A Practical Circuit Fingerprinting Method Utilizing Observability Don't Care Conditions

Carson Dunbar and Gang Qu
Electrical and Computer Engineering Department and Institute for Systems Research
University of Maryland, College Park, Maryland, USA
{cdunbar, gangqu}@umd.edu

***Abstract*— Circuit fingerprinting is a method that adds unique features into each copy of a circuit such that they can be identified for the purpose of tracing intellectual property (IP) piracy. It is challenging to develop effective fingerprinting techniques because each copy of the IP must be made different, which increases the design and manufacturing cost. In this paper, we explore the Observability Don't Care (ODC) conditions to create multiple fingerprinting copies of a design IP (e.g. in the form of gate level layout) with minute changes. More specifically, we find locations in the given circuit layout where we can replace a gate with another gate and some wires without changing the functionality of the circuit. However, as expected, this could introduce design overhead. Our experimental results show that, although we can embed fingerprints of up to 1438 bits, there is an average of 10.9% area increase, 50.5% delay increase, and 9.4% power increase on circuits in the MCNC and ISCAS 85 benchmark suites. We further propose a fingerprinting heuristics under delay constraints to help us reduce area and power overhead.**

I. INTRODUCTION

With the system on a chip (SoC) paradigm becoming more popular, especially for small embedded system developers, intellectual property (IP) blocks such as cores and memory units are becoming essential. These IPs enable the more efficient reuse based design methodology [1]. However, it also makes IP theft a profitable business and a major threat to IP developers, vendors, and SoC industry in general and hence motivates the IP protection problem [1, 2, 7, 10].

In addition to law enforcement mechanisms, such as patent, copyright, and encryption, watermarking and fingerprinting have proven to be effective countermeasures to theft. Watermarking enables the designer of the IP to prove the authorship of an IP, while fingerprinting allows the tracking of each individual sold IP. Therefore, when the IP designer suspects any IP piracy, he can verify this by retrieving the embedded watermark, and then use the fingerprint to locate the source of the IP piracy (i.e., which IP buyer has been involved in the IP piracy).

For this purpose, an IP fingerprinting technique needs to meet the following three fundamental requirements: (1) *correct functionality*. Failure to provide the desired functionality will make the IP useless; (2) *distinct fingerprints*. If two copies of the IP have identical fingerprints, then we won't be able to tell which one is the source of the IP piracy; (3) *heredity*. The fingerprint must stay in any illegally reproduced IP instances in order to enable the trace of IP piracy. As we will survey in the related work section, current circuit fingerprinting methods either violate these requirements or require non-trivial (re)design efforts.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA

http://dx.doi.org/10.1145/2744769.2744780

A. Improvements to the State of the Art

In this paper, we propose a practical method to generate a large number of distinct fingerprinted IPs with little design and re-design overhead. Our method utilizes a two-step process to introduce fingerprints to individual integrated circuits (ICs). First, an IC is designed with a number of flexibilities so every IC fabricated is identical. Second, in the post-silicon stage, the flexibilities are solidified such that each IC has an individual fingerprint. Introducing flexibility in circuits reduces the redesign for fingerprints by moving fingerprint application to the last stages of the VLSI design cycle.

We first use a small example to show the basic ideas behind our fingerprinting approach. Then we review the related work and provide the background in section II. The proposed method is elaborated in section III and our experimental setup is explained in section IV. Finally, we report the experimental results and conclude the paper in sections V and VI.

Illustrative example.

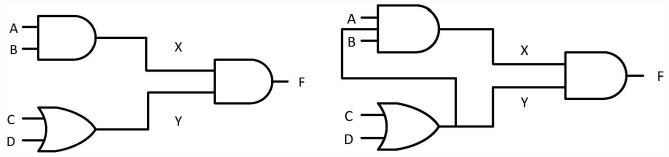

Fig 1. Two 4-input circuits that implement the same function.

The left circuit in Figure 1 realizes the function $(AB)(C + D) = F$. When the Y input to the AND is zero, the output F will be zero regardless of the value of X input; however, when Y=1, F will be determined by the X input. So when we direct signal Y to the AND gate that generates X, as shown in the right of Figure 1, we can easily verify that this circuit implements the same function F. However, these two circuits are clearly distinct. Moreover, if one makes a copy of any of these circuits, this distinction remains. Thus we can embed one bit fingerprint information by controlling whether X depends on Y. This fingerprint meets all the three requirements we listed earlier.

One key feature of this approach is that the changes we make on the circuit are minute. We can make a connection, as shown on the right circuit in Figure 1, during routing and placement; then determine whether to keep this connection based on the fingerprint bits at post-silicon phase. This avoids the expensive redesign and fabrication based on a new layouts as in the current fingerprinting approaches [2, 3].

Challenges and contributions. As simple as the above example suggests, there remain several technically challenging questions to develop a systematic fingerprinting technique based on this idea. For

example, the two circuits shown in Figure 2 also implement the same function F as those circuits in Figure 1 and they all have similar layout. The contribution of this paper is the discovery of the proposed practical fingerprinting method and its implementation.

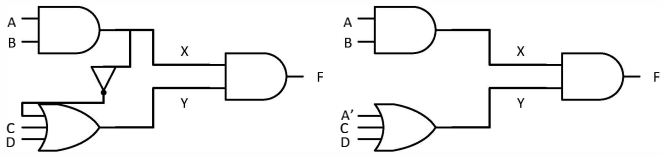

Fig 2. Two more implementation of the same function.

The first major challenge was how to efficiently identify the locations in the circuit where we can make changes to reflect the fingerprint? Second, at such fingerprinting locations, what kind of changes to the circuit can we make? Third, how much fingerprinting information can be embedded, or how many different fingerprinting copies can be generated by this approach? Finally, what is the impact to the design quality such as area, delay, and power after embedding fingerprints?

We propose to compute the Observability Don't Care (ODC) conditions at each gate and use such information to locate the gates that can be modified to embed fingerprints. Once the fingerprinting locations are identified, we analyze the circuit locally to determine what kinds of changes can be made. Because ODC conditions exist almost everywhere in any combinational circuit, this provides us a large space to embed fingerprints. When we design circuits, with the consideration of adding fingerprints after fabrication, we realize that reasonable area and power overhead have incurred, but the delay penalty is too large to accept. Therefore, we propose a delay-driven heuristics to create fingerprinting copies under delay constraints.

II. RELATED WORK

IP protection is becoming a more relevant topic in circuit design as many circuits are not being manufactured by the firms that designed them, as they used to be. With many parties being involved with the production of a single circuit, it is easier for someone to steal information about the circuit for their own benefit. In this section we will discuss the current work being done on circuit fingerprinting.

A. Circuit Fingerprints

Like a human fingerprints, which can identify any one person, circuit fingerprints attempt to identify individual ICs, or batches of ICs, that were manufactured. These can either be placed in the circuit beforehand or derived from process variations and other circuit characteristics.

Patel et al. created a technique in [2] to determine a circuit's fingerprint through its glitches. A glitch may be considered a temporary signal that is incorrect but overall does not affect the performance of the device. The authors state that if the glitches are not causing major issues in the circuit, that they can be used as a fingerprint. In addition, by changing the operation temperature they are able to create more temporary glitches that could help them identify specific ICs.

Jin et al., from [3], use a different technique that takes advantage of process variations as opposed to glitches that can be fixed. They use the delay path variations to create a fingerprint for a circuit. This helps them prevent Hardware Trojans, which would increase the path delay and change the output of the IC in a harmful way.

Both [2] and [3] are techniques for fingerprinting individual circuits. Neither technique is suitable for IP protection because if an adversary is willing to copy a circuit, the fingerprint will be completely different from the fingerprint in the circuit that they copied. If this is the case, someone trying to prevent duplication would not be able to prove that an adversary is copying their work because they would not have a record of the new fingerprint coming from their designs.

Due to its difficulty, fingerprinting of ASICs for IP protection does not have much research being done as watermarking. The first major paper on this specific topic was done by Caldwell et al. of [4], where the authors attempt to create fingerprints by using "specific VLSI CAD optimization" heuristics. They use the NP-hard problems of partitioning, satisfiability, graph coloring and standard-cell placement problems, similar to the watermarking work in [5], [6], [7], and [12], with different criteria to create independent fingerprints for each buyer of the device.

Building off the idea of the idea in [4], a conceptually different fingerprinting method for the graph coloring problem is proposed in [8], where the authors effectively add new constraints to the graph by either adding new states or adding new pathways which either manipulate cliques or bridge current nodes. By doing this, they are able to increase the solution space which means that they can create even more varied fingerprints, which was a possible problem of [4] if the design was highly specified before optimization.

The issue with both [4] and [8] is that these methods must be implemented in the earlier stages of the VLSI design cycle. This requires a significant amount of redesign to the circuit and will increase the production cost and time. This is where we propose to make our contribution and improvements to the state-of-the-art, in a similar manner to the work in [9].

Finally, we mention the physical unclonable function (PUF), a new security technology. Based on the unclonable intrinsic fabrication variation in delay, capacitance, or threshold voltage, the PUF circuitry can produce a unique response to a given challenge to authenticate a device or generate a bitstream as the cryptographic key. PUFs are generated in additional circuitry and thus they will not alter the functionality. Furthermore, the fabrication variation is believed to be unique. So the first two requirements of fingerprints we proposed earlier are satisfied. However, when an IP is illegally reproduced, the PUF information will be changed and thus violate the last requirement for fingerprint. Therefore, PUF can be used as a fingerprint for device authentication, but not for IP protection.

III. OBSERVABILITY DON'T CARE (ODC) FINGERPRINTING

The goal of this work is to define a new paradigm of fingerprinting that depends on IC post-silicon flexibility and improves on previous work in the field. Most current fingerprinting techniques, such as the work in [2] and [3], use process variation as a method of fingerprinting a circuit. This can be unreliable because it requires significantly accurate measurements to be made, both when determining the fingerprint initially and any time afterward. These techniques rely on process variation so if an adversary copies the circuit, the owner's fingerprint is no longer present, which violates the *heredity* requirement for fingerprinting and defeats the purpose of the fingerprint. Other methods like those mentioned in [4] and [8] are significantly more costly due to the need for high-level circuit redesign.

Our method attempts to use ODCs to create small changes in the circuit which can be implemented in the later stages of the VLSI design cycle. These changes will have no effect on the functionality of the circuit, but will allow a designer or manufacturer to determine the circuit's origin. They also allow the designer to create specific fingerprints for groups of ICs or individual ICs, depending on the IC's design and buyer information.

Although this work is centered on the concept of utilizing ODCs to create small modifications throughout a circuit, several challenges prevent it from being a simple task. First, not all gates that create an ODC can be used to create a circuit modification, so conditions need to be established to find locations suitable for modification. Once a location is established, a modification may be made. The second challenge is that the modification is dependent on the location and gates at that location. Finally, once all of the modifications have been made, it is important to prevent unacceptable overhead from occurring due to the modifications. In the rest of this section, we elaborate formally these challenges and our solutions to solve them.

A. Observability Don't Cares

Observability Don't Cares are a concept in Boolean computation. The conditions by which an ODC occurs are when local signal changes cannot be observed at a primary output. An example of this can be seen in Figure 3 below. When the bottom AND gate has an input equal to zero, it will generate a zero as its output. This output will be propagated to the next AND gate and generate another zero as the primary output for this circuit. The signals from C, A, and B are all blocked and cannot be observed from the primary output.

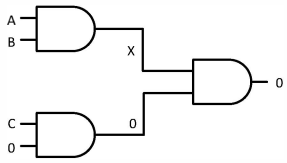

Fig 3. ODC on the AND gate

ODCs can be several layers deep and can cause several different signals to be blocked, depending on the input to the circuit.

Formally, the ODC conditions of a function *F* with respect to one of its input signal *x* can be defined as the following Boolean difference:

$$ODC_x = \left(\frac{\partial F}{\partial x}\right)' = \left(F_x \oplus F_{x'}\right)' = F_x F_{x'} + F_x{}' F_{x'}{}' \quad (1)$$

Basically, this states that when we have a function *F*, and a variable *x*, when the condition ODC_x is satisfied, the value of variable *x* will not have any impact to the value of the function *F*. In the above example, the final output is produced by a 2-input AND gate, for one of its input *x*, if the other input is *y*, then from equation (1), we have $ODC_x = y'$. Hence, when the other input has value zero (as shown in Figure 3), input *x* becomes an ODC.

B. Finding Locations for Circuit Modification based on ODCs

Every logic circuit that is created uses a library of gates that determines the logical relationships that can occur. Most libraries contain gates that create ODCs as defined using Equation (1), but not every instance of these gates will be able to be modified to accommodate a fingerprint. There are four necessary conditions that must be met for a gate to be considered a fingerprint location and are enumerated in the following definition.

Definition 1 (Fingerprint location): A fingerprint location is defined as two or more gates that can be considered for modification for a circuit fingerprint without changing the functionality of the circuit. These gates consist of a single primary gate and one or more gates that generate inputs for the primary gate that meet the following criteria:

1. The primary gate must have at least one input that is not a primary input of the circuit.
2. The primary gate must have at least one input which is the output signal of a fanout free cone (FFC), which means that this signal only goes into the primary gate.
3. The FFC in criterion 2 must have either a gate with non-zero ODC (such as those in Table I) or a single input gate (e.g. an inverter).
4. The primary gate must have a non-zero ODC with respect to one or more of its input signals other than the one from the FFC.

Criterion 1 is necessary for making local minor changes to the circuit (for fingerprinting purpose). Criterion 2 ensures that the changes made to the FFC will not affect the functionality of the circuit elsewhere. Criteria 3 and 4 provide a possible signal (in criterion 4) that can be added to a gate in the FFC (in criterion 3). Each ODC gate, in a circuit is analyzed using Definition 1 and if it satisfies all the criteria in the definition, it is then considered to be a fingerprint location, a location where the circuit can be modified to add the fingerprint.

C. Determining Potential Fingerprint Modifications

For each fingerprint location that is found, a modification can be applied to the gate's inputs. A generic modification for a fingerprint is depicted in Figure 4.

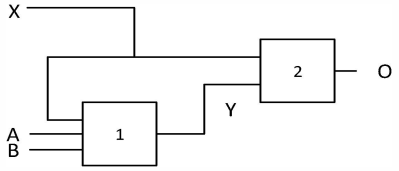

Fig 4. Generic fingerprint change

Figure 4 has two generic gates, represented as boxes 1 and 2, three primary inputs (X, A, and B), and one primary output (O). Gate 2 represents the primary gate, gate 1 represents the gate within the FFC that generates signal Y, and signal X is independent of the FFC that generates signal Y. Suppose that signal X satisfies ODC_Y, thus we can add signal X into the FFC of Y, for example gate 1 as shown in Figure 4, either in its regular form X or its complement form X'. However, when we make this addition, we need to guarantee that when signal X takes the value that does not satisfy ODC_Y, it will not change the correct output value Y. In the rest of this paper, signal X will be known as an ODC trigger signal, as defined below

Definition 2 (ODC Trigger Signal): An ODC Trigger signal is a signal that feeds into a gate, with a non-zero set of ODC conditions, which causes the ODC condition to activate. In the context of this work it also represents the signal that is used to modify the input gate to the primary gate for the fingerprint modification.

In order for this to work, the relationship between the signal X, gate 2, and gate 1 must be analyzed so that X only changes gate 1's output, Y, when it also triggers the ODC, criterion 3 in the definition of a fingerprint location. For every possible pair of gates that can be considered a fingerprint location, similar to gate 1 and gate 2, a structural change must be proposed in order to modify that location. This requires a maximum of n^2 proposed changes, where n is the number of ODCs and single input gates in a library. An example of this exists for the library we used, in Table II, later in this section. Modifications with certain gates may not always be feasible, especially if the overhead costs are too large.

For simple changes like the one in Figure 4 or those in the motivation example, each location like this can be considered a position to embed one bit in a bit string that represents the fingerprint. For each circuit that is manufactured, this fingerprint location can be either modified 1, or left alone 0. This means that for a circuit for n potential fingerprint locations there are at least 2^n possible fingerprints and n bits of data in the bit string.

In addition to the simple change in Figure 4, a fingerprint location may also consist of more than one input gate as stated in Definition 1. If this is the case, a modification similar to the one in Figure 4 may be applied to each of the input gate in the FFC that abides by Definition 1, criterion 3. If this situation occurs, k bits are added to the fingerprint bit string, and the number of potential fingerprints is multiplied by 2^k, where k is equal to the number of input gates in the FFC of the primary gate.

It is also possible to leverage certain gate combinations to add data to our fingerprint bit string and also to reduce the delay that can be caused by rerouting signals. If the ODC gate, that is being considered as a fingerprint location, is immediately preceded by another ODC gate with the same ODC trigger signal, such as the two ANDs in Figure 5 or for example a NOR preceded by an AND, the fact that the ODC would be triggered before the X signal is generated in Figure 5 can be utilized to directly feed one or two input signals into the gate from the fan out free cone. The OR gate in Figure 5 shows an example of this. Input signals, A or B are simply inverted and directed into the OR gate, removing the need to put X into the OR gate which cause a delay if the gates on the left hand side have equal delay.

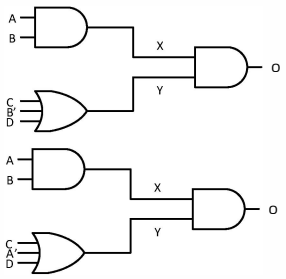

Fig 5. Fingerprint change that reroutes earlier signals

This additional method of fingerprint change allows us to add more data to our fingerprint bit string. For every gate combination like this we are able to add at least n(n+1)/2 potential changes, where n is the number of inputs to the ODC trigger gate, because all combinations of the inputs to the ODC trigger gate can be added to the inputs of the fan out free cone gate. This also means we can add $\log_2(n(n+1)/2)$ bits to our fingerprint bit string. In sum, with only one fingerprint location, we may be able to modify the circuit in potentially many different ways and thus embed multiple bits of fingerprint.

D. Maintaining Overhead Constraints

The fingerprint modifications proposed can cause a large overhead, relative to the circuit's initial performance. Rerouting paths, increasing the input size to input gates, and introducing new inverters are the cause of the overhead. Two heuristic methods have been considered for reducing this overhead, a reactive method and proactive method.

Of the two methods, the reactive method is easier to implement but is difficult to scale. This method involves taking a fully fingerprinted circuit and by removing one fingerprint modification at a time, analyzing the difference in overhead, whether it be area, delay, power, or something else. The modification that results in the largest change to the overhead is removed and the resulting circuit is tested again. This is done until a certain overhead constraint is met or there are no more modifications to remove.

The proactive method is more difficult to implement, but because it is done as modifications are applied it scales well with larger circuits. This heuristic requires that each modification is analyzed before being implemented. For area and power this is simple because any new gates or changes in gates will result in overhead that can be estimated using information about the cells in the library. Delay is more difficult to analyze because not every modification will slow the circuit down. As modifications are added the critical path may change which changes where new modifications should be considered. The delay can be estimated by determining the slack on each gate and updating the information every time a modification is made, but this can be time consuming for large gates that will have a large number of modifications. For this proactive method, modifications would be added until a certain overhead constraint was met, the opposite of the reactive method.

E. Security Analysis

As we have mentioned in the introduction, an IP will be protected by both watermark (to establish the IP's authorship) and fingerprint (to identify each IP buyer). When a suspicious IP is found, the watermark will be first verified to confirm that IP piracy has occurred. Next, the fingerprint needs to be discovered to trace the IP buyer who may be involved in the IP piracy. It is trivial for the IP designer to detect the fingerprint embedded by our proposed approach because the designer can compare the fingerprinted IP with the design that does not have any fingerprint to check whether and what change has occurred in each fingerprint location to obtain the fingerprint.

However, it is infeasible for an attacker to reveal the fingerprint locations from a single copy of the IC. This is because that when the fingerprint information is embedded at a fingerprint location, the FFC of the fingerprinted IP will include the signal that is not in the FFC in the original design when the fingerprint location is identified. Consider the left circuit in Figure 1 of the motivational example, the FFC that generates signal X contains only the 2-input AND gate with A and B as input. But when signal Y is added to this AND gate, the FFC will include the 2-input OR gate with C and D as input. This will make this portion of the circuit not a fingerprint location (criterion 4 is violated).

When the attacker has multiple copies of fingerprinted ICs, he can compare the layout of these ICs and identify the fingerprint locations where different fingerprint bits were embedded in these ICs. This collusion attack is a powerful attack for all known fingerprinting methods. Carefully designed fingerprint copy distribution schemes may help [2, 3, 8], but require a large number of fingerprinting copies. As we will demonstrate through experiments, our proposed approach has this capability and thus can reduce the damage of collusion attack. In addition, it is also known that as long as the collusion attacker does not remove all the fingerprint information, all the copies that are involved in the collusion can be traced [2, 3, 8].

IV. Experimental Setup

To prove the usefulness of this new method, a circuit modifier was constructed in C++, based on the description in the previous section. We started with the Microelectronics Center of North Carolina (MCNC) and ISCAS '85 benchmark circuits in the Berkeley Logical Interchange Format (blif), which specifies the circuits' logical behavior, not its physical layout. From here they were put through Berkeley's ABC [10] program with a library of gate cells. The ABC program can map a blif file to a Verilog netlist with the standard gates in the library. ABC also allowed us to get both the area and delay of the benchmark circuit.

A. Initial Fingerprinting

Our first goal was to create a program that could implement the details of section III A and B. We started by gathering all of the gate data, from the benchmarks, including: gate type, fan outs, fan ins, and gate depth.

Combining this information with the ODC calculations, done with equation 1 and the gates in our library, allowed us to determine what gate combinations were viable fingerprint locations.

```
Input: Circuit in Verilog netlist format
Output: Circuit in Verilog netlist format with fingerprints inserted
Variables: Gi stores gate information for each gate
1)  for each line in file:
2)     if(line == gate)
3)        add gate to Gi
4)  for each gate in Gi:
5)     store gate type, fan ins, fan outs in Gi
6)     determine and store depth in Gi
7)     if(gate creates ODC):
8)        for each gate2 that fans into gate
9)           if gate2 only feeds into gate
10)          mark gate2
11) for each gate in Gi:
12)    if(gate has marked fan ins && creates an ODC)
13)       choose fan in with greatest depth
14)       choose other gate with lowest depth
15)       add fingerprint to new file
16)       continue
17)    else
18)       Write original gate to new file
19) output new file
```

Fig 6. Pseudo-code of proposed method

The next problem was to take the fingerprint locations and modify them to get the maximum fingerprint size. For each fingerprint location we chose to work with the input gate within the fan out free cone, which had the highest depth. We chose the gate with the highest depth to our primary gate so that we knew it did not need a signal until the latest possible time, reducing delay. For the input signal, we chose the ODC trigger signal that occurred at the earliest depth to create our fingerprint location. The ODC trigger signal was chosen so that we could reduce our delay overhead. Once all of the fingerprint locations were discovered, a look up table was used to determine which fingerprint modifications could be made to the circuit. Figure 6 shows this entire process in pseudo-code.

Each of benchmark circuit netlists were run through our circuit modifier and the resulting performance metrics (area, delay, and power) were recorded. Table II is a listing of a subset of the benchmark circuits that were tested as well as their original area, delay, and power measurements, in columns 3-5. After the fingerprint modifications were applied, we re-measured the circuit, and got the new area, delay, and power, as seen in columns 8-10.

B. Reduction of Overhead

Our original results, shown in Table III, columns 6-10, had a large delay overhead. To compensate for this overhead, a program was written to implement the reactive method mentioned in section III C, tuned for delay overhead. This method was chosen because it gave us an approximate upper bound on fingerprint locations and did not require any sophisticated industrial tools.

The program went through our design and took turns removing each fingerprint location we created, one at a time, and tested the result against the original circuit. The result with the minimum delay was saved as our new circuit, and the process started over again with the new design as the base design. The program would then continue to attempt to remove finger print locations, one at a time, until we reached our overhead constraint.

When no fingerprint location could be found that reduced the system's delay, random fingerprint locations were removed until a better delay could be achieved again. Because this was done at random, not systematically, this program needed to be run several times to find more optimal solutions. As a result, the data we obtained may not be the optimal solution, but the data usually left a good number of fingerprint locations for each gate of a significant size.

V. EXPERIMENTAL RESULTS

As we can see from the results, our area overhead is acceptable except in a few of the circuits. The largest area overhead is for one of the smallest circuits and it can be reasoned that even small changes to the gates and possibly including new ones will cause a significant change in the area. The bottom of Table II shows our average results and an average of 12.60% overhead is acceptable, especially when a number of circuits have between 20 and 400 fingerprint locations that were changed.

On the downside though, the delay overhead is fairly poor. Table II shows that we have an average delay of 64.36%. At its worst it is almost 80%, which means the circuit would need to run at almost half its original speed. This is unacceptable. The reason for this result is that the circuit modifying program does not yet check for changes that are made on the critical path, where changes would cause increases to the delay.

The major result of this work, is the number of possible fingerprint locations. Both the circuit size and gate composition contributed greatly to the number of fingerprint locations. In circuits with a smaller number of gates, more variation occurred in the number of fingerprint locations. This is because the layouts are more varied as well. A circuit with a shallower depth will not have as many gates that have inputs from previous gates.

For the rest of the gates, the number of fingerprint locations was quite good. The number of locations allows for a minimum of 2^n possible fingerprint combinations, where n is the number of locations. This means that even if we only have 21 locations, as in gate C432, we can create more than 2 million possible fingerprints.

The reason we say that 2^n is a minimum number is because for every fingerprint location, there can be several configurations, as discussed in section III. These configurations can increase the total number of fingerprints significantly, as can be seen in the column six of Table II. Column 7 of Table II shows the $log_2(n)$ where n is the number of possible fingerprint combinations there are. This was done for two reasons: 1. the numbers were so large in some cases that the data could not be accurately represented in our tables and in the program we wrote and 2. this data gives an idea of how much information can be hidden in these fingerprints if you think of each combination as a bit of information. For most of the circuits, the number of possible combinations of changes to the circuit is far larger than 2^n.

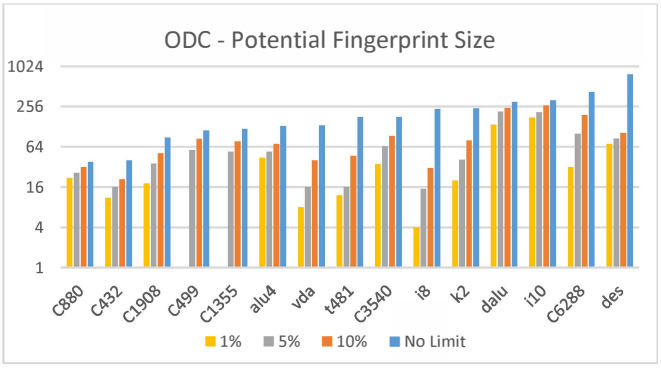

Fig 7. Fingerprint sizes for tested circuits before and after constraints

TABLE II. RESULTS OF A SUBSET OF MCNC/ISCAS 85 AND 89 BENCHMARKS BEFORE/AFTER ODC FINGERPRINT INJECTION.

	Gate Count (original)	*Area*	*Delay*	*Power*	*Fingerprint Locations*	*Log_2(Possible Fingerprint Combinations)*	*Area Overhead*	*Delay Overhead*	*Power Overhead*
C432	166	269584	9.49	1349.5	40	68.07	11.19%	54.69%	6.05%
C499	409	662128	7.62	2951.6	112	177.16	9.25%	31.23%	10.00%
C880	255	426880	6.95	2068	38	66.58	6.52%	47.05%	5.86%
C1355	412	668160	7.67	2988.2	118	187.36	9.86%	30.38%	9.44%
C1908	395	635216	10.66	2655.4	88	151.25	11.40%	46.53%	11.92%
C3540	851	1469488	11.64	7242.3	179	376.79	10.10%	50.52%	9.46%
C6288	3056	4797760	32.92	1	420	635.26	6.29%	34.33%	N/A
des	3544	5831552	6.64	23145.3	782	1438.62	11.87%	75.00%	8.13%
k2	1206	2039280	5.82	5482.4	241	470.25	13.36%	78.87%	8.64%
t481	826	1478768	6.49	4188.1	178	418.62	13.49%	74.42%	7.08%
i10	1600	2676816	12.65	9729.9	316	601.15	9.85%	48.70%	9.03%
i8	1211	2273600	4.73	9621.6	235	541.13	9.45%	67.44%	10.63%
dalu	836	1383184	10.1	5275	298	507.57	15.97%	47.13%	21.45%
vda	635	1088080	4.51	3270.4	134	277.42	14.24%	58.98%	9.75%
Avg Change							12.60%	64.36%	10.67%

For gates with an excessive number of fingerprint combinations, we can either eliminate some of the locations to reduce our overhead, or include additional functionality to our fingerprints, such as error correcting codes or redundancy, so that even if an adversary tampers with the circuit, we can figure out what they have done and what the original fingerprint was.

Table III shows the average results of our heuristic approach to delay overhead management. The rows show we put on a 10%, 5%, and 1% delay overhead constraint on delay. For most of the larger circuits we still had a good number of fingerprint locations left to create a robust fingerprint set. Our area, delay, and power overhead are also minimized. These gates had few fingerprint locations to begin with so attempting to put a delay constraint on them was not possible with the current system. Figure 7 also shows a comparison in terms of fingerprint size for our original results versus the results of our delay constrained fingerprint. It can be seen that, while there is a steep decline in fingerprint size when constraining the delay, the size of the fingerprint for the 5% and 10% delay constraint are still of a significant size. In addition, for the larger circuits, the 1% constraint still provides us with a large fingerprint.

TABLE III. AVERAGE RESULTS AFTER HEURISTIC OVERHEAD CONSTRAINT

	Fingerprint Reduction	*Area Overhead*	*Delay Overhead*	*Power Overhead*
10% Delay Constraint	49.00%	*5.04%*	***9.42%***	*4.99%*
5% Delay Constraint	64.30%	*3.57%*	***4.44%***	*2.46%*
1% Delay Constraint	81.03%	*2.40%*	***0.41%***	*2.65%*

VI. CONCLUSION

This work has shown that we can create a significant number of fingerprints that are hard to detect, and thus hard to remove even for the relatively small benchmark circuits. These fingerprints have a minimal overhead since they only require a minor change in the gate level design. In addition, if we utilize the amount of data we can hold in these fingerprints, we will have the opportunity to make the fingerprints more robust against attackers. There are several potential directions to continue this work. One of the most important and interesting one is to investigate how to implement these fingerprints further down the manufacturing line to make them practical for real life designs. Potential methods include using fuses as the connections for the added lines so we can decide which ones are active, determining where there are empty lanes to where we can add fingerprint locations and be able to add connections after fabricating the rest of the circuit or using engineering changes to remove or reroute lines in post-production.

VII. ACKNOWLEDGEMENT

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61228204 and by the Army Research Office under grant W911NF1210416.

VIII. REFERENCES

[1] A. Cui and C.-H. Chang, "Stego-signature at Logic Synthesis Level for Digital Design IP Protection," in *ISCAS*, Island of Kos, 2006.

[2] H. J. Patel, J. W. Crouch, Y. C. Kim and T. C. Kim, "Creating a unique digital fingerprint using existing combinational logic," in *Circuits and Systems, IEEE International Symposium on*, Taipei, 2009.

[3] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerpring," in *Hardware-Oriented Security and Trust, IEEE International Workshop on*, Anaheim, CA, 2008.

[4] A. E. Caldwell, H.-J. Choi, A. B. Kahng, S. Mantik, M. Potkonjak, G. Qu and J. L. Wong, "Effective Iterative Techniques for Fingerprinting Design IP," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, New York, NY, 1999.

[5] A. B. Kahng and e. al., "Copy Detection for Intellectual Property Protection of VLSI Desgin," in *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, California, 1999.

[6] I. Hong and M. Potkonjak, "Techniques for intellectual property protection of DSP designs," in *IEEE International Conference. Acoustics, Speech, and Signal Processing*, Munich, Germany, 1998.

[7] S. Megerian, M. Drinic and M. Potkonjak, "Watermarking Integer Linear Programming Solutions," in *IEEE/ACM Design Automation Conference*, New Orleans, LA, 2002.

[8] G. Qu and M. Potkonjak, "Fingerprinting intellectual property using constraint-addition," in *Proceedings of the 37th Annual Design Automationn Conference*, New York, NY, 2000.

[9] C. Dunbar and G. Qu, "Satisfiability Don't Care Condition Based Circuit Fingerprinting Techniques," in *Proceedings of 20th Asia and South Pacific Design Automation Conference (ASP-DAC), 2015* , Chiba, Japan, 2015.

[10] R. K. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification*, Berlin Heidelberg, Springer, 2010, pp. 24-40.

[11] A. L. Oliveira, "Techniques for the creation of digital watermarks in sequential circuit designs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 20, no. 9, pp. 1101-1117, 2001.

[12] E. Charbon, "Hierarchical watermarking in IC design," in *Proc. Custome Integrated Circuit Conf.*, Santa Clara, CA, 1998.

[13] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang and G. Wolfe, "Robust IP watermarking methodologies for physical design," in *Proceedings of the ACM/IEEE Design Automation Conference*, Piscataway, NJ, 1998.

[14] A. T. Abdel-Hamid, S. Tahar and E. M. Aboulhamid, "A survey on IP watermarking techniques," *Design automation for embeddedsystems,* vol. 9, no. 3, p. 211, 2004.

[15] N. Narayan and e. al., "IP Protection for VLSI Designs Via Watermarking of Routes," in *IEEE INternational ASIC/SOC Conference*, Washington, DC, 2001.

[16] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Sal-danha, H. Savoj, P. R. Sephan, R. K. Brayton and A. Sangiovannie-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," University of California, 1992.