Toward Accurate Timing Analysis for Transistor-Level Programmable Fabrics

Qiongdan Huang, Jingxiang Tian, Thomas Broadfoot, Xiangyu Xu, Bo Hu,
Mustafa Shihab, Apurva Jane, Vibhav Salimath, Yiorgos Makris and Carl Sechen
Electrical and Computer Engineering Department, The University of Texas at Dallas, Richardson, TX 75080, USA
E-mail: {qxh150430, jxt122130, tjb043000, xxx110230, bo.hu,
mms091120, axj173930, vks160030, gxm112130, and cms057000}@utdallas.edu

2022 IEEE 15th Dallas Circuit And System Conference (DCAS) | 978-1-6654-9885-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/DCAS53974.2022.9845530

***Abstract*—Transistor-level programmable fabrics [3] [4] have received interest recently as more compact embedded field-programmable gate arrays (eFPGAs) for hardware obfuscation, in which a crucial part of the design is implemented in the eFPGA and the rest of the design is implemented as an ASIC. However, conventional static timing analysis (STA) tools were developed either for ASICs or look-up table (LUT)-based FPGAs. Transistor-level programmable fabrics have pass transistors, keepers, and repeaters in the interconnect. Conventional STA tools require the interconnect parasitics to be expressed in SPEF (Standard Parasitic Exchange Format), consisting only of wiring parasitics. Furthermore, logic gates for transistor-level programmable fabrics may have a separate input to the pMOS pull-up network and the nMOS pull-down network. Dual inputs in this manner cannot be handled accurately enough since conventional methods have to take the longest delay among the inputs, which often overestimates the downstream delay. We propose an instance-based characterization solution which enables the use of an STA tool (PrimeTime from Synopsys) for static timing analysis of transistor-level programmable fabrics. We individually characterize each cell (logic gate) instance in the transistor-level programmable fabric, from predecessor cell-instance output to characterized cell-instance output, including all parasitics, pass transistors, repeaters and keepers. Experimental results corroborate that the proposed instance-based characterization method yields very accurate STA for transistor-level programmable fabrics.**

***Index Terms*—STA, eFPGA, TRAP**

I. Introduction

Reconfigurable computing has contributed to the eminent progression in a wide scope of utilizations – from high performance computing to secure systems. Despite the fact that LUT-based FPGAs have been the most popular reconfigurable computing platform, there are a few other alternative reconfigurable platforms based on fine-granularity programmability [1], [2]. Specifically, the TRAnsistor-level Programmable fabric (TRAP) [3], has received interest recently as it illustrates uniquely compelling features, such as: (1) A more compact embedded field-programmable gate array (eFPGA) for hardware obfuscation [4], in which a small but crucial part of the design is implemented on an on-chip eFPGA and the rest is implemented by a conventional application-specific integrated circuit (ASIC); (2) A better logic density and lower area overhead compared to conventional eFPGAs for the same technology. However, accurate timing analysis of transistor-level programmable fabrics has been a challenge.

Timing verification has always played an important role in digital circuit verification [5]–[7]. Among the various methods proposed in the literature for examining the timing of digital circuits, one of the most widely used timing verification methods, is staitic timing analysis (STA). Conventional STA tools are predominantly designed for ASICs. Furthermore, STA tools for FPGAs are proprietary, vendor-specific, and tailored to certain architectures.

The unique architecture of TRAP having pass transistors, half-keepers, and bi-directional repeaters in the interconnect network between logic gates [3] makes it difficult to apply conventional STA on it. Such pass transistors, half-keepers and bi-directional repeaters, when embedded in the parasitics, cannot be handled by existing STA approaches. Moreover, educated attempts at replacing them with an equivalent resistance results in gross delay estimation error. In addition, certain inputs of some of the logic gates in TRAP are dual inputs (i.e., separate connections to the pMOS and nMOS networks), implying that a driver must fan out and be connected to both of these inputs. Taking only the worst-case of the two delays to each of these dual inputs (as STA does) and applying it to both of them overestimates the delay. On top of that, an attempt to apply a transistor-level timing analysis tool (e.g., Synopsys' NanoTime), was not successful. NanoTime, which utilizes a topological algorithm to identify standard cells, fails to identify cells in TRAP since the pull-up and pull-down networks in TRAP are not simple duals of one another, although they are obviously logic duals of one another. Also, NanoTime documentation states that it doesn't handle circuits well "that use the source or drain terminals of pass transistors as inputs" [8] as it ignores parasitics on the side path [6]. Parasitics on the side path shown on Fig. 2 also contribute to delay. Additionally, TRAP is designed to seamlessly integrate with ASICs. This means that we need to use the same STA tool for both TRAP and the much larger ASIC. Consequently, we can infer that there exists no straightforward mechanism to applying existing STA tools to perform STA on TRAP.

To address these limitations, herein we propose a novel instance-based characterization (IBC) approach which characterizes cell instances from predecessor cell instance output to characterized cell-instance output, rather than from characterized cell input to output (for various loads), as done conventionally. Upon completion of cell instance characterization, a conventional STA tool (e.g., PrimeTime) can be used since the parasitics, pass transistors, half-keepers, etc., have already been included in the instance characterization step.

II. Transistor-Level Programmable Fabrics

TRAP is a CMOS computational fabric consisting of carefully arranged regular rows and columns of transistors which

978-1-6654-9885-2/22/$31.00 ©2022 IEEE

can be individually configured and appropriately interconnected in order to achieve post-fabrication implementation of target digital circuit [3], [4]. Fig. 1. depicts its structure. The transistor array and the interconnect are the most fundamental elements. Each row in TRAP comprises columns of transistors, and in particular, each column contains 3 pMOS and 5 nMOS transistors. In Fig. 1, the 3 pMOS devices in the first column happen to be colored blue, with two of them (P3, P1) being vertically oriented (from source to drain) and the topmost of which is connected to the power supply. The other pMOS device (P2) is horizontally oriented in the figure, and connects the center of the two vertically oriented pMOS transistors in this column to a symmetrical point in the next column. There are also two vertically oriented (drain to source) nMOS devices (N1, N3) in which the bottom-most one is connected to ground. Likewise a third nMOS (N2) that is horizontally oriented is attached between that pair of nMOS devices and a symmetrical pair in the next column. Two additional nMOS devices are connected at the center of the column, one (horizontally oriented NH) connecting to the center of the adjacent column, and one (vertically oriented NV) optionally providing a cell output at that column.

Except for the last two nMOS devices, which can only be programmed 'on' or 'off', each of the other 6 transistors in a column can receive a logic signal, or can be programmed to be 'on', or 'off'. In this manner, so-called standard cells of various widths and the same height (one row height) can be programmed. Note that the interface controls those transistors to either receive a logical signal, or be programmed. In Fig. 1., the implementation of a NAND3 gate is illustrated via the blue transistors with inputs A, B, and C, which either receive an input or are programmed to be always 'on'. Meanwhile, the black transistors are programmed 'off'. It is apparent that three columns are needed to implement a NAND3 gate in TRAP. Although D flip-flops (DFFs) can be implemented in the array, custom DFFs are also optionally available every three columns and, if used in a particular column, preclude the use of that column for logic cells.

An abstract of the interconnect array in TRAP is shown in Fig. 1. It consists of segments of vertical and horizontal metal layers overlaying the logic transistor array. nMOS transistor switches optionally interconnect two wire segments on different metal layers. Every 3 columns the horizontal metal tracks are segmented via pass transistors, and every 9 columns the pass transistors are replaced with bi-directional repeaters. Repeaters in TRAP are used to boost signals in order to present voltage drop after passing various pass transistors. Between each row (vertically) a bi-directional repeater connects a vertical metal segment in one row with the corresponding metal segment in the adjoining row.

III. Attempting Cell-based Characterization

We first considered an obvious potential solution to STA for TRAP, which is to attempt a cell-based characterization approach in which the pass transistors in the interconnect are replaced with an equivalent resistance of some sort. Fig.

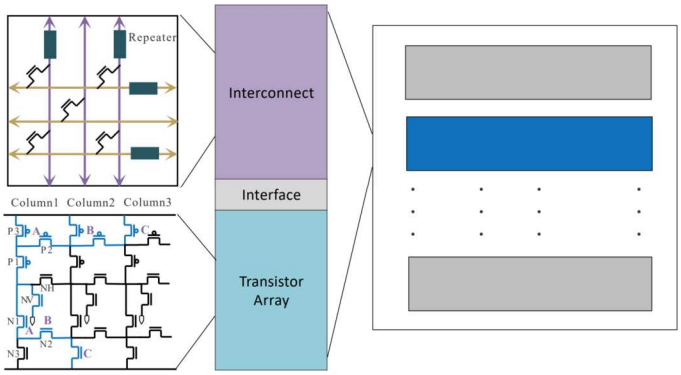

Fig. 1: Hierarchical organization of the TRAP fabric.

2 illustrates a NAND2 gate driving another NAND2 gate (whose inputs are each split into two, one going to the pMOS pull-up network and the other going to the nMOS network. The extracted parasitics between the two gates, for horizontal and vertical metal layers, are shown. In addition, turned-on pass transistors, with small red Vdd symbols on their gates, are shown, as are six turned-off pass transistors (which only contribute parasitic capacitance).

In this example the load NAND2 gate has separate connection paths to the nMOS and pMOS transistor networks for each of its inputs. We chose only the longer path among these two paths and applied that delay to both the nMOS and the pMOS parts of the load NAND2 gate inputs. We then performed parasitic extraction and obtained the SPICE circuit model of that signal path. Next, we replaced each pass transistor with a resistor R, the value of which was varied until the delay from the driver NAND2 gate input to the load NAND2 gate input matched the delay of the original extracted path, which included the actual pass transistors. We also simulated a variety of other such extracted parasitic paths with pass transistors and observed that, over all such cases, a value of R roughly in the neighborhood of 1000 Ω was the best choice. Therefore, for all circuits, we replaced the pass transistors with a resistor of magnitude 1000 Ω. Fig. 2 shows the parasitic network including pass transistors, while Fig. 3 shows the parasitic network after the turned-on pass transistors have been replaced by resistors. Note that the parasitics orthogonal to the obvious signal direction are due to connected wire segments (metal lines) which are not connected to any other instances, but which nonetheless must be charged/discharged.

After replacing the pass transistors with resistors, we generated the SPEF for each net in the netlist. In all cases, when a driving gate or repeater drives the nMOS and pMOS networks of a load gate input through divergent paths, we chose the longer of the two paths and discard the other when generating the SPEF. For TRAP it is necessary to also include the parasitics of all pass transistors which are programmed off when generating the SPEF.

After using the above process for extracting the SPEF, we characterized all gates that can be implemented on the transistor-level programmable fabric using commercial library characterization CAD tools (e.g., Synopsys' Silicon Smart).

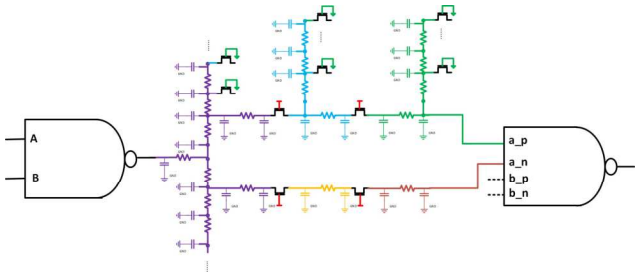

Fig. 2: Parasitic network for NAND2 driving a NAND2.

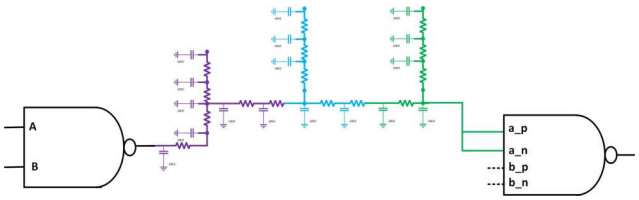

Fig. 3: Parasitic with 'on' pass transistors replaced by resistors.

Algorithm 1 TRAP library-based cell characterization

1: *Characterize* all cells and repeaters in the target library
   /* after programming of the fabric do the following */
2: **for** any gate with dual inputs **do**
       Retain only the path with longest estimated delay
3: **end for**
4: *Replace* each pass transistor with a 1000 Ω resistor
5: *Develop* SPEF for each net in the network
   /* include parasitics of off transistors on wire segments */
6: *Run* STA (i.e., PrimeTime) to obtain worst-case delay

This, in turn, enables the use of conventional STA tools (e.g., PrimeTime) to obtain the worst case delay of a circuit implemented using these gates. The entire procedure is summarized in Algorithm 1. Results evaluating its effectiveness will be provided in Section V, revealing that despite enabling the use of conventional STA, library-based characterization does not nearly reach the required levels of STA accuracy.

IV. Instance-based Cell Characterization

In an effort to obtain much more accurate STA results than what cell-based characterization offers, we propose an instance-based characterization solution. In this approach, each instance of a library cell used in a design is characterized individually, along with all its extracted parasitics. The characterization includes any repeaters and pass transistors associated with the fanout wiring of each cell instance. Thereby, once characterization is completed, conventional STA (e.g., using PrimeTime) can be run accurately without using any SPEF, since literally all of the parasitics have already been accounted for in the characterization step.

A key novel attribute of our instance-based cell characterization is that it takes place from the output of the predecessor gate to the output of the instance being characterized. The advantage of this approach is that it can handle seamlessly the previously discussed issue occurring when a gate input has separate paths connecting to the nMOS and pMOS networks. Indeed, in our scheme these dual-paths are part of the instance

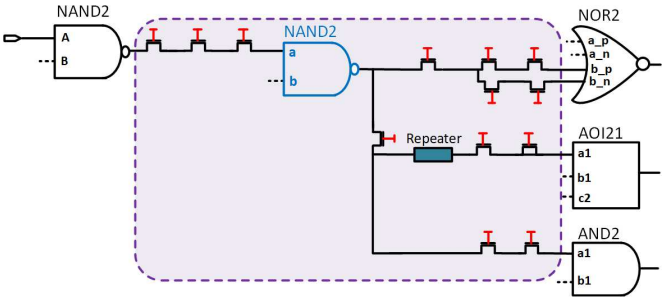

Fig. 4: Characterization circuit (purple box) for NAND2 instance (in blue) with multiple fanouts.

being characterized by SiliconSmart and, therefore, do not need to be dealt with by PrimeTime.

An important issue with this new characterization approach, which proceeds from predecessor output to characterized instance output, is the load faced by the instance being characterized. If no explicit load is made present, as would be done for library-based characterization, then SiliconSmart would have to be run for a range of load capacitances, and in most cases would not be accurate due to the resistive nature of the actual load. We therefore attach the actual R and C network load that the characterized instance drives, up to (and including the input capacitances) of the fanout instances being driven by the instance being characterized. In this way, SiliconSmart is presented with the actual load the instance being characterized sees. Off-path parasitics (e.g., Fig. 2) for the driver of the instance being characterized are included in the characterization of an instance. As an example of this approach, Fig. 4 shows the basic circuitry (inside the purple box), minus the R and C parasitics of the interconnect, used to characterize the NAND2 gate (in blue) toward the top center of the purple box.

The instance-based characterization approach is summarized in Algorithm 2 and detailed in the following subsections.

A. SPICE netlist and parasitic extraction for one block

TRAP features arrays of largely identical blocks. Therefore, it is sufficient to carry out a detailed SPICE netlist and parasitic extraction for one such block and use that extracted block as the reference when searching for the SPICE netlist and the parasitics for circuitry in any other block. For the purpose of timing analysis, only the SPICE netlist and the parasitics associated with the interconnect and devices that impact signal delay are retained. The SPICE netlist and PEX/PXI (Parasitic EXtraction and Parasitic eXtraction Interconnect) files are generated from the layout through an extraction program (i.e., Mentor's Calibre). Here, in Line 1 of Algorithm 2, we only generate the SPICE netlist and the PEX/PXI files for one block of TRAP, and then use it for all other blocks. Parasitics are extracted from the PEX and PXI files.

B. Programming bits used to configure the SPICE netlist

The same programming bits that are used to configure the transistor-level programmable fabric are used to configure the SPICE netlist. Each programming bit has a direct link to a programmable transistor. Lines running from an SRAM memory

Algorithm 2 Instance-based Cell Characterization and STA

Inputs: *Programming Bits*; *Structural Verilog*
1: *extract* SPICE netlist & parasitics for each block
/* Described in Section IV-A */
2: **for** each block **do**
3: *insert* SPICE netlist and parasitics
4: *configure* SPICE netlist using programming bits
/* Described in Section IV-B */
5: **end for**
6: **for** each block **do**
7: **for** each net in block **do**
8: Cross-reference with block's PEX and PXI files
9: *add* corresponding PEX and PXI entries
10: **end for**
11: **end for**
/* Described in Section IV-C */
12: *find* dual-graph of the SPICE netlist (Lines 1–4)
/* Described in Section IV-D */
13: **for** each instance in levelized order **do**
14: **for** each driver gate of the instance **do**
15: *find* all components, nets from driver to instance
/* using DFS and dual graph: Section IV-E */
16: **end for**
17: **for** each fanout gate of the instance **do**
18: *find* all components, nets from instance to fanout
/* using DFS and dual graph: Section IV-E */
19: **end for**
20: *create* SPICE netlist for characterization of instance
21: **for** each net in SPICE netlist **do**
22: *append* parasitics for net (Lines 5-10) to PEX/PXI files for instance
/* Described in Section IV-F */
23: **end for**
24: **end for**
25: *merge* instance characterization files into instance group files, if parallel processing will be used, where each group is for a particular logic level. Otherwise, without multithreads, each group contains one instance.
/* Described in Section IV-G */
26: *launch* SiliconSmart characterization; Process the instance groups in logic level order.
/* Described in Section IV-G */
27: *run* PrimeTime to obtain worst-case delay
/* Described in Section IV-H */

cell to a particular transistor are traced to that programmable transistor and the gate of that transistor is assigned VDD or GND in accordance with the associated programming bit value stored in that memory cell.

For each block in the transistor-level programmable fabric, we first insert the SPICE netlist obtained at Line 1 of Algorithm 2. Then we apply the programming bits for that block to configure the corresponding SPICE netlist. This step comprises Lines 2-5 in Algorithm 2.

C. PEX and PXI file creation

To enable characterization of the instances in a design, we need the configured SPICE netlist obtained from Lines 2-5 of Algorithm 2 along with the PEX and PXI files for the entire design. To create these files, we use as our source the PEX and PXI files for one block of the transistor-level programmable fabric, which were obtained as shown in Line 1 of Algorithm 2. Since each block in the fabric has identical parasitics, for each block and for each net in that block in turn, we cross-reference that net to the PEX and PXI files of the source block. Then, PEX and PXI entries are added for this net. Thereby, after all blocks have been so examined, PEX and PXI files for the entire design are created (Lines 6-11 of Algorithm 2).

D. Graph of active elements associated with an instance

The programmable interconnect consists of arrays of switches (pass transistors) and repeaters. The turned-on switches and bi-directional repeaters are separately termed as *active elements* while the turned-off switches and repeaters are termed *non-active elements*. Only the active elements indicate how signals travel in the design. If we find a bi-directional repeater is associated with a certain instance, we look-up the transistor configuration for the repeater in the SPICE netlist obtained at Line 1 of Algorithm 2.

Fig. 5 shows an example of a 3-gate circuit implemented on a transistor-level programmable fabric. Fig. 6 is the dual graph of Fig. 5, which is used to facilitate the search for the active elements associated with each instance (Line 12 of Algorithm 2). The example in Fig. 5 has one NAND2 instance, one NOR2 instance, one OAI21 instance, and 11 nets. Vertices represent nets and edges represent active elements (i.e., 'on' switches, repeaters and gate instances). Between the 'NET2' vertex and the 'NET3' vertex, transistor MMN1 is the edge connecting them. Repeater X0, as well as instances such as X1, X2, and X3 are also represented by edges.

E. Determining the SPICE netlist of each instance

As described earlier through the example of Fig. 4, an instance to be characterized starts from the output of its predecessor, goes through certain pass transistors and gates (e.g., repeaters), and ends with its fanout loads as its output. Fig. 7 shows an example of the process of determining the SPICE netlist for the NAND2 gate instance X1. The example in Fig. 7 is derived from the example shown in Fig. 5. The dual graph from Fig. 6 is used to trace the components associated with this instance. Starting and ending points, which are obtained by finding the inputs and output of the instance in the structural Verilog netlist, are provided to the graph. Depth-first search (DFS) is used, starting from IN1 (NET1) and proceeding to OUT1 (NET5). The search reveals the pass transistors and gates (e.g., repeaters) between these two points. The first search area is marked as the purple box in Fig. 7. The second step of the search will then start from the output of the target instance and end at the loads of the target instance. The load gates driven by a particular instance are readily available in the structural Verilog netlist. In this case, the search starts at OUT1 (NET5) and ends at NET6, NET7, NET8, and NET9. The second step of the search takes place inside the illustrated green box in Fig. 7. Together, these two search steps, which are described in Lines 13-20 of Algorithm 2, produce a list of active elements relevant to the characterization of the target instance. When extracting instance X1 of Fig. 7, the 'off' pass transistors attached to NET3, NET4, NET6, NET7, NET8, and NET9 also need to be extracted from the SPICE netlist generated through the steps described in Section IV-B.

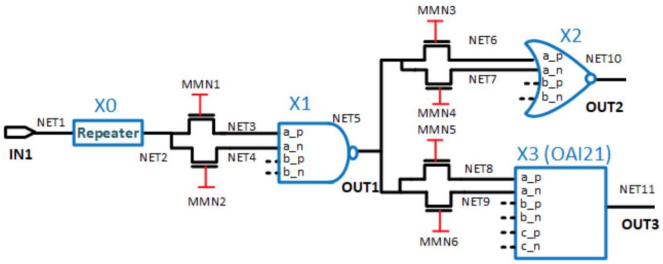

Fig. 5: TRAP circuit example.

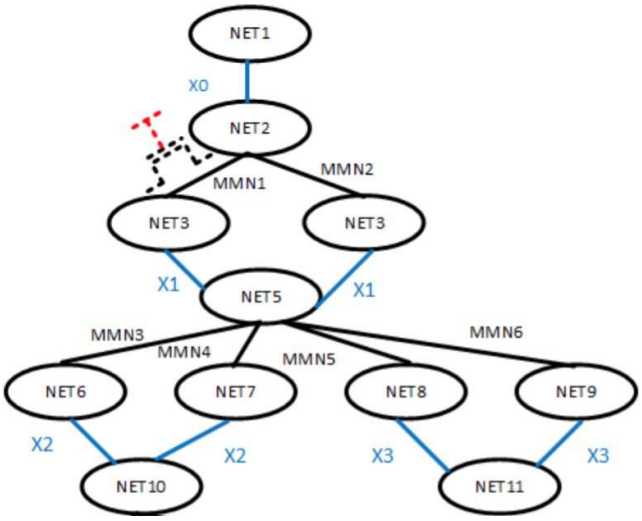

Fig. 6: Dual graph corresponding to Fig. 5

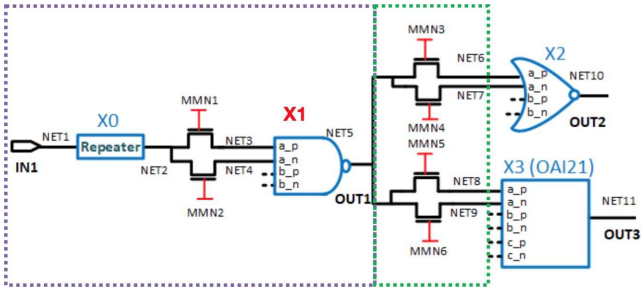

Fig. 7: Example netlist for characterizing instance X1.

F. PEX/PXI file creation for each instance

After a SPICE netlist is obtained for characterization of an instance, the nets are cross-referenced in the PEX and PXI files generated by Lines 6-10 of Algorithm 2 to provide the parasitics for this instance. Instance-specific PEX and PXI files are created in this process (Lines 21-24 of Algorithm 2).

G. Instance characterization and Liberty file creation

The SPICE netlist associated with each instance, along with the corresponding PEX and PXI files, enables SiliconSmart to characterize each instance and generate the Liberty file (.lib) for each instance. During the characterization for each instance, we applied the Pole Analysis via Congruence Transformations (PACT) algorithm to perform Model Order Reduction (MOR) on each instance's parasitics network [9]. This is also known as the SimLA option in Hspice. The PACT algorithm reduces the RC networks in a well-conditioned manner, while preserving network stability. With the MOR approach, each instance can be characterized in about 2 minutes, on average. We note that this is quite similar to the average characterization time for a standard cell in an ASIC library for one corner. We create a separate characterization directory for each instance to facilitate parallel characterization runs, if desired. After characterization is complete, a single Liberty file is created for the design. This process is described in Lines 25-26 of Algorithm 2.

H. Static timing analysis

The last step of our instance-based cell characterization approach, which is shown in Line 27 of Algorithm 2, is to invoke PrimeTime. This step uses the structural Verilog netlist and the Liberty file (.lib). We note that PrimeTime no longer needs to use SPEF, since all interconnect parasitics have already been included during instance characterization.

V. Experimental Results

In this section we show results for the set of combinational benchmark circuits shown in Table I for TRAP [3], [4] designed and fabricated using two different technologies, GF 65nm and GF 12nm, respectively. The benchmarks include 6 from the ISCAS-85 set (C432, C880, C1908, C1355, C3540, C6288), a 4-bit fast adder (74283), and a 13X13 multiplier. Benchmark 74283, C432, C880 and 13X13 multiplier are implemented through TRAP in GF 65nm. Benchmark C1908, C1355, C3540, C6288 are implemented through TRAP in GF 12nm. As mentioned earlier, for the intended hardware obfuscation application of the transistor-level programmable fabric, circuit sizes of at most a few thousand instances are anticipated. Note that even for these benchmarks, the number of extracted resistors and capacitors extends well into the millions and Hspice simulation time for one vector can last for hours.

In Table III, we first compare the conventional Library-based Characterization (LBC) STA method (Algorithm 1) with the proposed Instance-based Characterization (IBC) method (Algorithm 2). Hspice is the industry's gold standard for accurate circuit simulation and the first column shows the Hspice simulated result for the critical path identified by PrimeTime for the instance-based characterization method.

We then provide the critical path delay information obtained by the IBC method, as well as the percentile difference from the Hspice results. Lastly, we also provide the same information, as obtained by the LBC method. The IBC method critical path delay results are accurate, as they are within 4% of the Hspice circuit simulation results for 65nm and within 3% for 12nm.

In contrast, the LBC method accuracy is completely unacceptable, with its percentile difference from the Hspice results exceeding 50%. Although replacing the turned ON pass transistors by a suitable resistor value can work to some extent for circuit simulation, quite apparently having such large

TABLE I: Transistor-level Programmable Fabric Benchmarks.

	74283	C432	C880	Multiplier 13x13	C1908	C1355	C3540	C6288
Primary Input Count	9	36	60	26	33	41	50	32
Instance Count	41	212	383	537	162	168	424	1405
Transistor Count	24,584	121,877	187,965	254,468	38,302	38,205	112,983	309,367
Resistor Count	608,016	3,073,909	4,980,687	6,023,876	1,230,174	1,225,462	3,614,381	9,716,311
Capacitor Count	321,196	1,664,333	2,443,169	3,246,169	411,461	409,505	1,209,813	3,280,336
Hspice Run Time (One Vector)	14.9 m	1.5 h	2.0 h	2.7 h	24.6 m	24.0 m	1.5 h	3.2 h

TABLE II: Run time for IBC/LBC + STA fabric.

	74283	C432	C880	Multiplier 13x13	C1908	C1355	C3540	C6288
IBC + STA using 10 threads								
Algorithm 2 (Lines 1–24)	0.82 min	1.9 min	3.3 min	4.0 min	0.8 min	0.7 min	1.7 min	4.6 min
SiliconSmart Characterization	9.8 min	0.8 h	1.4 h	1.8 h	16.9 min	16.7 min	0.8 h	2.2 h
Total Time (Including PrimeTime)	10.6 min	0.8 h	1.4 h	1.8 h	17.7 min	17.4 min	0.8 h	2.2 h

TABLE III: Critical path delay comparison.

	Hspice (ns)	IBC (ns)	Δ	LBC (ns)	Δ
74283	16.0	16.67	4%	6.62	58%
C432	97.1	99.81	2.7%	43.23	55%
C880	87.6	83.4	2%	41.2	53%
Mult. 13x13	98.67	94.46	4.2%	44.23	55%
C1908	12.84	12.59	2.0%	5.78	55%
C1355	13.48	13.26	1.6%	6.34	53%
C3540	18.87	18.40	2.5%	8.30	56%
C6288	44.23	42.91	2.9%	19.46	56%

resistors intermingled with the extracted R's and C's in the SPEF causes significant problems for PrimeTime when it tries to compute the SPEF delay. Hence the LBC method must be ruled out as a viable option for static timing analysis of a transistor-level programmable fabric having pass transistors in the programmable interconnect network.

Table II show the total run time for processing the 8 benchmark circuits, including the total run time spent on the main parts of Algorithm 2 and cell instance characterization. Characterization of the instances (Line 26 in Algorithm 2), was done with k parallel processors (threads) as described earlier. We show results when IBC was performed using k=10 threads (i.e., the number of available SiliconSmart licenses available to us). The Hspice characterization time for one instance averages about two minutes using the PACT algorithm (SimLA option) mentioned earlier.

For the transistor-level programmable fabric, there are 21 logic cells available. The unique aspect of the fabric is that each input of each logic cell has an optional inversion that can be activated/deactivated via programming. To characterize all possible configurations of each cell in the standard library-based manner, the characterization run time is 1.5 hours (for one corner on ten threads). Obviously this characterization would only have to be done once for a given technology update. However, library-based characterization is nowhere close to accurate for this transistor-level programmable fabric. But this does at least show that instance-based characterization is feasible, with the run time comparable to the characterization time of the complete library of logic cells.

VI. Conclusion

Transistor-level programmable fabrics such as TRAP have received interest recently as more compact eFPGAs for hardware obfuscation. However, until now accurate STA for TRAP has been a challenge. We developed an instance-based characterization solution which enables the use of PrimeTime for static timing analysis for TRAP. TRAP has one or more pass transistors in the interconnect for each net. Furthermore, logic gates (or cells) may have a separate input to the pMOS pull-up network and the nMOS pull-down network. To address these limitations, we individually characterize each cell instance in TRAP, from predecessor cell-instance output to characterized cell-instance output, including all parasitics, pass transistors, half-keepers, etc. Experimental results show that the proposed instance-based characterization method yields very accurate STA for TRAP. Having the ability to accurately generate liberty file (.lib) entries for each cell instance in the eFPGA enables accurate STA for a mixed ASIC/eFPGA since the ASIC portion is characterized in the usual cell-based manner. Furthermore, since cell instances can be readily characterized in parallel using multi-core, multi-threaded processors, the computation time is readily feasible for the circuit sizes contemplated for TRAP in the intended application of hardware obfuscation. Our approach could be extended to enable accurate STA for any type of cell-based netlist in which the wiring parasitics either cannot be expressed in SPEF or cannot be accurately expressed in SPEF.

References

[1] F. Yuan, C. C. Wang, T. Yu, and D. Marković, "A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing," *IEEE JSSC*, vol. 50, no. 1, pp. 137–149, 2015.

[2] J. Langeheine, J. Becker, S. Folling, K. Meier, and J. Schemmel, "A cmos fpta chip for intrinsic hardware evolution of analog electronic circuits," in *Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001*, 2001, pp. 172–175.

[3] J. Tian et al., "A field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration," in *IEEE/ACM DATE*, 2017.

[4] M. Shihab et al., "Design obfuscation through selective post-fabrication transistor-level programming," in *IEEE/ACM DATE*, 2019.

[5] R. B. Hitchcock, "Timing verification and the timing analysis program," in *IEEE/ACM DAC*, 1982.

[6] J. Ousterhout, "A switch-level timing verifier for digital mos vlsi," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, no. 3, pp. 336–349, 1985.

[7] A. Devgan and C. Kashyap, "Block-based static timing analysis with uncertainty," in *IEEE/ACM ICCAD*, 2003.

[8] S. Inc, "Nanotime user guide," p. 33, 2016.

[9] K. Kerns and A. Yang, "Stable and efficient reduction of large, multiport rc networks by pole analysis via congruence transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 734–744, 1997.