Using Physical and Functional Comparisons to Assure 3rd-Party IP for Modern FPGAs

Adam Hastings, Sean Jensen, Jeffrey Goeders, Brad Hutchings
Department of Electrical and Computer Engineering
Brigham Young University
Provo, Utah, USA
Email: {adamhastings, seantalbotj, jgoeders, brad_hutchings}@byu.edu

***Abstract*—In modern FPGA design, 3rd-party IP is commonly used to reduce costs and time-to-market. However, the complexity of IP and associated CAD tools makes it easier for attackers to maliciously tamper with the IP (*i.e.* insert Hardware Trojans) in ways that are hard to detect. This work proposes techniques that allows a user to incorporate trusted 3rd-party IP into a design and verify that the incorporation occurs tamper-free. We present comparative results from utilizing this framework across a benchmark suite of 22 designs. We show that the approach reliably detects tampering without giving any false positives.**

I. Introduction

In modern FPGA design, 3rd-party Intellectual Property (IP) is often used to reduce the time and cost of the design process. However, the use of 3rd-party IP is not without risk: the inherent complexity of most 3rd-party IP modules makes it difficult for the user to determine whether or not the IP contains anything malicious, such as Hardware Trojans (HTs).

One step towards securing 3rd-party IP would be to use IP only from trusted 3rd-parties who have verified and can vouch for the safety of the IP. Indeed, this is a necessary first step, but it is not sufficient—3rd-party IP can still be compromised in other ways, such as by malicious attackers or malicious design tools, that can tamper with the IP by altering desired behaviour or by inserting unwanted circuitry. Unfortunately, detecting such tampering of IP is a non-trivial task—even benign FPGA design tools will naturally modify and optimize any IP they are given. It is therefore difficult to determine if any changes made to the final placed and routed IP were benign, or malicious.

This work addresses the problem of detecting malicious tampering in 3rd-party IP by using two assurance techniques that are based upon physical and functional comparisons. These techniques are designed to assure users that the 3rd-party IP used in their design is either physically or functionally equivalent to the original trusted and secured IP.

Both physical and functional assurance are presented together because they generally provide complementary benefits: users may want to apply physical assurance in some situations and functional assurance in others. For example, physical assurance has the advantage that it may be simpler to implement and provides a somewhat stricter definition of assurance than the functional-assurance technique. When using physical assurance, for example, the end-user can be certain that the instantiated 3rd-party IP is placed and routed identically and thus provides similar timing behavior as the originally provided trusted IP. Functional assurance, though more complex, can be applied across a broader set of circuits and is generally more flexible in its application than physical assurance.

A major benefit and contribution of these techniques is that they provide users the assurance that the 3rd-party IP has been included tamper-free, *without* requiring them to know or implement hardware-oriented security measures, such as HT detection and mitigation. Therefore, these techniques can be used by FPGA designers who may not have expertise in the field of hardware security, thus making hardware assurance more accessible.

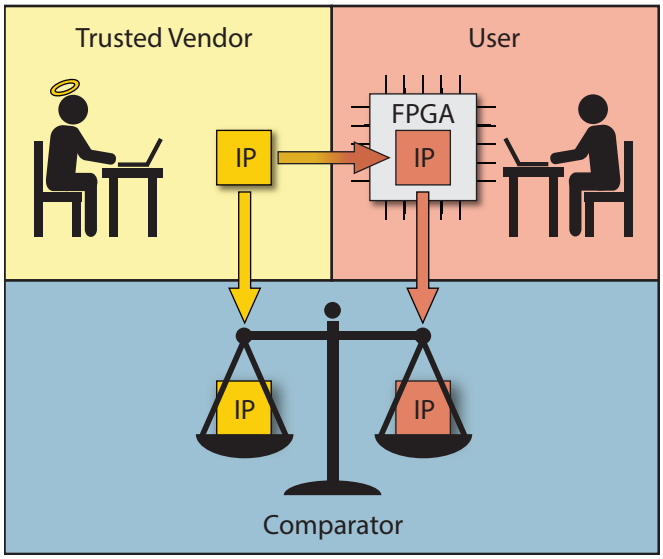


Fig. 1: General Assurance Strategy

II. General Assurance Strategy

Both physical and functional assurance techniques involve three parties, 1) a trusted vendor, 2) an end-user, and 3) a comparator as shown in Figure 1.

Trusted Vendor: The trusted vendor is the IP creator. Although the design tools that the vendor uses may be untrusted, we assume the vendor has sufficiently inspected and tested the IP to determine that it is free from malicious inclusions.

978-1-5386-6544-2/IVSW18/©2018 IEEE

For example, the vendor may employ hardware Trojan (HT) detection and mitigation techniques [1]–[9] to accomplish this goal. The vendor provides the design files to the user for inclusion in their design, which we refer to in the paper as the *trusted IP*. For physical assurance, the trusted IP design files implement a completely placed and routed circuit for a specific FPGA device. For functional assurance, the trusted IP design files consist of a verilog netlist.

User: The user instantiates the trusted IP into their design using the method required by the format of the IP. The user is not required to know HT detection and mitigation techniques, nor understand the details within the IP.

Comparator: The trusted IP files and the user's final placed and routed design are provided to a party that performs the comparison. This comparison process could be carried out by the vendor, user, or another trusted party. This latter option has the benefit that it will detect malicious hardware that may have been directly inserted by the user. The comparison between the trusted IP and the implementation of that IP in the user's design is completely automated. In the following, Section III will explain the physical assurance process and Section IV will explain the functional assurance process.

III. Physical Assurance Process

The physical assurance process ultimately compares the final placed and routed user circuit against the provided 3rd-party IP to determine the integrity of the trusted IP as implemented in the user design. The detailed steps of the process are outlined below and are illustrated in Figure 2a.

A. Trusted IP

1) Trusted third party creates a Vivado project for a specific FPGA part using the Xilinx Vivado Hierarchical Design (HD) flow. Vivado HD contains several different flows centered around the idea of a partitioned design. In Xilinx terminology, a *pblock* refers to a physical partition of the chip, containing a set number of resources.
2) The trusted third party creates a pblock, assigns their IP to this block, and performs synthesis and implementation. It is important to recognize that this will be *out-of-context* synthesis, meaning that the tools do not incorporate anything about the circuit into which the IP will be placed. This prevents the tools from making optimizations that would affect the behavior of the circuit, and that allows the IP to be used by any surrounding circuitry.
3) Once implemented, the IP is fully contained within this pblock partition and can be exported as a Vivado Design Checkpoint (.dcp file). This file is then provided to the user.

A major disadvantage of this approach is that the IP provider must choose the FPGA part and exact pblock location on the chip. This can be mitigated by providing multiple files for different parts, and a few different locations on the chip to give greater flexibility to the user.

B. User Implementation

1) The user creates their full design in HDL and synthesizes it using Vivado HD.
2) The user creates a pblock in their design in the same location as the one chosen by the trusted IP provided.
3) The trusted IP (.dcp file) is assigned to the pblock.
4) The user design is implemented using Xilinx HD, utilizing the IP from the trusted vendor.

C. Physical Comparison

The final phase of the process is the comparison flow. Either the trusted vendor or the user takes the two sets of physical information and compares them, looking for any differences. A discrepancy may mean a number of different things, as treated in Sections VI to IX. A perfect match means that the instantiated and trusted IP are physically (and therefore functionally) identical and indicates that no tampering has occurred.

The extracted physical information used in the comparison process includes everything necessary to fully reconstruct the design on the fabric. In detail, the extracted information includes:

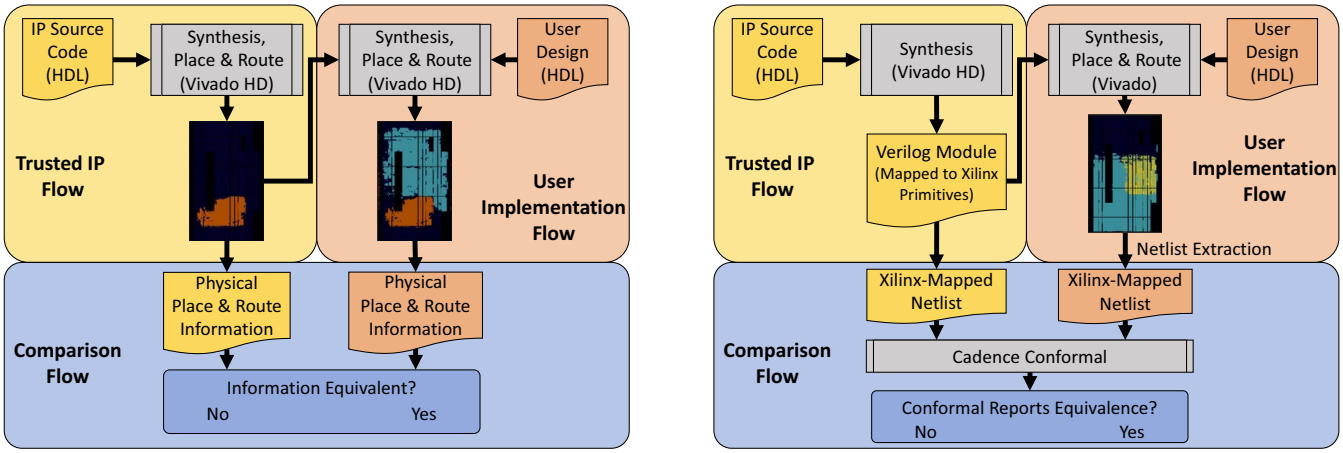
- Sites: a list of all of the physical resources allocated to the pblock
- Cells: the name, bel, configuration, location, and a mapping of net to cellpin
- Boundary nets: the names of the nets connecting the IP to the top-level logic
- Remaining Nets: the name, wires (if applicable), pips (if applicable), and pin count
- Bels: the name, type, and configuration

The goal of physical assurance is to compare the trusted IP and instantiated IP based on physical implementation details. If two designs use the same physical resources in the same way, the two designs are guaranteed to be (barring slight process variations) functionally identical. We have fully automated the extraction and comparison process using TCL scripts developed for this work. We assume the design tools accurately extract the trusted IP circuit elements. In the unlikely event that the reported results were spoofed, reverse-engineering the bitstream could be a work-around, which we discuss in Section IX.

By demonstrating physical equivalence between the trusted and instantiated IPs, it is inferred that the instantiated IP was not tampered with. This equivalence then assures the user that the instantiated IP is safe to use.

IV. Functional Assurance Process

The functional assurance process determines the integrity of the trusted IP in the user's implemented design by comparing a partial netlist extracted from the final user circuit against the original trusted IP netlist. The detailed process is outlined below and is illustrated in Figure 2b.



(a) Physical Assurance Flow

(b) Functional Assurance Flow

Fig. 2: IP Assurance Framework

A. *Trusted IP*

In the trusted IP flow, the trusted vendor creates the IP as a module using Vivado Hierarchical Design (HD). Using the standard out-of-context CAD flow, the IP is synthesized, optimized, and mapped to Xilinx FPGA primitives. Finally, the trusted vendor extracts a Verilog netlist for their IP using the `write_verilog` command, which creates a netlist of Xilinx primitives (LUTs, FFs, etc.) This netlist is then provided to the user.

Because our framework assumes that FPGA design tools are untrusted, it may cause concern that we trust Vivado to faithfully extract a netlist from the IP. Such concerns are not unfounded, but are addressed with possible workarounds in Section IX.

B. *User Implementation*

Using the standard Vivado flow, the user incorporates the netlist into their design. Because the trusted IP is in the form of a Verilog netlist, instantiating the IP is a straightforward and simple task; the user can instantiate the Verilog module in their RTL design in the same manner as any other Verilog module. In addition to instantiating the IP, the user must also apply a few additional constraints to prevent the CAD design tools from making further optimizations to the IP. This step makes it possible for the comparison flow to determine equivalence between the trusted and instantiated IPs. We accomplished this step by applying the `DONT_TOUCH` Xilinx attribute to the following objects:

- The cells within the instantiated IP
- The nets within the instantiated IP
- The hierarchical cell that represents the instantiated IP

For example, a user could apply these attributes to an IP named `aes128_0` by adding the following lines of code to their constraints file:

```
set_property DONT_TOUCH true [get_cells aes128_0/*]
set_property DONT_TOUCH true [get_nets  aes128_0/*]
set_property DONT_TOUCH true [get_cells aes128_0]
```

The `DONT_TOUCH` attribute prevents the Xilinx CAD tools from optimizing or changing anything it is applied to. This forces the CAD tools to place and route the IP netlist without changing its structure, and therefore its behavior. While this initially may seem drastic, it is important to recognize that the trusted IP has already undergone logic optimization in the trusted IP Flow. The `DONT_TOUCH` attribute simply prevents *further* optimizations to the IP, such as cross-boundary optimizations. Furthermore, applying `DONT_TOUCH` to the target IP has little effect on the optimization of the user's surrounding circuitry. The user's surrounding circuit will still be fully optimized, and in fact, can still leverage knowledge of the content of the trusted IP. Therefore, the primary drawback to this approach is that the trusted IP has no knowledge of the surrounding circuit when its logic is optimized. The effects on size and area of this limitation are discussed in section Section VIII.

Once the user has instantiated the trusted IP netlist in their design they can perform implementation (place & route) as usual. After implementation, the user uses the `write_verilog` command to extract a Verilog netlist from the implemented design. Like the trusted IP netlist, this user design netlist contains Xilinx primitives. As in the Trusted IP Flow, we assume that Vivado faithfully retrieves the correct netlist, while acknowledging that a compromised version of Vivado could potentially retrieve a spoofed netlist. As with the trusted IP flow, workarounds to this issue are discussed in Section IX.

C. *Functional Comparison*

The final phase of our implementation is the comparison flow. In this step, the trusted IP netlist and the user's netlist are compared using Conformal, a formal equivalence checker developed by Cadence. Specifically, our process tells Conformal to compare the IP module in the trusted IP netlist to the IP module in the user design netlist. Conformal, like other formal equivalence checkers, can be used to prove that the two

netlists are logically equivalent[1]. As discussed in Section II, by demonstrating functional equivalence, it is then inferred that the instantiated IP has not been tampered with. Likewise, if Conformal determines that the trusted and instantiated IPs are *not* the same, then it becomes clear that somewhere along the way, something in the instantiated IP was changed.

V. General Experiments

Experiments were conducted to answer the following questions for both the physical and functional assurance techniques:

- Can the the assurance approach successfully extract the IP from the user's implemented design, compare it against the original trusted IP, and determine that they are identical?
- Can the assurance approach successfully detect modifications to the trusted IP that may have occurred anywhere in the insertion and implementation process?
- How does this assurance approach impact the timing constraints and area of the final implemented design? Put another way, what does the end user have to "pay", in terms of speed and area, to achieve assurance using this technique?

In total, 53 experiments were conducted across 22 different benchmark circuits. Of the 22, 21 are synthetic and were created by interconnecting various Open-Cores (www.opencores.org) without regard for circuit function. From these 21 synthetic circuits, 43 experiments were conducted. For each experiment, a unique sub-block was extracted from the synthetic circuit to play the role of the trusted IP block. The 22nd benchmark circuit is a functional LEON3 processor (www.gaisler.com/leon3) from which 10 different sub-blocks were extracted. The extracted sub-blocks serving as the trusted IP ranged in size from 10 logic cells to nearly 100,000. The full benchmark suite is publicly available at GitHub (www.github.com/byuccl/ipassurance).

To set a baseline for timing and slice utilization, all benchmarks were implemented using a conventional synthesis, place, and route flow to determine whether timing constraints are met and to measure slice utilization. These baseline results are used to determine how much (if, at all) the pblock-based assurance approach may affect timing and area.

VI. Physical Assurance Results

Every benchmark in our suite was run through the physical assurance flow (Section III) and ultimately resulted in a perfect match between the trusted IP and the instantiated IP. In addition to this we ran experiments on the sensitivity of the assurance flow and collected statistics on its performance impact.

[1]For the purposes of this paper, it is assumed that logical equivalence implies functional equivalence. Although logically-equivalent netlists may differ in treatment of certain don't-care conditions, for example, it is assumed that these differences, if they occur, do not affect the functional behavior of the IP in any meaningful way.

A. Sensitivity

The physical assurance flow is extremely sensitive to tampering. We confirmed this by running sensitivity tests on the entire benchmark suite. Our automated tests performed the following modifications three times for each experiment:

- Randomly select a cell from the instantiating circuit and move it into the trusted IP pblock
- Randomly select a cell from the trusted IP and change its location within the pblock
- Randomly select a LUT or FF from the trusted IP and change its initialization equation/value
- Randomly select a net, change the route it takes from its source to its sink

These tests were chosen to mimic any change that might be made to the circuit, malicious or otherwise. Every modification that was performed was caught by the assurance process.

B. Operating Frequency and Area

Of the 10 experiments run on the LEON3 benchmark, all 10 passed timing in both the baseline and pblock-based designs. Of the 43 experiments created from the synthetic benchmarks, only one pblock-based design (dfadd) failed timing after its baseline design had passed timing. This suggests that in most cases, the pblock-based approach does not affect the CAD tools' ability to meet timing, although this can still happen in some situations.

The effects of the pblock-based assurance approach on slice utilization can be seen in Figure 3. On average, designs that used the physical-assurance method used 4.73% more slices than the baseline design, with a standard deviation of 0.059. This penalty is expected, as when the IP is synthesized out-of-context, it cannot employ any cross boundary optimizations, such as constant propagation or unused logic removal.

VII. Challenges and Limitations of Physical Assurance

Physical assurance, as a research approach, was selected as it provides a strict guarantee that the IP is implemented in the exact same manner as provided by the trusted vendor, down to the individual wiring of each net. Although this introduces some overheads, the prevailing notion was that, if functional assurance became impossible for some reason, physical assurance could serve as a final assurance back-stop that would work in most, if not all, cases.

As an idea, physical assurance is simple enough. However, in practice, due to inherent CAD tool complexity, direct comparison of the physical manifestations of the trusted and instantiated IP often failed, and required work-arounds. Some challenges we encountered were:

Global Resources: Circuits would fail to route if the trusted IP contained a global resource such as a clock buffer, as many global resources cannot be included in pBlock partitions. To avoid this problem, the trusted vendor would be prevented from instantiating global resources in their pBlock, and would instead have to instruct the user to instantiate the resource in their surrounding design.

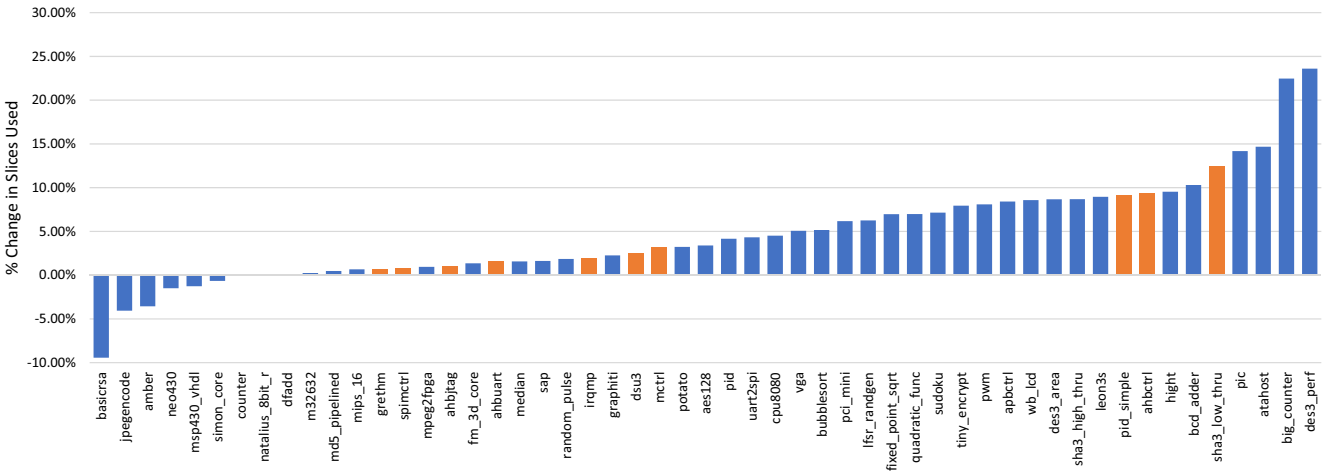


Fig. 3: Impact of physical assurance flow on slice utilization. Synthetic IP benchmarks (blue) and LEON3 IP benchmarks (orange).

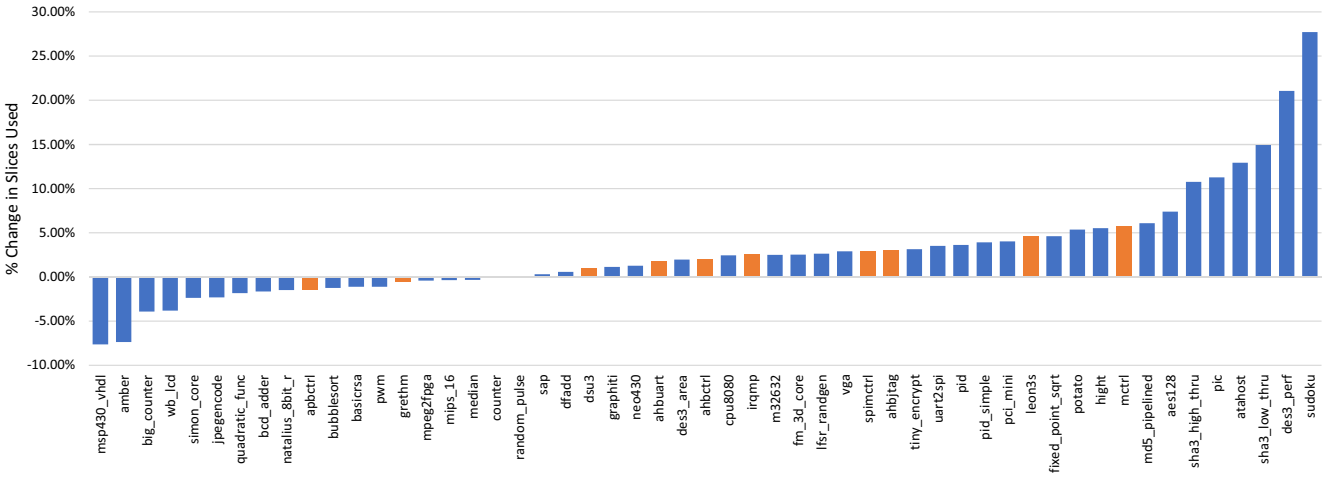


Fig. 4: Impact of functional assurance flow on slice utilization.

Pblock location: In certain cases routing would fail likely because the location of the pblock increased congestion. This problem can be mitigated if the provider of the trusted IP generates several versions of the trusted IP, each with different pblock locations.

Subtle physical optimizations: Despite the fact that pblock partitioning is meant to prevent any changes, in some cases, Vivado would occasionally permute the inputs at the periphery of the trusted-IP pblock. This problem caused most of the benchmarks to initially fail an equivalence test. This problem was overcome by extracting the LUT pin mapping from the original trusted IP, and then locking these pins prior to routing the instantiated IP.

Finally, in one synthetic benchmark containing a pipelined version of the MD5 algorithm (md5_pipelined), Vivado performed a minor optimization (again, near the periphery of the pblock) that occurs because two nets are aliases of one another. After manually inspecting the design, it was determined that the trusted IP and the instantiated IP were indeed functionally equivalent; however, they were not physically equivalent. After some experimentation, we found that this minor physical difference could be eliminated if the placement of the pblock was modified. This is somewhat similar to the problem where pblock placement interfered with placement and routing (though movement of the pblock as a solution is probably not the best solution). This problem only occurred in one synthetic benchmark and other mitigating strategies are being examined.

VIII. Functional Assurance Results

In this section we evaluate our functional assurance flow (Section IV). Using this flow, we were able to determine functional equivalence between all 53 pairs of trusted and instantiated IPs.

A. Sensitivity

Similar to the physical assurance testing, we developed a sensitivity analysis to demonstrate that functional assurance

successfully detects unwanted modifications. To further validate functional assurance, we tampered with the designs in the following ways:

- Pick a random Lookup Table (LUT) in the instantiated IP and modify its logic function.
- Leak a random wire in the instantiated IP to a secretly added backdoor port.

We tampered with each of our instantiated IPs once for each of the above modifications. This gave us a total of 106 tampered designs which we used to test the sensitivity of our approach. Our approach caught all of the malicious modifications that we made. Specifically, we observed that Conformal reported a failure in determining equivalence between the trusted and instantiated IP. This suggests that our approach can be used to detect malicious modifications to a trusted IP.

B. Operating Frequency and Area

Of the 10 experiments run on the LEON3 benchmark, all 10 passed timing in both the baseline and pblock-based designs. Of the 43 experiments created from the synthetic benchmarks, no designs failed timing after its baseline design had passed timing.

The effects of the functional assurance process on slice utilization can be seen in Figure 4. The greatest performance decrease was observed with the `sudoku` design, which was 27.73% larger than its baseline design. Some increase in logic size is expected as our proposed flow prevents some logic optimization to the trusted IP. It also makes sense that this varies from design to design as many optimizations, such as logic minimization based on constant or unconnected inputs, will depend on the surrounding design. On the other end of the spectrum was the `msp430_vhdl` design, which was actually 7.63% smaller than its baseline design. While somewhat unexpected, this is not altogether surprising; the stochastic nature of our CAD tools will naturally produce circuits of varying sizes across different configurations, regardless of our assurance-based approach.

On average, designs that used the functional assurance method used 2.81% more slices than the baseline design, with a standard deviation of 0.06. We believe that this increase in slice count is a very reasonable price to pay for the assurance our approach provides.

IX. Challenges and Limitations of Functional Assurance

Compared to physical assurance, functional assurance provides greater ease and flexibility to the user as they do not not have to physically partition their design, but can rather just instantiate the trusted IP netlist. However, it should be noted that this additional flexibility comes at a price; it only protects against tampering at the logical (*i.e.* RTL, gatelist, or netlist) level of the IP. It does not protect against tampering at the implementation level of the design, *i.e.* placement and routing. For example, a malicious CAD tool or attacker could tamper with the IP by placing its cells unnecessarily far apart from each other, thus slowing down timing and degrading the IP's performance. Similar attacks could be performed on the routing of the IP. The functional assurance framework does not address these kinds of attacks. Note that the previously-discussed physical assurance flow is less succeptible to so-called timing hacks.

One benefit of providing assurance at the functional level is that many of the challenges present in the physical assurance process (discussed previously in Section VII), such as global resources or pblock location, no longer pose an issue. However, we did encounter a number of new challenges that arise when performing functional assurance:

Renaming Rules: We found that when exporting the instantiated IP as Verilog netlists, Vivado would sometimes unexpectedly rename input and output ports. This happened to only a handful of ports on only a few of the 53 instantiated IP blocks. This unexpected renaming caused problems downstream in the assurance process—because Conformal relies on the primary input and output names of the trusted and instantiated IPs to match, such renaming breaks Conformal's ability to determine equivalence. We overcame this challenge by writing a netlist parser that would find where instances of renaming had occurred. The parser would then generate a list of TCL commands which were issued to Conformal to inform it of equivalence between renamed ports. Only after using this work-around could Conformal accurately determine equivalence between the trusted and instantiated IPs.

Unused Outputs: When creating trusted IP, the vendor uses the Vivado Hierarchical Design to create out-of-context modules. Because these out-of-context IP blocks do not belong to any top-level design, its output pins are undriven. However, when incorporating the out-of-context IP into a user design, we observed that Vivado made all output pins be driven by either a signal, or by a constant 1 or 0. This becomes an issue for any unused outputs in the instantiated IP (which, despite being unused, are not optimized away due to the `DONT_TOUCH` attribute discussed in Section IV-B). Vivado handles these unused outputs by tying them to ground, but at the expense of the downstream Conformal equivalence checking. A naïve use of Conformal would show that the outputs tied to 0 are logically different from the undriven outputs in the trusted IP, and that the trusted and instantiated IPs are nonequivalent.

We overcame this challenge by writing scripts that would analyze the instantiated IPs, find the unused outputs, and auto-generate a list of TCL commands that would instruct Conformal to ignore the unused outputs when making the comparison. Only after informing Conformal of the unused outputs could it accurately determine equivalence between the trusted and instantiated IPs.

Untrusted Tools: In our approach, we used potentially untrusted 3rd-party tools to implement our framework. However, it is possible (although more difficult) to successfully implement our framework *without* relying on untrusted 3rd-party tools. This can be achieved by using the following techniques.

1) Verifying Netlist Extraction: In our approach, we assumed that Vivado accurately extracted the correct netlist when

asked to do so. However, the netlists retrieved from Vivado could have been spoofed. In this case, it would be possible to bypass Vivado by reverse-engineering the generated bitstream back into a netlist. While certainly a challenge, this step is by no means impossible, especially if the user uses the trusted IP netlist to create heuristics for reconstructing the instantiated IP's netlist.

2) Verifying Equivalence Checking: In the functional comparison Flow, we assume that Conformal accurately determines functional equivalence without spoofing the result. However, this step can also be accomplished without the use of 3rd-party tools by means of graph matching algorithms. By representing the netlist as a graph, where primitives are vertices and nets are edges, a graph isomorphism algorithm [10] can determine identical structure (and therefore functional equivalence) between netlists. Because primitives and nets have unique names, this process of determining a graph isomorphism between the trusted and instantiated IPs is a computationally feasible problem.

X. Conclusion and Future Work

In conclusion, our physical and functional assurance flows provide hardware engineers assurance that their trusted 3rd-party IP has not been tampered with. The physical assurance flow verifies that the trusted IP is instantiated in the user's design precisely as intended, down to each individual wire placement. However, this flow requires that users provide physical partitioning in their design for the trusted IP. On the other hand, functional assurance provides greater ease of use, allowing for simple instantiation of the trusted IP netlist, while sacrificing some guarantees of the physical implementation. We believe that both flows can be of use, depending on the desired assurances and ease of use.

In this work we demonstrated that 3rd-party IP assurance is not only possible, but that it is also feasible to fully automate the process. This means that designers and engineers can easily determine the integrity of the trusted 3rd-party IP in their designs without needing hardware security expertise. While the two approaches presented in the paper require moderately more logic resources (4.7% for functional and 2.8% for physical assurance, on average), we believe this slight overhead is worth the assurances that are provided to the designer.

Future work will focus on improving the flexibility of our approach as well as improving the cost in terms of speed and area. Specifically, we intend to remove the Xilinx `DONT_TOUCH` attribute, and see if it's still possible to determine equivalence between the trusted IP and an untampered instantiated IP. Early results suggest that this is possible, but not as straightforward as the techniques outlined here.

References

[1] C. Krieg, C. Wolf, A. Jantsch, and T. Zseby, "Toggle MUX: How X-optimism can lead to malicious hardware," in *Design Automation Conference (DAC)*, Jun. 2017, pp. 1–6.

[2] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A Novel Technique for Improving Hardware Trojan Detection and Reducing Trojan Activation Time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 112–125, Jan. 2012.

[3] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "veritrust: Verification for hardware trust," in *Design Automation Conference (DAC)*.

[4] P. Kitsos, K. Stefanidis, and A. G. Voyiatzis, "TERO-Based Detection of Hardware Trojans on FPGA Implementation of the AES Algorithm," in *Euromicro Conference on Digital System Design (DSD)*, Aug. 2016, pp. 678–681.

[5] J. He, Y. Zhao, X. Guo, and Y. Jin, "Hardware Trojan Detection Through Chip-Free Electromagnetic Side-Channel Statistical Analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2939–2948, Oct. 2017.

[6] L. Pyrgas, F. Pirpilidis, A. Panayiotarou, and P. Kitsos, "Thermal Sensor Based Hardware Trojan Detection in FPGAs," in *Euromicro Conference on Digital System Design (DSD)*, Aug. 2017, pp. 268–273.

[7] M. Lecomte, J. Fournier, and P. Maurine, "An On-Chip Technique to Detect Hardware Trojans and Assist Counterfeit Identification," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3317–3330, Dec. 2017.

[8] X. Zhang, A. Ferraiuolo, and M. Tehranipoor, "Detection of Trojans Using a Combined Ring Oscillator Network and Off-chip Transient Power Analysis," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 9, no. 3, 25:1–25:20, Oct. 2013.

[9] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Multiple-parameter side-channel analysis: A non-invasive hardware Trojan detection approach," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, Jun. 2010, pp. 13–18.

[10] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.