High-Level Synthesis of Benevolent Trojans

Christian Pilato[1], Kanad Basu[2], Mohammed Shayan[2], Francesco Regazzoni[3] and Ramesh Karri[2]
[1]Politecnico di Milano, Milan, Italy
[2]New York University, New York, NY, USA
[3]Università della Svizzera italiana, Lugano, Switzerland

***Abstract*—High-Level Synthesis (HLS) allows designers to create a register transfer level (RTL) description of a digital circuit starting from its high-level specification (e.g., C/C++/SystemC). HLS reduces engineering effort and design-time errors, allowing the integration of additional features. This study introduces an approach to generate *benevolent Hardware Trojans* (HT) using HLS. Benevolent HTs are Intellectual Property (IP) watermarks that borrow concepts from well-known malicious HTs to ward off piracy and counterfeiting either during the design flow or in fielded integrated circuits. Benevolent HTs are difficult to detect and remove because they are intertwined with the functional units used to implement the IP. Experimental results testify to the suitability of the approach and the limited overhead.**

I. Introduction

Modern System-on-Chip (SoC) architectures have led to the demise of the traditional design model for integrated circuits (IC) in which a company designs, manufactures, tests and packages all the disparate pieces of the SoC. Instead, design companies procure third-party IPs and combine them in their SoC. This IP-reuse paradigm has several benefits. Re-using IPs is generally cheaper than designing them from scratch, shortening the design cycle and allowing designers to meet the rigid time-to-market pressures. Although an IP-based design has benefits, IP protection becomes an important concern. IP watermarking is a technique to protect the rights of the IP vendor [1]. IP watermarking hides a "signature" in the IP without altering the main function of the design. This "signature" can be verified later, for instance, during litigation to determine the real designer.

In parallel, the complexity of SoC architectures is pushing towards effective design methodologies, such as high-level synthesis (HLS) [2]. HLS allows engineers to use automatic tools to translate high-level descriptions into register-transfer level (RTL). An HLS-based design serves multiple purposes [3]. Complex functionalities can be described in software and generated during HLS. By translating software into RTL, the design costs and also some design errors can be reduced [4].

We propose an approach that, on one side, borrows concepts from *Hardware Trojans* (HTs) to create IP watermarks and, on the other side, raises the abstraction level to HLS. Hardware Trojans are malicious and deliberate design modifications, which are inserted by an adversary. A class of them reveals under rare conditions. Our proposed method is similar but with a benign effect. So we call this alteration *benevolent HT*. We automate the generation of these HT-based watermarks by extending a traditional HLS flow.

After discussing prior research on IP watermarking and HLS-based secure design (Section II) and the useful background on IP watermarking, HTs, and HLS (Section III), we present our major contributions. First, we develop the microarchitecture of a benevolent HT used as watermark (Section IV). Then, we describe an extended HLS flow to automate the insertion of benevolent HTs (Section V). Finally, we present our experimental results (Section VI) and conclude the paper (Section VII).

TABLE I: Comparison of Watermarking Techniques.

Technique	**Level**	**Ref.**	**Overhead**	**Dispersal**	**Verification**
Graph coloring	Synthesis	[1]	High	High	Difficult
FSM state-based	Synthesis	[5], [6]	High	High	Difficult
FSM transitions	Synthesis	[7], [8]	High	High	Difficult
Scan-chains	DFT	[9], [10]	Low	Low	Easy
Hybrid	Synth+DFT	[11]	High	Low	Difficult
Insertion-based	PnR	[12]	Low	Low	Destructive
Multivariate sign	HLS	[13]	High	High	Easy

II. Related Work

A. IP Watermarking Techniques

Table I compares existing watermarking techniques, acting at different levels of the design flow. These methods can be roughly classified into two groups – Finite State Machine (FSM)-based or scan chain-based. FSM-based watermarks take advantage of undefined transitions [7], [8] or undefined states [5], [6] to create a unique sequence, which does not interfere with the normal chip functioning. These watermarks incur high area overhead and are susceptible to removal attack. Scan chain-based watermarks, which use re-ordering of scan chains [9], [10], can lead to routing congestion. Our proposed approach raises the abstraction level, operating during HLS. This allows us to reuse functional units and minimize the overhead. The signature verification is enabled in a way similar to the one used to activate HTs.

B. Security-Aware HLS

Recent research explores how HLS can design IP components with enhanced security features, while minimizing the overhead [14]. HLS-based techniques have been used to build a trustworthy system using untrustworthy third party IPs (3PIPs) [15], [16]. HLS was also used for HT detection and recovery [13], [17]. In this paper, we aim at implementing anti-piracy techniques using HLS.

III. Background

A. Threat Model: IP Infringement

Illegitimate use of IP components creates serious economic damages to IP providers. Our threat model assumes that the adversary has access to the netlist or the layout files to reverse engineer, copy, and sell the IP without authorization. IP watermarking attempts to protect against IP infringement.

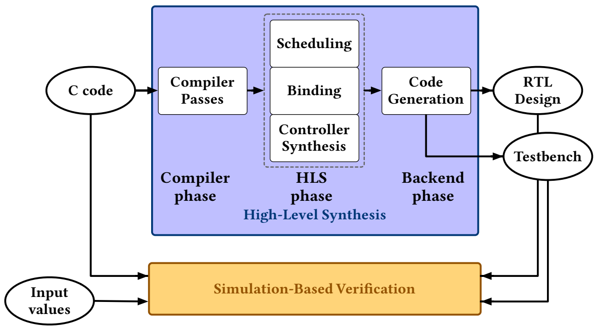


Fig. 1: HLS flow with simulation-based verification.

If the malicious party is unaware of the watermark, she will copy it along with the original design. The watermark can be used later to identify the real IP author during litigation.

B. Hardware Trojans

A HT is a malicious and deliberate alteration of the original design. A HT is usually composed of two components: trigger and payload [18]. The trigger is the part that activates the Trojan. It is usually a rare event. The payload implements the malicious effect, such as leakage of sensitive data or denial of a service. A HT should be hard to identify to avoid detection and removal. The HT designer should find a condition that is hard to identify and hide the resulting trigger logic into the design. The payload should be buried in the design as well, for example by reusing the existing logic.

C. High-Level Synthesis

HLS starts with a software description and transforms it to an RTL. The RTL description is the starting point for the subsequent stages of the design flow like logic synthesis, placement and routing. HLS has three essential steps as portrayed in Figure 1. In the first step, state-of-the-art compilers perform optimizations. In the second step, HLS creates the RTL microarchitecture. The third step generates the RTL in an HDL of choice. Verification of the RTL using simulation vectors ensures that it adheres to the software description.

The two principal units created by an HLS design flow are the controller and the datapath. In each clock cycle, the controller issues commands to the datapath to control the logic. The datapath includes all functional units in the design plus the registers that save the temporary values of the intermediate calculations. Multiplexers determine the computation pathways corresponding to the control inputs.

IV. IP Watermark using Trojan Design Methodologies

Design principles used to insert malicious HTs can be reused to design watermarks that we call benevolent HTs. The main difference with malicious HT insertion is that, in case of IP watermarking, the IP designer is inserting the alteration, identifying rare conditions and dispersing the logic, while the attacker aims at achieving the malicious effect, i.e., the detection and removal of the HT. We base our watermark[1] design on the observation that a circuit has two modes of operation: *functional* and *test*. While the *functional mode* concerns itself with the normal system functionalities, the *test mode* is used to perform manufacturing test on the real device (e.g., to detect manufacturing defects). The system does not perform any real functions in the *test mode*. We design an HT which is active only in test mode. The HT will not interfere with the IP functionality because testing is explicitly requested by the user. We explore two methods to activate the test mode:

1) **explicit:** we directly use an input pin (e.g., an unused bit of the configuration register) or a scan-chain flip-flop.
2) **implicit:** we provide a specific input sequence to activate the *test mode*.

The identification of particular sequence of inputs to activate the test mode (in case of *implicit*) is easier in case of benevolent HTs. This is because the IP designer, who is in charge of inserting the HT, is fully aware of the IP functionality. If a design has all input combinations exhaustively utilized, this method requires an additional condition. This approach uses the same logic used for implementing the IP functionality, as described in Section V-B. Our HT-based watermarks do not affect IP verification or validation, which is performed with traditional methods.

A. Properties of IP Watermarks

Effectiveness of a watermark is often evaluated by analyzing the following properties [8], [19], [20], [21]:

1) **Ownership credibility:** The IP ownership should be easy to verify with minimum chance of collisions with signature generated with other methods.
2) **False positive rate:** There should not exist different input sequences that generate the same signature.
3) **Resist removal:** It should be hard to remove.
4) **Resist re-synthesis:** If the watermark is completely created by signals which are redundant to the function, logic synthesis optimizations may remove the watermark.
5) **Resist a new watermark:** Even if an IP infringer can include his/her watermark on top of the owner's watermark, the original watermark should be still verifiable.
6) **Resist denial:** An IP infringer may reduplicate a watermark, referring to it as a chance occurrence with numerous trial-and-errors. This should not be allowed.
7) **Design metrics degradation:** The IP watermark should have a minimal impact on the design metrics (e.g., minimal resource overhead).

To summarize, a watermark should be stealthily embedded in the design, well dispersed, incur a low design overhead and have minimal side-channel effects. As seen in Table I, existing IP watermarks do not comply with all criteria. On the other hand, HT-based watermarks are a promising solution that can be used to satisfy these properties.

V. HLS-based Design of Benevolent HTs

We extend the classic HLS flow to insert benevolent HTs as shown in Figure 2. Since the HTs must not interfere with the IP execution during functional mode, we insert them after the baseline microarchitecture has been defined, i.e., before the *backend steps*. The trigger implementation, explained in Section V-A, can use any of the two methods described in

[1]In the rest of the paper, we will use *watermark* and *hardware Trojan* (HT) interchangeably.

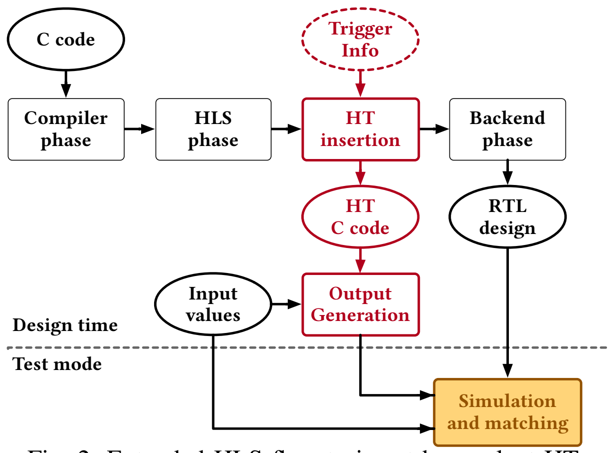


Fig. 2: Extended HLS flow to insert benevolent HTs.

Section IV, while the payload is generated on the top of the existing functional units as described in Section V-B. As output, the extended HLS methodology produces the HDL description of the component augmented with the HT-based watermark ready for logic synthesis. We discuss the proof of authorship of our watermarking solution (Section V-C) and possible protection techniques (Section V-D).

A. Trigger Generation

For *explicit activation*, we use a combination of the configuration bits to specify whether the IP is in test mode or not. The attacker does not know this combination and he/she has to reconstruct it. When this sequence of bits is set, the IP switches to test mode and the HT is activated. Thus, in order to verify the watermark, the user needs to just set this bit and provide the desired input sequence needed for validation.

For *implicit activation*, we add a simple FSM to the design that analyzes the set of input values provided during normal execution until the specific trigger sequence is recognized. The particular sequence is specified by the IP designer as an additional input to the extended HLS flow (*Trigger Info*, as shown in Figure 2). Once the FSM identifies the sequence, the circuit switches from functional to test mode. As discussed in Section IV, no valid computation is done in test mode. Hence, the input sequence must be of no interest and the IP results must be discarded.

Remarks: *The explicit activation does not depend on the specific IP to validate. It leverages the configuration register, which is already present in the chip. Therefore, this method incurs no extra hardware overhead. Unlike malicious HTs, the trigger sequence is not inserted by the attacker but decided by the designer. Therefore, it is easier for the designer to identify the trigger during watermark verification, while it is harder to match during normal execution. The hardware overhead of the trigger in case of implicit activation depends on the length of the input sequence to detect.*

B. Payload Generation

The payload is added to the microarchitecture generated by HLS to carry out its watermarking function while limiting the overhead. It re-uses the functional units of the real design to implement complex sequences of operations and generate the output signature, i.e., a value returned by the IP and read by the user. Since the complexity of the IP function can be high and must differ from the watermarking function, we developed an algorithm to select a subset of the IP operations to perform during watermarking. The payload microarchitecture is then generated by reusing the functional units, interconnections and registers to minimize the area overhead.

Our approach is based on the following observations:

- It is not necessary to watermark all modules of the design. *We restrict the analysis only to a subset of the modules to minimize the overhead.*
- The payload function should be complex enough to create a unique signature. *The payload should involve a significant subset of IP operations.*
- The watermarking procedure should be self-contained, using only internal functional units and limiting interactions with the system. *We exclude operations with side-effects (e.g., accesses to external memories) from the payload.*

We generate the payload as follows:

1) We identify a subset of modules to watermark by analyzing the call graph[2] of the input `C` code. There are many ways to select the functions to embed the IP watermark: e.g., we can identify the functions with the largest number of operations or the most frequently executed ones.
2) We list all functions in reverse topological order. When analyzing one module, all the submodules are analyzed to properly propagate input and output values.

Starting from the innermost function, all the functions are analyzed. If the current function must include the IP watermark, we create the corresponding payload function as follows. We start by creating a copy of the current Control Data Flow Graph (CDFG) and a “random key” that has as many bits as the number of CDFG operations. This vector represents the list of operations to keep or remove in the payload function. Each bit of the key is randomly set to 0/1 with probability p_w (provided by the designer before starting HLS), while control operations are always set to 1 (to avoid changing the behavior) and side-effect operations, like memory operations, are always set to 0 (to eliminate them). When eliminating an operation, the corresponding output is generated by taking one of its inputs with equal probability. The value produced by a memory-read operation is substituted by a random constant, while the results of all memory-store operations of a function are added and propagated to the output port. This modification may require extra functional units or registers if all adders are already used in the clock cycles where the operations are needed. The payload controller sits between the functional controller and the datapath. The resulting microarchitecture is shown in Figure 3, where some control signals are controlled by the functional controller (and the payload controller does not change them) and separate control signals are handled solely by the payload controller. One can multiplex the signals with different values between the function and payload controllers.

The activation of the functional controller is filtered by the payload. When `start` is high and `trigger` is low, the

[2]The call graph represents calling relationships between subroutines.

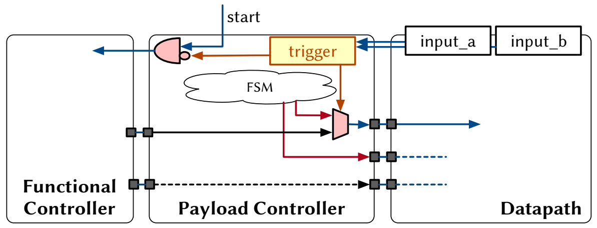


Fig. 3: Separate function and payload controllers.

functional controller executes, bypassing the payload. When `trigger` is high, the functional controller is immobilized and the payload controller controls the datapath.

Each module must be connected to its outermost function to provide input values and to propagate the output as shown in the microarchitecture example of Figure 4. The module inputs are directly connected to the input ports of the submodules (with multiplexers having the trigger signal as selector). If the function has no return operation, we add an additional output port to propagate the output values up to the top module so that it can be read during test mode (see `Module B` of Figure 4). The output values are combined to have a single return value for each module, as shown in `Module A` of Figure 4. When propagating this signal, an output port is added to each function above the modified module in the call graph hierarchy or, if the modules have a return port, multiplexers are added to steer the logic based on the *trigger* signal.

After establishing the payload function, we pre-compute the output values produced by the newly-generated payload CDFG. These golden values will be later used for signature verification. We recreate the resulting payload function in `C` code (*HT C Code* in Figure 2). This code executes on the pre-defined collection of input values to establish the corresponding golden outputs (*Output Generation* in Figure 2).

Remarks: *Since the synthesis tool is oblivious to the functions needed for "useful" computation, the watermark is robust to re-synthesis. The watermark has low overhead since it reuses existing functional units to compute the signature, limiting the congestion. Application of input trigger signals can furnish the output values associated with the payload. This helps us to verify the watermark since the golden values are generated on the payload CDFG resulting from the simplification.*

C. Analysis – Proof of Authorship

The proof of authorship of a watermark (P_c) is usually referred as the "probability of collisions" [22]. Essentially, P_c is the probability of creating a design that carries our watermark by coincidence. In this section, we compute an upper bound on P_c and we show that is convincingly low.

In our HT-based watermarking, there is a collision if a watermark is created with the same payload CDFG. We assume that the designer most likely identify the same function to watermark since it is deterministic (e.g., the function with most operations). We also assume that control operations are always maintained and operations with side effects are always removed. Let N be the number of operations of the function to watermark, N_c be the number of control operations, and N_s be the number of operations with side effects, the number

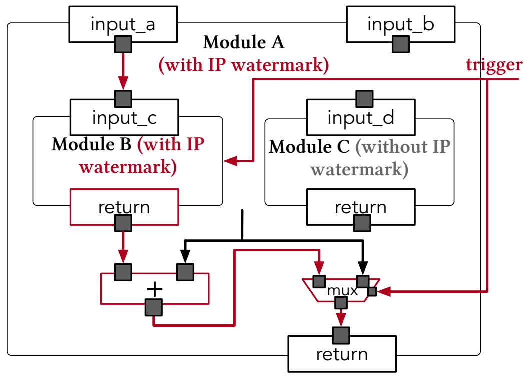


Fig. 4: Connections between modules implement a distributed payload (in red). This includes return port on module B.

of remaining operations is $N_o = N - N_c - N_s$. For these operations, we create a random vector to select which functions to retain. Let p_w be the probability to retain each operation in the payload function. The *probability of collision* P_c can be computed as $P_c = p_w^{N_o}$. This probability is low with a high number of operations N_o.

Remarks: *The "random key" creates a different payload function during each HLS process and the probability to replicate it is extremely low (see the results in Section VI-A).*

D. Protecting the Benevolent HTs

To impede HT disclosure and prevent a malevolent IP customer from bypassing the watermark, we disperse the HT payload over numerous functions and re-use functional units (see Section V-B). Proper select signals are used to control the operations to execute in each mode. However, an attacker can isolate the payload controller and attempt to reverse engineer the watermark by following the trigger signal.

To further thwart the reverse engineering, we merge the HT payload and the functional controllers of the IP. Merging the states of the two controllers is simple because each state of the functional controller has a corresponding state in the payload controller that controls the pathways depending on the operations of the watermarking function.

An alternative approach for the benevolent HT involves introducing a black box in the design where one can insert the payload controller, provided that it can be completely isolated from the rest of the design. This is relevant for FPGA architectures. The IP provider gives the RTL of the IP simultaneously with the directives to set up a bitstream partition of a given size and precise positions for the netlists. The IP buyer receives no other details. The IP consumer receives the payload controller as a partial FPGA bitstream, as in a framework for hardware digital rights management in [23].

Remarks: *Since the payload function is intertwined with the datapath and the two controllers are merged together, isolating the watermark to reverse engineer and remove it is harder. Since most of the functional units are used in both functional and test modes, we expect the side-channel fingerprints to be very similar in the two cases.*

VI. Experimental Evaluation

We insert benevolent HTs during HLS using the BAMBU open-source HLS framework [24]. BAMBU accepts a C-based specification and synthesizes to different FPGA targets. We implemented the benevolent HT insertion method in Section V as an additional pass on the HLS results generated by BAMBU, before generating the structural RTL microarchitecture. To evaluate the overhead introduced by our HT-based watermarks, we targeted a Xilinx Virtex-7 FPGA (xc7vx485t) at 100 MHz and we analyzed the resources required after logic synthesis using Xilinx Vivado 2018.2. We validated the function of the watermark by comparing software results with the ones after RTL simulation using Mentor ModelSim SE 10.3.

We employed BAMBU and its modified version to create IP components for selected kernels from CHStone [25] and MachSuite [26] benchmark suites. All benchmarks are specified in `C`. Table II reports the hardware resources used by the baseline IP. Default HLS options have been used for all benchmarks except `backprop`. In this case, we allocated all data to the external memory due to synthesis problems. The resulting IP implementation does not use FPGA BRAM resources but exchange data with the external memory.

We select one function per benchmark to inject a benevolent HT. For each benchmark, we select the function with most operations to minimize the probability of collisions while keeping the overhead low. The number of operations N of the function to watermark is reported in Table II (Ops). Since the payload is identical for the two types of HTs (explicit and implicit), we evaluated HTs with implicit activation to assess the cost of the trigger circuitry (absent in the explicit activation). For each benchmark, we chose a sequence of three 32-bit values as a trigger to switch on the HT, while the probability p_w of keeping each operation is set to 0.75. This is a reasonable trade-off between the probability of collisions and the resulting area overhead.

A. Evaluating the Watermarking Function

BAMBU generates HDL ready for logic synthesis and hardware test benches for simulation-based validation on a multitude of inputs furnished by the user. We used these artifacts to evaluate the effectiveness of our watermarking method by analyzing the properties presented in Section IV-A.

In the first set of experiments, we did RTL simulations on random sequences of input values to test Trojan activation based on the three 32-bit trigger sequences. For each benchmark, we created 1,000 random combinations of significant inputs and none of these sequences activated the HTs. The IP consistently worked in functional mode with no performance overhead and is unlikely to turn on the HTs by chance.

We presented the input sequences to each benchmark to simulate the IP in test mode. After each simulation, we checked the simulation results against the golden ones obtained in software after the HT generation. The results matched in all instances. Different output values are generated for each sequence of inputs (**False positive rate**). We repeated HLS+simulation several times to generate different payload variants based on the random key. Given each design and each sequence of inputs, different output values are always

TABLE II: Characteristics of IP after HLS+FPGA synthesis and number of operations of the watermarked function.

Benchmark	LUT	FF	DSP	BRAM	Ops
`adpcm` [25]	35,638	10,889	102	104	622
`backprop` [26]	10,126	8,655	59	0	725
`fft` [26]	5,309	4,149	40	0	725
`gsm` [25]	5,145	2,758	38	18	369
`jpeg` [25]	36,678	15,130	18	174	1,280
`mips` [25]	3,086	1,040	8	14	328
`motion` [25]	7,662	2,919	0	26	350
`viterbi` [26]	763	987	2	0	103

generated after each HLS process (**Ownership credibility**), making it extremely difficult to replicate with trial-and-errors (**Resist denial**). Indeed, the probability of collisions P_c ranges between 10^{-12} (`viterbi`, which is the benchmark with less operations) and 10^{-150} (`jpeg`, which is the benchmark with most operations) (**Proof of authorship**).

We performed experiments on logic synthesis to testify that the payload is hard to remove. Specifically, we synthesized the HDL resulting from our extended HLS flow and no simplifications are performed on the payload even with all optimizations active (**Resist re-synthesis**). The IP watermark has a minimal resource overhead, as discussed in Section VI-B (**Design metrics degradation**).

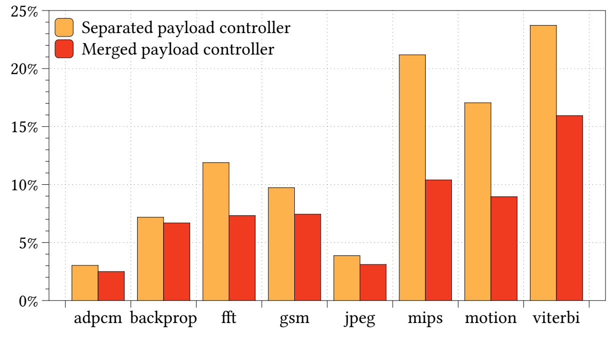
Logic equivalence checkers (LEC) cannot be used to detect HT-based watermarks. The benevolent HT is embedded by the designer, who is aware of the modifications, while the attacker has no golden model of the high-level specification. The designer can discount the LEC warnings corresponding to the HT-based watermarks. Other LEC indications can be addressed by the designer [27] (**Resist removal**).

Finally, an IP infringer may apply existing low-level watermarking methods on the resulting chip design to claim IP ownership. However, our watermarking procedure operates at the functional level by embedding a payload function into the design. Therefore, our watermark is not compromised by the extra watermark and it can be still verified (**Resist a new watermark**).

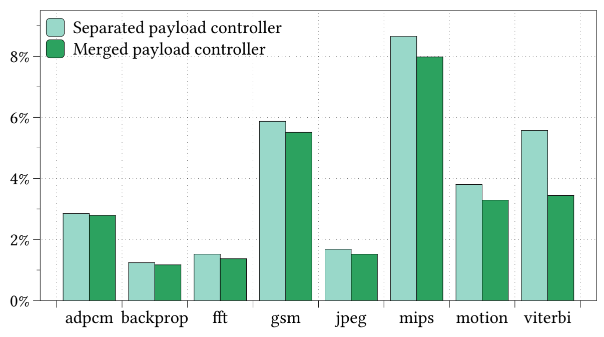
B. Resource Overhead

We performed HLS+FPGA synthesis of each IP design with its HT-based watermark to test the resource overhead compared to the baseline implementation. We conducted these experiments with separated and merged functional and payload controllers (see Section V-D). The trigger modules use few flip-flops (FF) and look-up tables (LUT) to encode the states and the transition function, resulting in a small overhead ($\sim$1%).

Figure 5(a) shows the overhead of LUTs for the payload with separate and merged controllers. Most of the overhead is due to the extra logic needed in the datapath to realize the payload function. The overhead increases as we decrease the probability p_w to keep operations in the payload function since more logic (i.e., multiplexers) must be created to skip the operations and propagate the chosen input value to the corresponding register. The overhead is small ($\sim$8% on average). In the largest benchmarks (`adpcm` and `jpeg`), the LUT overhead is around 3% even with separate controllers. The overhead is large in three benchmarks: `motion`, `mips`



(a) Look-up Tables (LUT)



(b) Flip-Flops (FF)

Fig. 5: Resource overhead of HT-based IP watermarks with separate and merged controllers: (a) LUT, (b) FF.

and `viterbi`. In `motion`, there are several bitwise operations. When eliminated, the multiplexers needed to drive the signals have a larger impact. `mips` and `viterbi` are small benchmarks and the additional resources contribute to a greater overhead, especially due to the additional controller. Merging the controllers has no impact on the datapath and also reduces the resources required to control the functional units. This optimization has more impact on small benchmarks.

Figure 5(b) shows the overhead of FFs for the payload with separate and merged controllers. The overhead is small in all cases ($\sim$3% even with separate controllers). In control-dominated designs (like `gsm` and `mips`), the FF overhead is larger ($\sim$6-8%) since the datapath is small. In data-dominated designs (`jpeg`), controller FFs have a limited impact ($\sim$1%). In case of separate controllers, we require few more FFs to encode the states in the payload controller.

VII. Conclusions and Future Work

This paper offered a technique to implement IP watermarking using benevolent HTs. These HTs are blended into the IP component during HLS by reusing the resources, which reduced the hardware overhead. We suggested two alternative triggers based on either an external pin attached to the configuration registers and scan chains and on a pre-defined string of inputs. On average, the resource overhead for the offered technique is around 8% and 3% for LUTs and FFs. Benevolent HTs will be combined with other IP-protection countermeasures (e.g., chip obfuscation) to provide high robustness against state-of-the-art security properties.

Acknowledgment

This work is supported in part by National Science Foundation (NSF) (A#: 1526405).

References

[1] F. Koushanfar, I. Hong, and M. Potkonjak, “Behavioral synthesis techniques for intellectual property protection,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 3, pp. 523–545, 2005.

[2] Xilinx Inc., “Vivado design suite user guide - designing with IP (UG896),” 2017.

[3] R. Nane *et al.*, “A survey and evaluation of FPGA high-level synthesis tools,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, 2016.

[4] M. Gupta, “Using 3rd party ip in asic/soc design,” 2017. [Online]. Available: https://www.design-reuse.com/articles/31313/using-3rd-party-ip-in-asic-soc-design.html

[5] A. L. Oliveira, “Techniques for the creation of digital watermarks in sequential circuit designs,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1101–1117, 2001.

[6] M. Lewandowski *et al.*, “A novel method for watermarking sequential circuits,” in *Proc. of HOST*, 2012, pp. 21–24.

[7] I. Torunoglu and E. Charbon, “Watermarking-based copyright protection of sequential functions,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 3, pp. 434–440, 2000.

[8] A. Cui *et al.*, “A robust FSM watermarking scheme for IP protection of sequential circuit design,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 30, no. 5, pp. 678–690, 2011.

[9] A. Cui and C.-H. Chang, “Intellectual property authentication by watermarking scan chain in design-for-testability flow,” in *Proc. of ISCAS*, 2008, pp. 2645–2648.

[10] A. Cui and C.-H. Chang, “A post-processing scan-chain watermarking scheme for vlsi intellectual property protection,” in *Proc. of APCCAS*, 2012, pp. 1–4.

[11] A. Cui, C.-H. Chang, and L. Zhang, “A hybrid watermarking scheme for sequential functions,” in *Proc. of ISCAS*, 2011, pp. 2333–2336.

[12] B. Le Gal and L. Bossuet, “Automatic low-cost ip watermarking technique based on output mark insertions,” *Design Automation for Embedded Systems*, vol. 16, no. 2, pp. 71–92, 2012.

[13] A. Sengupta and D. Roy, “Antipiracy-aware ip chipset design for ce devices: A robust watermarking approach [hardware matters],” *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 118–124, 2017.

[14] C. Pilato *et al.*, “Securing hardware accelerators: a new challenge for high-level synthesis,” *IEEE Embedded Systems Letters*, 2017.

[15] J. Rajendran, H. Zhang, O. Sinanoglu, and R. Karri, “High-level synthesis for security and trust,” in *Proc. of IOLTS*, 2013, pp. 232–233.

[16] J. J. Rajendran, O. Sinanoglu, and R. Karri, “Building trustworthy systems using untrusted components: a high-level synthesis approach,” *IEEE Trans. on VLSI Syst.*, vol. 24, no. 9, pp. 2946–2959, 2016.

[17] X. Cui *et al.*, “High-level synthesis for run-time hardware trojan detection and recovery,” in *Proc. of DAC*, 2014, pp. 1–6.

[18] M. Tehranipoor and F. Koushanfar, “A survey of hardware trojan taxonomy and detection,” *IEEE design & test of computers*, vol. 27, no. 1, 2010.

[19] X. Huang *et al.*, “A new watermarking scheme on scan chain ordering for hard IP protection,” in *Proc. of ISCAS*, 2017, pp. 1–4.

[20] A. B. Kahng *et al.*, “Constraint-based watermarking techniques for design IP protection,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1236–1252, 2001.

[21] A. T. Abdel-Hamid *et al.*, “IP watermarking techniques: Survey and comparison,” in *Proc. of IWSOC*, 2003, pp. 60–65.

[22] A. B. Kahng *et al.*, “Constraint-based watermarking techniques for design IP protection,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1236–1252, Oct. 2001.

[23] M. Barbareschi, A. Cilardo, and A. Mazzeo, “Partial FPGA bitstream encryption enabling hardware DRM in mobile environments,” in *Proc. of CF*, 2016, pp. 443–448.

[24] C. Pilato and F. Ferrandi, “Bambu: A modular framework for the high level synthesis of memory-intensive applications,” in *Proc. of FPL*, Sep. 2013, pp. 1–4.

[25] Y. Hara *et al.*, “Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis,” *Journal of Information Processing*, vol. 17, pp. 242–254, 2009.

[26] B. Reagen *et al.*, “MachSuite: Benchmarks for accelerator design and customized architectures,” in *Proc. of IISWC*, Oct. 2014, pp. 110–119.

[27] A. Saifhashemi, H.-H. Huang, P. Bhalerao, and P. A. Beerel, “Logical equivalence checking of asynchronous circuits using commercial tools,” in *Proc. of DATE*, 2015, pp. 1563 – 1567.