Timing Constraint-driven Technology Mapping for FPGAs Considering False Paths and Multi-Clock Domains [*]

Lei Cheng, Deming Chen, Martin D.F. Wong
Univ. of Illinois at UC, Champaign, IL USA
{lcheng1,dchen,mdfwong}@uiuc.edu

Mike Hutton, Jason Govig
Altera Corp., San Jose, CA USA
{mhutton,jgovig}@altera.com

Abstract

Modern FPGA chips contain multiple dedicated clocking networks, because nearly all real designs contain multiple clock domains. In this paper, we present an FPGA technology mapping algorithm targeting designs with multi-clock domains such as those containing multi-clocks, multi-cycle paths, and false paths. We use timing constraints to handle these unique clocking issues. We work on timing constraint graphs and process multiple arrival/required times for each node in the gate-level netlist. We also recognize and process constraint conflicts efficiently. Our algorithm produces a mapped circuit with the optimal mapping depth under timing constraints. To the best of our knowledge, this is the first FPGA mapping algorithm working with multi-clock domains. Experiments show that our algorithm is able to improve circuit performance by 16.8% on average after placement and routing for a set of benchmarks with multi-cycle paths, comparing to a previously published depth-optimal algorithm that does not consider multi-cycle paths.

1. Introduction

Modern designs tend to have many different timing constraints, arising from multiple related (and not related) clock domains. They also have exceptions (can also be considered as constraints) such as multi-cycle, maximum path delay, and false-path exceptions [1]. A false path in a circuit is a path which cannot be activated by any input vector. Fig. 1 shows a false path example. In practice, designers can specify don't care conditions, such that some activated paths will become false paths under don't care conditions. A multi-cycle path in a circuit is a path that does not have to propagate signals in single clock cycle. A false path can be treated as a path under a clock domain with a multi-cycle timing constraint of ∞. A multi-cycle path and a single-cycle regular path can be seen to function with related clocks under two clock domains [1]. Logic synthesis with multiple clock domains brings up new challenges for optimization while trying to fulfill these complicated timing constraints.

Researchers have proposed algorithms to identify false paths using various path sensitization methods [2, 3, 4, 5, 6, 7]. There are also some algorithms to identify multi-cycle paths in a circuit. In [8], the authors presented a method to detect multi-cycle paths based on the analysis of the state transition graph of the controller of a microprocessor. In [9], the authors provided a multi-cycle path detection algorithm based on symbolic state traversal of finite state machines. The multi-cycle paths can also be detected using a SAT-based method [10]. Timing constraints due to multi-clock domains are heavily used in static timing analysis (STA). Most literatures to date focus on analysis with false path constraints [11, 12, 13]. In [14], the authors proposed a unified framework for STA considering both false path and multi-cycle path timing constraints. However, all of these STA algorithms may suffer an exponential runtime for a node involved in multiple timing constraints. In [1], the authors proposed an efficient STA method using the edge mask data structure, in which each edge has two associated vectors representing timing information from the source and destination direction respectively. In this paper, we use a method similar to [1] for timing information propagation.

[*]This work is partially sponsored by Altera Corporation through a research grant. We used machines donated by Intel.

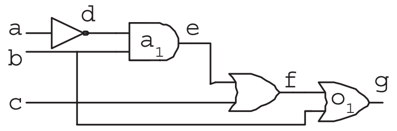

Figure 1: Example of a false path. In this example, the path $adefg$ is a false path. For this path to be active, b is required to be 1 at the gate a_1, and b is also required to be 0 at the gate o_1, which is not possible.

In this work, we design a technology mapping algorithm with multi-clock domain consideration for FPGAs because modern FPGA chips contain multiple dedicated clocking networks. For example Altera Stratix II devices [15], contain 16 dedicated full-chip global clocks and 32 regional clock lines. These clocks can be controlled either by pins or internal signals, or generated by on-chip PLLs from other clock signals. FPGA technology mapping converts a given Boolean network into a functionally equivalent network comprised only of LUTs. It is a critical synthesis step in the FPGA design flow.

Through communication with FPGA vendors, we are aware that synthesis with multi-clock domains and the corresponding timing constraints are very important to FPGA customers. Timing constraints can also be specified in commercial FPGA tools, such as Altera Quartus II [16] and Xilinx ISE [17]. However, we are not aware of any work on FPGA technology mapping with multi-clock domains. Previous depth-optimal FPGA technology mapping algorithms are all working with a single clock domain [18, 19, 20, 21]. In this work, we propose an algorithm targeting FPGA designs under multi-clock domains. We significantly extend the cut-enumeration-based mapping framework to carry out the multi-clock-domain mapping process. We work on timing constraint graphs and process multiple arrival/required times for each node in the gate-level netlist. We also recognize and process constraint conflicts efficiently. Our algorithm produces a mapped circuit with the optimal mapping depth under multi-clock timing constraints. In addition, due to the lack of real benchmarks with multi-clock domains, we develop a circuit model to incorporate multi-cycle paths and designed twenty such benchmark circuits using the twenty largest MCNC benchmarks as the base. These benchmarks just serve as the evaluation instrument, and our algorithm is general enough to handle other multi-clock-domain criteria, such as maximum path delay and false paths. Experiments show that our algorithm is able to improve circuit performance by 16.8% on average, comparing to a previously published depth-optimal algorithm that does not consider multi-cycle paths.

The rest of this paper is organized as follows. We provide the problem formulation and some related definitions in Section 2. In Section 3, we present the details of our FPGA technology mapping algorithm. The results are shown in Section 4, and we conclude this paper in Section 5.

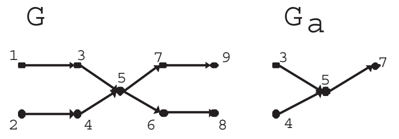

Figure 2: Graph G represents a circuit. Graph G_α is a timing constraint graph for timing constraint α. $B(G_\alpha) = \{3,4\}$, $E(G_\alpha) = \{7\}$. The path $3 \rightarrow 5$ is a prefix path of G_α. The path $5 \rightarrow 7$ is a suffix path of G_α. The path $4 \rightarrow 5 \rightarrow 7$ is a complete path of G_α. The complete path $1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 9$ of G is a path with timing constraint α, because it covers a complete path $3 \rightarrow 5 \rightarrow 7$ of G_α, while the complete path $1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 8$ of G is a regular path.

2. Problem Formulation and Definitions

A Boolean network can be represented by a DAG where each node represents a logic gate, and a directed edge (i, j) exists if the output of gate i is an input of gate j. A PI node has no incoming edges and a PO node has no outgoing edges. We treat the flip-flop outputs as special PIs and the flip-flop inputs as special POs, and make no distinction in terms of notation. We use $input(v)$ to denote the set of nodes which are fanins of gate v. Given a Boolean network N, we use O_v to denote a *cone* rooted on node v in N. O_v is a subnetwork of N consisting of v and some of its predecessors, such that for any node $w \in O_v$, there is a path from w to v that lies entirely in O_v. The cone O_v is K-feasible if the size of $input(O_v)$ is not larger than K, where $input(O_v)$ denotes the set of distinct nodes outside O_v which supply inputs to the gates in O_v. A *cut* C is a partitioning (X, X') of a cone O_v such that X' is a cone of v. Node v is the cut *root*. The *cut-set* of the cut, denoted $V(X, X')$ or $V(C)$, consists of the inputs of cone X', or $input(X')$. A cut C is K-feasible if $|V(C)| \leq K$, which means X' can be implemented by a K-LUT. A Boolean network is l-bounded if $|input(v)| \leq l$ for each node v. In this work, all initial networks are 2-bounded. If a network is not 2-bounded, we can transfer it into a 2-bounded network using gate decomposition.

We assume that the timing model of a circuit is a directed acyclic graph $G = \{V, E\}$, where V is a set of vertices and E is a set of edges (see Fig. 2 [14] for an example of timing constraint graph and related definitions). The edges of the graph are associated with delays. The *begin set* $B(G)$ is a set of vertices in G which have no incoming edges. The *end set* $E(G)$ is a set of vertices in G which have no outgoing edges. A *prefix path* of G is a path starting from a vertex in $B(G)$, and a *suffix path* of G is a path that ends at a vertex in $E(G)$. A *complete path* of G is both a prefix path and a suffix path of G. A timing constraint α is specified by a constraint type and a timing constraint graph G_α, where G_α is a subgraph of G. A complete path of G is a path of timing constraint α if and only if it covers a complete path of G_α. The constraint type of a timing constraint specifies whether the timing constraint is a false-path constraint or a multi-cycle constraint with exact number of cycles. Our algorithm assumes timing constraint graphs are given.

We use a widely accepted unit delay model [18], where each LUT on a path contributes one unit delay. If we treat regular paths as a special case of multi-cycle paths with one clock cycle requirement, and false paths as a special case of multi-cycle paths with infinity clock cycle requirement, the optimum mapping depth of a 2-bounded circuit N under timing constraints is the minimum number d_o such that there is a mapped circuit of N, in which the depth of every constraint path is limited by $d_o \cdot cycles$, where $cycles$ is the clock cycle requirement of the timing constraint of the path. The mapping problem for depth-optimal FPGA with multi-clock domains is to cover a given 2-bounded Boolean network with K-feasible cones, or equivalently, K-LUTs in such a way that the optimal mapping depth is guaranteed under timing constraints due to multi-clock domains.

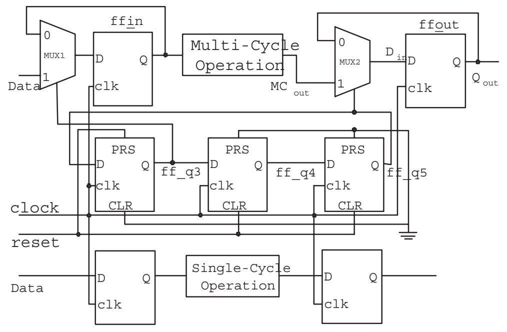

Figure 3: Example of a multi-cycle timing constraint.

3. FPGA Technology Mapping with Timing Constraints

3.1 Overall Algorithm

In previous cut-enumeration-based delay optimum FPGA technology mapping algorithms [18, 19, 20, 21], minimum mapping depth is computed at the end of the cut enumeration process. This minimum mapping depth is assigned to be the required time for each PO, and required times are propagated from POs to PIs in the cut selection phase. In our algorithm considering timing constraints, there are multiple arrrival/required times on a node. Our algorithm takes a circuit and timing constraints in SDC (Synopsys Design Constraints) format as inputs. We convert every timing constraint into a timing constraint graph, and generate timing constraint cores for all constraints (see Section 3.6 for definition of timing constraint cores). Then different arrival times corresponding to different timing constraints are propagated from PIs to POs in the topological order. At the end of cut enumeration, we compute the optimum normalized mapping depth (see Section 3.5 for definition). Then, we set different required times on the POs according to the various timing constraints applied on them. When we propagate required times from POs back to PIs in the reverse topological order, we choose the cut for each node that fulfills all the timing requirements and has a small cost. The details will be provided in the rest of this section. In Section 3.2, we present a multi-cycle circuit model that is used to generate testing circuits in our experiments; in Section 3.3, we introduce our data structure; in Section 3.4, we present the basic procedure of propagating constraint delays during cut enumeration; in Section 3.5, we present how to carry out cut selection and generate mapping solution for a node considering various constraint delays; in Section 3.6, we introduce the concept of constraint cores that can help to resolve constraint conflicts; and we provide analysis of our algorithm in Section 3.7.

3.2 A Multi-Cycle Circuit

Fig. 3 shows a circuit model with multi-cycle paths [9] to serve the purpose of evaluating our algorithm. We generate our testing circuits using this model, because we cannot access any real circuit with timing constraints. Even though this circuit model only shows multi-cycle timing constraints, our algorithm is general enough and will work for any type of circuits with multi-clock domains. The circuit model can also help to understand why there are multiple arrival/required times on one node when various timing constraints are considered. The upper part of the figure shows the data path, and the middle part is the control flip-flops. The initial state of the control flip-flops after the reset signal is (ff_{q3},ff_{q4},ff_{q5})=$(1,0,0)$, and these flip-flops change as $(0,1,0), (0,0,1), (1,0,0), (0,1,0), (0,0,1), \ldots$ synchronized with the clock signal. The gate $MUX1$ selects input from pin $Data$ only when $ff_{q3} = 1$, and the gate $MUX2$ selects input from pin MC_{out} only when $ff_{q5} = 1$. It is easy to know that the combinatorial path from *ff_in* to *ff_out* is a two-cycle path. Fig. 4 shows a decomposition of

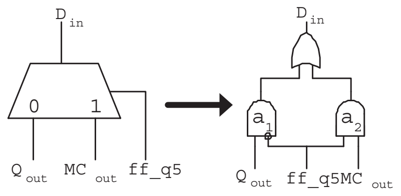

Figure 4: Decomposition of gate MUX2 in Fig. 3. In this example, both AND gate a_2 and the OR gate cross two different timing constraints, one is for regular paths, and the other is for multi-cycle paths. The AND gate a_1 has only one arrival/required time. The AND gate a_2 and the OR gate both have two arrival/required times.

$MUX2$ in Fig. 3. In Fig. 4, the AND node a_2 has two arrival times, one from ff_q5 which comes from a regular path, and the other from MC_{out} which comes from a two-cycle path. The OR gate in Fig. 4 also has two arrival times, but the AND gate $a1$ only has one arrival time for regular paths. Similarly, both the AND gate a_2 and the OR gate have two required times, and the AND gate $a1$ has only one required time.

3.3 Data Structure

When we consider many timing constraints in the technology mapping algorithm, different nodes from the same cut set may have different sets of timing constraints. In Fig. 5(a), for example, both nodes v_1 and v_2 belong to the cut set of the cut C_1. Node v_1 is not covered by any timing constraint graph, but node v_2 is contained by a timing constraint graph G_α. However, the fact that a cut node is covered by a constraint graph does not imply that there always exists a constraint path going through this cut node and the cut root. In Fig. 5(a), node v_3 is covered by the constraint graph G_α, but the cut root u_2 is not covered by any constraint graph, so any path through nodes v_3 and u_2 is a regular path.

With above observations, we design the following data structure in our algorithm. Given a cut C rooted on a node u, for each node $v \in V(C)$ (note that $V(C)$ is the cut set of C), there is a *timing set* $\mathcal{D}(C,v)$, and each item, named *delay item*, in a timing set $\mathcal{S} \in \mathcal{D}(C,v)$ provides the information of a constraint path going through both v and u. More specifically, $\mathcal{S}$ is a triple $\mathcal{S} =< \alpha, f, d >$, where α identifies which timing constraint the path goes through ($\alpha = 0$ for regular paths); f is a flag indicating whether the path contains a complete constraint path ($f = f_c$), or a prefix constraint path ($f = f_p$) so far; and d is the arrival time in the corresponding timing constraint. Given a timing set $\mathcal{D}(C,v)$, let $f_S(\alpha, C, v)$ denote the delay item

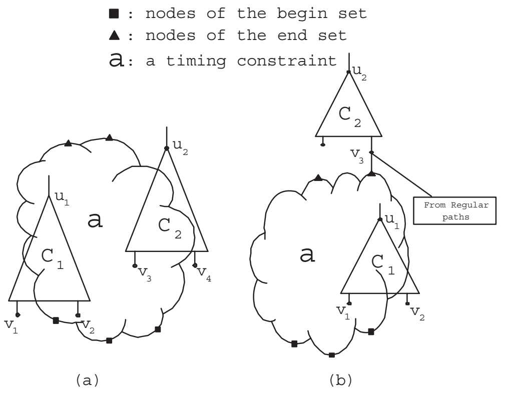

Figure 5: Examples for data structure illustration.

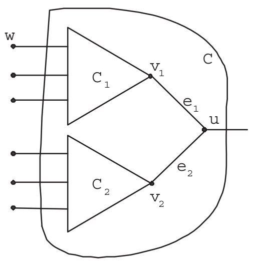

Figure 6: Cut C is generated by combining C_1 and C_2.

corresponding to α, and $f_a(\alpha, C, v)$, $f_f(\alpha, C, v)$, $f_d(\alpha, C, v)$ denote the timing constraint identity field, flag field, and arrival time field of $f_S(\alpha, C, v)$, respectively. If $\mathcal{D}(C,v)$ do not have an item corresponding to a timing constraint α, then $f_S(\alpha, C, v)$, $f_a(\alpha, C, v)$, $f_f(\alpha, C, v)$ do not exist, and $f_d(\alpha, C, v)$ is infinity. Note that the triple form of a delay item, $< \alpha, f, d >$, is used to construct a delay item, while $f_S(\alpha, C, v)$ is used to select a delay item from $\mathcal{D}(C,v)$. In Fig. 5(b), for cut C_1 rooted at node u_1, $\{v_1, v_2\} \subseteq V(C_1)$, we have $\mathcal{D}(C_1, v_1) = \{< \alpha, f_p, x_1 >\}$ and $\mathcal{D}(C_1, v_2) = \{< 0, , x_2 >\}$(for regular paths, we do not care about the flag field), where x_1 and x_2 are some delay values. For cut C_2 rooted at node u_2, $v_3 \in V(C_2)$, we have $\mathcal{D}(C_2, v_3) = \{< \alpha, f_c, x_3 >, < 0, , x_4 >\}$ since there are both complete paths of timing constraint α and regular paths going through v_3 to u_2. Given $\mathcal{D}(C_2, v_3) = \{< \alpha, f_c, x_3 >, < 0, , x_4 >\}$, we have $f_S(\alpha, C_2, v_3) =< \alpha, f_c, x_3 >$, $f_S(0, C_2, v_3) =< 0, , x_4 >$, $f_d(\alpha, C_2, v_3) = x_3$, and $f_d(0, C_2, v_3) = x_4$.

3.4 Arrival Time Generation

Timing sets are computed along cut enumeration. For a PI node u, it only has one trivial cut C, and $V(C) = \{u\}$. If the PI node u belongs to both the begin set and the end set of a timing constraint α, then $< \alpha, f_c, 0 >\in \mathcal{D}(C,u)$; if u only belongs to the begin set of a timing constraint α, then $< \alpha, f_p, 0 >\in \mathcal{D}(C,u)$; if u does not belong to the begin set of any timing constraint, then $\mathcal{D}(C,v) = \{< 0, , 0 >\}$.

For an internal node u with two fanins v_1 and v_2, a cut C rooted at u is generated by combining one cut C_1 rooted at v_1 and another cut C_2 rooted at v_2. We only process the case when C is K-feasible. Let e_1 denote the edge from v_1 to u, and e_2 denote the edge from v_2 to u (see Fig. 6). Since we use the topological order traversal, the timing sets of $V(C_1)$ and $V(C_2)$ are known when we combine cuts C_1 and C_2 to from cut C. The timing sets of $V(C)$ are either computed from those of $V(C_1)$ and $V(C_2)$, or generated by the cut itself when u belongs to the begin set of some timing constraints, as shown by Algorithm 1. For any node $w \in V(C_1)$, it is also true that $w \in V(C)$. Initially, timing set $\mathcal{D}(C,w)$ is empty. Algorithm 2 examines every delay item $\mathcal{S} =< \alpha, f, d >\in \mathcal{D}(C_1, w)$. For each delay item $\mathcal{S} =< \alpha, f, d >$, if $\alpha = 0$ (corresponding to regular paths), we add delay item $< \alpha, f, d >$ to $\mathcal{D}(C,w)$. If $f = f_c$, there exists a path from a PI to v_1 through w which covers a complete path of timing constraint α, and we also add $< \alpha, f_c, d >$ to $\mathcal{D}(C,w)$. If $f = f_p$ and e_1 does not belong to timing constraint α, there is no path from a PI to u through w that covers a complete path of α, so we add $< 0, , d >$ to $\mathcal{D}(C,w)$. If $f = f_p$, and e_1 is an edge of the timing constraint α, we add $< \alpha, f_p, d >$ to $\mathcal{D}(C,w)$. Furthermore, if u belongs to the end set of the timing constraint α, we change flag f_p to f_c. After we process all the nodes in $V(C_1)$, we process nodes in $V(C_2)$ in a similar way in Algorithm 1. At the end of Algorithm 1, we check whether u belongs to the begin set of a timing constraint. If u belongs to the begin set of a timing constraint β, we will add $< \beta, f_p, d >$ to $\mathcal{D}(C,w)$ for all $w \in V(C)$, where d is the maximum of all delay values in $\mathcal{D}(C,w)$, and we also remove the delay item corresponding to the regular paths from $\mathcal{D}(C,w)$ (note that the regular path timing information may be regenerated later if there is a prefix path going out of the timing constraint graph

Algorithm 1: Generate timing sets for a cut

Input: C, C_1, C_2 : three cuts, C is formed by combining C_1 and C_2
Output: timing sets

```
begin
    $u \leftarrow$ root of $C$;
    for $w \in V(C)$ do
        $\mathcal{D}(C, w) \leftarrow \emptyset$;
    end
    /*timing sets propagation*/
    for $w \in V(C_1)$ do
        propogate_timing_set$(w, e_1, C, C_1)$;
    end
    for $w \in V(C_2)$ do
        propogate_timing_set$(w, e_2, C, C_2)$;
    end
    /*timing sets generation by the cut itself*/
    for timing constraints $\beta$ satisfying $u \in B(G_\beta)$ do
        for $w \in V(C)$ do
            $d \leftarrow 0$;
            for $< \beta, f, d' > \in \mathcal{D}(C, w)$ do
                if $d' > d$ then $d = d'$;
            end
            $f = f_p$;
            if $u \in E(G_\beta)$ then $f = f_c$;
            $\mathcal{D}(C, w) \leftarrow \{< \beta, f, d >\} \cup \mathcal{D}(C, w)$;
            Remove delay_item $f_S(0, C, w)$ from $\mathcal{D}(C, w)$;
        end
    end
end
```

Algorithm 2: propogate_timing_set(w, e, C, C')

```
begin
    $u \leftarrow$ root of $C$;
    for $< \alpha, f, d > \in \mathcal{D}(C', w)$ do
        if $\alpha = 0$ or $f = f_c$ then
            add_delay_item$(< \alpha, f, d >, \mathcal{D}(C, w))$;
        else
            /*$f = f_p$*/
            if $e \notin E(G_\alpha)$ then
                add_delay_item$(< 0, , d >, \mathcal{D}(C, w))$;
            else
                if $u \in E(G_\alpha)$ then $f \leftarrow f_c$;
                add_delay_item$(< \alpha, f, d >, \mathcal{D}(C, w))$;
            end
        end
    end
end
```

Algorithm 3: add_delay_item$(< \alpha, f, d >, \mathcal{D}(C, w))$

```
begin
    if $f_S(\alpha, C, w)$ exists then
        if $f_d(\alpha, C, w) > d$ then
            $d \leftarrow f_d(\alpha, C, w)$;
        Remove $f_S(\alpha, C, w)$ from $\mathcal{D}(C, w)$;
    end
    $\mathcal{D}(C, w) \leftarrow \mathcal{D}(C, w) \cup \{< \alpha, f, d >\}$;
end
```

without going through end set nodes of timing constraint β). When u belongs to the begin set of a timing constraint, we also check whether u belongs to the end set of the timing constraint. If u belongs to both the begin set and the end set of a timing constraint, we change the flag from f_p to f_c in the above process.

After we compute the timing sets related to all the cuts of a node u, we will compute the arrival times of different timing constraints on u. We assign a timing set $\mathcal{D}_n(u)$ to node u, which is very similar to the timing set we defined before. Let $\mathcal{C}(u)$ denote all the K-feasible cuts rooted at u. For every timing constraint α that covers u, we add $< \alpha, f, d_\alpha >$ to $\mathcal{D}_n(u)$, where $d_\alpha = MIN\{MAX\{f_d(\alpha, C, w) \mid w \in V(C)\} \mid C \in \mathcal{C}(u)\}$ and f is equal to any $f_f(\alpha, C, w)$ if $f_S(\alpha, C, w)$ exists (we can prove that $f_f(\alpha, C, w)$ is always the same for any $C \in \mathcal{C}(u)$ and $w \in V(C)$ if $f_S(\alpha, C, w)$ exists). In a cut-enumeration-based algorithm, there is a trivial cut C' for each node u, and $V(C') = \{u\}$. We have described how to generate timing set for the trivial cut of a PI node. For an internal node u, $\mathcal{D}(C', u) = \{< \alpha, f, d{+}1 > \mid \forall < \alpha, f, d > \in \mathcal{D}_n(u)\}$, because a trivial cut always starts a new level in cut-enumeration-based algorithms.

It is possible that one path belongs to multiple timing constraints, which introduces timing constraint conflicts. We will talk about how to handle this issue in our algorithm in Section 3.6.

3.5 Mapping Solution

At the end of the cut enumeration, we know all the timing information for every timing constraint. For a PO node u, we check every delay item $\mathcal{S} =< \alpha, f, d >$ from its timing set $\mathcal{D}_n(u)$. If $\alpha = 0$, there is at least one regular path from a PI node to u; if $\alpha \neq 0$ and $f = f_c$, then there is at least one path from a PI node to u containing one complete path of the timing constraint α. We define a *normalized depth* for each delay item. If $\alpha = 0$ (or $\alpha \neq 0$ and $f = f_p$), the normalized depth is d. If $\alpha \neq 0$ and $f = f_c$, the normalized depth is $\lceil d/cycles \rceil$, where *cycles* is the number of cycles allowed in the multi-cycle path constraint α, and *cycles* is infinity for a false-path constraint. The *optimum normalized mapping depth* is the maximum of all normalized depths. For illustration purpose, we can think of optimum normalized mapping depth as the minimum clock period of the mapped circuit under unit delay model. For a regular path with depth d, the clock period should be equal to or larger than d to satisfy the path timing. For a multi-cycle path with depth d, the clock period should be equal to or larger than $\lceil d/cycles \rceil$ to satisfy the path timing.

Same with the previous cut enumeration based technology mapping algorithm [18], we carry out cut selection procedure in the reverse topological order from POs to PIs. However, we set different required times on the POs according to the various timing constraints applied on them. Let d_o denote the optimum normalized mapping depth. The POs with regular paths will have a required time with value of d_o for regular paths. The POs with a timing constraint α will have a required time with value of $d_o * cycles$ for the timing constraint α. The required times will be propagated from POs to PIs in the cut selection stage. To pick which cut to map a node is the key for high mapping quality. Different required times due to different timing constraints offer opportunities to save mapping cost under the timing constraints. In our algorithm, when we map a node u, we will examine all the cuts on u and try to find the best one that fulfills all the timing requirements and also reduces the mapping cost. The mapping cost of a cut is computed in a similar way as that in [18]. To make sure the timing constraints are met, each input on the examined cut needs to be checked against all of its required times associated with its timing constraints. After the best cut is picked, required times on u will be propagated to the inputs of the cut according to the timing constraints on the inputs respectively.

3.6 Timing Constraint Cores

There is a *constraint conflict* if there is a path that belongs to more than one timing constraints. However, different timing constraints should have different priorities. If a path belongs to multiple timing constraints, we should only consider the timing constraint with the highest priority. However, we cannot examine paths one by one to determine whether a path belongs to multiple timing constraints due to exponential number of paths. It is also very hard to determine whether multiple arrival times at a PO come from the same path or different paths. Thus, it is very important to design an effective method to handle this problem. In this subsection, we introduce a concept called *timing constraint core* to resolve constraint conflicts.

Definition 1. *Given a timing constraint α, the timing constraint core of α, $K(\alpha)$, is the set of nodes that cannot be reached without going through $B(G_\alpha)$ and $E(G_\alpha)$ from outside of G_α.*

With this definition, we immediately have the following theorem:

Theorem 1. *Given a timing constraint α, every complete path going through one node of $K(\alpha)$ is a path of the timing constraint α.*

For a node u, we define the core timing constraint of u as follows:

Definition 2. *Given a node u, its core timing constraint, $K_n(u)$, is the timing constraint α with the highest priority satisfying $u \in K(\alpha)$. If u does not belong to any timing constraint core, $K_n(u)$ does not exist.*

During the cut enumeration process, when we process a node u, we filter out all the timing information related to timing constraints with lower priorities than $K_n(u)$, if $K_n(u)$ exists. By filtering out low priority timing constraints, we automatically resolve constraint conflicts among low priority timing constraints and $K_n(u)$. It is possible that there is a high priority timing constraint graph G_β covering u. In this case, it is true that $u \notin K(\beta)$, otherwise $K_n(u) \neq \alpha$. There must exist a complete path going through u that does not cover a complete path of β, but covers a complete path of α. So, we need to propagate timing information for both α and β when we process u.

3.7 Complexity and Optimality

Regarding to the runtime of our algorithm, we have the following theorem:

Theorem 2. *The runtime of processing a cut is linear to the number of total timing constraints.*

With this theorem, we know that the runtime of our algorithm is increased by at most n_t times, where n_t is the total number of timing constraints, compared to the previous algorithm [18] without considering timing constraints.

Furthermore, we can show that our algorithm is able to produce optimum normalized mapping depth as defined in Section. 3.5.

Theorem 3. *Our FPGA technology mapping algorithm is able to produce a mapped circuit with the optimum mapping depth when timing constraints are considered.*

4. Experimental Results

Our experiments are carried out on a desktop PC with a 2.4 GHz Intel(R) Xeon(TM) CPU. The OS is Red Hat Linux 8.0, and we use gcc to compile our program. Because we cannot get access to any real circuit with timing constraints, we designed a new set of benchmark circuits from the 20 largest MCNC benchmark circuits. We use the template in Fig. 3 to derive our benchmark circuits. We randomly pick two circuits from the 20 largest benchmark circuits, and place the circuit with larger mapping depth into the multi-cycle part of Fig. 3, and place the other circuit into the single-cycle part. In this way, we generate 20 new benchmark circuits, run both our algorithm and DAOmap on these circuits, and report the results. Note that we compare with DAOmap because DAOmap is a recent depth-optimal mapping algorithm without resynthesis (i.e., without changing the original circuit structure during mapping). We believe our algorithm can be extended to work on other mapping frameworks, such as the ABC mapper [21], where resynthesis choices are carried out. This will be considered in our future work.

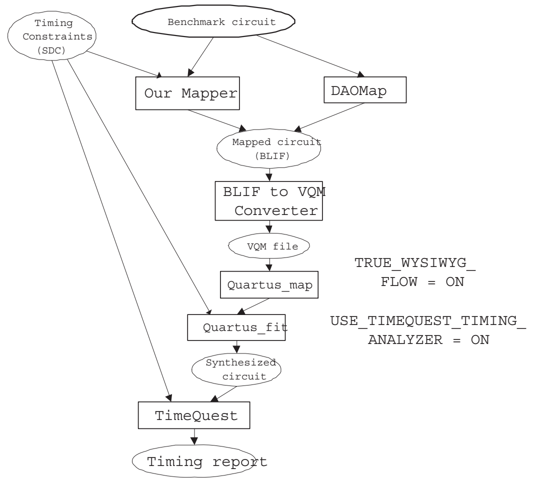

Figure 7: Experimental results.

We use $K = 5$ for all circuits in the experiment. The mapped circuits are in Berkeley Logic Interchange Format (BLIF). Fig. 7 shows our experimental flow. In order to use the commercial timing analysis engine, TimeQuest Timing Analyzer (available from Altera's Quartus II software), we write a converter from BLIF format to Verilog Quartus Mapping (VQM) format. We build a project for each VQM file. We set TRUE_WYSIWYG_FLOW to ON so that Quartus II will not optimize and remap our circuits, and we set USE_TIMEQUEST_TIMING_ANALYZER to ON so that Quartus II will consider timing constraints during placement and routing stage. We use Synopsys Design Constraints Format (SDC) to specify timing constraints, and the multi-cycle part in Fig. 3 is set to have a multi-cycle timing constraint of two clock cycles. We use the same Stratix II device for both DAOmap and our algorithm. We run Quartus II commands quartus_map [1], and quartus_fit to synthesize, place and route the circuits. Then, we use the TimeQuest Timing Analyzer to report timing information for all circuits. In order to find the minimum clock period of a mapped circuit, we run the experimental flow for multiple iterations, and we set a different clock period value in each iteration (using the SDC command create_clock). For each iteration, the TimeQuest Timing Analyzer reports whether the clock period is achievable or not. If the specified clock period is not achievable under the multi-cycle timing constraints, we increase the clock period value in a step of 0.5ns, and repeat the flow again. If the specified clock period is achievable, we decrease the clock period value also in a step of 0.5ns, and run the flow. The initial clock period is 6ns. After a few iterations, we will know the minimum clock period of the mapped circuit achievable after placement and routing.

Table 1 shows our experimental results. The first column shows two circuits of the 20 largest MCNC benchmarks we used to generate a circuit with timing constraints. The first circuit corresponds to the single cycle part, and the second circuit corresponds to the multi-cycle part of the new circuit. The column PIs (POs) shows the numbers of primary inputs (outputs) in our new circuits. The column $LUTs_{DAO}$ ($LUTs_{our}$) shows the numbers of LUTs of the mapped circuits produced by DAOmap (our mapper). The column clk_{DAO} (clk_{our}) shows the minimum clock periods achievable for the mapped circuits by DAOmap (our mapper). The column $Impr_{perf}$ shows the performance improvements of our algorithm over DAOmap. Since the circuit performance is measured by the clock frequency, the reciprocal of the clock period, the performance improvement is computed by

[1]The command quartus_map is used to prepare the input files for later process. It does not optimize or remap our circuits since we set TRUE_WYSIWYG_FLOW to ON.

Table 1: Experimental Data.

Circuit	PIs	POs	$LUTs_{our}$	$LUTs_{DAO}$	$clk_{our}(ns)$	$clk_{DAO}(ns)$	$Impr_{perf}(\%)$
alu4+clma	60	11	4499	4685	6	7.5	25
alu4+diffeq	254	58	1695	1691	5.5	6.5	18.18
alu4+tseng	219	129	1658	1646	5.5	7	27.27
apex2+s298	50	17	2558	2554	6.5	8	23.08
apex2+tseng	475	509	2681	2666	6.5	7.5	15.38
apex4+elliptic	815	125	2566	2587	6	6.5	8.33
apex4+frisc	338	138	1812	1804	6	6.5	8.33
des+clma	351	346	6380	6524	6.5	8.5	30.77
ex1010+tseng	215	131	4126	4049	7	8	14.29
ex5p+diffeq	248	113	1447	1425	6	7	16.67
ex5p+elliptic	814	170	2404	2435	6	6.5	8.33
misex3+diffeq	455	427	2523	2519	5.5	6.5	18.18
misex3+tseng	219	135	1603	1584	5.5	6	9.09
pdc+tseng	453	546	4560	4514	8	9	12.5
seq+diffeq	482	448	2716	2706	5.5	6.5	18.18
seq+tseng	246	156	1796	1771	5.5	6.5	18.18
spla+clma	62	49	6162	6341	7	8	14.29
spla+diffeq	256	96	3330	3347	7	8	14.29
spla+s298	28	60	3954	3970	7	8.5	21.43
spla+tseng	453	552	4077	4079	7.5	8.5	13.33
Average			3127	3145			16.76

the formula $\frac{1/clk_{our}-1/clk_{DAO}}{1/clk_{DAO}}$. From the table, we know that our algorithm is able to improve circuit performance by 16.8% on average. The reason that our algorithm is able to achieve better circuit performance is very intuitive. Our mapping algorithm is able to map the multi-cycle parts of a circuit with larger depths, and map the normal parts with shorter depths, while previous algorithms do not have such a capability. In previous cut-enumeration-based algorithms, timing constraint paths and regular paths have no difference when they are used to compute the optimum mapping depth, so the optimum mapping depth computed by previous algorithms is generally larger than that computed by our algorithm. Since this optimum mapping depth is set as the required time of the circuit, previous algorithms tend to map a regular path with a larger depth than our algorithm. As a result, they are likely to produce a mapped circuit with larger clock period after placement and routing.

5. Conclusions

In this paper, we presented an FPGA technology mapping algorithm considering multi-clock domains. We developed a method that could propagate timing information for various timing constraints through the network. We worked on timing constraint graphs and processed multiple arrival/required times for each node in the network. We also introduced a concept called timing constraint core, which could be used to resolve constraint conflicts. Our algorithm could produce a mapped circuit with the optimal mapping depth under multi-clock timing constraints. Compared to a previous FPGA technology mapping algorithm that did not consider timing constraints, our algorithm is able to improve circuit performance by 16.8% on average after placement and routing.

6. References

[1] M. Hutton et al. Efficient static timing analysis and applications using edge masks. In *FPGA*, pages 174 – 183, 2005.

[2] J. Benkoski et al. Timing verification using statically sensitizable paths. *TCAD*, 9(10):1073–1084, 1990.

[3] D.H.C. Du, S.H.C. Yen, and S. Ghanta. On the General False Path Problem in Timing Analysis. In *DAC*, pages 555 – 560, 1989.

[4] S. Perremans, L. Claesen, and H. De Man. Static timing analysis of dynamically sensitizable paths. In *DAC*, pages 568 – 573, 1989.

[5] P.C. McGeer and R.K. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *DAC*, pages 561 – 567, 1989.

[6] D. Brand and V.S. Iyengar. Timing analysis using functional analysis. Technical report, BM Thomas J. Watson Res. Center, 1986.

[7] H.-C. Chen and D.H.C. Du. Path sensitization in critical path problem. *TCAD*, 12(2):196–207, 1993.

[8] A.P. Gupta and D.P. Siewiorek. Automated Multi-Cycle Symbolic Timing Verification of Microprocessor-based Designs. In *DAC*, pages 113 – 119, 1994.

[9] K. Nakamura et al. Waiting false path analysis of sequential logic circuits for performance optimization. In *ICCAD*, pages 392 – 395, 1998.

[10] K. Nakamura et al. Multi-clock path analysis using propositional satisfiability. In *ASPDAC*, pages 81 – 86, 2000.

[11] K.P. Belkhale and A.J. Suess. Timing analysis with known false sub graphs. In *ICCAD*, pages 736 – 739, 1995.

[12] E. Goldberg and A. Saldanha. Timing analysis with implicitly specified false paths. In *VLSI Design*, pages 518 – 522, 2000.

[13] D. Blaauw, R. Panda, and A. Das. Removing user-specified false paths from timing graphs. In *DAC*, pages 270 – 273, 2000.

[14] S. Zhou et al. Efficient static timing analysis using a unified framework for false paths and multi-cycle paths. In *ASP-DAC*, pages 24 – 27, 2006.

[15] Altera StratixII Device. [online]http://www.altera.com/products/devices/stratix2/st2-index.jsp

[16] Altera Quartus II Software. [online]http://www.altera.com/products/software/products/quartus2/qts-index.html

[17] Xilinx ISE Software. [online]http://www.xilinx.com/ise/logic_design_prod/foundation.htm

[18] D. Chen and J. Cong. DAOmap: A Depth-optimal Area Optimization Mapping Algorithm for FPGA Designs. In *ICCAD*, Nov. 2004.

[19] J. Lamoureux and S. J. E. Wilton. On the Interaction between Power-Aware CAD Algorithms for FPGAs. In *ICCAD*, 2003.

[20] V. Manohararajah, S.D. Brown, and Z.G. Vranesic. Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping. *TCAD*, 25(11):2331 – 2340, 2006.

[21] A. Mishchenko, S. Chatterjee, and R.K. Brayton. Improvements to Technology Mapping for LUT-Based FPGAs. *TCAD*, 26(2):240–253, 2007.

[22] P. Ashar, S. Dey, and S. Malik. Exploiting multicycle false paths in the performance optimization of sequential logic circuits. *TCAD*, 14(9):1067–1075, 1995.