Time-borrowing platform in the Xilinx UltraScale+ family of FPGAs and MPSoCs

Ilya Ganusov and Benjamin Devlin
Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124-3400
Email: {ilya.ganusov, ben.devlin}@xilinx.com

***Abstract*—This paper presents enhancements to the Xilinx UltraScale+ clocking architecture to support fine-grain time-borrowing. Time borrowing improves performance by redistributing timing slack between fast and slow paths. The UltraScale+ architecture introduces programmable hardware delays and pulse generators embedded in the clocking tree to support time-borrowing based both on clock skew scheduling and pulsed latches. This programmable hardware allows borrowing from a few picoseconds to multiple nanoseconds between sequential pipeline stages without any changes to RTL, placement or routing. Vivado algorithms automatically determine when to skew flip-flop clock or convert them to pulsed latches to achieve the highest possible performance. Using the default Vivado flow, this programmable time-borrowing platform delivers 5.5% F_{max} increase on average over a suite of 89 industrial designs. It is especially effective on high-speed applications, delivering up to 13.7% F_{max} increase on individual designs. We also demonstrate that using non-default features, such as delays cascades or increasing hold margin, can increase average performance gains to 7.4% and 8.5%, respectively. This platform incurs minimum area (less than 0.1% of total chip area) while staying robust in the presence of tight hold constraints and increasing process variation.**

I. Introduction

This paper describes enhancements to the clocking architecture of Xilinx UltraScale+ FPGAs and SoCs [1] that provide a powerful platform for performance optimizations based on time-borrowing. Xilinx UltraScale+ devices are manufactured in the TSMC's 16-nm FinFET+ process node and feature a number of architectural changes compared to the previous generation of FPGAs, including higher-speed transceivers and IO interfaces, flexible clocking tree with time-borrowing technology described in this paper, new UltraRAM memory, and upgraded DSP functionality. These enhancements, coupled with the process advantage, deliver an average of 60% performance improvement compared to the 7-series generation of FPGAs, which was implemented in the TSMC's 28-nm process node [19].

The clocking infrastructure of Xilinx FPGAs was completely redesigned in the 20-nm UltraScale generation of FPGAs to be more regional and segmentable [21]. These changes significantly increased the number of unique clock domains supported on a single device while reducing clock skew and jitter within each of them. Minimizing clock skew is important since it causes imbalance in timing slack available between sequential pipeline stages and decreases the achievable F_{max} for a given design. Extra clock skew can also cause hold violations when datapath delay is faster than the clock skew on the same path. Fixing hold violations on an FPGA usually involves detouring fast paths in the interconnect or inserting dummy lookup tables (LUTs) to slow the datapath, which often negatively affects F_{max} as well.

In addition to clock skew, there are multiple other reasons for uneven distribution of available timing slack in FPGA designs. For example, sequential pipeline stages can have different logic depth, causing one pipeline stage to violate timing requirements, while the next stage with fewer LUTs can meet timing with significant margin, as shown in Figure 1. Uneven slack distribution can also be caused by floorplanning restrictions, control set constraints on register placement, routing congestion, or random noise in place-and-route tools.

Time borrowing techniques address this problem by redistributing timing slack between consecutive pipeline stages so that ideally all pipeline stages become equally critical [3]. This can be achieved either by changing clock arrival times to flip-flops (adding useful skew) or by using latches (or pulsed latches). In some research literature time-borrowing based on clock skew manipulation is referred to as time-stealing while latch-based borrowing is called cycle-stealing. For the remainder of this paper, we refer to both approaches as time-borrowing.

The UltraScale+ clocking architecture provides hardware support for time-borrowing based both on skewed flops and pulsed latches. Clock skew scheduling relies on fine-grain programmable delay elements embedded in the clocking tree. These delay elements can adjust clock skew of flip-flops by as little as 40ps. Time borrowing based on pulsed latches relies on a pulse generator that re-uses delay lines. These programmable delays can also be cascaded to increase borrowing time up to several nanoseconds. All register elements in the UltraScale+ configurable logic block (CLB) can be configured as flip-flops or latches and can use either of the mechanisms to improve performance. Time-borrowing algorithms in Vivado automatically determine when to skew flip-flop clocks or convert them to pulsed latches to achieve the highest performance for a given design.

We demonstrate that the UltraScale+ time-borrowing feature incurs minimum area and power overheads while providing 5.5% F_{max} improvement on average on the industrial bench-

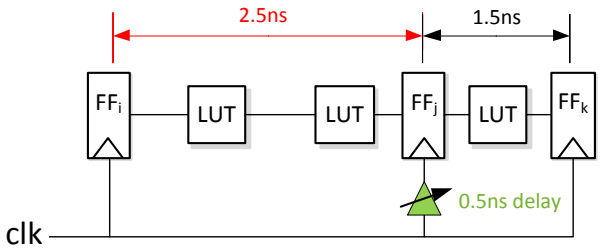

Fig. 1. Example of time-borrowing based on skewed flops

mark suite, with up to 13.7% F_{max} increase on individual designs. We also show that enabling more aggressive optimizations using cascading features and hold router can provide up to 7.4% and 8.5% F_{max} increase on average, respectively. Finally, we show that the UltraScale+ time-borrowing feature incurs minimal area overhead and provides the most optimal F_{max} increase per unit of additional area overhead.

The rest of this paper is organized as follows. Section II discusses how and why time-borrowing can improve performance and how it relates to retiming. Section III describes implementation details of the hardware added to support time-borrowing in the UltraScale+ architecture. Section IV describes two of the slack scheduling algorithms used in this paper, though many more are possible. Section V discusses experimental setup and measurements highlighting various aspects of the technology. Finally, Section VI discusses related work and Section VII concludes the paper.

II. Time-Borrowing Concept

The maximum frequency of a synchronous pipeline is determined by the longest datapath delay between sequential elements of that pipeline. The example in Figure 1 shows two sequential pipeline stages with two paths. The first datapath contains two LUTs and the related interconnect and has a propagation delay of 2.5ns. The next pipeline stage has only one LUT and a shorter interconnect resulting in a datapath delay of 1.5ns. Since all sequential elements are controlled by the same clock, they will have the same timing requirement determined by the period P of that clock. Now let us assume that the timing requirement for this circuit is 2ns (500MHz), in which case the first pipeline stage will have a negative slack of -0.5ns since it needs 2.5ns to complete computation, while the second pipeline stage will have a positive slack of +0.5ns since it only needs 1.5ns.

Time-borrowing can improve performance of this circuit by essentially "borrowing" available positive slack in one pipeline stage to compensate for the negative slack in another pipeline stage. The example in Figure 1 shows how this can be accomplished by intentionally adding 0.5ns to the clock arriving to the middle flop. This delay effectively "stretches" the time available for computation between FF_i and FF_j by 0.5ns, so the first stage's timing requirement will be stretched to 2ns+0.5ns=2.5ns, which is exactly equal to the datapath delay of the first stage allowing it to meet the timing

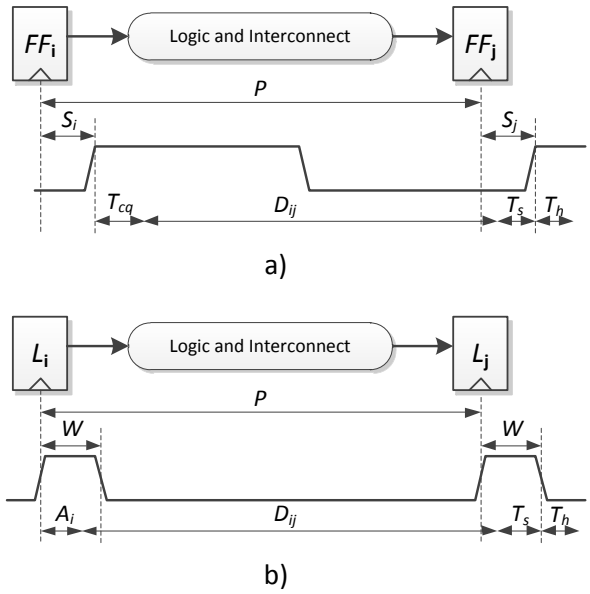

Fig. 2. Setup and hold timing constraints for time-borrowing based on skewed flip-flops (a) and pulsed latches (b)

requirement. At the same time, the extra delay on FF_j clock also creates negative clock skew between flops FF_j and FF_k and tightens its timing constraint by 0.5ns and eliminates any extra positive timing slack that was available before clock skew was introduced. Therefore, adding clock delay to FF_j allowed to improve performance by effectively shifting available slack from one pipeline stage to another.

The UltraScale+ architecture employs time-borrowing based both on skewed flip-flops and pulsed latches. The next subsections more formally review timing constraints for both of the techniques with the help of Figure 2. Cycle time of a clock is denoted by P, clock-to-Q delay by T_{cq}, data-to-Q delay by T_{dq}, setup time by T_s, hold time by T_h, and data propagation delay between sequencing elements i and j by D_{ij}.

A. Time-borrowing based on clock skew

In edge-triggered sequential elements, data is launched from flip-flop i at the rising edge of a clock and has to arrive at destination flip-flop j earlier than the setup time before the next rising edge of clock, as shown in Figure 2a. In the absence of clock skew, the setup time constraint for this timing path will look as follows:

$$T_{cq} + D_{ij} \leq P - T_s \tag{1}$$

If we delay the clock signal to flip-flop j by S_j, it will effectively relax the timing requirement by the same amount since the capture edge will be delayed relative to the launch edge. Delaying clock to flip-flop i would have the opposite effect:

$$T_{cq} + D_{ij} \leq P - T_s + S_j - S_i \tag{2}$$

Based on this constraint, it is easy to see that the relative skew between S_j and S_i determines how much slack is borrowed from the most critical path starting at j. To satisfy the hold constraint, the fastest data from flip-flop i has to arrive at j no earlier than its hold time after the rising edge of the clock signal so that j can hold its current data in a stable state. Adding extra skew on capture flop j increases the minimum datapath delay required to meet the hold constraint, while extra S_i skew helps to satisfy this requirement:

$$T_{cq} + D_{ij} \geq T_h + S_j - S_i \tag{3}$$

These constraints show that adding skew to the endpoint of a path will always improve the setup constraint on that path while tightening the hold constraint. At the same time, adding skew to the startpoint will tighten the setup constraint and relax hold constraints. Therefore, time-borrowing techniques based on skewed flops can effectively re-distribute both setup and hold slack to achieve the best performance.

B. Time-borrowing based pulsed latches

When we use pulsed latches as sequencing elements, data can arrive at any time when the clock is high, unless it is later than the setup time before the falling edge of the clock. Let A_i be the latest data arrival time at endpoint latch i from all startpoint latches k that are connected to i through logic and interconnect:

$$A_i = \max(T_{cq}, A_k + T_{dq}) \tag{4}$$

where T_{cq} corresponds to data arriving at pulsed latch i before rising edge and $A_k + T_{dq}$ after rising edge. The setup time constraint between latches i and j can be described by:

$$A_i + D_{ij} \leq P + W - T_s \tag{5}$$

where W is the width of the pulse when the clock is high, as shown in Figure 2b. In other words, A_i represents the amount of time borrowed from the current path by the previous path, while W is the bound on how much time the current path can borrow from the next path. The actual amount of borrowing can take any value between 0 and pulse width W with infinite resolution. This property represents the main advantage of pulsed latches over skewed flops, since in practice clock skew values S_j and S_i in flop constraints (2) and (3) can be generated only in quantized steps and their precision is further diluted by process variation.

The hold time constraint of pulsed latches can be described by

$$T_{cq} + D_{ij} \geq W + T_h \tag{6}$$

This hold constraint shows the main disadvantage of pulsed latches - they tighten hold by the full width of the pulse independent of how much time is actually borrowed. Unlike skewed flip-flops, they also lack a mechanism to improve hold by delaying the startpoint of the hold path.

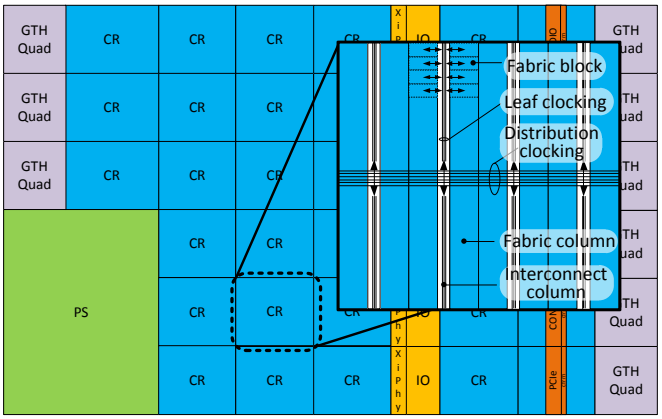

Fig. 3. UltraScale+ MPSoC floorplan

III. Hardware Implementation

A. UltraScale clocking architecture

The clocking infrastructure of Xilinx FPGAs was completely redesigned in the 20-nm UltraScale generation of FPGAs [21] to be more regional and segmentable. Prior to UltraScale, Xilinx FPGA devices contained a central clocking column which was the point of origin of all global clocks in the programmable logic (PL). This column also divided the entire device into segmentable clocking regions (CRs) consisting of two columns and multiple rows.

The UltraScale architecture changed clocking architecture in two major ways. First, instead of CRs spanning half a device width, UltraScale CRs have increased granularity, allowing for increasing the number of CR columns when increasing device size. Second, it increased clock routing flexibility by introducing two types of global routing tracks called routing and distribution. The purpose of routing clock tracks is to bring the clock to the centroid of the clock region, from which it can fan out to all of the loads using distribution tracks. This allows the architecture to both minimize clock skew and improve clock routing flexibility. The re-buffers of the routing and distribution clock tracks are at the boundaries of CRs, as shown in Figure 3.

The CR is made up of any mix of many columns of fabric blocks, where each column consists of a single type of block. For example, a column could be comprised of DSPs, CLBs, BRAM, or UltraRAM. In between these fabric columns, there are interconnect columns which provide connectivity on the FPGA. Running horizontally in the middle of the CR are 24 distribution tracks that connect to 16 leaf clock tracks at each fabric column that span the height of one fabric column in the CR, in north and south directions. Each individual fabric block will then interface with one interconnect block to either the east or west, allowing connectivity for both general data signals, global control signals, and clocking signals. A cluster of logic will see the smallest skew when placed closely inside the same CR, connected to the same leaf track, inside the

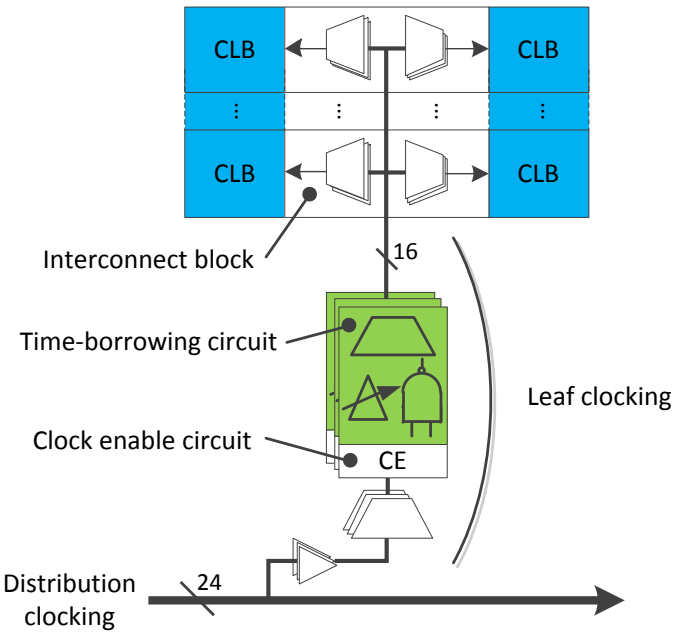

Fig. 4. Clock region distribution track, leaf track, and time-borrowing circuit connectivity

same fabric block. Loads that are on different leaf tracks or span multiple CRs will see higher clock skew.

B. Time-borrowing circuit

The UltraScale+ time-borrowing implementation uses 16 time-borrowing circuits (consisting of multiple programmable delays and a pulse generator) at each point where the distribution track connects to the leaf clocks, as shown in Figure 4. This means for every 960 flip-flops there are 16 time-borrowing circuits. A finer granularity is explored in section V-D, but shown to be less effective. The time-borrowing circuit is shown in Figure 5. The *Clk in* connection takes the input clock, *Cascade in* takes a *Cascade out* connection from a neighboring time-borrowing circuit, and *Clk out* is the corresponding delayed or pulsed clock signal connected to flip-flop loads. Delay is generated in this circuit by a inverter delay-chain, where the exit point can be programmed in 2^N increments of the smallest selectable delay for a single time-borrowing circuit. The output of the delay circuit line can also optionally connect to a NAND gate in order to produce a pulse on the positive edge of *Clk in*. This is used when loads are converted from flip-flops into pulsed latches. The *Cascade in* and *Cascade out* connections allow for multiple neighbouring time-borrowing circuits to be joined together to form larger delays and pulses that otherwise are not possbile from a single circuit. The UltraScale+ clocking provides 16 leaf tracks for complex designs that use multiple clocks, but in the average design many of these tracks are unused and can take advantage of this cascade feature.

We designed for delay taps that gave post-layout extracted delay values of 41ps, 96ps, 168ps, and 295ps in the slow corner, and using the bypass connection when no extra delay is

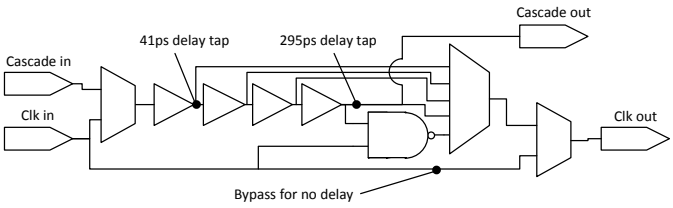

Fig. 5. Time-borrowing circuit diagram

required. The NAND pulse generator is capable of generating pulses of width equal to the largest delay of 295ps, which was chosen to satisfy minimum pulse width requirements for both pulse latches and flip-flops.

The cascade feature allows for larger delays to be formed in multiples of 295ps, with a smaller adjustment of 41ps, 96ps, 168ps, and 295ps in the final time-borrow circuit used to drive the leaf clock. When the cascade feature of the delay circuit is used, it is still possible to use the respective leaf clock track by connecting *Clk in* to *Clk out* through the output multiplexor. The delay used in ILP problem formulation will also include fast-corner derate for hold timing, min-max variation spread, and change in clock pessimism removal (CPR) if clock tracks are changed for a given startpoint and endpoint path. CPR helps by removing the skew introduced from min-max variation in a given hold or setup analysis corner up to the common node before clock paths diverge.

Figure 6 shows an example of creating delayed version of user clock *Clock A*. Here we show three CLB blocks that are receiving the same clock, but additional delay is given to the third CLB by passing it through the time-borrowing circuit. There are two penalties apparent from doing this. First, CPR is impacted for loads with paths between the top two CLBs and the third CLB. In this case, the common node is moved upstream in the clock network to the distribution track and will be penalized by the min-max variation seen over the delay circuit and clock leaf track. The second penalty is the increased power consumed by using the delay circuit and additional leaf track.

IV. Time-Borrowing Algorithm

The purpose of the time-borrowing algorithm is to determine the configuration of programmable clock delays and pulse generators to achieve the highest F_{max}. In this paper, we determine the optimal configuration by combining the setup and hold constraints described in equations 2, 3 with the capabilities of available hardware, and expressing them as a mixed-integer linear program (MILP), described in equations 7-17.

For each timing path starting at sequential element i and ending at sequential element j, $S_{j_{min}}$ denotes clock delay assigned to j, $S_{i_{max}}$ denotes clock delay assigned to i, $W_{j_{min}}$ denoted the amount of time-borrowing from the timing paths starting at j, and W_i denotes the amount borrowed by a path ending in i. Constants $SKEW_{min}$ and $SKEW_{max}$ denote minimum and maximum value of each delay tap due

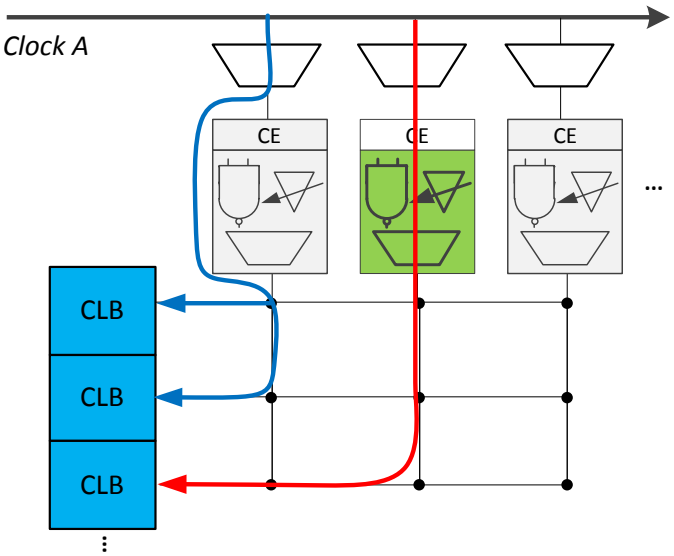

Fig. 6. Example of creating delayed version of a *Clock A* using the time-borrowing circuit

to process variation, while $PULSE_{min}$ and $PULSE_{max}$ indicate minimum and maximum values for the generated pulse widths. Each binary variable $isSkew_{j_n}$ is true if n_{th} delay tap has been chosen to skew clock on j. Similarly, the binary variable $isLatch_j$ indicates if sequential element j is converted to a pulsed latch. Continuous variables Wj and Wi determine the amount of time-borrowing on pulsed latches j and i, respectively. Given these definitions, the MILP objective it to minimize P while satisfying constraints 7-17 for each path from i to j:

$$D_{ij} \leq P - T_s + S_{j_{min}} - S_{i_{max}} + W_{j_{min}} - W_{i_{max}} \quad (7)$$

$$D_{ij} \geq T_h + S_{j_{max}} - S_{i_{min}} + isLatch_j \cdot PULSE_{max} \quad (8)$$

$$S_{j_{min}} = \sum_n isSkew_{j_n} \cdot SKEW_{min_n} \quad (9)$$

$$S_{j_{max}} = \sum_n isSkew_{j_n} \cdot SKEW_{max_n} \quad (10)$$

$$S_{i_{min}} = \sum_n isSkew_{i_n} \cdot SKEW_{min_n} \quad (11)$$

$$S_{i_{max}} = \sum_n isSkew_{i_n} \cdot SKEW_{max_n} \quad (12)$$

$$W_{j_{min}} = isLatch_j \cdot W_j, 0 \leq W_j \leq PULSE_{min} \quad (13)$$

$$W_{i_{max}} = isLatch_i \cdot W_i, 0 \leq W_i \leq PULSE_{max} \quad (14)$$

$$0 \leq \sum_n isSkew_{j_n} + isLatch_j \leq 1 \quad (15)$$

$$0 \leq \sum_n isSkew_{i_n} + isLatch_i \leq 1 \quad (16)$$

$$isSkew \text{ and } isLatch \text{ are integral} \quad (17)$$

In designs with multiple clock domains, the objective function is extended to minimize a sum of periods of all clock domains that do not meet timing. To define the MILP problem for each design mapped onto UltraScale+ FPGAs, we first extract timing graphs with the relevant setup and hold information. In particular, we collect all setup paths that are within 2x of the maximum optional clock skew or pulse width delay from the worst negative slack (WNS) path. We also collect all hold paths where hold slack is less than the maximum clock skew delay/pulse width. Since the same path can often cause hold violations in different corners, the hold subgraph needs to be extracted in both fast and slow corners. The Vivado timing engine has been optimized specifically for this extraction task to provide full timing subgraphs at reasonable runtimes.

All sequential endpoints are enumerated and assigned $isSkew$ binary variables. Sequential elements that share the same clock control set or latch configuration are assigned to the same binary variable. In addition to $isSkew$ binary variables, all sequential elements in CLE (with the exception of some LUTRAM and SRL pins) are also assigned $isLatch$ variables.

The Vivado Design Suite [23] version 2016.1 implements two versions of the time-borrowing algorithm. The first version is part of the default Vivado flow and uses a modified version of MILP formulation described by equations (7-17) combined with a set of heuristics to optimize runtime and memory requirements while achieving optimal or close to optimal F_{max} results. The second version can be enabled with non-default switches to run a globally optimal MILP problem described in this paper to get optimal results at the expense of extra runtime. The algorithm and experimental results described in this paper are part of this second non-default Vivado flow. Both the default and non-default flows deployed in production continuously evolve with each Vivado release to achieve the highest performance at the lowest runtime cost.

V. Experimental Results

A. Experimental Setup and Benchmark Suite

We evaluate our time-borrowing approach using the Vivado Design Suite [23] version 2016.1. The time-borrowing algorithm is run post-route using production values for delay taps and generated pulses. In this version of Vivado, the placer

TABLE I
Benchmark Design Suite

Metric	Min	Max	Avg
Clock domains	1	28	2
Fmax (MHz)	77.3	850	338
LUT	11,455	462,336	130,538
FF	3,403	553,759	110,739
BRAM	4	984	255
DSP	0	1,752	174
Nets	134,108	3,171,239	879,642
total designs	89		

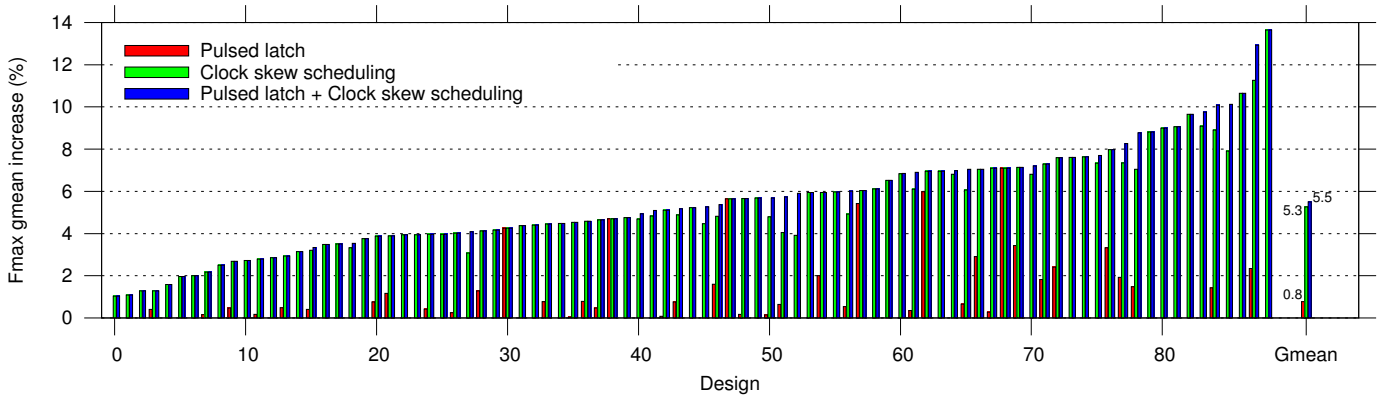

Fig. 7. F_{max} improvement over all designs

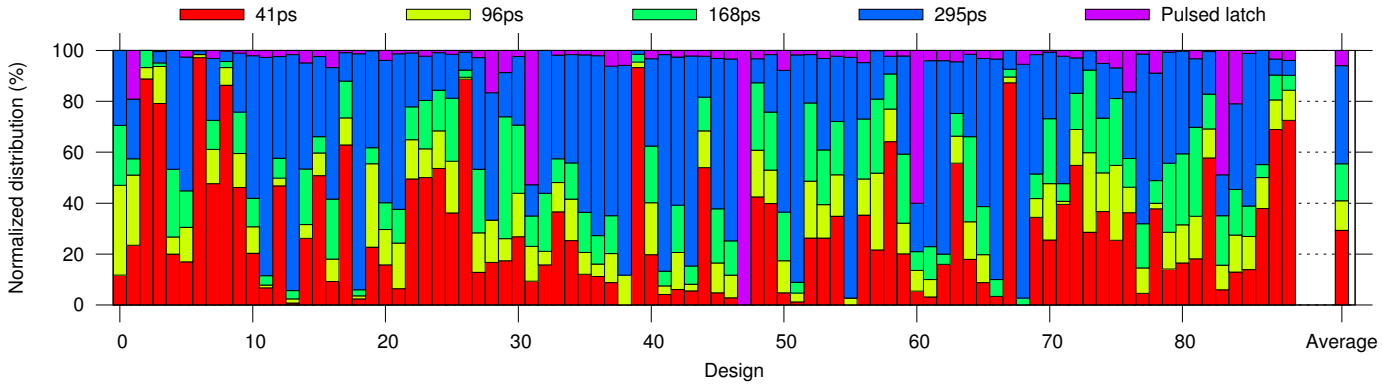

Fig. 8. Distribution of delay taps used over all designs

and router are not aware of the subsequent time-borrowing optimization.

Our design suite consists of 89 representative customer designs and Xilinx IP blocks selected from a wide variety of FPGA applications - from wired and wireless communications to test and measurement to emulation. Their identifying characteristics have been removed from our results. Designs vary in size from 11k to 462k LUTs and the average design contains approximately 130k LUTs. Other characteristics of the benchmark suite are captured in Table I. F_{max} improvements are measured by the geometric mean (gmean) in improvement over negative setup slack clock domains only. All designs in the benchmark are constrained so they have a negative setup slack clock domain, but designs will also often contain related positive slack clock domains by design. There are no hold violations in any design and we do not allow the time-borrowing algorithm to create hold violations by default, although this is explored in later sections.

We use a global time-borrowing algorithm to assign optimal skew values, using standard ILP solvers, such as open source CBC solver [12] and a commercial Gurobi solver [4]. In order to reduce runtime, we disabled the cascade feature by default. The spread between min and max delays for programmable delay line in the setup and hold corners is around $\pm 10\%$, and difference between the slow and fast corners is around 65%. We use a CPR penalty of 80ps when moving loads that were originally on the same leaf clock track onto separate leaf clock tracks. Each leaf clock region can use a maximum of 16 tracks.

All the benchmark designs are implemented (synthesized, optimized, placed, and routed) on the UltraScale+ series of devices. We chose the minimum-sized device from the UltraScale+ device family [22] to accommodate each design. In terms of utilization, we allow up to 90% utilization of LUTs, and up to 95% utilization of FFs and other hard blocks (DSP, BRAM, etc.).

B. F_{max} improvement results

Figure 7 shows the F_{max} improvements over the design suite with three variations. We measured gmean F_{max} improvements of 0.8%, 5.3%, and 5.5% when allowing conversion to pulsed latches only, only clock skew scheduling, and a combination of both, respectively. Figure 7 shows the F_{max} improvement for each design as well as the gmean over the entire design suite. There are no designs where conversion to pulsed latches gives the greatest F_{max} improvement, mainly due to hold violations that arise from their use. There are 12 designs where the combination of clock skew scheduling and pulsed latches gives $>1\%$ higher improvement than either of the techniques alone, as flip-flops can be used on hold-critical paths, and latches used opportunistically.

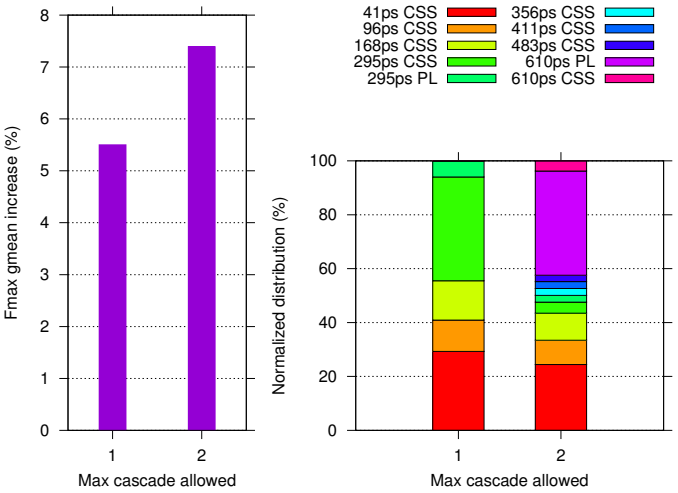

Fig. 9. F_{max} improvement when allowing cascade connections and resulting tap distribution change

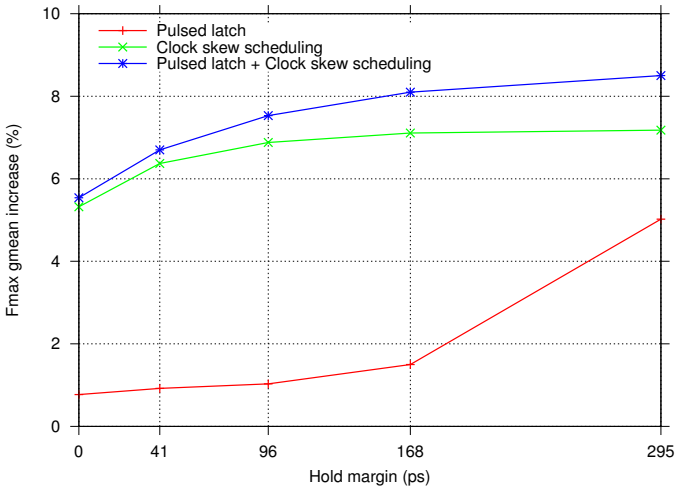

Fig. 10. F_{max} improvement sensitivity to hold margin

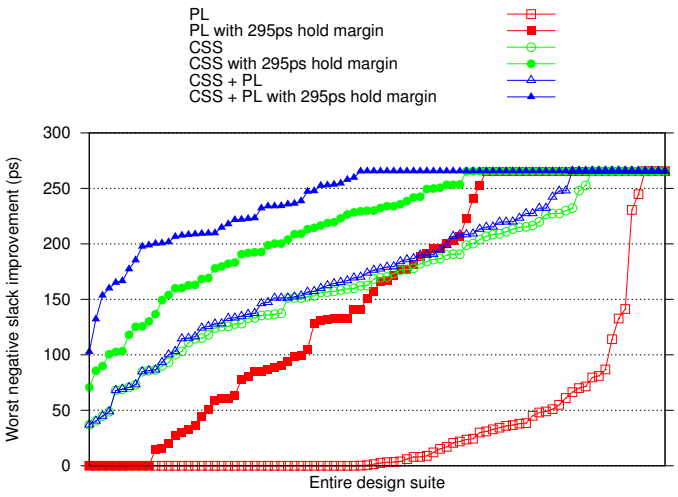

Fig. 11. Average setup WNS improvement over all designs (each line sorted independently) with and without hold margin

Figure 8 shows the distribution of delay taps used for each design. The largest 295ps delay tap is used the most, followed by the smallest 41ps delay tap. The largest delay tap would be used to give the biggest F_{max} improvements, with the smaller fine grain delay tap likely used to fix hold violations where start and end points are shared on both hold and setup critical paths. We can see there is no correlation between the F_{max} improvement and the distribution of delays used. Although pulsed latches provide optimal improvement to setup violations, because they also introduce hold violations they are not as widely used. Also due to control set restrictions a slice must be all flip-flops or all pulsed latches.

Figure 9 shows the improvement and delay tap distribution when utilizing the cascade connection. By allowing two neighboring time-borrowing circuits to cascade, we are able to increase the gmean F_{max} improvement from 5.5% to 7.4%, delivering an additional 1.9% performance improvement.

C. Sensitivity analysis to hold

We can evaluate the impact of hold constraints by measuring F_{max} after allowing the time-borrowing algorithm to violate hold. The Vivado hold router can often fix hold violations by detouring fast nets in the interconnect. Relaxing the hold constraint in the MILP solver provides an upper bound on the F_{max} improvements if such detouring is successful. Figure 10 shows the impact on F_{max} when we allowed for hold margin in steps equal to the delay tap value. Hold margin is the amount of additional slack a path is given when calculating hold requirements. For example, a hold margin of 295ps means all hold paths in the design are given an additional 295ps of slack (derated down into the fast-min corner) when calculating hold timing. In this case, the 295ps delay tap or pulsed latch can be implemented with no worry about introducing hold violations. This is why pulsed latches do not see a large improvement until the amount of hold margin is equal to the pulse length of 295ps, in which case there is no hold penalty for using a pulsed latch. The reason this still under-performs when compared to clock skew scheduling is due to the fact that some blocks such as DSPs, BRAMs, SRLs, and LUTRAMs cannot be converted into pulsed latches. These results also show that by allowing 295ps of hold margin we can increase the gmean F_{max} improvement to 8.5%. The time-borrowing algorithm is currently run as a post-route step in Vivado, but this shows that allowing for hold violations and then following up with a post-time-borrow pass of the hold router can nearly double performance improvement from time-borrowing.

Figure 11 shows the related average improvement in setup WNS when we allowed hold margin. Pulsed latches show the biggest improvement when hold margin is introduced, and a combination of pulsed latches and clock skew scheduling allows over half the designs to get the maximum benefit possible. This shows that higher speed designs will be able to see the largest F_{max} improvements when hold violations are allowed.

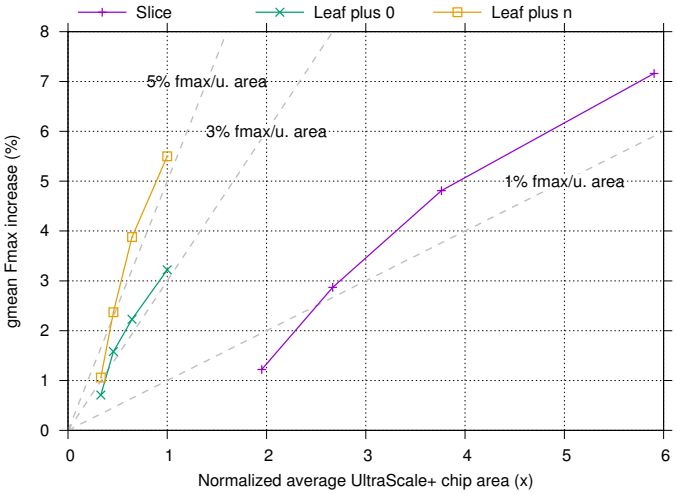

Fig. 12. F_{max} improvement vs area for different implementation locations and delay line lengths

D. Correlation between location, cost, and performance

The location and number of delays of the time-borrowing circuit will impact both the cost and the performance gain possible. Figure 12 shows the F_{max} improvement possible if we had hypothetically introduced the time-borrowing circuit at the slice level, for example right before the flip-flops in the CLE, or higher up the clock structure network at the distribution to leaf clock junction. At both of these locations we can allow the time-borrow circuit to use no extra clock tracks (leaf plus 0), or up to the maximum of 16 tracks (leaf plus n). Each time-borrow circuit can have fewer delay taps in order to decrease its area, at the expense of maximum F_{max} improvement. Implementing the time-borrowing circuit at the slice level allows for larger F_{max} improvements of up to 7.2%. However, compared to the leaf-level implementation, the area overhead grows by nearly 6x. The contour of that product of F_{max} per unit area shows the most optimal location is the leaf, with all delay taps present.

VI. Related Work

Time-borrowing using clock skew has been studied extensively in the research literature. Fishburn [3] was one of the first to recognize that clock skew can be used to reduce the clock period and propose MILP formulation to improve both performance and silicon yield in ASICs. Singh and Brown [17] proposed to use 4 shifted global clock lines in FPGAs which downstream flops can choose to exploit time-borrowing. Yeh et al. [24] suggested to put programmable delay elements along the clocking H-tree, while Dong and Lemieux [2] propose to embed them at the level of individual flip-flops.

Timing optimization based on latches has also been studied extensively. Most prior work has represented the latch-based optimization problem using linear constraints and solved it using linear programming (LP) [14], [18], [24] or graph algorithms [16], [11]. The authors optimize circuit performance by using two clocks with adjustable duty cycles. The use of pulsed latches in ASICs have been explored by Lee et. al [7], [6]. Using flip-flop-like timing constraints, their optimization strategy relies on exploiting the difference between pulse widths and clock delays to maximize performance potential. They propose to use multiple pulse widths and skewed clocks, which is similar to our approach.

Pulsed latches have recently been explored in the FPGA context. Teng and Anderson [20] propose to generate clock pulses at the clock root level and selectively convert critical downstream flip-flops to latches. This approach allows to exploit pulse latch-based time-borrowing without any extra hardware. The downside of this approach is extra variation on the clock path which would incur significant penalty on hold, which is a very critical limitation for pulsed latch use. Altera's Stratix V FPGA architecture [9] uses pulsed latches for all sequential elements in reconfigurable logic blocks. Similar to our findings, they show that pulsed latch time-borrowing in modern FPGAs is severly limited by hold.

Retiming physically relocates flip-flops or latches across combinational logic to balance the delays between combinational stages. Sequential elements can move backward or forward. A forward push of a flip-flop gives the combinational stage feeding into the flip-flop more time to complete, whereas a backward move has the opposite effect. Retiming was first introduced by Leiserson and Saxe [8]. Their initial work has been extended in multiple directions such as more efficient algorithms (e.g., [7]), retiming using level-sensitive latches (e.g., [11], and retiming for low power [13], [5]. Altera's Stratix 10 FPGA [10] is planning to extensively rely on retiming to improve application F_{max}.

Sapatnekar formally shows that retiming and time-borrowing are equivalent and should lead to similar performance improvements [15]. At the same time, retiming changes the position and number of flip-flops, making the design debugging process more difficult as a designer may not be able to correlate the retimed design with the original RTL specification. Furthermore, time-borrowing via retiming is inherently quantized because it is impossible to relocate a flip-flop to be in the middle of a logic gate if such granularity is necessary. Nevertheless, retiming is useful in certain contexts and the Xilinx Vivado suite provides retiming capabilities in physical synthesis tools [23].

VII. Conclusion

This paper presents enhancements to the Xilinx UltraScale+ clocking architecture to support fine-grain time-borrowing. Time-borrowing improves performance by re-distributing timing slack between fast and slow paths. The UltraScale+ architecture introduces programmable hardware delays and pulse generators embedded in the clocking tree at the junction of clock distributuion tracks and leaf clock drivers. These modifications enable time-borrowing based both on clock skew scheduling and pulsed latches. This programmable hardware allows borrowing from a few picoseconds to multiple nanoseconds between sequential pipeline stages without any changes to RTL, placement or routing. Vivado algorithms automatically

determine when to skew flip-flop clock or convert them to pulsed latches to achieve the highest possible performance.

Using the default Vivado flow, this programmable time-borrowing platform delivers an average of 5.5% Fmax increase over a suite of 89 industrial designs. It is especially effective on high-speed applications, delivering up to 13.7% Fmax increase on individual designs. We also demonstrate that using non-default features, such as delay cascades or increasing hold margin, can increase average performance gains by 7.4% and 8.5%, respectively, at the expense of additional runtime. This platform incurs minimum area overhead (less than 0.1% of total chip area) while staying robust in the presence of tight hold constraints and increasing process variation.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful feedback and suggestions. We would also like to thank Satish Sivaswamy, Jindrich Zejda, Walt Manaker, Atul Srinivasan, Pari Kannan, Robert Ondris, Dinesh Gaitonde, Ray Chen, and Manu Jose for their contributions to the evaluation and commercialization of time-borrowing in the UltraScale+ architecture.

REFERENCES

[1] S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig. A 16-nm multiprocessing system-on-chip field-programmable gate array platform. *IEEE Micro*, 36(2):48–62, Mar 2016.

[2] X. Dong and G. G. F. Lemieux. Pgr: Period and glitch reduction via clock skew scheduling, delay padding and glitchless. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 88–95, Dec 2009.

[3] J. P. Fishburn. Clock skew optimization. *IEEE Trans. Comput.*, 39(7):945–951, July 1990.

[4] I. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[5] K. N. Lalgudi and M. C. Papaefthymiou. Fixed-phase retiming for low power design. In *Low Power Electronics and Design, 1996., International Symposium on*, pages 259–264, Aug 1996.

[6] H. Lee, S. Paik, and Y. Shin. Pulse width allocation and clock skew scheduling: Optimizing sequential circuits based on pulsed latches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(3):355–366, March 2010.

[7] S. Lee, S. Paik, and Y. Shin. Retiming and time borrowing: Optimizing high-performance pulsed-latch-based circuits. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 375–380, Nov 2009.

[8] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

[9] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu. Architectural enhancements in stratix v^{TM}. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 147–156, New York, NY, USA, 2013. ACM.

[10] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, and J. Van Dyken. The stratix^{TM}10 highly pipelined fpga architecture. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 159–168, New York, NY, USA, 2016. ACM.

[11] B. Lockyear and C. Ebeling. Optimal retiming of level-clocked circuits using symmetric clock schedules. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1097–1109, Sep 1994.

[12] R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, Jan 2003.

[13] J. Monteiro, S. Devadas, and A. Ghosh. Retiming sequential circuits for low power. In *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, pages 398–402, Nov 1993.

[14] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(3):322–333, Mar 1992.

[15] S. S. Sapatnekar and R. B. Deokar. Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1237–1248, Oct 1996.

[16] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Graph algorithms for clock schedule optimization. In *Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers., 1992 IEEE/ACM International Conference on*, pages 132–136, Nov 1992.

[17] D. P. Singh and S. D. Brown. Constrained clock shifting for field programmable gate arrays. In *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays*, FPGA '02, pages 121–126, New York, NY, USA, 2002. ACM.

[18] B. Taskin and I. S. Kourtev. Delay insertion method in clock skew scheduling. In *Proceedings of the 2005 International Symposium on Physical Design*, ISPD '05, pages 47–54, New York, NY, USA, 2005. ACM.

[19] B. Taylor, R. Crotty, G. Starr, and R. Wittig. Xilinx 28nm Generation Programmable Families. In *Hot Chips 22: Stanford University, Stanford, California, August 15–17*, 2010.

[20] B. Teng and J. H. Anderson. Latch-based performance optimization for fpgas. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 58–63, Sept 2011.

[21] Xilinx. *UltraScale Architecture Clocking Resources*. Xilinx, ug572 (v1.3) edition, November 24, 2015.

[22] Xilinx. *UltraScale Architecture and Product Overview*. Xilinx, ds890 (v2.8) edition, June 3, 2016.

[23] Xilinx. *Vivado Design Suite User Guide: Design Flows Overview*. Xilinx, ug892 (v2016.2) edition, June 8, 2016.

[24] C.-Y. Yeh and M. Marek-Sadowska. Skew-programmable clock design for fpga and skew-aware placement. In *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, FPGA '05, pages 33–40, New York, NY, USA, 2005. ACM.