High-Level Synthesis of Key Based Obfuscated RTL Datapaths

Sheikh Ariful Islam and Srinivas Katkoori
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
Email: {sheikhariful, katkoori}@mail.usf.edu

***Abstract*—Reverse engineering (RE) a register transfer level (RTL) description allows an attacker to counterfeit intellectual property (IP) as well as introduce hardware trojans. To mitigate this risk, RTL obfuscation can be employed. Most of the existing obfuscation methods are targeted at gate-level and layout-level. In this work, we propose key based RTL obfuscation scheme at an early design phase during high-level synthesis (HLS). Given a control data flow graph (CDFG), obfuscation points are identified during scheduling and obfuscation logic is inserted during the datapath generation phase. In order to keep performance overhead low, such insertion is done only on noncritical paths. We implemented the proposed obfuscation technique in an in-house HLS system and the obfuscated RTL designs were synthesized to gate-level with Synopsys Design compiler targeting 90nm CMOS technology library. Based on the experimental results on four datapath intensive benchmarks, we demonstrate that proposed approach obfuscates the design with extremely low probability of reverse engineering. For a 32-bit obfuscation key, the average area, delay, and power overheads are 2.45%, 2.65%, and 2.61% respectively, which are reasonable.**

I. Introduction

In complex system-on-chip (SoC) design at advanced technology nodes, third party components (IPs, design automation tools, library files, etc.) account for a significant portion of chip design due to stringent time-to-market requirement. In the heart of this design paradigm, fabless design house is performing fabrication and testing in foreign foundries due to the expensive cost of owning a foundry [1]. IP infringement has been identified as major security challenge in this loose control of system design. Concurrent threats on hardware allow an 'untrusted foundry' to overbuild, counterfeit, insert trojans, reverse engineer (RE) to leak valuable information without any visible knowledge of original IP owner. Due to extended life cycle and attack space in hardware design cycle, IP protection at early design phase is essential to overcome the above security challenges.

Hardware obfuscation is one of the possible countermeasures to create obscure description (both structural and functional) for native source language (e.g., VHDL, Verilog, SystemC). Design protection through code obfuscation for software has been investigated extensively so far [2]. Though semantic-preserving transformations are performed for software obfuscation, a similar line of obfuscation methods entail both semantic- and functionality-preserving for hardware. One benefit of hardware obfuscation is to render 'unintelligible' netlist and hence slow down the reverse-engineering approaches for smart hackers/attackers.

At gate- and layout-level, there is a wealth of techniques in literature for IP protection using several techniques such as hardware metering [3]–[5], digital watermarking [6]–[8], and fingerprint analysis [9]–[11]. The underlying assumption of key-based obfuscation methods is to prevent RE of the state transition functionality extraction for RTL and gate-level design and this has been possible by embedding additional states for correct key initialization [12]–[16]. Such key-based approaches have two common characteristics: (a) system initialization on correct key value else entering into 'black hole state'; and (b) a large number of sequential elements to prevent brute-force attack.

A naive implementation of key-based obfuscation is to embed a large number of state elements incurring significant design overhead. Although this simple technique would expose the attacker to perform brute-force attack, security primitives should be implemented with minimal design overhead. As a result, carefully determining obfuscation points is of paramount interest. However, no systematic approach is available to construct the obfuscated RTL netlist from a high-level design description (C, C++, SystemC). This motivates us to pursue an approach to obfuscate RTL design at an early phase of the design cycle. Our goal for obfuscation is achieved through enhanced complexity of reverse engineering task.

To the best of the authors' knowledge, this is the first work that attempts key based obfuscation during HLS. The main contribution of this work is to automate the obfuscation of an RTL design during the design phase by determining possible obfuscation points early on. The proposed algorithm finds operations on non-critical paths of the input CDFG (control-data flow graph) for obfuscation logic insertion during the datapath generation. Experimental results for four datapath intensive benchmarks implemented in 90nm technology node show that the proposed obfuscation for the 32-bit key can be achieved with acceptable 2.45% area, 2.65% delay, and 2.61% power overheads.

Section 2 presents an overview of hardware obfuscation which forms the basis of providing obfuscation during behavioral synthesis. Section 3 presents in detail the proposed obfuscation method during high-level synthesis. Section 4 reports the experimental results. Section 5 draws conclusions.

978-1-5386-1214-9/18/$31.00 ©2018 IEEE

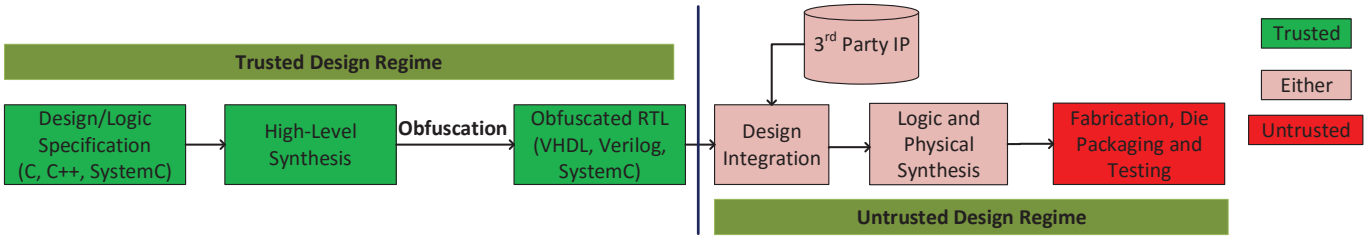


Fig. 1: Obfuscation during IC design in trusted/untrusted design environment.

II. Background and Related work

This section summarizes the RTL, gate-level, and layout-level approaches of current hardware obfuscation methods for mitigating reverse engineering.

Hardware obfuscation is a technique for protection against leaking design specification during untrusted execution. It increases the difficulty to read, interpret and eventually modifying the structure and functionality of inherent IP/IC. Hardware obfuscation can be described as follows. An RTL output denoted by $out = f(dp_0, ctrl_0)$ can be obfuscated to produce $out = f'(dp_1, ctrl_1, k)$ where $\{dp_0, dp_1, k\}$ are, respectively, regular datapath, obfuscated datapath and key value. Similarly, $\{ctrl_0, ctrl_1\}$ denote regular controller and obfuscated controller respectively. The regular RTL design f and obfuscated RTL design f' must be structurally different but functionally equivalent for a unique obfuscation key value.

To protect soft IP's behavioral description from an adversary, Chakraborty and Bhunia proposed *mode-based* FSM design by extending the bit-length of a host register to select appropriate mode [13]. As correct values (keys) need to be passed along the design phase for correct functionality during system integration, an attacker in integration house or foundry can overbuild the same IP. A similar line of defense to construct obfuscated RTL IP following gate-level netlist modification and de-compilation is proposed in [14]. Li and Zhou examine in detail the penalty for piracy based on *stuttering* for sequential circuits [15]. Their approach uses common input combinations for all FF's in the design to determine normal and slow mode operation for the design. However, protection scheme of a hard IP deliverable to system integrator by the IP owner is not specified. Comparator-free dynamic codeword generation and encoded within transition function for correct circuit functionality is proposed in [17]. The authors mention that traversing a particular path makes it feasible to create code-word to be interlocked within the design. A functional mode is carried out even with wrong code-word to create incorrect circuit behavior thus creating more confusion for the attacker.

The concept of 'ObfusFlow (Obfuscated Design Flow)' at the gate level design was proposed by Chakraborty and Bhunia [12]. The output of an embedded FSM having the same number of inputs is XOR-ed with the node(s) having larger fan-in and fan-out to determine the exact input sequence of internal state elements. An extended version of this technique namely 'HARPOON' can ensure both authentication and obfuscation for gate level netlist [16].

There have been recent efforts to obfuscate design at the layout level. Circuit camouflaging against image-recognition based attack analysis to recover the original chip design includes creating cells that seem identical in terms of size and layer spacing [18]–[20].

III. Proposed Approach

A. Attack model

An attacker with the help of compromised design tool or extraction tool [21] can retrieve the high-level functionality of a design. Usage of such disassembling tool can increase attacker confidence in reverse engineering against obfuscation approach. We assume an attack model wherein the early design phase until RTL design is in trusted zone and untrusted later on (See Figure 1). It is reasonable as system integrator find their way to reveal key-less IPs specification at a reasonable performance cost while an untrusted foundry can perform IP piracy. Therefore, designs sent out to a foundry for fabrication following integration must be obfuscated to protect from malicious attack scenario. Further, in order to minimize de-obfuscation intent, such obfuscation should be provided in early design phase. Compared to prior work [13], we incorporate obfuscation during the datapath generation phase of high-level synthesis, while in [13], the RTL design is modified during post-synthesis. Our approach provides more freedom for obfuscation. Works exist that introduce exor and inverters for obfuscation purposes [13], [22]; compared to these works, using muxes provides a function agnostic way and thus is applicable to wide variety of applications.

Given a specification, HLS tool performs the structural synthesis. Obfuscation technique is then applied to RTL description. The obfuscated netlist is passed down to logic synthesis tool to create an obfuscated logic model and physical synthesis produces the geometric obfuscated layout of the design and its interface to other IPs. As integrator and foundry are commonly untrusted due to their knowledge of the firm and hard silicon features respectively, the correct de-obfuscation keys from early design space will be provided only after the design is manufactured.

B. Proposed framework

Figure 2 shows the HLS framework for obfuscation as a synthesis objective. The input is user-specified area, clock frequency, and obfuscation key length. It incorporates an overhead estimator to generate the revised area and frequency constraint for a given key length. The revised constraints are used as

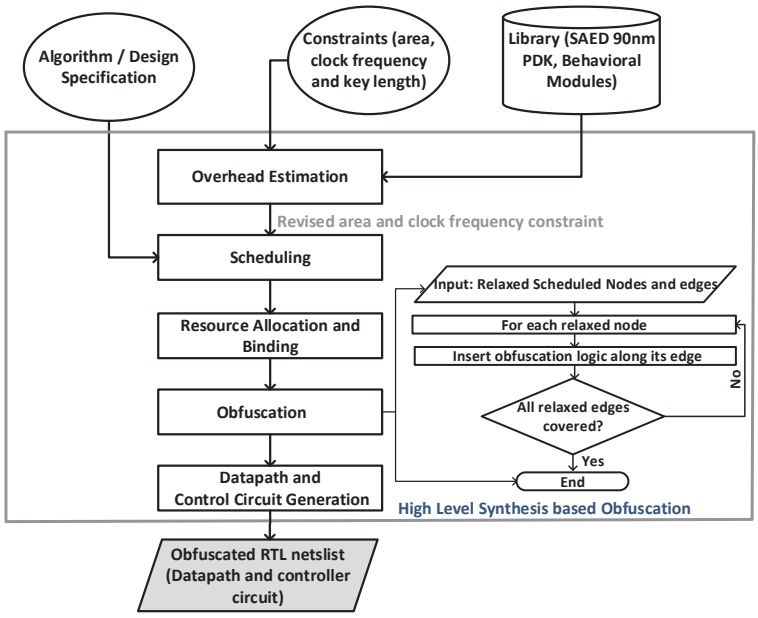


Fig. 2: High Level Synthesis for Obfuscated RTL design flow

Algorithm Obfuscate
Input: G=Scheduled CDFG, K=Random Key
Output: Obfuscated RTL Datapath

```
begin
 // Step 1 - Identify obf. set
 foreach operation op in G do
   if(op is not on critical path) then
     Add op to Obf_op_set{}
   end if
 end for
 // Step 2 - Insert obf. logic randomly
 foreach bit k_i in K do
   op <- Randomly pick an obfuscation op
         from Obf_op_set{}
   I <- Randomly select from {input1,
         input2, output} of the op for
         obfuscation logic insertion.
   Insert a mux with two inputs
   if(k_i = 0) then
     input0 of mux <- I
     input1 of mux <- random line
   else
     input0 of mux <- random line
     input1 of mux <- I
   end if
   mux select <- k_i
  end for
end
```

Fig. 3: Proposed RTL Obfuscation Algorithm

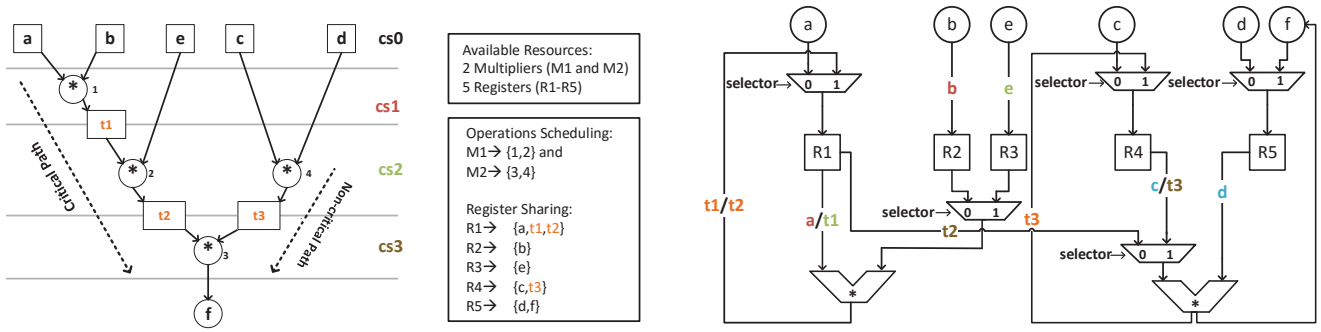
input to perform architectural synthesis. The resultant intermediate representation is passed down to obfuscation phase. The probable obfuscation points are identified and obfuscation logic from which obfuscated RTL netlist is generated.

Figure 3 shows the pseudo-code of the proposed obfuscation algorithm. It accepts a scheduled CDFG (G) and a random key (K). There two major steps. In the first step, the graph is analyzed for operations for possible obfuscation at their inputs or output. In order not to degrade the performance of the design, only the operations on non-critical paths are considered. In the second step, obfuscation logic is inserted in the design. For each bit of the obfuscation key, randomly we select an operation and its input or output line and replace it with a 2-input multiplexer. The select line of the mux is controlled by the key bit. Randomly a line in the design is selected and fed to the input besides the original input. After inserting obfuscation logic for all bits in the key, the obfuscated datapath is generated. Note that in our approach the obfuscation logic is inserted in the datapath only. Although the controller is untouched, in order to RE the design, the attacker must know the original algorithm and the schedule, allocation, and binding information. We assume that this information is kept confidential and unavailable during the untrusted design phase.

The time complexity of the algorithm can be estimated as follows. The first step of obfuscation operation set identification requires computation of mobility of each operation. This can be done by performing ASAP and ALAP scheduling on the CDFG. For directed acyclic graphs, the worst case complexity of these algorithms is same as that of topological sort i.e., $O(|V|+|E|)$, where V is the number of operations and E the number of edges in G. The second step is of constant time complexity. Therefore, the overall time complexity is $O(|V|+|E|)$.

C. Motivational example

To illustrate early design obfuscation approach, consider a simple multiplicative function, namely, $f = e*(a*b)*(c*d)$. Assume that two multipliers and five registers are available. We assume input values are readily available at the beginning



(a) Scheduling, allocation, and binding

(b) Datapath

Fig. 4: Synthesis of arithmetic function: f= e*(a*b)*(c*d)

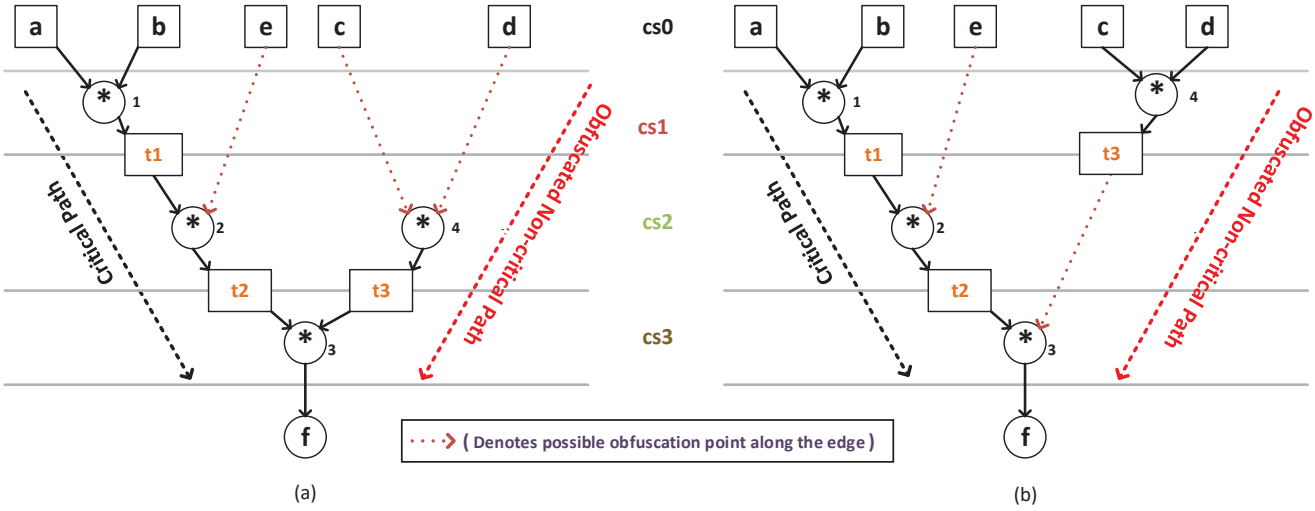


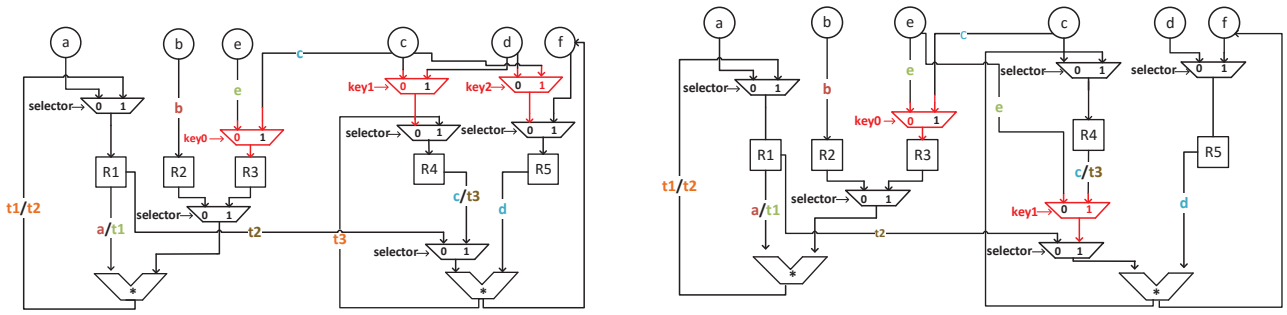
Fig. 5: Variation of scheduling for relaxing nodes

of execution (control step 0, $cs0$). The maximum number of multiplication operations that can be performed in a given cycle is two. Let us consider the critical path along $\{a, t1, t2, f\}$. A resource constrained scheduling of function f is shown in Figure 4a. Based on available resources, an example datapath is as shown in Figure 4b.

Operation 4 has the flexibility to be scheduled in either $cs1$ or $cs2$ as shown in Figure 4a. As the inputs are not required until $cs2$, for schedule shown in Figure 5a obfuscation can be performed on the inputs of Operation 4, thus tolerating delay on the inputs. While in schedule shown in Figure 5b, as its output is not required until $cs3$, the output can be obfuscated. The corresponding datapaths are as shown in Figures 6a and 6b. This example illustrates the obfuscation possibilities arising due to scheduling. In the sequel, we refer to operations with mobility as $relaxed$.

Before delving into the details of the proposed obfuscation, it is important to stress the impact of mobility that use time difference between start times for as soon as possible (ASAP) and as late as possible (ALAP) scheduling. In our approach, we perform the mobility calculation [23] of each node after scheduling is completed. The nodes having zero mobility restricts their scheduling at a particular time to meet latency constraint and scheduling for the nodes having mobility greater than one can be relaxed. In our example, operations 1, 2, and 3 have zero mobility as they satisfy the latency bound (four control steps) and operation 4 has the mobility of one, so it is a relaxed node.

As relaxed scheduling consists possible obfuscation points, we limit our discussion to impose obfuscation logic in those partial nodes of data flow graph. As shown in Figure 5, small dotted lines represent the flexibility of obfuscation logic. We can embed steering logic along the edges represented by nodes $\{c, d, e\}$ to increase the difficulty of an attacker from logging original structural information. For example, obfuscated datapath in Figure 6a is produced by a scheduled DFG in Figure 5a. Here, key 0, 1 and 2 belong to obfuscation points for nodes $\{e, c, d\}$. Input value in red mark represents the correct 1-bit key for each mux. For this obfuscated datapath, we introduce three 2-input multiplexers (mux). However, if additional inputs are available for mux, the resultant obfuscated datapath can reveal very limited to no local structure compared to 2-input mux for the brute-force attack. Either primary input c or d can be fed into $key0$ based mux along with primary input e. We arbitrarily



(a) Key insertion in scheduled datapath for Figure 5a.

(b) Key insertion in scheduled datapath for Figure 5b.

Fig. 6: Possible Obfuscation points of arithmetic function: f= e*(a*b)*(c*d).

TABLE I: Scheduling freedom for f (cs=control step).

Scheduling variant	Unobfuscated function (f)	Obfuscated function (f')	key Value (k)
Fig. 6a	e*(a*b)*(c*d)	$(\overline{k_0}$e+ k_0c)*(a*b)*$(\overline{k_1}$c + k_1d)* $(\overline{k_2}$c + k_2d)	$\{k_0, k_1, k_2\} = \{0, 0, 1\}$
Fig. 6b	e*(a*b)*(c*d)	$(\overline{k_0}$e+ k_0c)*(a*b)*$(\overline{k_1}$e+ $(k_1$(c*d))	$\{k_0, k_1\} = \{0, 1\}$

choose primary input c here. Similarly, Figure 5b reconstructs the original datapath in Figure 6b for two (2) obfuscation points. The criterion for higher obfuscation is determined by the number of available inputs for mux and relaxed nodes. An example of multiplicative function generation based on mux size and a number of obfuscation points is shown in Table I.

D. Difficulty of attack

We evaluate the proposed scheduling based obfuscation scheme in terms of three metrics: (a) potency, (b) resilience, and (c) cost [24]. Potency refers to the difficulty level of an attacker to understand an obfuscated program. As we cannot quantify this subjective metric and we want the attacker to carry out the circuit functionality with wrong output, a formal equivalence tool (BDD, OBDD, ROBDD) cannot construct correct data-flow graph due to large space of obfuscation points.

Resilience determines the withstanding capability of an obfuscated datapath against attacker attempt. Given the number of registers (R) each being n-bit width and a number of obfuscation keys (k), an attacker has to figure out the possible registers location holding those k-keys. One out of $\sum_{k=1}^{R} \binom{R*n}{k}$ possibilities can determine correct key locations in finite amount of datapath registers. The search is augmented by $k!$ different ways of arranging k keys into R registers. We ensure the resilience of obfuscated dataflow graph by the following equation:

$$P(R, n, k) = \frac{1}{\sum_{k=1}^{R} \binom{R*n}{k} * k!} \quad (1)$$

A lower value of P requires brute-force attack in de-obfuscating RTL datapath for correct obfuscation points. With a larger number of register bits and key size, the attackers need more resources (computational time and effort) to reverse engineer the design.

IV. Experimental Results

An in-house HLS tool is used to insert obfuscation logic into regular RTL datapath. A functional testing procedure to assess the obfuscated RTL against reverse engineering is presented. Finally, the results of applying the proposed obfuscation scheme on HLS benchmarks are analyzed for design overhead given a key length and latency bound.

A. Experimental Setup

The experimental flow of obfuscating a high-level design description is as follows:

1) Input to the HLS tool is a VHDL behavioral description that is converted into a CDFG. The CDFG is scheduled with Force-Directed Scheduling algorithm for a given latency constraint. Resource allocation and binding are automatically performed. Then structural datapath and behavioral controller are generated in VHDL.
2) The RTL design is synthesized with Synopsys Design Compiler in 90nm technology to generate gate level netlist. Synopsys VCS-MX was used to check the functional equivalence of synthesized netlist and original RTL.
3) To generate the obfuscated design, obfuscation is carried out before VHDL generation with the algorithm described in Section III(B). The obfuscation elements are preserved as keys are provided through primary inputs (PI). This renders a functionally equivalent structurally modified RTL netlist.

B. Results and analysis

We implemented our proposed obfuscation technique in C programming language and evaluated it on four datapath intensive HLS benchmarks, namely Elliptic Wave Filter (Elliptic), Fast Fourier Transform (FFT), Finite Impulse Response (FIR) Filter, and Lattice Filter. The RTL designs are targetted

TABLE II: Comparison of design attributes of non-obfuscated and obfuscated designs.

Design	Non-obfuscated						Obfuscated			Obfuscation Overhead			Resilience (P)
	Latency Bound (λ)	# Operations (A=+, M=*, S=-)	# Registers (Datapath + Controller)	Area (um^2)	Delay (ns)	Power (uW)	Area (um^2)	Delay (ns)	Power (uW)	Area Overhead (%)	Delay Overhead (%)	Power Overhead (%)	
Elliptic	15	(26+, 8*)	43	110941	28.04	536.03	114474	28.84	548.82	3.18	2.85	2.38	$5.2e{-}97$
FIR	5	(4+, 5*)	19	76806	25.80	507.70	78013	26.58	509.05	1.59	3.02	0.26	$18.8e{-}86$
FFT	10	(20+, 16*, 4-)	56	67152	19.62	320.25	69096	20.26	331.91	2.89	3.26	3.64	$6.6e{-}44$
Lattice	10	(8+, 5*)	21	64796	26.65	360.86	66197	27.05	375.94	2.16	1.50	4.17	$7.0e{-}87$
Average										2.45	2.65	2.617	

to Synopsys SAED 90nm PDK standard cell library. PPA (power, performance, and area) analysis of the user-specified architecture optimization is carried out for each design. A direct comparison with prior RTL obfuscation technique [13] is not possible, as our RTL designs are generated from algorithmic descriptions, while in [13] the RTL code is manually written and is not partitioned into datapath and control.

Table II reports design overhead compared to non-obfuscated design. For a key length of 32 bits, we incur, on average 2.45% area overhead. We observe the power overhead 2.61% for same key length sequence. The delay overhead of 2.65% is reasonable for possible obfuscation points and available resources. These are reasonable trade-offs for the high resistance to malicious reverse engineering.

In the last column of Table II, we report the probability values for resilience i.e., obfuscation efficiency using Equation (1) for 32-bit keys. It is intuitive that for such extremely low values an attacker will enumerate all possible netlists, thereby likely facing high resistance to reverse engineering. Superior resilience can be provided by a random distribution of key in both controller and datapath circuitry using our scheme.

V. Conclusions

Obfuscation of an RTL datapath early in the design flow can help to build in high resistance to reverse engineering effort. In this paper, we introduce for the the first time, a technique to obfuscate an RTL design during high-level synthesis. Operations on non-critical paths in the CDFG are targeted so that there is little or no performance overhead. Experimental results for four data-path intensive benchmarks is highly promising. Further, we show that the probability of reverse engineering which bits are the obfuscation key bits is extremely low.

References

[1] R. Kumar. Simply fabless! *IEEE Solid-State Circuits Magazine*, 3(4):8–14, Fall 2011.

[2] G. McGraw. Software security. *IEEE Security & Privacy*, 2(2):80–83, 2004.

[3] Y. Alkabani and F. Koushanfar. Active Hardware Metering for Intellectual Property Protection and Security. In *USENIX security symposium*, pages 291–306, 2007.

[4] R. Maes, D. Schellekens, P. Tuyls, and I. Verbauwhede. Analysis and design of active IC metering schemes. In *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*, pages 74–81. IEEE, 2009.

[5] F. Koushanfar. Integrated circuits metering for piracy protection and digital rights management: An overview. In *Proceedings of the GLSVLSI*, pages 449–454. ACM, 2011.

[6] F. Koushanfar, I. Hong, and M. Potkonjak. Behavioral Synthesis Techniques for Intellectual Property Protection. *ACM Trans. Des. Autom. Electron. Syst.*, 10(3):523–545, July 2005.

[7] I. Hong and M. Potkonjak. Behavioral Synthesis Techniques for Intellectual Property Protection. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 849–854, New York, NY, USA, 1999. ACM.

[8] M. Lewandowski, R. Meana, M. Morrison, and S. Katkoori. A novel method for watermarking sequential circuits. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 21–24, June 2012.

[9] J. B. Wendt, F. Koushanfar, and M. Potkonjak. Techniques for foundry identification. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[10] A. E. Caldwell, Hyun-Jin Choi, A. B. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J. L. Wong. Effective iterative techniques for fingerprinting design IP. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(2):208–215, Feb 2004.

[11] C. Bell, M. Lewandowski, and S. Katkoori. A multi-parameter functional side-channel analysis method for hardware trust verification. In *2013 IEEE 31st VLSI Test Symposium (VTS)*, pages 1–4, April 2013.

[12] R. S. Chakraborty and S. Bhunia. Hardware protection and authentication through netlist level obfuscation. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 674–677, Nov 2008.

[13] R. S. Chakraborty and S. Bhunia. RTL Hardware IP Protection Using Key-Based Control and Data Flow Obfuscation. In *Proceedings of the 2010 23rd International Conference on VLSI Design*, VLSID '10, pages 405–410, Washington, DC, USA, 2010. IEEE Computer Society.

[14] R. S. Chakraborty and S. Bhunia. Security through obscurity: An approach for protecting Register Transfer Level hardware IP. In *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 96–99, July 2009.

[15] L. Li and H. Zhou. Structural transformation for best-possible obfuscation of sequential circuits. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 55–60, June 2013.

[16] R. S. Chakraborty and S. Bhunia. HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, Oct 2009.

[17] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall. Interlocking Obfuscation for Anti-tamper Hardware. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 8:1–8:4, New York, NY, USA, 2013. ACM.

[18] R.P. Cocchi, L.W. Chow, J.P. Baukus, and B.J. Wang. Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing, August 13 2013. US Patent 8,510,700.

[19] L.W. Chow, J.P. Baukus, B.J. Wang, and R.P. Cocchi. Camouflaging a standard cell based integrated circuit, April 3 2012. US Patent 8,151,235.

[20] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang. Circuit camouflage integration for hardware IP protection. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–5, June 2014.

[21] T. Meade, S. Zhang, and Y. Jin. Netlist reverse engineering for high-level functionality reconstruction. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 655–660, Jan 2016.

[22] S. Koteshwara, C. H. Kim, and K. K. Parhi. Key-based dynamic functional obfuscation of integrated circuits using sequentially triggered mode-based design. *IEEE Transactions on Information Forensics and Security*, 13(1):79–93, Jan 2018.

[23] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.

[24] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Transactions on software engineering*, 28(8):735–746, 2002.