# Robust Optimization of Multiple Timing Constraints

Michael Wainberg and Vaughn Betz, *Member, IEEE*

***Abstract*—Modern Field-Programmable Gate Array (FPGA) circuit designs often contain multiple clocks and complex timing constraints, and achieving these constraints requires timing optimization at all stages of the CAD flow. To our knowledge, no prior published work has either described or quantitatively evaluated how to compute connection timing criticalities for circuits with multiple timing constraints in order to best guide CAD optimization algorithms. While single-clock techniques have a simple extension to multi-clock circuits, this formulation is not robust for circuits with multiple constraints of different magnitudes, or impossible constraints. We describe a robust method of timing optimization for circuits with multiple timing constraints, implemented in the open-source VPR (Versatile Place and Route) FPGA CAD tool. Our formulation can optimize multiple constraints well, even in the case where some constraints are impossible, and achieves over 20% greater clock speed with aggressive constraints than a straight-forward extension of single-clock work.**

***Index Terms*—Circuit optimization, field programmable gate arrays, multi-clock circuits, timing analysis, timing constraint**

## I. Introduction

As the number of transistors in Field-Programmable Gate Arrays (FPGAs) continues to grow according to Moore's Law, there has been a corresponding increase in the size and complexity of FPGA circuit designs. This presents new challenges to the Computer-Aided Design (CAD) tools used to map designs onto FPGAs.

One complexity is that modern designs often include registers on many different clocks: for example, a typical DDR3 interface contains 5 to 7 clocks, and designs with multiple memory and serial interfaces may have dozens of clocks [1]. I/O ports add to the effective number of clocks in a design, since an I/O port is often equivalent to an off-chip register with additional delay through the I/O. We define a *clock domain* as a set of registers clocked on the same clock, or a set of input or output ports connected to registers on a common off-chip clock.

CAD tools must analyze the timing of all paths between registers or I/Os which have been given a *timing constraint* by the designer. A timing constraint is a desired maximum delay along any path between a particular pair of *source* and *sink* clock domains (for the purposes of this paper, hold-time analysis is neglected). Timing information is used in two ways: to guide circuit *optimization*, as circuit elements may be rearranged to reduce the delay of paths close to violating timing constraints; and to *analyze* the degree of attainment of timing constraints in the final circuit implementation to report to the designer.

Increasing the number of clocks and I/Os in the circuit leads to a corresponding growth in the number of timing constraints, since each clock domain will in general require a distinct timing constraint, as will any clock domain pairs employing synchronous data transfer between them. Multi-clock timing optimization is recognized as an important problem in industrial CAD tools, and at least one patented technique [2] exists for this task. However, we are not aware of any published research on multi-clock timing optimization, or any evaluations of the best way to perform it.

A robust timing optimizer should provide an accurate metric for the timing importance of each connection between circuit elements in the presence of a variety of timing constraints of different magnitudes, in order to guide the various steps of CAD optimization. Even early in the CAD flow, when delay estimates are imperfect, a robust optimizer should give a rough estimate of whether each connection can achieve its timing constraint.

Timing optimization should be robust even when given timing constraints which are impossible to achieve. A typical FPGA design workflow involves rapid iteration of design changes and CAD analysis, and some paths may initially be given impossible timing constraints, to be corrected later in the design process. CAD tools must avoid over-optimizing domains with impossible constraints at the cost of causing other domains to fail their timing constraints, as this obscures the root cause of the timing issue. This could lead designers to waste time redesigning portions of a circuit that can already meet timing, for instance by repipelining modules, rewriting combinational logic, introducing multicycle constraints, or instantiating multiple copies of a module. Robust timing optimization should let as few constraints fail as possible, and ensure that those which do are failing due to an actual timing issue with the design, rather than with the CAD optimizer.

In this paper, we describe a robust method of timing optimization for circuits with multiple timing constraints (*multi-clock circuits*), used in the open-source VPR (Versatile Place and Route) FPGA CAD tool [3]. We first describe a standard procedure for single-clock timing analysis and optimization. We then summarize a method for specifying multiple timing constraints, and timing analysis of multi-clock circuits. We then illustrate the complications which limit the robustness of straight-forward multi-clock timing optimization. Finally, we describe various implementations of multi-clock timing optimization which address these complications, and compare their performance at timing- and wirelength-driven optimization in VPR, as well as their efficacy at optimizing a mix of achievable and impossible constraints. We find a formulation which can optimize multiple constraints well, even in the case

Manuscript received [TODO]; revised [TODO]. The authors gratefully acknowledge the financial support of NSERC, Altera, SRC and Texas Instruments.

V. Betz is with the Department of Electrical and Computer Engineering, Toronto, ON (e-mail: vaughn@eecg.utoronto.ca).

Digital Object Identifier [TODO]

where some constraints are impossible. While our focus of study is FPGAs, this work is expected to also be applicable to Application-Specific Integrated Circuits (ASICs).

## II. Background: Single-clock timing analysis and optimization

### A. Timing analysis

Timing analysis uses a directed graph representing timing dependencies between connected circuit elements [4], [5]. Nodes denote input and output pins of registers, look-up tables (LUTs) and other primitive blocks; edges denote connections between circuit elements, and their weights represent the time delay between nodes.

In single-clock circuits, timing analysis is usually performed using a pair of breadth-first traversals. First, a forward breadth-first traversal is performed from timing graph source nodes (register outputs, chip inputs and memory outputs) to sink nodes (register inputs and chip outputs), to compute the *arrival time* at each node. The arrival time $T_{arr}(i)$ is the time after a clock edge at which node $i$'s value will stabilize, or equivalently the longest path to this node from any source node in its transitive fanin, and is calculated recursively as [6]:

$$T_{arr}(i) = max_{\forall j \in fanin(i)}\{T_{arr}(j) + delay(j,i)\}. \quad (1)$$

At source nodes, the base case is:

$$T_{arr}(source) = 0. \quad (2)$$

This equation captures the notion that the value of a node $i$ only stabilizes once the values of all nodes $j$ in its immediate fanin have stabilized.

Second, a backward breadth-first traversal is performed from sink nodes to compute the required arrival time, also called the *required time*, at each node. The required time $T_{req}(j)$ is the time after a clock edge at which node $j$'s value must stabilize, so that all registers in its transitive fanout are able to meet their setup time requirements. It is defined as the timing constraint minus the delay of the longest path from node $j$ to any sink node in its transitive fanout, and is calculated as:

$$T_{req}(i) = min_{\forall j \in fanout(i)}\{T_{req}(j) - delay(i,j)\}. \quad (3)$$

At sink nodes, the base case is:

$$T_{req}(sink) = cons \quad (4)$$

where $cons$ is the timing constraint.

From arrival and required times, a *slack* is calculated for each edge, representing the amount of delay which could be added to a connection before any path using the connection would violate the timing constraint. The *critical path delay* can then be defined as the delay of the lowest-slack path. Slack is equal to the difference between the required time and the arrival time, or more specifically, the difference between the required time at the end of a connection and the arrival time at its start, minus the delay along the connection. For a connection from i to j, then,

$$slk_{single-clock}(i,j) = T_{req}(j) - T_{arr}(i) - delay(i,j). \quad (5)$$

### B. Timing optimization

FPGA CAD tools use timing information to optimize the layout of a design on the chip by reducing the delay of more timing-critical paths. Timing analysis is interleaved with optimization during all stages of the physical CAD flow — clustering, placement and routing — so that the optimizers can obtain feedback on the results of their actions. Most optimization algorithms use either connection weights or net weights to optimize timing, for instance in placement [7]. While raw slacks are the most useful piece of information for the designer and are consequently given in the final timing *analysis*, timing *optimizers* often use a related metric called *criticality*, which quantifies the timing importance of each connection [8], [9]. Criticality can be used directly as a connection weight, or as input to a function that computes connection weights.

In general, criticality has been defined to satisfy two properties: it should increase with decreasing slack, since connections with lower slack are more important to optimize; and it should be normalized to between 0 and 1, as explained in Section IV-B.

A definition of criticality for single-clock circuits used in the Pathfinder FPGA router [10] and since widely adopted for other CAD phases such as placement and clustering [6] is the following:

$$crit_{single\text{-}clock\text{-}pathfinder}(i,j) = 1 - \frac{slk(i,j)}{D_{max}} \quad (6)$$

where $D_{max}$ is the critical path delay, or equivalently the maximum arrival time over all nodes. As required, criticality increases with decreasing slack; it can never be less than 0, since the maximum slack along any edge can never exceed $D_{max}$; and it can never be greater than 1, *provided that slack is non-negative*. For slack to be non-negative, the timing constraint, i.e. the required time at sink nodes, must be greater than or equal to $D_{max}$; in single-clock circuits, we may simply set the constraint to equal $D_{max}$.

A nearly equivalent formulation proposed by Maidee et al. [11], [12] normalizes by the maximum slack in the circuit, instead of the maximum delay:

$$crit_{single\text{-}clock\text{-}maidee}(i,j) = 1 - \frac{slk(i,j)}{max_{\forall i,j}\{slk(i,j)\}}. \quad (7)$$

Criticality has been enhanced by incorporating *path counting*, which considers all of the paths using a connection, not just the most critical one. Path counting was proposed by Kong [13] and used in Chen and Cong's SPCD algorithm [14], [15]. Other enhancements have been developed to account for process variation in the delay of each connection, using statistical methods [16], [17], [18].

Instead of calculating criticality directly, Ren et al. [19] try a variety of connection weights by repeatedly invoking timing analysis and placement optimization. They then choose the

weights which optimize a *figure of merit*, which they define as the total amount by which the slacks of sink nodes fall short of a designer-specified threshold. That is:

$$FOM = \sum_{t \in sink \,:\, slk(t) < threshold} slk(t) - threshold \quad (8)$$

When this threshold is 0, the figure of merit reduces to total negative slack.

This prior work has focused on the single timing constraint case, and as will be shown in Section IV, generalizing these equations to the multi-clock case raises new questions and complexities.

### C. Clock delay

Clock delay can be accounted for in timing analysis by subtracting clock skew from the delay of each path. However, a more convenient formulation, which does not require keeping track of paths, consists of adding the clock delay at each source/sink node to its arrival/required time:

$$\begin{aligned} T_{arr}(source) &= clock\_delay(source) \\ T_{req}(sink) &= cons + clock\_delay(sink). \end{aligned} \quad (9)$$

This formulation ensures that slack has the proper value in the presence of clock delay.

The value by which slack is divided in (5) to form criticality must also be modified, since criticality will be negative whenever slack exceeds $D_{max}$ due to clock skew. However, slack is guaranteed to never exceed $T_{req,max}$, the maximum required time at any sink node in the circuit (since slack is essentially required time minus arrival time, and arrival time can never be negative). Therefore, a definition which correctly normalizes criticality is:

$$crit_{single\text{-}clock}(i,j) = 1 - \frac{slk(i,j)}{T_{req,max}}. \quad (10)$$

Note that this definition reduces to (6) in the absence of clock delay.

## III. Multi-clock timing analysis

The first step in extending the timing analysis described in the previous section to multi-clock circuits is to allow the specification of complex timing constraints. To this end, we have enabled VPR to support a subset of the Synopsys Design Constraint (SDC) language [20]. Each clock in the design may be given a distinct target maximum clock period and be phase-shifted with respect to other clocks. Paths between pairs of clock domains may be ignored if their data transfer is asynchronous.

The clocks of off-chip registers leading to and from I/O ports may also be analysed: the I/Os are treated as registers clocked on a *virtual clock* (i.e. one that does not appear in the design). The maximum off-chip delay from any register to the I/O port is treated as an additional clock delay to the I/O, or is mapped to a connection inside the I/O port.

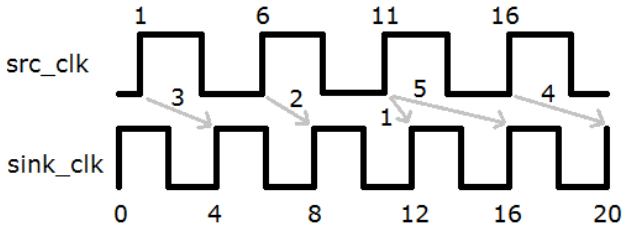


Fig. 1. Example calculation of a timing constraint, between a source domain with period 5 ns and a 1 ns phase shift, and a sink domain with period 4 ns and no phase shift. The LCM clock period is 20 ns, and the timing constraint for transfers from *src_clk* to *sink_clk* is 1 ns.

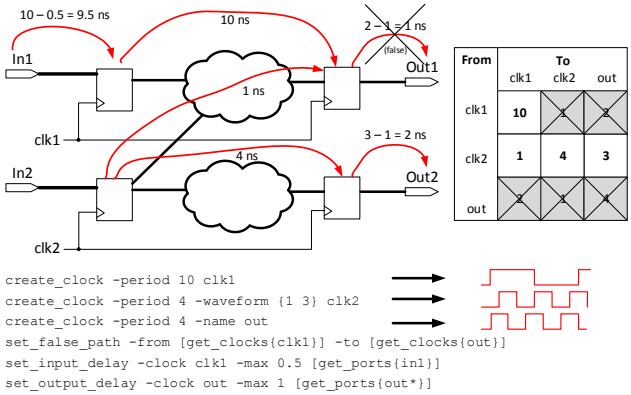


Fig. 2. SDC specification and constraint matrix for an example circuit with 3 constrained clocks. Constraints for each path are indicated by arrows; path delays are unmarked. Greyed-out entries correspond to pairs of constrained clock domains which do not have any paths analyzed between them.

Each pair of source and sink domains may have a timing constraint, leading to a $C$ by $C$ matrix of timing constraints for a circuit with $C$ clocks (real and virtual). For diagonal or intra-domain matrix entries, the constraint is just the target clock period specified in the SDC file. However, for off-diagonal entries, the constraint is implicit: it is the minimum time between a source edge and the next sink edge, assuming both clocks run at their target periods. This time can be found by iterating over all sink edges and finding the nearest previous source edge, as shown in Fig. 1. Only edges up to the least common multiple (LCM) of the clock periods need be considered, since the pattern of edges will repeat after that. Fig. 2 shows an example SDC specification for a three-clock circuit, along with the timing constraint matrix and the maximum permissible delays for each path. The timing constraint matrix formalism is flexible enough to represent phase-shifted clocks (as shown in Fig. 2), clocks with irregular duty cycles, and even negative-edge-triggered flip-flops (which are equivalent to positive-edge-triggered flip-flops on a 50% phase-shifted clock with the same target period as the original clock).

Since each connection in a multi-clock circuit may now be on paths between multiple domain pairs, it may have multiple values for slack, one for each constraint. However, only the worst-case, minimum slack is important to the designer. Computing the minimum slack requires performing a pair of forward and backward breadth-first traversals for

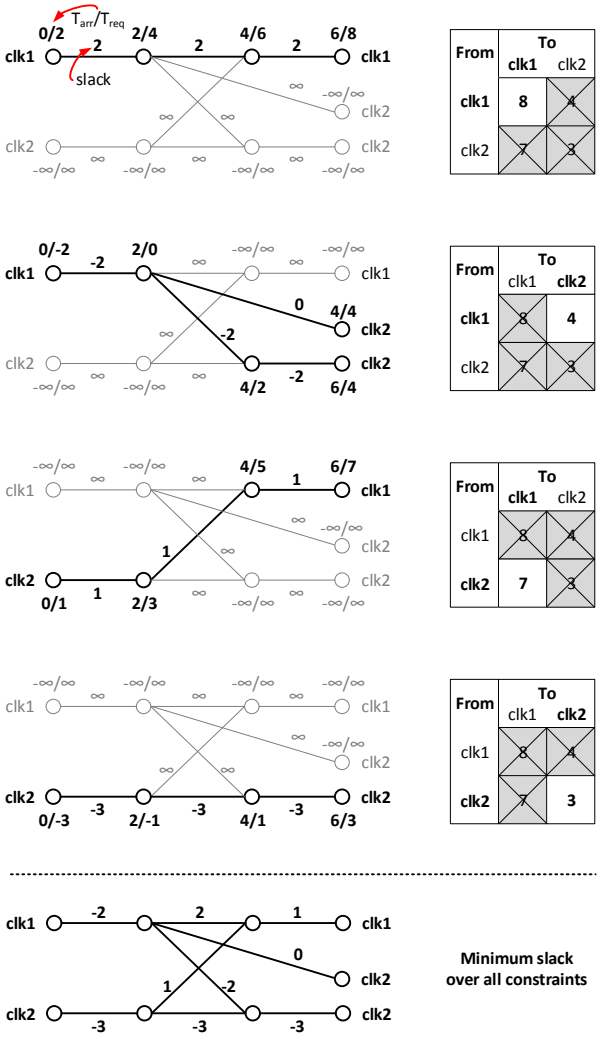


Fig. 3. An example slack calculation for a circuit with two clocks, where each connection has a delay of 2 ns. The first four panels show the results of each pair of forward and backward breadth-first traversals, corresponding to the highlighted entry in the constraint matrix. The numbers on each node are in the format arrival time / required time. The slack for each edge is set to the minimum of the slacks for each constraint, as shown in the fifth panel.

each pair of source and sink clock domains with a timing constraint between them, using the relevant entry from the timing constraint matrix to compute required arrival times. The slack of each connection is then set to the minimum of its slacks for each constraint:

$$\begin{aligned} slk_{cons}(i,j) &= T^{cons}_{req}(j) - T^{cons}_{arr}(i) - delay(i,j) \\ slk(i,j) &= min_{\forall cons}\{slk_{cons}(i,j)\}. \end{aligned} \tag{11}$$

Fig. 3 shows the slack calculation for an example two-clock circuit.

## IV. Complications in Multi-Clock Timing Optimization

### A. Normalizing slacks to form criticality

The first issue is how to normalize slacks to form criticality. There are two plausible formulations which extend (10) to multiple clock circuits. One formulation is a *global* normalization, in which the least slack over all constraints is divided by the greatest required time over all constraints:

$$\begin{aligned} crit_{global}(i,j) &= 1 - \frac{slk(i,j)}{max_{\forall cons}\{T^{cons}_{req,max}\}} \\ &= 1 - \frac{min_{\forall cons}\{T^{cons}_{req}(j) - T^{cons}_{arr}(i) - delay(i,j)\}}{max_{\forall cons}\{T^{cons}_{req,max}\}}. \end{aligned} \tag{12}$$

The other formulation is a *per-constraint* normalization: a criticality is calculated for each constraint based on the slack and the maximum required time for that constraint, and the final criticality is set to the worst-case (maximum) criticality over all constraints:

$$\begin{aligned} crit_{per\text{-}cons}(i,j) &= max_{\forall cons}\left\{1 - \frac{slk_{cons}(i,j)}{T^{cons}_{req,max}}\right\} \\ &= max_{\forall cons}\left\{1 - \frac{(T^{cons}_{req}(j) - T^{cons}_{arr}(i) - delay(i,j))}{T^{cons}_{req,max}}\right\}. \end{aligned} \tag{13}$$

However, global normalization as in (12) erroneously favours the optimization of domains with short paths. To see why, consider a circuit with two non-overlapping clock domains *clk* and *clk2*. Say that the critical path delays of the *clk* and *clk2* domains are respectively 10 and 4 ns, and that barely achievable timing constraints are set for each domain (equal to the critical path delays). Because the constraints are of equal difficulty, the two domains should be optimized equally.

According to (12), both domains are normalized by the largest required time over all constraints, 10 ns. As a result, every connection on *clk2* has a criticality of at least 0.6, since *clk2* paths can have a slack of at most 4 ns. Meanwhile, the criticality of connections on *clk* can span the full range from 0 to 1. The net effect is over-optimization of the *clk2* domain at the expense of *clk*. For this reason, per-constraint normalization as in (13) is preferable.

### B. Avoiding greater-than-one criticality

Greater-than-one criticality is the result of negative slacks, which occur whenever a timing constraint is smaller than the delay of a path subject to the constraint. Even if a constraint is achievable, it may not be achieved until late in the CAD flow, so negative slacks and greater-than-one criticalities may still be present in earlier stages.

As stated in Section 2, it is desirable to normalize criticality to be between 0 and 1. The reason for this is that at some stages of optimization, for example clustering and placement in VPR, it is beneficial to use a power of criticality ($criticality^{criticality\_exponent}$) as the connection weight, to give a particularly high priority to the optimization of more critical connections [6]. For instance, VPR's clusterer uses a criticality exponent of 8. This means that connections with criticalities near one will have a connection weight near one, while those less than one will have a reduced connection weight.

Criticalities greater than one are problematic because they result in very large connection weights when taken to a high power. Large connection weights are not inherently unusable, since the timing cost functions used by clustering, placement and routing typically use fractional changes in timing cost [6], [16]. Rather, the problem is that these values become *relatively* much larger than the weights of connections with criticality less than one.

For example, in Fig. 3, the top center connection has a slack of 2 corresponding to a required time of 8 (from the first traversal pair), hence a criticality of 0.75. Meanwhile, the top left connection has a slack of -2 corresponding to a required time of 4 (from the second traversal pair), hence a criticality of 1.5. Both connections are fairly critical and should be optimized. However, since the first connection has half the criticality of the second, it will have a connection weight 256 times smaller than the second, assuming a criticality exponent of 8, and be effectively ignored for optimization purposes.

Reducing the gap between failing and achievable constraints can be accomplished by reducing the criticality exponent. However, this would defeat the purpose of the criticality exponent, by reducing its effectiveness at separating criticalities near one from those less than one. Although it is possible to apply different transformations to criticalities of different magnitudes, rather than a uniform power-law transformation, it is unclear what type of transformation to apply in order to get good connection weights in all circumstances. It appears more feasible to deal with greater-than-one criticalities at the source, rather than pushing the problem to the timing cost calculation.

In the following section, we discuss several ways to avoid greater-than-one criticality by modifying slacks to be non-negative, which we term *slack modification*. However, note that the slacks reported to the designer in timing *analysis* are never modified, only the slacks used to calculate criticality during timing *optimization*. We call a combination of slack modification and criticality normalization a *criticality formulation*.

## V. Formulations of Slack Modification

### A. Unmodified slacks (U)

The simplest solution is to not modify slacks at all. Although this does not avoid greater-than-one criticality, connection weights can at least be bounded by an appropriate choice of *criticality denominator* (the delay value which slack is divided by when calculating criticality). For instance, since negative slacks cannot exceed the maximum arrival time, setting the criticality denominator to the maximum of all arrival *and* required times ensures that criticality is at most 2. However, this still allows connection weights to become quite large after taking criticality to a high criticality exponent.

### B. Clipped slacks (C)

Another solution is to *clip* any negative slacks to 0, while leaving positive slacks unchanged:

$$slk_c(i,j) = max(min_{\forall cons}\{slk_{cons}(i,j)\}, 0). \quad (14)$$

While this works when timing constraints are achievable, it fails when given impossible timing constraints, as the optimizers lose the ability to distinguish paths which severely fail timing from those that only slightly fail. In the limiting case of a 0 ns timing constraint, all paths will have negative slack and will be clipped to 0, so that the optimizers will have absolutely no information about which connections are more important to optimize. While this is an extreme example, large timing failures do nonetheless occur early in the design cycle, as well as early in the CAD flow. Fig. 4 shows the result of applying shifted slacks to the circuit of Fig. 3.

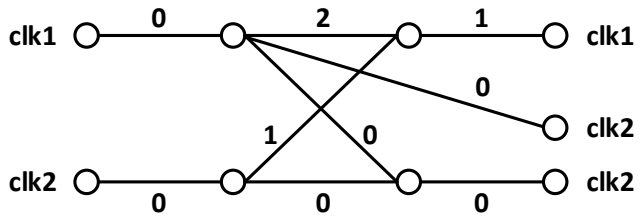


Fig. 4. The circuit from Fig. 3 with slacks labeled, after clipping negative slacks to 0. Note that most connections have 0 slack, and optimizers cannot determine which of these connections are more critical to optimize.

### C. Shifted slacks (S)

Another method of slack modification is to *shift* up every slack by the magnitude of the most negative slack in the entire design (and increase each constraint's criticality denominator by the same amount). If no negative slacks exist, no shifting occurs.

$$\begin{aligned} shift &= \left| min(0, min_{\forall i,j}\{slk(i,j)\}) \right| \\ slk_{cons,s}(i,j) &= T^{cons}_{req}(j) - T^{cons}_{arr}(i) - delay(i,j) + shift \\ crit_s(i,j) &= max_{\forall cons}\left\{1 - \frac{slk_{cons,s}(i,j)}{T^{cons}_{req,max} + shift}\right\} \end{aligned} \quad (15)$$

A disadvantage of shifting slacks is that it requires an extra cost of either runtime or memory: either two pairs of graph traversals per constraint instead of one (the first to find the most negative slack, the second to shift slacks), or the intermediate storage of slacks for every constraint. Fig. 5 shows the result of applying shifted slacks to the circuit of Fig. 3.

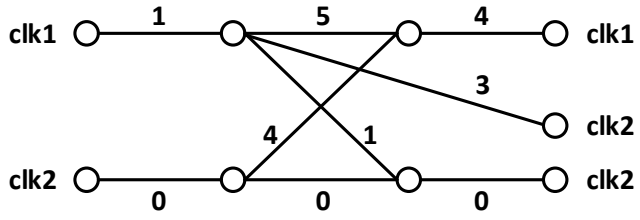


Fig. 5. The circuit from Fig. 3, after shifting all slacks up by 3 ns (since the most negative slack is -3 ns). Note that shifting slacks preserves information about which connections are more critical.

### D. Relaxed slacks (R)

A variation of shifted slacks is to shift each constraint's slacks by the most negative slack in that constraint, if any negative slacks exist. The simplest way to do this computationally is to *relax* the required time at every sink node by setting it to the maximum of the constraint (plus the clock delay at the sink node) and the maximum arrival time at any sink node in the constraint's traversal:

$$T^{cons}_{req,r}(sink) = max(cons + clock\_delay(sink), T^{cons}_{arr,max}). \quad (16)$$

Aside from this change, slack and criticality are calculated in the default way, according to (11) and (13). Fig. 6 shows the result of applying relaxed slacks to the circuit of Fig. 3.

Relaxing slacks is equivalent to shifting them when the circuit has a single constraint, or when all constraints have already been achieved and no slacks need to be shifted or relaxed. The only difference between them is that when a circuit contains multiple tight or impossible constraints, relaxation favours the optimization of the domains with more lax constraints. This is because relaxation gives a larger shift to domains which fail timing by a larger amount, while shifted gives the same shift to all domains (see Section VI-A2 for a more detailed discussion).

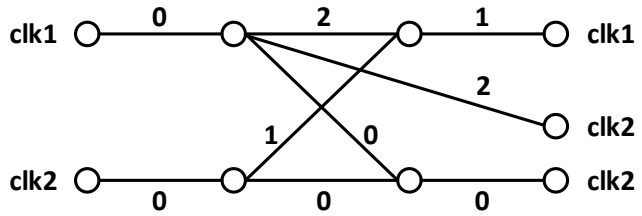


Fig. 6. The circuit from Fig. 3, after relaxing slacks. Note that relaxing slacks preserves much more information than clipping, because it maintains the ordering of slacks within each domain; however, unlike shifting, it does not preseve the ordering of slacks between domains.

Figures 7 to 10 show pseudo-code for multi-clock timing analysis (for brevity, only per-constraint criticality normalization is shown). The control variables $relaxed$, $shifted$ and $clipped$ indicate aspects of the algorithm which are specific to each slack modification method. The timing graph is iterated over in breadth-first fashion one 'level' at a time: first the source nodes, then the nodes in the immediate fanout of the source nodes, and so on.

## VI. Results

We conducted three sets of tests to compare the slack modification and criticality normalization methods discussed in the previous two sections, using three sets of benchmark circuits: MCNC, spliced MCNC, and VTR. All tests were conducted using VPR 7.0 [3] with default settings, on the *k6_frac_N10_40nm* architecture, a 40-nm architecture with ten fracturable 6-LUTs per logic block and length-4 routing wires. All quoted results are for the second run, to model the typical case where there is some spare routing in the chip [5]. Runtime results are quoted for single-core execution on an Intel Xeon E5-2650 processor clocked at 2.0 GHz.

**Inputs:** timing graph describing the circuit (see Section II-A); the (estimated) delay of every edge
**Output:** criticality of each edge (netlist connection)

```
for all edge ∈ timingGraph do
    edge.slack = ∞
    edge.criticality = 0
for all sourceDomain, sinkDomain ∈ clockDomains do
    for all node ∈ timingGraph do
        node.T_arr = −∞
        node.T_req = ∞
    forwardTraversal()
    backwardTraversal()
    updateSlackAndCrit()
```

Fig. 7. Pseudo-code for multi-clock timing analysis, using per-constraint criticality normalization.

```
for all sourceNode ∈ timingGraph do
    if node.domain == sourceDomain then
        if node.type == REGISTER then
            node.T_arr = node.clockDelay
        else if node.type == INPUT then
            node.T_arr = getOffChipDelay(node)
for all node ∈ levels(timingGraph) do
    if node.T_arr ≠ −∞ then
        // node is on a path from sourceDomain
        for all toNode ∈ fanout(node) do
            toNode.T_arr = max(toNode.T_arr,
                node.T_arr + edge(node, toNode).delay)
```

Fig. 8. Pseudo-code for the forward traversal for each constraint

```
cons = getTimingConstraint(sourceDomain, sinkDomain)
maxT_arr = getMaxT_arrThisConstraint(timingGraph)
for all node ∈ reversed(levels(timingGraph)) do
    if node.isSink() then
        if node.T_arr ≠ −∞ and node.domain == sinkDomain then
            // node is on sink domain and on a path from source domain
            if relaxed then
                node.T_req = max(cons + node.clockDelay, maxT_arr)
            else
                node.T_req = cons + node.clockDelay
    else
        // not a sink
        if node.T_arr ≠ −∞ then
            // node is on a path from sourceDomain (valid T_arr)...
            if ∃ toNode.T_req ≠ ∞ ∈ fanout(node) then
                // ...and to sinkDomain (valid T_req)
                for all toNode ∈ fanout(node) do
                    node.T_req = min(node.T_req,
                        toNode.T_req − edge(node, toNode).delay)
```

Fig. 9. Pseudo-code for the backward traversal for each constraint

```
critDenom = getMaxT_reqThisConstraint(timingGraph)
if shifted then
    shift = getMostNegativeSlackAllContraints(timingGraph)
    shift = min(shift, 0)
    critDenom = critDenom − shift
for all node ∈ timingGraph do
    for all toNode ∈ fanout(node) do
        edge = edge(node, toNode)
        slack = toNode.T_req − node.T_arr − edge.delay
        if shifted then
            slack = slack − shift
        else if clipped then
            slack = max(slack, 0)
        edge.slack = min(edge.slack, slack)
        edge.crit = min(edge.crit, 1 − slack/critDenom)
```

Fig. 10. Pseudo-code for the slack and criticality update for each constraint

In the following analysis, $(a, b)$ denotes a clock constraint of $a$ ns and an I/O constraint of $b$ ns, and $M(a, b)$ denotes a trial using criticality formulation $M$ and constraint $(a, b)$. For instance, $R(0, 0)$ denotes a trial using relaxed slack modification with per-constraint normalization where both the clock and I/O domains are given a 0 ns constraint.

Cross-domain paths are analysed for all circuits where they exist. Most of the time, cross-domain constraints are derived implicitly from target clock periods (see Section III). However, there is one exception where they are specified explicitly: if an I/O domain is assigned an extremely lax constraint of 1000 ns, clock-to-I/O and I/O-to-clock paths are also assigned a constraint of 1000 ns so that all paths involving I/Os are marked as irrelevant for the purpose of timing optimization.

### A. MCNC benchmarks

The first set of tests compares the achieved clock period, I/O delay and wirelength of the 10 largest single-clock MCNC benchmark circuits (bigkey, clma, diffeq, dsip, elliptic, frisc, s298, s38417, s38584.1, tseng) [21] using different criticality formulations for timing optimization. I/O delay refers to the delay of paths from inputs to outputs, not cross-domain paths between I/Os and registers; clock period accounts for the effect of clock skew. Table I summarizes the results when the clock and I/O domains are given each permutation of an impossible 0 ns constraint and a lax 1000 ns constraint, or when both domains are given an 'achievable' (Achv.) constraint equal to their critical path delay with $R(0, 0)$.

*1) Ineffective techniques: unmodified and clipped:* Without slack modification (U), the optimizers are largely unable to distinguish between tight and lax constraints, giving the same weight to each domain regardless of its constraint and severely under-optimizing the clock domain compared to other methods. The $U$ columns of Table I show that not modifying slacks achieves 20-21% larger clock periods, 3-8% lower I/O delay, and 11-12% lower wirelength than $R(0, 0)$, with all tested constraints. Clipping slacks (C) does show some responsiveness to timing constraints. However, it also under-optimizes the clock domain, possibly because clipping under-optimizes paths which severely fail timing (since their large negative slack gets clipped to 0) and the clock domain contains more such paths. With $(0, 0)$ constraints, clipped and unmodified slacks result in a clock domain critical path delay over 15% worse than for any other slack modification technique, as shown in the first row of Table I(a).

*2) Shifted vs relaxed slacks:* Shifted (S) and relaxed (R) slacks perform the best overall when the circuit is given a $(0, 0)$ constraint. As expected, shifted optimizes tight constraints over lax constraints more than relaxed. Since the average clock domain path delay of the 10 circuits is approximately twice their average I/O-to-I/O path delay, a 0 ns constraint is even tighter for the clock domains than for the I/O domains. This means that shifted prioritizes the clock domain at the cost of neglecting the I/O domain.

[1]Criticality formulations are, from left: unmodified, clipped, shifted, relaxed, shifted with global normalization, relaxed with global normalization. 'Achv.' means that the constraint for each domain is set to the critical path delay achieved with $R(0, 0)$.

TABLE I
GEOMEAN TIMING, WIRELENGTH AND RUNTIME OF MCNC BENCHMARK CIRCUITS UNDER VARIOUS CRITICALITY FORMULATIONS[1], RELATIVE TO $R(0, 0)$ (BOLDED). ROWS WITH LAX CONSTRAINTS FOR EACH DOMAIN ARE GRAYED OUT AS THEY ARE NOT BEING OPTIMIZED.

(a) Achieved clock period

| Clock cons. | I/O cons. | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|---|
| 0 ns | 0 ns | 1.21 | 1.20 | 0.97 | **1.00** | 1.05 | 1.01 |
| Achv. | Achv. | 1.21 | 1.16 | 1.02 | 0.99 | 1.07 | 1.01 |
| 0 ns | 1000 ns | 1.20 | 1.12 | 0.96 | 0.98 | 1.08 | 1.09 |
| 1000 ns | 0 ns | 1.20 | 1.34 | 1.28 | 1.32 | 1.31 | 1.31 |
| 1000 ns | 1000 ns | 1.21 | 1.19 | 1.19 | 1.19 | 1.16 | 1.16 |

(b) Maximum I/O delay

| Clock cons. | I/O cons. | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|---|
| 0 ns | 0 ns | 0.92 | 0.98 | 1.06 | **1.00** | 1.07 | 0.99 |
| Achv. | Achv. | 0.94 | 0.96 | 0.95 | 1.04 | 0.94 | 1.01 |
| 0 ns | 1000 ns | 0.97 | 1.41 | 1.23 | 1.24 | 1.25 | 1.31 |
| 1000 ns | 0 ns | 0.94 | 0.92 | 0.96 | 0.95 | 0.91 | 0.97 |
| 1000 ns | 1000 ns | 0.94 | 1.08 | 1.08 | 1.08 | 1.11 | 1.11 |

(c) Total wirelength (number of logic blocks traversed)

| Clock cons. | I/O cons. | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|---|
| 0 ns | 0 ns | 0.88 | 0.89 | 1.00 | **1.00** | 1.00 | 0.99 |
| Achv. | Achv. | 0.88 | 0.91 | 0.99 | 0.99 | 0.98 | 0.98 |
| 0 ns | 1000 ns | 0.89 | 0.90 | 0.98 | 0.99 | 0.90 | 0.90 |
| 1000 ns | 0 ns | 0.88 | 0.92 | 0.94 | 0.94 | 0.91 | 0.92 |
| 1000 ns | 1000 ns | 0.88 | 0.86 | 0.86 | 0.86 | 0.85 | 0.85 |

(d) Total runtime (minutes) for (0,0)

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Raw | 0.34 | 0.32 | 0.41 | 0.39 | 0.36 | 0.37 |
| Relative to R | 0.87 | 0.83 | 1.05 | **1.00** | 0.92 | 0.96 |

(e) Timing analysis runtime (minutes) for (0,0)

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Raw | 0.06 | 0.06 | 0.11 | 0.07 | 0.07 | 0.07 |
| Relative to R | 0.94 | 0.82 | 1.57 | **1.00** | 0.95 | 1.00 |

(f) (0,1000) : (0,0) runtime ratio

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Total | 1.02 | 1.06 | 0.97 | 0.90 | 0.98 | 0.91 |
| Timing analysis | 0.89 | 1.02 | 0.93 | 0.82 | 0.87 | 0.82 |

For example, consider giving a $(0, 0)$ constraint to the circuit in Fig. 11, which has a clock domain with a single 10-ns connection and an I/O domain with a single 4-ns connection. Shifted slacks would optimize the clock domain significantly more than the I/O domain: this method would increase both slacks by 10 ns, giving the clock domain's connection a slack of 0 ns and a criticality of 1, with the I/O domain connection having a slack of 6 ns and a criticality of only 0.4. Conversely, relaxed slacks would optimize both domains equally, since it would increase the clock domain's slack by 10 ns and the I/O domain's by 4 ns, giving both connections a slack of 0 ns and a criticality of 1.

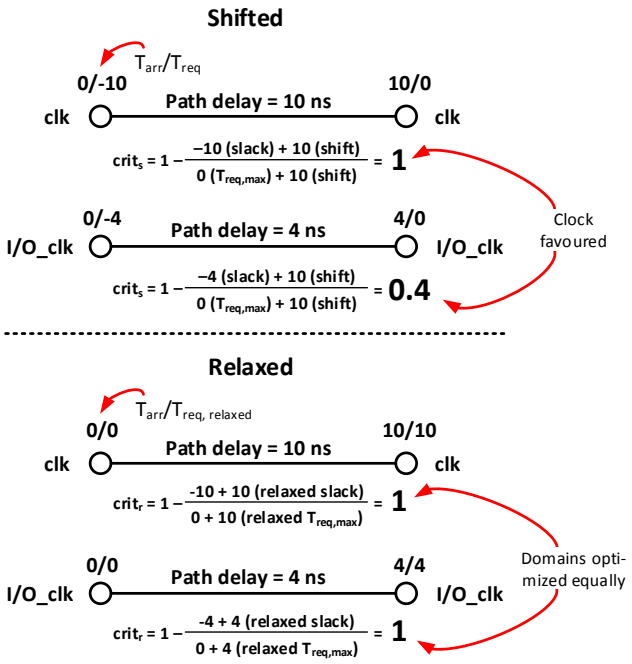


Fig. 11. An example of how shifted preferentially optimizes the clock domain over the I/O domain when given 0 ns constraints on both domains.

Because of this effect, $S(0,0)$ achieves a 3% increase in clock speed compared to $R(0,0)$, as shown in the first row of Table I(a), but causes a 6% degradation in I/O delay, as shown in the first row of Table I(b). However, note that since the clock domain paths are longer, there is approximately a 1-to-1 critical path delay tradeoff between the two domains in absolute terms.

With a lax clock constraint (bottom two rows of Table I), shifted and relaxed perform similarly, as the difference in priorities of the two methods does not come into effect when most connections in the circuit (those on the clock domain) already have an extremely low criticality.

*3) Global vs per-constraint criticality normalization:* As discussed in Section IV-A, global criticality normalization (denoted by $_g$) favours the optimization of domains with shorter paths, in this case the I/O domains, compared to per-constraint normalization. This effort only slightly decreases the I/O delay for most constraints, likely because the I/Os do not require further optimization, but does degrade the clock's critical path delay, sometimes significantly (compare $S$ with $S_g$ and $R$ with $R_g$ in Tables I(a) and I(b)). Perversely, this effect becomes more pronounced with a lax I/O constraint, which should optimize the clock more: global normalization performs 3-8% worse on the clock domain with a $(0,1000)$ constraint than with a $(0,0)$ constraint (compare $S_g$ and $R_g$ between the first and third rows of Table I(a)), even as per-constraint normalization performs 1-2% better. This behaviour is clearly suboptimal and indicates the lack of robustness of global criticality normalization.

*4) Timing-wirelength tradeoff:* Relaxed and shifted slacks show the correct tradeoff between timing and wirelength optimization. For these methods, wirelength appropriately increases with the number of domains given tight constraints, since critical connections are routed along slightly more direct paths, even if this forces non-critical connections to take significantly more circuitous paths. For instance, increasing both constraints from $(0,0)$ to $(1000,1000)$ reduces wirelength usage by 14-15% but increases clock delay by 10-23% (compare $S$, $R$, $S_g$ and $R_g$ between the first and fifth rows of Tables I(a) and I(c)). Unmodified and clipped slacks do not show the correct tradeoff.

*5) Runtime:* The runtime of timing analysis is similar for each criticality formulation (Table I(e)) and between $(0,0)$ and $(0,1000)$ constraints (Table I(f)). The most substantial difference is that timing analysis for $S$ is approximately 50% longer because of the extra graph traversals (see Section V-C). Geomean timing analysis runtime is 26% of geomean total runtime for $S$ and 18-19% for the other criticality formulations (compare Tables I(d) and I(e)).

### B. Spliced MCNC Benchmarks

The second set of tests compare the best two criticality formulations from the first set, shifted and relaxed slack modification with per-constraint normalization. These tests were performed on true multi-clock circuits given two difficult, but unequal, constraints. Due to the paucity of available large multi-clock benchmark circuits, we created benchmarks by splicing together pairs of the same 10 MCNC benchmarks, resulting in two-clock circuits with one clock from each of the original circuits. We used the 10 pairs of circuits with the largest combined number of logic blocks.

VPR was run three times on each spliced circuit. For the first run, both sub-circuits were assigned a clock constraint of 0 ns to indicate that they should be optimized to run as fast as possible. For the remaining two runs, each of the two sub-circuits was assigned a timing constraint equal to its achieved critical path delay from the first run, a barely achievable constraint, while the other sub-circuit's constraint remained at 0 ns, an impossibly tight constraint. The timing and wirelength results for the last two runs are shown in Table II. For example, the first two rows give results for the circuit formed by splicing together the single-clock *bigkey* and *clma* MCNC benchmark circuits: in the first row, *bigkey* is given an impossible constraint, while *clma* is given an achievable constraint; in the second row, their roles are reversed.

Here, unlike for the first set of tests (Section VI-A), shifted fails to improve the speed of the tighter clock compared to relaxed, but it does result in a 7% degradation in the more lax clock.

The reason for this behaviour is that, unlike Section VI-A, the two constrained clock domains are entirely disjoint. This means that they can be instantiated in separate locations of the chip, without interfering with each other (except when there is competition for localized on-chip resources, such as RAM blocks or I/O ports). As a result, relaxed can achieve close-to-optimal timing on both domains *at the same time*—optimization is not a zero-sum game. It is therefore futile for shifted to devote more attention to the tighter domain as this only degrades the more lax domain, which still requires good optimization with its barely achievable constraint.

TABLE II
CRITICAL PATH DELAY AND WIRELENGTH OF SPLICED TWO-CLOCK CIRCUITS, WITH ONE SUB-CIRCUIT (NAME IN BOLD) GIVEN A 0 NS CONSTRAINT AND THE OTHER GIVEN AN ACHIEVABLE CONSTRAINT, UNDER RELAXED (R) AND SHIFTED (S) SLACK MODIFICATION WITH PER-CONSTRAINT CRITICALITY NORMALIZATION.

| Circuit | Impossible delay (ns) | | Achievable delay (ns) | | Wirelength | |
|---|---|---|---|---|---|---|
| | R | S | R | S | R | S |
| **bigkey**+clma | 1.22 | 1.22 | 11.05 | 10.95 | 88801 | 87729 |
| bigkey+**clma** | 10.65 | 10.91 | 1.22 | 1.22 | 88537 | 88486 |
| **clma**+diffeq | 10.48 | 10.35 | 6.81 | 6.57 | 91033 | 90610 |
| clma+**diffeq** | 7.02 | 6.88 | 10.82 | 10.66 | 90956 | 86928 |
| **clma**+elliptic | 10.85 | 10.55 | 9.01 | 9.86 | 111456 | 108805 |
| clma+**elliptic** | 9.01 | 8.78 | 10.61 | 11.95 | 110179 | 106733 |
| **clma**+frisc | 10.98 | 10.64 | 11.96 | 13.20 | 116823 | 116519 |
| clma+**frisc** | 12.14 | 12.42 | 11.39 | 11.39 | 115958 | 108933 |
| **clma**+s298 | 10.42 | 10.55 | 9.80 | 11.23 | 93063 | 89808 |
| clma+s298 | 9.71 | 10.08 | 10.96 | 11.66 | 90520 | 84210 |
| **clma**+s38417 | 10.62 | 10.44 | 7.37 | 8.81 | 121014 | 123223 |
| clma+**s38417** | 7.39 | 7.64 | 10.34 | 11.90 | 122401 | 115747 |
| **clma**+s38584.1 | 10.48 | 11.45 | 5.54 | 7.24 | 125406 | 124253 |
| clma+**s38584.1** | 5.50 | 5.91 | 10.56 | 10.77 | 125264 | 121157 |
| **elliptic**+s38584.1 | 9.12 | 8.85 | 6.27 | 5.63 | 68627 | 65821 |
| elliptic+**s38584.1** | 6.27 | 5.27 | 9.14 | 9.10 | 68245 | 67901 |
| **frisc**+s38584.1 | 12.05 | 12.41 | 5.50 | 5.86 | 72502 | 67256 |
| frisc+**s38584.1** | 5.91 | 6.09 | 12.05 | 12.73 | 73819 | 71734 |
| **s38417**+s38584.1 | 7.49 | 7.62 | 5.50 | 6.95 | 82446 | 80512 |
| s38417+**s38584.1** | 5.36 | 5.63 | 7.78 | 8.30 | 80996 | 77349 |
| **S / R geomean** | **1.00** | | **1.07** | | **0.97** | |

Although realistic multi-clock circuits usually do not have completely disjoint clock domains, the argument still holds for them. This is because realistic circuits tend to have weakly coupled clock domains, with few cross-domain paths compared to intra-domain paths.

The MCNC and spliced MCNC tests consisted of two runs of VPR: the circuit was first routed to find the minimum channel width, then re-routed at a fixed channel width of 1.3 times the minimum.

### C. VTR Benchmarks

While the MCNC and spliced MCNC benchmarks are sufficient to illustrate the effects of various criticality formulations, they are not representative of the size and complexity of modern-day circuit designs. Consequently, tests similar to those in Section VI-A were conducted on each of the 9 VTR benchmark circuits [3] with more than 5000 6-LUTs (bgm, blob_merge, LU8PEEng, LU32PEEng, mcml, mkDelayWorker32B, stereovision0, stereovision1, and stereovision2); the largest contains over 100,000 6-LUTs. By comparison, all of the MCNC benchmarks used in this paper have fewer than 10,000 4-LUTs, and half have under 2000.

As the architecture used for the earlier tests lacks the hard multipliers and memories required by some of the VTR benchmarks, these tests used the *k6_frac_N10_frac_chain_mem32K_40nm* architecture, which adds carry chains, 32KB of RAM and hard multipliers. Since only one of the benchmark circuits has I/O-to-I/O paths, an alternative metric of I/O delay was used instead: the geometric mean of the maximum clock-to-I/O and I/O-to-clock paths.

To reduce CPU time by an order of magnitude, a fixed channel width of 154 was used for all runs, 10% greater than the channel width required to route any of the circuits at $R(0,0)$. Additionally, the number of routing iterations attempted before declaring failure was doubled to 100 from the default of 50. Still, for the least robust slack modification techniques (clipped and no modification), the LU8PEEng and LU32PEEng circuits failed to route entirely at this channel width with certain timing constraints (denoted with an asterisk in Table III).

The clock period and wirelength results for the VTR benchmarks do not differ significantly from the MCNC results, confirming that the broad trends noted in Section VI-A hold for a variety of circuit sizes. Unmodified slacks are once again almost completely unresponsive to variations in timing constraints (see the $U$ columns of Table III), and both unmodified and clipped slacks have at least 17% greater clock delay than relaxed and shifted slacks with a $(0,0)$ constraint (first row of Table III(a)).

$S(0,0)$ performs slightly (2%) better than $R(0,0)$ on the clock domain and much worse (26%) on the I/O domain (first row of Tables III(a) and III(b)).

Global normalization does not degrade the clock domain as in Section VI-A, and sometimes even substantially outperforms per-constraint normalization on the I/O domain (compare $S$ with $S_g$ and $R$ with $R_g$ in Tables I(a) and I(b)), but per-constraint normalization is still preferred overall because of global normalization's worse performance in Section VI-A.

Finally, relaxed and shifted slacks once again exhibit the timing-wirelength tradeoff described in Section VI-A.

There is more variability in the VTR benchmarks' I/O results (Table III(b)), because these benchmarks have much shorter I/O paths than internal paths: with $R(0,0)$, their geometric mean ratio of clock period to I/O delay is approximately 6.8, compared to only 2.4 for the MCNC benchmarks. This allows several criticality formulations to reduce I/O delay by over one-third when the clock is given a lax constraint (compare rows 1 and 4 of Table III(b)), since the I/O paths are so short to begin with.

Runtime results are similar between the MCNC and VTR benchmarks, aside from the increased runtime for the larger VTR benchmarks. In particular, geomean timing analysis runtime is a similar percentage of total runtime (31% for $S$, 16-20% for the other criticality formulations; compare Tables III(d) and III(e)), suggesting that the runtime of timing analysis does not grow much faster with circuit size than the runtime of other parts of the CAD flow. The time complexity of timing analysis is the same as that of breadth-first search, $O(|V|+|E|)$ where $|V|$ and $|E|$ are respectively the number of timing graph nodes and connections. However, since all practical digital circuits have constant-bounded fanin, $|E| = O(|V|)$ and hence the time complexity simplifies to $O(|V|)$.

TABLE III
GEOMEAN TIMING, WIRELENGTH AND RUNTIME OF VTR BENCHMARK CIRCUITS UNDER THE SAME CRITICALITY FORMULATIONS AS TABLE I

(a) Achieved clock period

| Clock cons. | I/O cons. | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|---|
| 0 ns | 0 ns | 1.24* | 1.17* | 0.98 | **1.00** | 0.99 | 0.98 |
| Achv. | Achv. | 1.26* | 1.03* | 1.00 | 0.99 | 1.02 | 0.99 |
| 0 ns | 1000 ns | 1.30 | 1.22 | 1.00 | 1.00 | 1.18 | 1.16 |
| 1000 ns | 0 ns | 1.23* | 1.36 | 1.23 | 1.25 | 1.35 | 1.34 |
| 1000 ns | 1000 ns | 1.23* | 1.14 | 1.14 | 1.14 | 1.18 | 1.18 |

(b) Maximum I/O delay (geomean of clock-to-I/O and I/O-to-clock)

| Clock cons. | I/O cons. | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|---|
| 0 ns | 0 ns | 1.12* | 1.08* | 1.26 | **1.00** | 1.17 | 0.98 |
| Achv. | Achv. | 1.08* | 0.96* | 0.96 | 1.05 | 0.81 | 0.96 |
| 0 ns | 1000 ns | 1.13 | 1.42 | 1.42 | 1.46 | 1.39 | 1.41 |
| 1000 ns | 0 ns | 1.12* | 0.63 | 0.88 | 0.91 | 0.62 | 0.59 |
| 1000 ns | 1000 ns | 1.10* | 1.25 | 1.25 | 1.25 | 1.28 | 1.28 |

(c) Total wirelength (number of logic blocks traversed)

| Clock cons. | I/O cons. | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|---|
| 0 ns | 0 ns | 0.88* | 0.86* | 0.98 | **1.00** | 0.98 | 1.00 |
| Achv. | Achv. | 0.88* | 0.93* | 0.99 | 0.99 | 1.00 | 1.00 |
| 0 ns | 1000 ns | 0.91 | 0.90 | 0.98 | 1.00 | 0.90 | 0.92 |
| 1000 ns | 0 ns | 0.87* | 0.94 | 0.88 | 0.88 | 0.94 | 0.94 |
| 1000 ns | 1000 ns | 0.87* | 0.84 | 0.84 | 0.84 | 0.84 | 0.84 |

* LU8PEEng and LU32PEEng failed to route at 154 channels, and are excluded from these averages.

(d) Total runtime (minutes) for (0,0)

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Raw | 4.56 | 4.61 | 4.13 | 3.69 | 3.64 | 3.50 |
| Relative to R | 1.24 | 1.25 | 1.12 | **1.00** | 0.99 | 0.95 |

(e) Timing analysis runtime (minutes) for (0,0)

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Raw | 0.81 | 0.75 | 1.26 | 0.72 | 0.74 | 0.70 |
| Relative to R | 1.12 | 1.03 | 1.74 | **1.00** | 1.02 | 0.96 |

(f) (0,1000) : (0,0) runtime ratio

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Total | 1.00 | 0.97 | 0.98 | 0.96 | 1.16 | 1.23 |
| Timing analysis | 0.95 | 0.98 | 0.97 | 0.97 | 0.98 | 1.16 |

### D. Titan benchmarks

A final set of tests were conducted on two benchmarks from the timing-driven Titan suite [22]: sparcT1_chip2, a multi-core microprocessor with 377,734 6-LUTs and 824,152 total blocks; and mes_noc, a 9-clock on-chip network with 274,321 6-LUTs and 549,045 total blocks. These are respectively the largest single-clock circuit and the largest multi-clock circuit able to pass the VPR flow according to [22]. (While [22] lists sparcT1_chip2 as having two clocks, one clock is an inverted version of the system clock and has no inter- or intra-domain paths, so sparcT1_chip2 is a single-clock circuit for the purposes of timing analysis). Two sets of constraints were tested: 0 ns for all clocks and I/Os (*All tight*), and 0 ns for only the system clock with 1000 ns for the I/Os and (for mes_noc) the other clocks (*System clock tight*), as presented in Table IV. Circuits were routed at a fixed channel width of 300, as used for this benchmark suite in both [22] and [23], and a very high maximum of 400 routing iterations.

With these much larger circuits, routability is a dominant concern. Circuits were deemed unroutable if VPR's 'safe' routing failure predictor found routing resource overuse to be so high that the circuit would be unlikely to route successfully, or if routing failed to converge after 48 hours of the CAD flow (since none of the successful runs required more than 13 hours, this is not an onerous restriction). Only $S$, $R$ and $S_g$ routed both circuits successfully with *All tight*, with $S$ and especially $S_g$ exceeding the performance of $R$ on both the system clock and I/O domains while underperforming $R$ on the other clock domains. However, only $R$ still routed successfully with *System clock tight*, suggesting that $R$'s more balanced optimization is necessary for robust optimization of these larger circuits. Further, $R$ demonstrated an 8% improvement in system clock period and 2% improvement in wirelength with only the system clock tight, showing that its optimization focus is correctly changing as the timing constraints change. Relative runtime results are very similar to the VTR benchmarks', further indicating that the runtime performance of timing analysis scales well with circuit size.

### E. Which criticality formulation is most robust?

The MCNC (Section VI-A) and VTR (Section VI-C) benchmark tests show that shifted ($S$) and relaxed ($R$) are easily the most successful methods of slack modification, since they have the best overall timing results and also appropriately trade off timing for wirelength when given more lax constraints.

The MCNC tests also confirm that global criticality normalization under-optimizes domains with longer paths, as discussed in Section IV-A, perversely increasing the clock delay of these circuits when I/O constraints are loosened because their clock domain paths are longer than their I/O paths. Although global criticality normalization does outperform per-constraint normalization on the VTR benchmarks' I/O paths, it is not robust for all circuits.

The spliced MCNC tests (Section VI-B) show that shifted does not robustly optimize multiple tight or impossible constraints of different degrees of difficulty when clock domains are disjoint, because its extra focus on the tightest constraints becomes unnecessary when the clock domains can be optimized independently.

The Titan tests (Section VI-D) show that only relaxed permits all circuits to route under both of the tested timing constraints. The other five criticality formulations cause convergence issues in the router.

Hence, **relaxed slack modification with per-constraint slack normalization is the most robust criticality formulation**.

## VII. CONCLUSION

The robust optimization of multiple timing constraints, including impossible constraints and constraints of varying

TABLE IV
GEOMEAN TIMING, WIRELENGTH AND RUNTIME OF TITAN BENCHMARK CIRCUITS UNDER THE SAME CRITICALITY FORMULATIONS AS TABLE I

(a) Achieved system clock period

| Constraints | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| All tight | – | – | 0.96 | **1.00** | 0.90 | – |
| System clock tight | – | – | – | 0.92 | – | – |

(b) Achieved geomean period of other clocks (mes_noc only)

| Constraints | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| All tight | – | – | 1.06 | **1.00** | 1.06 | 1.11 |
| System clock tight | – | – | – | 2.67 | – | – |

(c) Maximum I/O delay (geomean of clock-to-I/O and I/O-to-clock)

| Constraints | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| All tight | – | – | 0.97 | **1.00** | 0.89 | – |
| System clock tight | – | – | – | 0.92 | – | – |

(d) Total wirelength (number of logic blocks traversed)

| Constraints | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| All tight | – | – | 0.97 | **1.00** | 0.99 | – |
| System clock tight | – | – | – | 0.98 | – | – |

(e) Total runtime (minutes) for *All tight*

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Raw | – | – | 603.16 | 530.45 | 544.28 | – |
| Relative to R | – | – | 1.14 | **1.00** | 1.03 | – |

(f) Timing analysis runtime (minutes) for *All tight*

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Raw | – | – | 131.51 | 71.52 | 79.19 | – |
| Relative to R | – | – | 1.84 | **1.00** | 1.11 | – |

(g) System clock tight : all tight runtime ratio

| | U | C | S | R | $S_g$ | $R_g$ |
|---|---|---|---|---|---|---|
| Total | – | – | – | 1.09 | – | – |
| Timing analysis | – | – | – | 1.04 | – | – |

* A "–" indicates that either or both circuits failed to route.

magnitudes, requires a rethinking of how to classify the timing criticality of each connection. In this paper, we have described a formulation of criticality, relaxed slack modification with per-constraint normalization, which optimizes multiple timing constraints well (including impossible constraints) and makes appropriate timing-wirelength tradeoffs. This formulation achieved over 20% greater clock speed with aggressive constraints than a formulation without slack modification.

What has the designer gained from this work? First, the luxury of being inexact: the ability to set highly unachievable multi-clock constraints (such as the 0 ns constraints tested in this paper) and still be assured that the CAD tools will produce robust results. Second, and more unexpectedly, an equally large increase in clock speed with *achievable* constraints where negative slack is not expected to be an issue, because even achievable constraints are often not achieved until late in the CAD flow, so circuits can still be de-optimized in early stages by a poor criticality formulation. Hence, the applicability of this work is not restricted to edge cases early in the design process; on the contrary, our criticality formulation achieves a significant across-the-board improvement in multi-clock circuit speeds.

Future work could reduce the runtime of multi-clock timing analysis by restricting the timing graph traversals for each constraint to only the portions of the graph which are affected by that constraint. Additionally, path counting (e.g. [13]) and statistical timing techniques (e.g. [16]) could be combined with our criticality formulation and applied to multi-clock timing analysis.

## ACKNOWLEDGMENTS

The authors would like to thank Kevin Murray for guidance regarding the Titan benchmarks, as well as the numerous researchers who have helped create and enhance the VPR CAD tool over many years.

Computations were performed on the GPC supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto [24].

## REFERENCES

[1] M. Hutton, D. Karchmer, B. Archell, and J. Govig, "Efficient Static Timing Analysis and Applications Using Edge Masks," in *FPGA '05*, pp. 174–183.
[2] V. Betz and D. Galloway, "Method of optimizing the design of electronic systems having multiple timing constraints," 2004, US Patent 6,763,506.
[3] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, pp. 6:1–6:30, 2014.
[4] R. B. Hitchcock, G. L. Smith, and D. D. Cheng, "Timing Analysis of Computer Hardware," *IBM J. Res. Dev.*, vol. 26, no. 1, pp. 100–105, 1982.
[5] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, Massachusetts: Kluwer Academic Publishers, 1999.
[6] A. Marquardt, V. Betz, and J. Rose, "Timing-driven Placement for FPGAs," in *FPGA '00*, pp. 203–213.
[7] H. Ren, "Sensitivity guided net weighting for placement driven synthesis," in *Proc. Int. Symp. on Physical Design*, 2004, pp. 10–17.
[8] A. S. Marquardt, V. Betz, and J. Rose, "Using Cluster-based Logic Blocks and Timing-driven Packing to Improve FPGA Speed and Density," in *FPGA '99*, pp. 37–46.
[9] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *FPGA '12*, pp. 77–86.
[10] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *FPGA '95*, pp. 111–117.
[11] P. Maidee, C. Ababei, and K. Bazargan, "Fast Timing-driven Partitioning-based Placement for Island Style FPGAs," in *DAC '03*, pp. 598–603.
[12] ——, "Timing-driven partitioning-based placement for island style FPGAs," *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 24, no. 3, pp. 395–406, 2005.
[13] T. T. Kong, "A Novel Net Weighting Algorithm for Timing-driven Placement," in *ICCAD '02*, 2002, pp. 172–176.
[14] G. Chen and J. Cong, "Simultaneous Placement with Clustering and Duplication," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 3, pp. 740–772, 2004.
[15] ——, "Simultaneous Timing-driven Placement and Duplication," in *FPGA '05*, pp. 51–59.
[16] Y. Lin, M. Hutton, and L. He, "Placement and Timing for FPGAs Considering Variations," in *FPL '06*, 2006, pp. 1–7.
[17] Y. Lin and L. He, "Stochastic Physical Synthesis for FPGAs with Pre-routing Interconnect Uncertainty and Process Variation," in *FPGA '07*, pp. 80–88.

[18] S. Sivaswamy and K. Bazargan, "Statistical Analysis and Process Variation-Aware Routing and Skew Assignment for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 1, pp. 4:1–4:35, 2008.
[19] H. Ren, D. Pan, and D. Kung, "Sensitivity guided net weighting for placement-driven synthesis," *IEEE Trans. Comput.-Aided Des. Integr. Syst.*, vol. 24, no. 5, pp. 711–721, 2005.
[20] S. Gangadharan and S. Churiwala, *Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)*. Springer London, Limited, 2013.
[21] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *MCNC*, 1991.
[22] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing Driven Titan: Enabling Large Benchmarks and Exploring the Gap Between Academic and Commercial CAD," *ACM Trans. Reconfigurable Technol. Syst.*, 2014.
[23] ——, "Titan: Enabling large and complex benchmarks in academic CAD," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. IEEE, 2013, pp. 1–8.
[24] C. Loken, D. Gruner, L. Groer, R. Peltier, N. Bunn, M. Craig, T. Henriques, J. Dempsey, C.-H. Yu, J. Chen, L. J. Dursi, J. Chong, S. Northrup, J. Pinto, N. Knecht, and R. V. Zon, "SciNet: Lessons Learned from Building a Power-efficient Top-20 System and Data Centre," *Journal of Physics: Conference Series*, vol. 256, no. 1, p. 012026, 2010.

**Michael Wainberg** received the B.A.Sc. degree in Engineering Science (Physics) from the University of Toronto in 2014.

**Vaughn Betz** (S'88–M'91) received the B.Sc(EE) from the University of Manitoba in 1991, the MS(ECE) from the University of Illinois at Urbana-Champaign in 1993, and the PhD(ECE) from the University of Toronto in 1998.

Dr. Betz co-founded of Right Track CAD and was VP of Engineering until its acquisition by Altera in 2000. He was at Altera from 2000 to 2011, ultimately as Senior Director of Software Engineering. He is currently an Associate Professor at the University of Toronto researching FPGA architecture, CAD and FPGA-based computation.