Automated Design of FPGAs Facilitated by Cycle-Free Routing

Ang Li
Dept. of Electrical Engineering
Princeton University
Princeton, NJ, USA
angl (at) princeton (dot) edu

Ting-Jung Chang
Dept. of Electrical Engineering
Princeton University
Princeton, NJ, USA
tingjung (at) princeton (dot) edu

David Wentzlaff
Dept. of Electrical Engineering
Princeton University
Princeton, NJ, USA
wentzlaf (at) princeton (dot) edu

***Abstract*—As device technology advances into the sub-10nm era, the design costs of *Field Programmable Gate Arrays* (FPGAs) built out of fully-custom, hand-layout blocks have increased dramatically. At the same time, *Embedded FPGAs* (eFPGAs) are picking up steam in heterogeneous system-on-chip designs, in which case supporting customizable FPGA architectures and reducing design costs are more crucial than squeezing the last bit of performance out of the transistors.**

To reduce the cost and complexity of FPGA designs, prior works have proposed to build FPGAs using *Electronic Design Automation* (EDA) tools and standard-cell libraries. Though functionally viable, this approach faces two challenges: 1) An accurate timing model is crucial for FPGA implementation tools to produce correct and optimal results. However, post-layout *Static Timing Analysis* (STA) with EDA tools is error-prone on FPGAs, because the typical FPGA routing graphs contain many cycles at design time. 2) Conventional FPGA design relies heavily on iterative/empirical improvements to achieve optimal floorplanning and time-budgeting. Without such insights, blocks may be shaped and constrained sub-optimally.

This work addresses the first challenge by proposing an algorithm to derive cycle-free sub-graphs. A cycle-free sub-graph is achieved by logically ranking the routing tracks and selectively removing some switch block connections. Each sub-graph enables accurate, per-switch, post-layout STA, and the union of multiple sub-graphs covers all the timing arcs of the FPGA. Furthermore, our proposed approach addresses the second challenge by enabling the creation of intrinsically cycle-free FPGAs that facilitate a flat multi-block or full-chip design flow. By blending the blocks, the EDA tools can exploit more optimization opportunities and automatically adapt to heterogeneous blocks. Our experiments show that the routability of cycle-free routing graphs is comparable to conventional FPGA routing graphs, and the Quality of Results (QoR) of the FPGA layout is superior to the result of previous approaches.

***Index Terms*—FPGA, FPGA Routing Architecture, EDA Flow**

This material is based on research sponsored by the NSF under Grant No. CCF-1453112, Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7852. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA), the NSF, or the U.S. Government.

I. Introduction

Field Programmable Gate Arrays (FPGAs) have scaled in logic capacity and performance over the past few decades. This scaling has been enabled by using regular architectures built out of small repeating blocks, as shown in Fig. 1. To optimize *Performance, Power, and Area* (PPA), these blocks are typically designed and optimized extensively by hand. Unfortunately, this key design aspect has become more of a challenge than a blessing as device technology has advanced into the sub-10nm era [1]. Conventional FPGA design costs, including floorplan elaboration, cell customization, and custom timing/power analysis, have increased dramatically as process design rules and fabrication limitations have evolved.

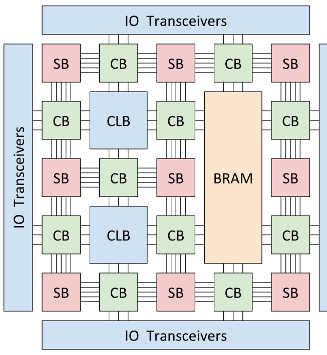
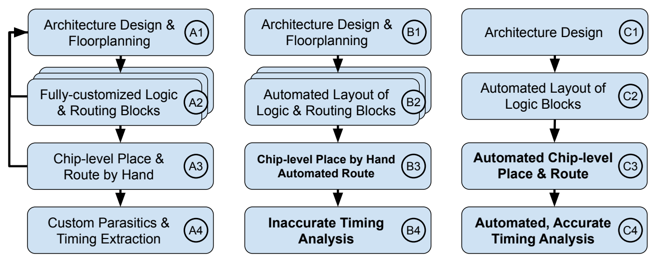


Fig. 1: FPGA Architecture. **CLB**: Configurable Logic Blocks containing Look-Up Tables and registers; **CB**: Connection Blocks containing programmable switches connecting CLB pins and routing tracks; **SB**: Switch Blocks containing programmable switches connecting routing tracks.

At the same time, *embedded FPGAs* (eFPGAs) have emerged as a promising component in heterogeneous *System-on-Chips* (SoCs), providing high performance and versatility for a broad spectrum of applications [2]–[5]. However, the conventional approach to designing FPGAs is costly when exploring novel FPGA architectures, integrating new hardwired blocks, or migrating between device technologies. Moreover, the lack of synergy between the design flows of FPGAs and other types of cores poses a challenge to chip-level timing closure and verification.

Addressing these issues, prior works [6]–[9] have proposed to build FPGAs using EDA tools and standard-cell libraries. Fig. 2a and 2b compare the conventional design flow and the EDA flow. At the cost of some PPA degradation, this design flow can be easily adapted to various FPGA architectures across different device technologies. When augmented with just a narrow selection of hand-optimized switches and configuration cells, the gap between FPGAs built this way and commercial FPGAs can be narrowed down to 60% overhead in area and 30% overhead in average switch delay [8], [10].


Architecture Design & Floorplanning
A1
Fully-customized Logic & Routing Blocks
A2
Chip-level Place & Route by Hand
A3
Custom Parasitics & Timing Extraction
A4
Architecture Design & Floorplanning
B1
Automated Layout of Logic & Routing Blocks
B2
Chip-level Place by Hand Automated Route
B3
Inaccurate Timing Analysis
B4
Architecture Design
C1
Automated Layout of Logic Blocks
C2
Automated Chip-level Place & Route
C3
Automated, Accurate Timing Analysis
C4

(a) Conventional
+ Optimal PPA
– High design cost
– Custom STA

(b) Prior Works
– Poor PPA
● Moderate cost
– Inaccurate STA

(c) This Work
● Moderate PPA
+ Low design cost
+ Accurate STA

Fig. 2: Comparison between FPGA design flows

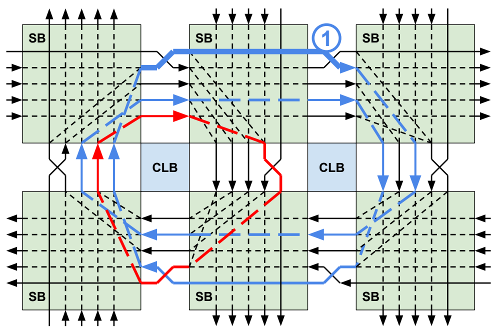


Fig. 3: Two example cycles (highlighted with colored lines) in an FPGA using the Wilton switch block. For clarity, blocks are not to-scale, and irrelevant connections are omitted. ① shows a long track ($L = 2$) which runs through the switch block in the middle.

Besides PPA degradation, the design flows proposed by prior works face another challenge: post-layout *Static Timing Analysis* (STA). An accurate and complete timing model is crucial for FPGA implementation tools to produce the correct and optimal results. However, the routing resources of FPGAs, namely the routing tracks and the programmable switches, are a rats nest of cycles, i.e., combinational loops, as shown in Fig. 3. A valid bitstream eliminates these cycles by fixing each switch to one specific configuration. At design time, nonetheless, the automated STA tools must measure all possible configurations. Unfortunately, these tools cannot calculate the propagation delay of point-to-point links when cycles are present in the circuit. They have no choice but to cut timing paths at random until a cycle-free timing graph is derived. This random cutting causes potentially critical paths to be lost, and produces inaccurate or even incorrect results.

In this paper, we propose an algorithm for designing **cycle-free FPGA routing graphs**. These routing graphs can be very similar to conventional/commercial routing architectures or can be novel routing architectures. Our proposed algorithm leverages the insight that while typical FPGA routing graphs appear to contain copious cycles, the regularity in the switch blocks like the Wilton switch block [11] actually brings them very close to being cycle-free with the removal/rearranging of a few key connections. This work takes inspiration from the computer architecture *Network on Chip* (NoC) community, in particular the turn model [12], an analytic framework for designing deadlock-free wormhole routing algorithms for NoCs. Our algorithm ranks the FPGA routing tracks logically, then constrains the routing graph so that connections are valid only if the output track is ranked higher than or equal to the input track while respecting the turn model.

This technique addresses the STA challenge by providing a systematic approach to identifying the minimally needed number of cycle-breaking connections. Disabling the timing paths corresponding to these connections enables automated, accurate, per-switch STA. Moreover, this technique facilitates the creation of intrinsically cycle-free FPGAs, which enables a flat multi-block or full-chip design flow, as shown in Fig. 2c. This flat design flow shifts the burden of floorplanning and constraining the blocks from FPGA designers to the EDA tools and opens up more optimization opportunities.

Our work makes the following contributions:

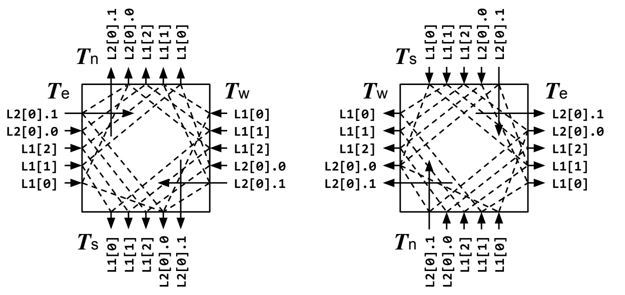
1) Proposes an algorithm to construct cycle-free FPGA routing graphs, either derived from conventional, existing FPGAs, or being a novel routing graph design.
2) Quantitative evaluation of the routability of cycle-free FPGA architectures with VTR [13].
3) Quantitative evaluation of the flat multi-block design flow enabled by cycle-free FPGA architectures.

II. Related Work

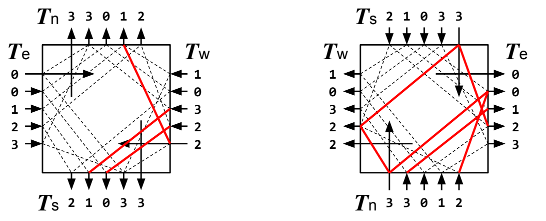
For the past few decades, commercial FPGAs have employed a fully-custom design flow to achieve optimal PPA. However, as device technology advances into the sub-10nm era, the complexity of manual design and custom verification has rendered this flow less cost-effective [1]. Addressing this issue, prior works have proposed various design automation approaches [6]–[10], [14], [15].

Neumann *et al.* [14] proposed to design FPGAs using automated layout generators, which are impractical for state-of-the-art technology nodes. A more technology-agnostic approach is to generate RTL netlists and then use EDA tools. Aken'Ova *et al.* [10] proposed to build eFPGAs out of synthesizable netlists with extensive physical-level floorplanning. Kuon *et al.* [7] adopted a similar strategy, generating RTL for repeatable tiles that are laid out with their specially-designed layout tools. Detailed planning and specialized tools are required in these works, resulting in limited portability.

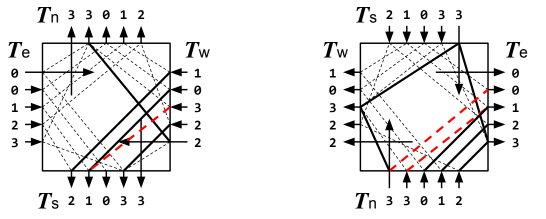
While classic FPGAs mostly use bidirectional routing tracks, pass-transistors, and tri-state buffers, modern FPGAs converge to unidirectional tracks and buffered switches, resembling standard ASIC designs. Following this trend, recent studies proposed to adopt the standard ASIC design methodology and use standard-cell libraries. Liu *et al.* [15] presented an open-source FPGA generator that produces synthesizable RTL for a customizable FPGA architecture. Each block is laid out individually, replicated, and assembled at the top level. No information was provided about how timing characteristics are extracted. Kim *et al.* [6] disclosed more details in their approach to building a fully-synthesizable FPGA with a similar bottom-up methodology. To deal with cycles within and between blocks, they generate SDC constraints to disable timing paths that are used less frequently. Prashanth *et al.* [9] proposed another systematic approach to disabling timing paths in a flat full-chip layout flow. These approaches cut many fewer connections than random, yet still leave a large portion of switches un-constrained. Tang *et al.* [8] augmented the standard-cell libraries with a few custom cells and reported great improvements on PPA. No constraints are applied during layout, yet they expect better *Quality of Results* (QoR) if constraints are applied properly.



(a) A Wilton switch block for uni-directional tracks. The left sub-figure shows the connections from horizontal tracks to vertical tracks, and the right sub-figure shows the opposite. Straight connections are omitted. Long arrows represent L2 tracks that pass through the block.



(b) Logical ranking of a Wilton switch block. Cycle-breaking connections are highlighted with red solid lines.



(c) Logical ranking of a cycle-free variation of the Wilton switch block. Rearranged connections are highlighted with black solid lines; removed connections are highlighted with red dashed lines.

Fig. 4: Logical ranking of a Wilton switch block and its cycle-free variation

III. Cycle-Free Routing Graph

A. Background

In FPGAs composed of uni-directional routing tracks, three types of cycles may exist at design time:

(i) Cycles inside CLBs. For example, a LUT output may be connected back to its inputs through the local crossbar when registers are bypassed.
(ii) Cycles consisting of routing tracks and programmable switches only, as shown in Fig. 3.
(iii) Cycles consisting of routing tracks and combinational paths through CLBs.

Type (i) cycles can be eliminated by tweaking the CLB structure or disabling the paths that bypass the registers. We will cover type (iii) cycles in the next section when we discuss layout methodologies. In this section, we propose an algorithm to analyze FPGA routing graphs and identify key connections that cause type (ii) cycles. Since CLBs and connection blocks do not contribute to the formation of type (ii) cycles, our analysis focuses on the switch blocks. Moreover, because of the regularity in FPGA architectures, inspecting the connection patterns of repeated switch blocks is sufficient to analyze the millions of tracks and switches across an entire FPGA.

B. Conventional Switch Blocks

Fig. 4a shows the connection pattern of the Wilton switch blocks [11] used in Fig. 3. Routing tracks are grouped into 4

Listing 1: Logical Ranking Algorithm

```
INIT:
  for trackset in [$T_n$, $T_e$, $T_s$, $T_w$]:
    for track in trackset:
      rank[track] = None
  init_track = $T_e$[0]
  init_rank = 0
  q = Queue()
  q.enque(Tuple(init_track, init_rank))
BFS:
  while q is not empty:
    track_i, rank_i = q.deque()
    for track_o in $F_{sb}$(track_i):
      if rank[track_o] is None:
        rank_o = rank_i
        if ((track_i$\in T_n$ and track_o$\in T_e$) or
            (track_i$\in T_w$ and track_o$\in T_s$)):
          rank_o = rank_o + 1
        rank[track_o] = rank_o
        q.enque(Tuple(track_o, rank_o))
```

sets based on their directions, namely the northbound set T_n, eastbound set T_e, southbound set T_s and westbound set T_w. Each track in a set is uniquely identified as $L[i].s$, in which L represents the length of the track; i indexes tracks of the same length; and s differentiates the sections of a long ($L > 1$) track. The connection pattern can be expressed as a multi-mapping function between the sets: $F_{sb} = T_n \cup T_e \cup T_s \cup T_w \to T_n \cup T_e \cup T_s \cup T_w$. The switch block does not contain any cycles within itself, yet multiple block instances connected together form numerous cycles across the entire FPGA.

C. Cycle Analysis Based on Logical Ranking

In on-chip network designs, assigning priorities to different channels and forbidding any traffic from depending on lower-priority traffic are a common approach to eliminating cyclic dependencies. Inspired by this approach, we propose to rank the FPGA routing tracks logically. All connections that satisfy any of the following criteria cause cycles. These connections are referred to as cycle-breaking connections hereinafter.

1) The output track is ranked lower than the input track
2) The output track is ranked the same as the input track, and the connection is a north-east or west-south turn (or another pair of turns that respect the turn model [12])

A cycle-free sub-graph can be derived by removing all cycle-breaking connections from all switch blocks. To prove this conclusion, we first study how cycles are formed: A cycle is a route, i.e., a sequence of tracks connected one after another, in which the last track is connected to the first track. Removing all connections satisfying criterion 1 guarantees that the logical rank either increases monotonically or remains the same along any route. In the former case, the last track cannot be connected to the first track since it is ranked higher. In the latter case, note that according to the turn model, a route must take at least one of the two turns specified in criterion 2 to return to its starting position. Therefore, removing all connections satisfying criterion 2 guarantees that a route cannot return to its starting position if all of its tracks are ranked the same, thus impossible to form a cycle.

Different sets of cycle-breaking connections are identified with different logical rankings. Listing 1 shows an example algorithm for finding a logical ranking that minimizes the number of cycle-breaking connections based on *Breadth-First Search* (BFS). Fig. 4b shows the logical ranking assigned to the Wilton switch block used in Fig. 3 and the cycle-breaking connections identified with this ranking.

D. *Cycle-Free Switch Block*

Switch blocks with no cycle-breaking connections are **Cycle-Free Switch Blocks**. Existing switch blocks can be modified into cycle-free switch blocks by removing or rearranging the cycle-breaking connections. Fig. 4c shows the cycle-free variation of the Wilton switch block. Novel cycle-free switch blocks can also be designed from scratch by assigning the logical ranking first and avoiding cycle-breaking connections when creating the connection pattern.

IV. Design Methodology

A. *Chip-Level STA of Conventional FPGAs*

Due to the cycles mentioned above, prior works are limited to a bottom-up design methodology. Each block is laid out individually, converted to a black-box macro, then replicated and stitched together at chip level. Our proposed algorithm can be applied to guide the post-layout STA in this flow:

1) Run block-level STA on the logic blocks including CLBs and other IP blocks such as BRAM and DSP blocks. Timing characteristics of these blocks are saved for the packer in the FPGA implementation toolchain.
2) Create abstract timing models for the logic blocks. Remove combinational timing arcs from the models. Sequential startpoints/endpoints are preserved.
3) Run block-level STA on the routing blocks.
4) Create abstract timing models for the routing blocks.
5) Disable timing arcs that correspond to cycle-breaking connections identified by a logical ranking. Run chip-level STA and measure the propagation delay of the remaining connections. These numbers are saved for the router in the FPGA implementation toolchain.
6) Repeat Step. 5 with a different logical ranking until all connections are analyzed.

Note that type (iii) cycles defined in Sec. III-A are eliminated in this STA because combinational paths through CLBs are analyzed in Step. 1 and removed in the following steps.

B. *Flat Multi-Block Design Flow for Cycle-Free FPGAs*

Our proposed approach enables the design of **Cycle-Free FPGAs**, that is, FPGAs in which all switch blocks are cycle-free. Multi-block sub-arrays of such an FPGA (if not the full chip, considering EDA runtime) may be placed and routed flat, which often leads to better QoR and reduces human labor involved in floorplanning and constraining the blocks. The following summarizes this design methodology:

1) Lay out the logic blocks individually. Create abstract timing models without combinational timing arcs.
2) Divide the FPGA into multi-block sub-arrays, e.g. 4×4 CLBs and the routing blocks around them. Lay out the sub-arrays using the abstract timing models for the CLBs. Allow the EDA tools to ungroup the routing blocks.
3) Assemble the sub-arrays at the chip level and run STA.

V. Routability Evaluation

In this section, we evaluate the impact of using cycle-free switch blocks on the routability of FPGA routing graphs.

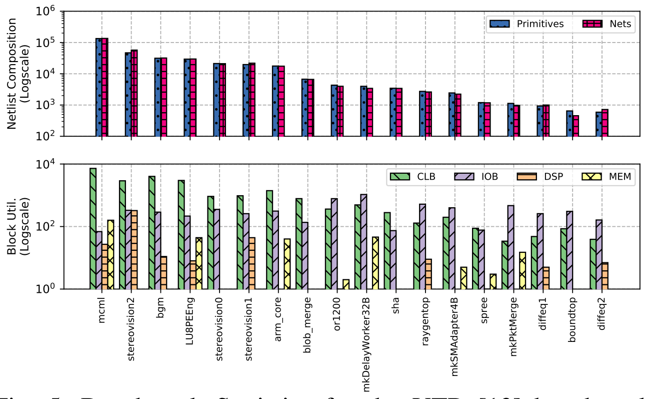


Fig. 5: Benchmark Statistics for the VTR [13] benchmark suite. The figure on the top shows the number of primitives (LUTs and flipflops) and nets used by each benchmark after synthesis. The figure on the bottom shows the utilization of blocks per benchmark after packing.

Parameter	Value	Note
N	8	#LUTs per logic block
K	6	LUT size. Support dual-LUT5/arithmetic modes
L	1, 4	Track lengths. Each type takes 50% channel width
F_s	3	Switch block connectivity
$F_{c,in}$	0.4	Connection block input connectivity
$F_{c,out}$	0.2	Connection block output connectivity
Hard Logic	-	DSP (Fracturable Multiplier) and BRAM

TABLE I: VTR architecture parameters

A. *Benchmarks*

We use the benchmarks shipped with the *Verilog-to-Routing* (VTR) [13] project for our evaluation. Fig. 5 shows the statistics of the VTR benchmarks, including the number of primitives and nets used by each design, as well as the utilization of different blocks. This benchmark suite covers designs of various scale, connectivity, and resource requirements, so we believe the results presented in this section are representative of a wide range of applications.

B. *Methodology*

We first use the *Princeton Reconfigurable Gate Array* [16] (PRGA) framework to generate a scalable architecture specification with the parameters shown in Table I. We run the VTR flow with the generated specification to implement each benchmark individually and determine the minimally sufficient FPGA size and routing channel width. We then relax the channel width by 30% and use the PRGA framework to generate routing resource graph specifications for three types of switch blocks (SB): 1) universal SB [17] (baseline), 2) vanilla Wilton SB [11], 3) cycle-free Wilton SB (this work). We rerun the routing step of the VTR flow with these specifications, reusing the same packing and placement results to eliminate other factors affecting the routing quality. We then compare the critical path delay.

C. *Evaluation Results*

Fig. 6 shows the evaluation results. Compared to the universal switch block, the vanilla Wilton switch block and its cycle-free variation reduce the critical path delay by 9.3% and 10.0% (geometric mean), respectively. Moreover, among the 7 large benchmarks that require more than 10K primitives and nets, the cycle-free Wilton switch block achieves better critical path delay in 5 benchmarks. In summary, the Wilton switch block's cycle-free variation has no negative impact on the routability of the routing graph.

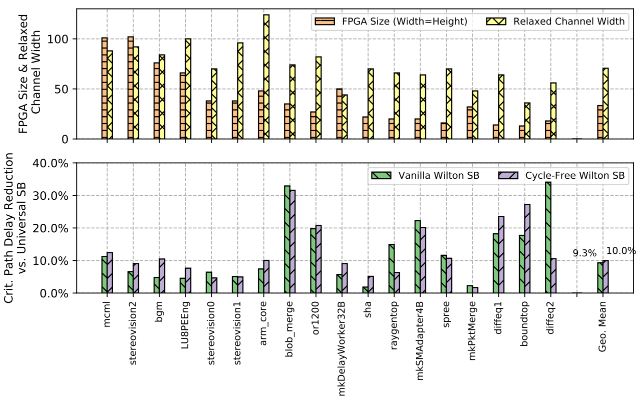
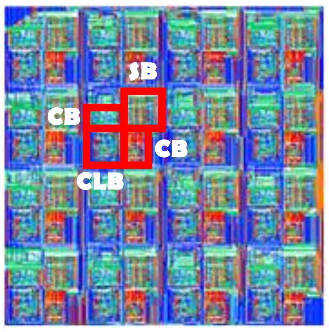
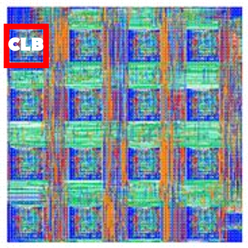


Fig. 6: Routing Results. The figure on the top shows the width, height, and channel width of the FPGA used to route each benchmark. The figure on the bottom shows the critical path delay reduction of the vanilla and cycle-free Wilton switch box compared to the universal switch box.



(a) A 4x4 sub-array of a conventional FPGA designed with the bottom-up design flow proposed by prior works



(b) A 4x4 sub-array of a cycle-free FPGA designed with the flat design flow

Fig. 7: Layout Photos

VI. Design Flow Evaluation

In this section, we evaluate the bottom-up design flow proposed by prior works and our proposed flat design flow enabled by cycle-free FPGAs.

A. Methodology

We first use the PRGA [16] framework to generate synthesizable RTL for two 4x4 sub-arrays from two FPGAs: a) a conventional FPGA equipped with vanilla Wilton switch blocks, b) a cycle-free FPGA equipped with cycle-free Wilton switch blocks. Other architectural parameters of the FPGAs are shown in Table. I, and the channel width is 200 (36 L1 tracks and 16 L4 tracks in each direction). Design (a) is laid out with the bottom-up design flow proposed by prior works, as summarized in Sec. IV-A, and design (b) is laid out with the flat design flow proposed in Sec. IV-B. Note that the CLB structure is the same for the two FPGAs, and both flows need the layout and abstract timing model of the CLB, therefore we only lay out the CLB once and use it in both designs. Fig. 7 shows the layout photos of the two sub-arrays.

B. Design Effort

One important saving of the flat design flow is the human labor. To improve the QoR of the conventional FPGA, we have to try different shapes and relative positions of the blocks. In each iteration, we manually place the pins of each block and align them at the top level in order to reduce

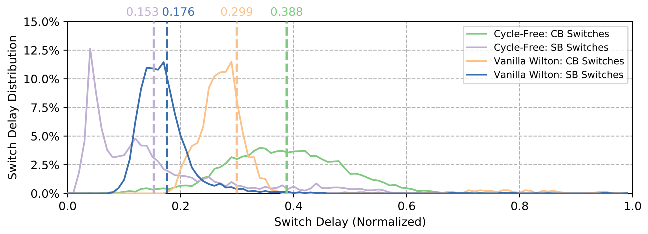


Fig. 8: Programmable switch delay distribution. Delay values are normalized to maximum delay. Average delays are marked with vertical dashed line. **CB** (connection block) switches connect block pins and routing tracks. **SB** (switch block) switches connect one track to another.

congestion. In comparison, the design flow is fully automated for the cycle-free FPGA except for CLB placement.

C. Area

The core utilization rate is set to 0.6 for all the blocks and the cycle-free sub-array. The post-layout area of design (b) (cycle-free) is 43.8% less than that of design (a) (conventional). However, design (a) can be smaller if the blocks are floorplanned in non-rectangular shapes, carefully abutted, and routed at the top level (which is difficult for advanced technology nodes due to the complex design rules). In the extreme case, if we remove all the routing/buffering space added at the top level, we can reduce the total area of design (a) by 37.2%. Even so, design (b) is still 10.5% smaller.

D. Performance

The key difference in performance is the propagation delay across the programmable switches. Fig. 8 shows the distribution of the switch delays in the two designs. On average, the SB switch delay of design (b) (cycle-free) is 13.1% lower than design (a), but the CB switch delay is 29.8% higher. As elaborated in Sec. IV-B, we intentionally remove all combinational paths through CLBs to eliminate type (iii) cycles defined in Sec. III-A. As a consequence, any timing path optimized by EDA tools may include many SB switches, but cannot include more than two CB switches: 1) A CLB output switch if the timing path starts from a register output in a CLB; 2) A CLB input switch if the timing path ends at a register input in a CLB. A possible improvement is to wrap the CLB and all the connection blocks around it into one physical block, then optimize the CB switches within the physical block. This possibility is left for future work.

VII. Conclusions

In this paper, we proposed a generic approach to construct cycle-free FPGA routing graphs. The proposed approach enables automated, accurate, per-switch, post-layout STA for conventional FPGAs. Moreover, it opens up the possibility of designing intrinsically cycle-free FPGAs that facilitate a flat multi-block or full-chip design flow. We quantitatively showed that using cycle-free switch blocks has no negative impact on the routability of the FPGA routing graphs. Furthermore, we compared the area and performance of a conventional FPGA built with the design flow proposed by prior works and a cycle-free FPGA built with our proposed flat multi-block design flow. Experiments show that our proposed flat design flow may save 10.5-43.8% area, 13.1% switch delay in switch blocks, but increase the switch delay in connection blocks by 29.8%.

References

[1] S. M. S. Trimberger, “Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology,” *IEEE Solid-State Circuits Magazine*, vol. 10, no. 2, pp. 16–29, 2018.

[2] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[3] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, “Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?” *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, pp. 225–236, 2010.

[4] M. Abramovici, C. Stroud, and M. Emmert, “Using Embedded FPGAs for SoC Yield Improvement,” in *Proceedings of the 39th Annual Design Automation Conference*, ser. DAC ’02. New York, NY, USA: Association for Computing Machinery, 2002, p. 713–724.

[5] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, “Going Deeper with Embedded FPGA Platform for Convolutional Neural Network,” pp. 26–35.

[6] Jin Hee Kim and J. H. Anderson, “Synthesizable FPGA Fabrics Targetable by the Verilog-to-Routing (VTR) CAD Flow,” in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2015, pp. 1–8.

[7] I. Kuon, A. Egier, and J. Rose, “Design, Layout and Verification of an FPGA Using Automated Tools,” in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 215–226.

[8] X. Tang, E. Giacomin, G. D. Micheli, and P. Gaillardon, “FPGA-SPICE: A Simulation-Based Architecture Evaluation Framework for FPGAs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 637–650, March 2019.

[9] P. Mohan, O. Atli, O. O. Kibar, and K. Mai, “A Top-Down Design Methodology for Synthesizing FPGA Fabrics Using Standard ASIC Flow,” in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 313.

[10] V. Aken Ova and R. Saleh, “A “Soft++” EFPGA Physical Design Approach with Case Studies in 180nm and 90nm,” in *Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, ser. ISVLSI ’06. USA: IEEE Computer Society, 2006, p. 103.

[11] S. J. E. Wilton, “Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory,” Ph.D. dissertation, Toronto, Ont., Canada, Canada, 1997, aAINQ28082.

[12] C. J. Glass and L. M. Ni, “The Turn Model for Adaptive Routing,” in *[1992] Proceedings the 19th Annual International Symposium on Computer Architecture*, May 1992, pp. 278–287.

[13] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. 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, Jul. 2014.

[14] B. Neumann, T. von Sydow, H. Blume, and T. G. Noll, “Design Flow for Embedded FPGAs Based on a Flexible Architecture Template,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 56–61.

[15] H. J. Liu, “Archipelago - An Open Source FPGA with Toolflow Support,” Master’s thesis, EECS Department, University of California, Berkeley, May 2014.

[16] A. Li and D. Wentzlaff, “PRGA: An Open-Source Framework for Building and Using Custom FPGAs,” in *The First Workshop on Open-Source Design Automation; Florence, Italy*, 2019, pp. 1–6.

[17] Y.-w. Chang and D. F. Wong, “Universal Switch Modules for FPGA Design Chinese University of Hong Kong,” vol. 1, no. 1, pp. 80–101, 1996.