N-Variant IC Design: Methodology and Applications

Yousra Alkabani
CS Dept., Rice University
6100 Main St., MS-132
Houston, TX 77005
yousra@rice.edu

Farinaz Koushanfar
ECE and CS Dept(s)., Rice University
6100 Main St., MS-380
Houston, TX 77005
farinaz@rice.edu

ABSTRACT

We propose the first method for designing *N-variant* sequential circuits. The flexibility provided by the N-variants enables a number of important tasks, including IP protection, IP metering, security, design optimization, self-adaptation and fault-tolerance. The method is based on extending the finite state machine (FSM) of the design to include multiple variants of the same design specification. The state transitions are managed by added signals that may come from various triggers depending on the target application. We devise an algorithm for implementing the N-variant IC design. We discuss the necessary manipulations of the added signals that would facilitate the various tasks. The key advantage to integrating the heterogeneity in the functional specification of the design is that we can configure the variants during or post-manufacturing, but removal, extraction or deletion of the variants is not viable. Experimental results on benchmark circuits demonstrate that the method can be automatically and efficiently implemented. Because of its lightweight, N-variant design is particularly well-suited for securing embedded systems. As a proof-of-concept, we implement the N-variant method for content protection in portable media players, e.g., iPod. We discuss how the N-variant design methodology readily enables new digital rights management methods.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids; K.6.5 [**Security and Protection**]: Physical Security

General Terms

Design, Security

Keywords

N-variant Design, Digital Rights Management, Physical Security

1. INTRODUCTION

N-variant design is the generation of $N \geq 2$ realizations of the same initial design description. The advantage of the technique is that it provides improved flexibility, robustness, attack resiliency, and design diversity. The strength and usefulness of N-variant designs was previously demonstrated for programs [6], virtual machines [10], and for achieving architectural heterogeneity [10]. While the method was initially intended for providing fault-tolerance, recent applications in security of computer systems, software and data has amplified its importance. Many attacks that take advantage of the specific and stationary nature of the underlying platform, may be eliminated by using N-variants [6, 10].

We propose the first methodology for designing N-variant ICs. The method works at the functional specification level, or by scripts that automatically perform pre-synthesis alterations. We construct a single hardware design consisting of multiple variants that are planned to have several exploitation sets. Note that it is also possible to do N-variant design post-synthesis. The drawback is that the design would become vulnerable to removal attacks. The advantage of pre-synthesis alternation is that all of the variants become an integral part of the pertinent design's functionality, making the removal attack detrimental to the whole structure.

N-variant IC design has a number of important applications.
(i) Hardware IP protection and digital rights management: New semiconductor business models can be enabled by the N-variant design, e.g., different versions of one design can be sold to various vendors, enabling an automatic way to trace the ICs.
(ii) Usage and content metering: Alternation of the variants can be used for enumerating the usage of IC components that run software/ media files, or in conjunction with unclonable chip IDs for metering the usage of an ICs.
(iii) Security: There is a need to prevent the exploits that target a homogeneous design from working on all the ICs [10]. Selection of different variants provides an effective countermeasure against the attacks.
(iv) Post-silicon optimization: Because of the manufacturing variability, for each IC, the designer can use the testing results to select the variant that has the best power/delay characteristics [3, 14].
(v) Self-adaptation: Due to the impact of variability in operational conditions, e.g., aging, the IC can be designed such that it can adapt its structure over time.
(vi) fault-tolerance: The inherent redundancy provided by N-variants enables fault-tolerance.

Integration of N-variants at the functional level is done by altering the FSM and adding several states to it. By managing (controlling) the state-transitions inputs, one would be able to select different variants of the design. The management inputs may come from various triggers, that depend on the target task for the N-variant design. We discuss in detail how the management may be altered for the various applications that we are considering.

The key advantage of using FSM is that it is not extractable from the synthesized design. Thus, even for a party who has access to the synthesized hardware IP, changing the FSM or extracting the original single-variant design would need an effort equivalent to redoing all the stages of design and implementation. For our purposes, the FSM can be safely assumed *inextricable*. Another important benefit of FSM is that certain aspects of FSM are inexpensively *verifiable* post-silicon. The inextricability and verifiability properties of the FSM were previously used for watermarking and for hiding information inside the design, and for remote activation and disabling [1, 2, 14, 15, 17, 20]. As it was shown in the context of watermarking and IC activation/disabling [1, 2, 12], careful constraint manipulation and don't care planning can greatly reduce the overhead of FSM modifications. The new method is particularly well-suited for lightweight embedded systems applications. This is because the N-variant design enables lightweight mechanisms for protection and security of IPs, software and content. The overhead of implementing traditional cryptographic protocols is huge, often overwhelming the constrained resources of an embedded system [19]. As a proof-of-concept, we demonstrate application of N-variant IC design for content usage metering of portable multimedia devices with an embedded MPEG compression module.

1.1 Motivational example

Figure 1 illustrates a motivational example, where the FSM of the design is shown by a state transition graph (STG) that has four states: s_0, s_1, s_2, and s_3. This FSM is the first variant. We replicate this design three times to have: s'_1, s'_2, s'_3, and s'_4 as the first copy (the second variant), s''_1, s''_2, s''_3, and s''_4 as the second copy (the third variant), and s'''_1, s'''_2, s'''_3, and s'''_4 as the third copy (the fourth variant). Thus, we construct a 4-variant circuit. The different copies can share FFs at the synthesis step to reduce the overhead. By careful state assignment, one can ensure that the FFs needed for each variant to function properly are different in at least one FF. This way, even if a FF is corrupted, there are still other copies that can function properly. In Figure 1, we show an example of how the FFs can be shared between the different variants. Assume that we implement the design using 4 FFs. The four FFs denoted by F_1, F_2, F_3, and F_4 are shown in front of each variant. The FFs in white affect the functionality of the variant, while the FFs in gray do not affect the functionality of the variant (although they can keep flipping all the time for obfuscation reasons depending on the application). For instance, the first variant can use F_1, and F_2 shown in white, while F_3, and F_4 do not affect its functionality. However, the second variant is affected by F_1 and F_4. Thus, if F_2 is corrupted, the first variant will not function properly, but the second will. For the circuit to properly function, it is enough to have one correctly functional variant selected. However, depending on the application, we can choose to switch between different variants, or to prefer a variant over the other based on their performances. Thus, the selection function shown in Figure 1 is application dependent as we discuss in Section 4.

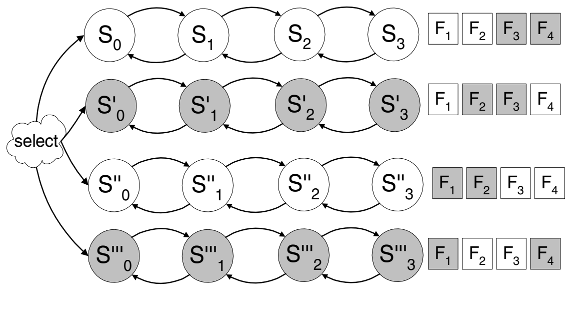

Figure 1: A 4-variant Design

1.2 Paper organization

In the next section, we discuss the background and related work along the lines of FSM and the N-variant concept. In Section 3 we devise an algorithm for efficient integration of the N-variants. Section 4 demonstrates how the variant selection inputs can be managed to facilitate various applications. Experimental results evaluating the overhead of the method on standard benchmarks are presented in Section 5. The proof-of-concept implementation for metering multimedia files is also reported. Section 6 concludes the paper and outlines a number of future research directions.

2. BACKGROUND

We describe the background and related literature on FSM and N-variant systems that has influenced and inspired our work.

2.1 FSM

A FSM is a dynamic discrete system with limited number of states that maps input sequences into output sequences. It can be used to represent a sequential function, e.g., sequential circuits. A FSM is typically defined by a 6-tuple M=(Σ,Δ,Q,q_0,δ,λ), where

- $\Sigma \neq \emptyset$ is a finite set of input alphabets;
- $\Delta \neq \emptyset$ is a finite set of output alphabets;
- Q={q_0,q_1,. . . }$\neq \emptyset$ is a bounded set of states;
- $q_0 \subset Q$ is the set of initial states;
- δ (q, a) is transition function on input a and set $Q \times \Sigma \rightarrow Q$;
- λ (q, a) is output function for input a and set $Q \times \Sigma \rightarrow \Delta$.

The STG is used to represent the state transitions and input/output relations of the FSM; nodes correspond to states and edges define the input/output conditions for state transitions.

FSMs are used for information hiding and watermarking purposes. Oliveira proposed to alter the STG such that a signature is mapped to a spacial topological property in the sequence of states traversed by a sequence of inputs [15]. Yuan and Qu exploited the existence of redundant transitions and manipulate them to hide information inside the FSM without altering the minimized FSM [20]. Alkabani et al. created the first method for remote unique enabling/disabling of ICs by use a combination of physically unclonable functions and FSM state manipulations [1,2]. The lock is interleaved with combinational and sequential transitions and can be used for enabling/disabling of the IC. The unlocking is done by sequence of inputs that can only be generated by knowing the FSM structure.

The classic way for fault-tolerance FSM design is triple modular redundancy (TMR), where three copies of FSM are concurrently run and the correct output value is determined by voting [16]. Even though TMR may be viewed as a basic N-variant design method, we emphasize that the new N-variant methodology is not a simple replication of the FSMs. Instead, the variants are elegantly interwind within the original design. N-variant design has applications that the traditional fault-tolerant design methods do not support.

Our N-variant design algorithm is devised to have a low-overhead. A class of methods that is relevant to our work is state assignment and state minimization of large FSM networks [7, 11]. The concepts and methods used in FSM minimization may help to even lower the overhead of N-variant designs in the future.

2.2 N-variant approach

N-version software development was introduced as early as 1968 [13]. The goal was to achieve fault-tolerance [4, 18]. N-versioning collects N copies of the target software written by *different programmers* and then runs them in parallel, using the redundancy in the results to achieve fault-tolerance. N-versioning is different from N-variant since N-variant systems include multiple copies within the *same design* that appear different. So far, the main usage of N-variants has been for security purposes [6, 10]. The motivating idea is that most systems are homogeneous and the same exploit can attack all instances of the system. The specific machine-level characteristics of the attacked computer is used by the binary exploits, e.g, byte order, calling conventions, program load addresses, etc. If the system is not exactly the same, the exploit would not be able to adjust itself and the attack will be halted.

Achieving a heterogeneous computer system by using randomization was first introduced by Forrest et al [8]. Cox et al. present an architectural framework that systematically uses automated diversity that provides detection and disruption of a large class of attacks [6]. Holland et al. extended the idea of achieving heterogeneity by generating the architecture-dependent parts of kernel and standard C library from machine description [10]. Their approach successfully suppresses the risks of code injection attacks and state corruption attacks.

As we have mentioned in Section 1, N-variant IC design can enable many more tasks than just security and attack resiliency. Perhaps the most interesting applications are in the domain of IP protection and enabling new business models, and for post-silicon optimization and self-adaptations of the ICs. Furthermore, because the N-variants are embedded into the hardware, the overhead is smaller than embedding them into the virtual machine and/or the architecture. The low overhead of the N-variant IC renders it suitable for embedded systems applications.

3. N-VARIANT IC DESIGN

In this section we discuss how the N-variant circuit can be designed. Figure 2 demonstrates the flow of our design. The implementation steps can be summarized as follows:

1. A single copy of the FSM is implemented as a group of FFs connected to the input and output through combinational circuits *logic*$_1$ and *logic*$_2$ respectively, as shown in Figure 2(a).
2. Modify the FSM as follows (Figure 2(b)):
   (a) Select the parameter m representing the number of replications of the FFs in the circuit.
   (b) Partition *logic*$_1$ into *logic*$_{1a}$ and *logic*$_{1b}$ and replicate *logic*$_{1a}$ for m times.
   (c) Partition *logic*$_2$ into *logic*$_{2a}$ and *logic*$_{2b}$ and replicate *logic*$_{1a}$ for m times.
   (d) Design the obfuscation logic and the selection logic to maintain the circuit's correct functionality. The obfuscation logic is implemented at the input side and is used to distinguish between different variants, and to generate dummy values in the unused FFs. The selection logic is used at the output side to select the output of the target variant. Note that when a variant is not selected it does not have to generate correct outputs.

Selection of the number of replications and the way the combinational circuits are partitioned, form the state-space of all possible implementations. Each choice requires a minimal complexity of the obfuscation and selection logics. Note that the choice is also affected by the application targeted by the N-variant design method.

It is clear that the overhead of the implementation varies greatly with the choice of m, the partitioning of the combinational circuit, and the complexity of both the obfuscation and selection logic blocks. The area and power overheads of the methods are affected mainly by m, and by the partitioning of the combinational circuits. The delay is only affected by the selection and obfuscation logic. At one extreme, one can set $m = N$, $logic_{ia} = logic_i$, and $logic_{ib} = \emptyset$, while implementing the selection logic as a multiplexer and discarding the obfuscation logic. This configuration produces the highest area and power overheads, and the lowest delay. At the other extreme, one can set $m = 1$, $logic_{ia} = \emptyset$, $logic_{ib} = logic_i$. This leads to an obfuscated circuit, and produces the lowest area and power overheads, and the highest delays. The potential impact of each of the possible design parameters is as follows.

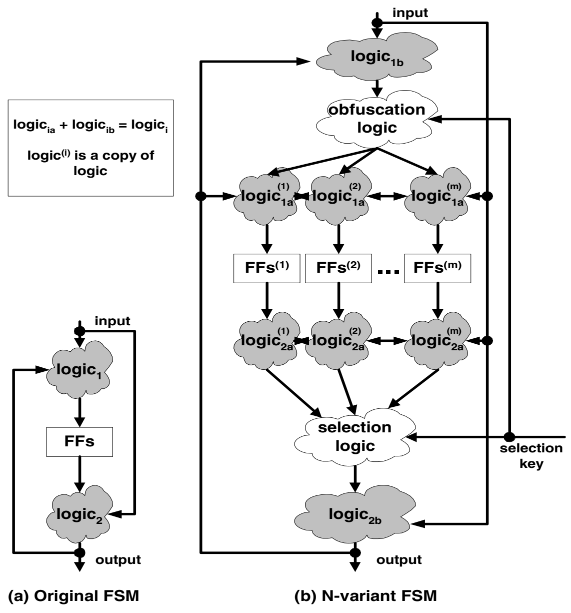

Figure 2: Design of an N-variant circuit.

• *FFs.* The FFs represent the memory for the FSM states, they can be replicated m times where $m \geq 1$; $m = 1$ means the FFs are all shared and this removes a big part of redundancy in our implementation. However, the method can still have multiple variants by manipulating the logic blocks.

• *Logic*$_1$. This circuit component represents the combinational part of the circuit at the input side. It is divided into two parts: $logic_{1a}$ and $logic_{1b}$. $Logic_{1a}$ is to be replicated m times, and $logic_{1b}$ is shared. Increasing the size of $logic_{1a}$ leads to larger area and power overheads. However, it does not affect the delay overhead. Despite the introduction of more area and power overheads, it can be useful for applications with timing constraint.

• *Logic*$_2$. This circuit represents the combinational part of the circuit at the output of the FFs. It is divided into two parts: $logic_{2a}$ and $logic_{2b}$. $Logic_{2a}$ is to be replicated m times, and $logic_{2b}$ is shared. Just like the case in $logic_{1a}$, increasing the size of $logic_{2a}$ leads to larger area and power overheads and can be useful for applications targeting performance improvement and fault-tolerance.

• *Obfuscation logic.* The obfuscation is used to adjust the inputs for different variants. The more FFs are shared between variants, the more complex this part will be. However, in cases where few FFs are shared, this part is used to generate dummy values in the unused FFs. This can be useful for security applications.

• *Selection logic.* This block is responsible for ensuring that the correct output from the target variant is selected. In case of many FF replications, it can be simply implemented as a multiplexer. However, as the number of shared FFs increases, it becomes increasingly important to interleave this block with the obfuscation logic to guarantee that the correct outputs are obtained regardless of the selected variant.

The nature of the original circuit also affects the significance of the overhead. For instance, if the combinational logic is very small in area, power, and delay, even the slightest replications and the simplest obfuscation and selection logic will yield a significant overhead compared to the original one. Note that the area and power overhead of an FSM (representing the control part of a system) is extremely small compared to the overall area and power overhead of the system.

4. APPLICATION-SPECIFIC MANAGEMENT OF THE N-VARIANTS

In this section, we describe a number of management methods that can enable each of the applications described in Section 1. The variant selection contains the following components and structures:

• *Trigger.* A trigger prompts selection of a new variant. It may come from various sources, including the clock signals, an internal counter, a sequence of inputs, or an external signal.

• *Storage.* The memory can be used in case selection of a variant is dictated by a stored key, or in case a counter should save its state in presence of rests. Also, many binary attacks may exploit a fixed memory address. To defend against this attack, all or some parts of the memory needs to by duplicated.

• *Driver.* The driver uploads the new variant in case a trigger is activated. A driver may be as simple as a multiplexer uploading the variant selection key from a stored location, or it may be a driver FSM, e.g., an obfuscated counter.

The above components can be used in various ways to manage the variant selection suited for a particular application. A few examples are as follows.

(i) Hardware IP protection and digital rights management. An IC vendor could configure its chips by using a fixed key specific to each customer. This provides a new mechanism to trace back the ICs in the supply chain. Also, many existing hardware IP protection methods can be readily used in N-variant settings. For example, the sequence of traversal among a few variants or within a variant could be utilized as a watermark or for hiding information [15,20]. As another example, the N-variant structure could be integrated with the unique random number generated on each chip, for fingerprinting or locking the state transitions by the manufacturer [1,2].

(ii) Usage and content metering. The N-variants can be used to enforce licensing agreement for the hardware, software, or content usage. Once the license agreement term is over, the IC would enter a nonfunctional state. The IP rights owner is the only entity that can load a new functional variant if a new license is obtained. A trigger by a clock or an internal counter can be used to save the number of uses or the time constraints. In case a counter is used, the states should be saved in an on-chip nonvolatile memory so the chip's reset would not affect the usage metering. Similar triggers were used for licensing of FPGA IPs [5], where it was noted that the trigger is vulnerable to memory reset/clock reset attacks. One possible countermeasure against this attack is to randomize the counter, so that resetting to zero would not traverse to the zero usage state. Many other counter obfuscation methods are possible. For example, one may initialize the counter by using a PUF, so different configurations will be needed depending on the unique IDs of each chip.

(iii) Security. The system could be made heterogeneous by frequently alternating the FSM variant [10]. For example, in case of exploits that use a fixed memory address, the memory can be duplicated creating a master and a slave copy. Every time a new variant is selected, the master and the slave copies would switch. The slave would copy its content from the master. If an attack exploits an address on the master copy, it would halt once the master copy is altered.

(iv) Post-silicon optimization. Because of the variability in CMOS technology, the variant with the best power or delay performance would be different on each IC. A spectrum of new post-manufacturing optimizations can be enabled by using the test data to determine the variant that has the best performance on each IC. For example, one may design the chips to have the maximum possible frequency, by selecting low feature sizes for the gates. Due to variability, some of the delays would be longer than planned, slowing down a few critical paths.

(v) Self-adaptation. The variants could be strategically uploaded using an offline or online monitoring method that manages the upload based on a certain target. For example, to be resilient against aging, one can equalize the amount of usage of the various components. Offline simulation of randomly selected variants can determine which components are used more in one variant. The variants with the largest difference in their usage patterns may be periodically loaded to equalize the aging effect.

(vi) fault-tolerance. The flexibility and redundancy of the N-variant design can be utilized for providing fault-tolerance. For example, if a FF is shown to be faulty during the test phase, the configurations that do not use this FF will be uploaded. The fault-tolerance can be implemented in more sophisticated ways, by adding a monitoring mechanism. As an example, a BIST structure can be included which periodically tests the circuit for faults due to aging, NBTI, or HCI effects. The correct variants could be managed accordingly.

5. EXPERIMENTAL RESULTS

In this section, we implement the N-variant method described in section 3 on benchmark designs and evaluate its performance. A program is written in C to generate N-variants for each design. The Berkeley SIS tool is used to simulate the results and to estimate the overhead. In the first subsection, we show the experimental results for applying the method on standard MCNC'91 benchmarks. In the second subsection, we demonstrate a DCT example as a proof-of-concept for applying the method to embedded multimedia applications. All the resulting circuits are mapped to the MCNC library.

5.1 Implementation of the N-variant method

Figure 3 shows an implementation of the N-variant method for $m = 2$. Our design targets embedding at least one redundant component for each part, while maintaining low area, power, and delay overheads. We present the results for $m = 2$ and $m = 4$ that translate to doubling and quadrupling the number of FFs respectively. The obfuscation logic is implemented as a block of XOR gates that place different values in the unused FFs for each variant. The selection logic is implemented as a multiplexer block. The number of variants generated by this implementation for a circuit with k FFs is 2^k for $m = 2$, and is 4^k for $m = 4$. Our selection of each parameter of the N-variant design (see Section 3) and the impact of the choice on the target applications are as follows:

• *FFs.* We selected $m = 2$ and $m = 4$. Selecting $m = 2$ ensures that the scheme has at least one spare FF for each variant; $m = 4$ provides more redundancy.

• *Logic*$_1$. We selected *Logic*$_{1a} = \emptyset$, and *logic*$_{1b}$=*logic*$_1$. The choices are made to reduce the area and power overheads. However, fault-tolerance and post-silicon optimization are now only dependent on FF redundancies.

• *Logic*$_2$. We devised the following parameters: *Logic*$_{2a} = \emptyset$, and *logic*$_{2b}$=*logic*$_2$. The selection is made to reduce the area and to reduce the delay and power overhead. Again, fault-tolerance and post-silicon optimization are only dependent on FFs.

• *Obfuscation logic.* For $m = 2$ and $m = 4$, only a half, or a quarter of the FFs are shared respectively. We select the obfuscation function to be a block of XOR gates that generate dummy values in the unused FFs. We limit the maximum number of variants to be 2^k (for $m = 2$) and 4^k (for $m = 4$) to keep the function simple and to constrain the delay overhead. Note that some of the variants can be reserved as trap non-functional FSMs for security and IP protection applications.

• *Selection logic.* This part is implemented as a multiplexer and is kept simple to avoid large delay overheads.

Note that the parameter choices used in one implementation are driven by the designer's target application. The real results may be affected by many factors such as the original shape of the circuit, the gate library used, and different optimizations done on the circuit after modification.

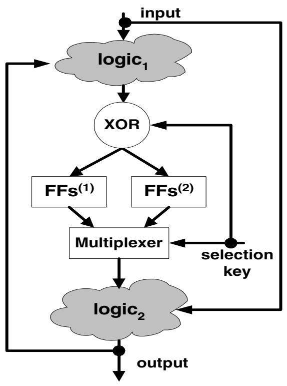

Figure 3: Implementation choice for the experiments.

Tables 1 and 2 show the respective area, power, and delay overheads for $m = 2$ and $m = 4$. The evaluations on the used benchmarks show a maximum area and power overhead of 20% and 19% respectively, while the delay overhead was at most 17% for $m = 2$. The area and power overhead for $m = 4$ goes up to 48% and 62%, and the maximum delay overhead is 21%. It should be noted that the benchmarks with the highest overhead are very small. It is also interesting to note that sometimes the overhead is negative (the circuit is improved) because the resulting circuit maps to fewer gates in the MCNC library. The average area overhead is -0.65% and 12% for $m = 2$ and $m = 4$. The average power overhead increased from -3% for $m = 2$ to 15% when $m = 4$. As expected the average delay overhead did not change by increasing m; it remained 10% in both cases for the evaluated benchmarks. All modifications to the circuits are done on the netlist before mapping. The relative values are more important than the absolute values as the values in the library can be scaled down.

It is worth noting here that the delay overhead is the only important metric in real designs. This is because the FSM which constitutes the control part of the design is typically only a small part of the design, significantly less than 1% of the power/area [9]. Thus, even if the FSM's area or power is doubled, it will not significantly affect the overall design.

5.2 Embedded multimedia application

We report the results of implementing the N-variant method for content usage metering for embedded multimedia applications. The key observation that facilitates ultra-low overhead implementation is that the N-variants do not need to be instrumented in all parts of the design; only the critical parts of the design can be strategically selected and augmented. In our multimedia example, we implement N-variants in the DCT part of the design, the essential component for audio, image, and video signal processing. Figure 4 shows a typical MPEG video compression flow which contains both DCT and IDCT components.

	Original				m=2		m=4	
BM	PI	PO	FFs	Area	Area	%	Area	%
dk16	2	3	5	461	482	4.6	535	16.1
keyb	7	2	5	461	490	6.3	535	16.1
planet	7	19	6	887	914	3.0	975	9.9
planet1	7	19	6	887	914	3.0	975	9.9
pma	8	8	5	346	382	10.4	419	21.1
s1488	8	19	6	880	915	4.0	973	10.6
s208	11	2	5	148	178	20.3	219	48.0
s420	19	2	5	148	151	2.0	191	29.1
s820	18	19	5	429	256	-40.3	296	-31.0
s832	18	19	5	427	271	-36.5	316	-26.0
styr	9	10	5	633	662	4.6	716	13.1
tma	7	6	5	287	318	10.8	362	26.1

Table 1: Area overhead of the N-variants implementation.

	Orig.	$m = 2$	%	$m = 4$	%	$m = 8$	%
Area	821	871	6.1	955	16.3	1117	36.1
Power	2946	3084	4.7	3490	18.5	4198	42.5
Delay	123.8	143	15.5	140	13.1	139.7	12.8

Table 3: The overhead for 2^9-variants ($m = 2$), 2^{18}-variants ($m = 4$), and 2^{36}-variants ($m = 8$) DCT.

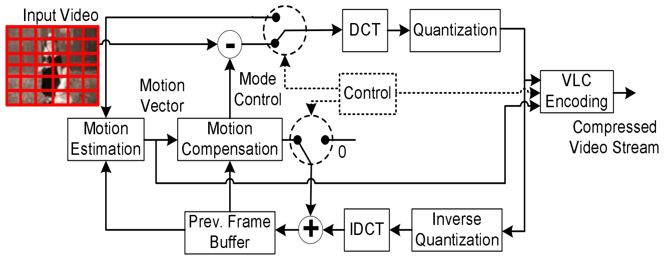

Figure 4: Flow of MPEG video compression flow.

Table 3 shows the results for applying the implementation described above to the DCT control circuit. The original DCT circuit has 9 FFs. We evaluated N-variant implementations with 2^9-variants ($m = 2$), 2^{18}-variants ($m = 4$), and 2^{36}-variants ($m = 8$). The area, power, and delay overheads for $m = 2$ are 6%, 5% and 15.5%, respectively. Changing m and keeping the obfuscation and selection logic the same, has a significant impact on the area and power overheads. For $m = 4$ the area and power overheads increase to 16% and 18%, and for $m = 8$ the area and power overheads reach 36% and 42%. However, the change in the delay overhead is insignificant.

6. CONCLUSION

We introduce N-variant design methodology, a novel design method that creates $N \geq 2$ copies of the same design on one IC. The strength and usefulness of N-variant design was previously shown for systems, software, and architectural components. However, no scheme for generating N-variant ICs was introduced, be-

	Original		**m**=2				**m**=4			
BM	**P**ower	**D**elay	**P**ower	%	**D**elay	%	**P**ower	%	**D**elay	%
dk16	1649.8	97.7	1712	3.8	106.6	9.1	1935.1	17.3	107.6	10.1
keyb	1546.5	61.5	1635.5	5.8	67.6	9.9	1837.3	18.8	68.4	11.2
planet	3103.5	187.9	3156	1.7	219.4	16.8	3471.1	11.8	195.6	4.1
planet1	3103.5	187.9	3156	1.7	219.4	16.8	3471.1	11.8	195.6	4.1
pma	1250.3	61	1309.8	4.8	67.9	11.3	1550.4	24.0	73.4	20.3
s1488	3007.3	134.9	3088.7	2.7	152.7	13.2	3378.3	12.3	144.8	7.3
s208	480.4	25.4	573.6	19.4	26.9	5.9	779.1	62.2	30.9	21.7
s420	480.4	25.4	462.9	-3.6	26.9	5.9	667.1	38.9	30.5	20.1
s820	1423.9	33.7	790.7	-44.5	36	6.8	1007.9	-29.2	31.8	-5.6
s832	1445.1	33.6	855.4	-40.8	36.7	9.2	1089.9	-24.6	36.8	9.5
styr	2170.2	128.2	2272	4.7	144.9	13.0	2470.6	13.8	134.9	5.2
tma	989.3	64.8	1062.1	7.4	71.1	9.7	1287.9	30.2	72.7	12.2

Table 2: Power and delay overheads of the N-variant implementation.

yond simple replications for fault-tolerance. We devise an implementation algorithm that works by extending the FSM such that it would include multiple variants of the same design. We demonstrate how the N-variants enable a number of important applications, including hardware and content IP protection, IP metering, security, design optimization and self-adaptation. We also discuss how the input signals could be managed for each target application. The experimental results on benchmark circuits demonstrate the efficiency of the algorithm in terms of area, power and delay. We also present a proof-of-concept implementation of N-variant approach on MPEG decoders by only manipulating one of its circuit component, namely DCT. Experimental evaluations shows the suitability of the method for embedded systems applications.

The N-variant design approach has a great potential to impact various applications that lay the ground for future research. Potential research directions include implementing the method on ICs, careful development of the digital rights management methods that exploit the N-variants as the underlying mechanisms, finding more efficient implementation methods, devising on-chip monitoring mechanisms so that the N-variants could be automatically used for self-adaptation, and design or implementation of post-silicon optimization methods that exploit the multiplicity of the variants. Last but not least, better identification and classification of possible attacks against the introduced security and protection mechanisms, particularly those for monitoring the vulnerability of the system against the binary attacks must be performed.

Acknowledgment

This work is partly supported by the Defense Advanced Research Projects Agency (DARPA)/MTO Young Faculty Award (YFA), grant number W911NF-07-1-0198, and by the National Science Foundation (NSF), grant number 0716674.

7. REFERENCES

[1] Y. Alkabani and F. Koushanfar. Active hardware metering for intellectual property protection and security. In *USENIX Security*, pages 291–306, 2007.

[2] Y. Alkabani, F. Koushanfar, and M. Potkonjak. Remote activation of ICs for piracy prevention and digital right management. In *ICCAD*, 2007.

[3] Y. Alkabani, T. Massey, F. Koushanfar, and M. Potkonjak. Input vector control for postsilicon leakage current minimization in the presence of manufacturing variability. In *DAC*, 2008.

[4] A. Avizienis. The N-version approach to fault-tolerant software. *IEEE Trans. on Software Engineering*, 11(12):1491–1501, 1985.

[5] N. Couture and K. Kent. Periodic licensing of FPGA based intellectual property. In *FPT*, pages 357–360, 2006.

[6] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems: A secretless framework for security through diversity. In *USENIX Security*, pages 105–120, 2007.

[7] S. Devadas. Approaches to multi-level sequential logic synthesis. In *DAC*, pages 270–276, 1989.

[8] S. Forrest, A. Somayaji, and D. Ackley. Building diverse computer systems. In *hot*OS, page 67, 1997.

[9] J.L. Hennessy and D.A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers, 1996.

[10] D. Holland, A. Lim, and M. Seltzer. An architecture a day keeps the hacker away. *SIGARCH Computer Architecture News*, 33(1):34–41, 2005.

[11] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *DAC*, pages 684–690, 1994.

[12] D. Kirovski and M. Potkonjak. Local watermarks: Methodology and application on behavioral synthesis. *IEEE Trans. on CAD*, 22(9):1277–1283, 2003.

[13] K. Knowlton. A combination hardware-software debugging system. *IEEE Trans. on Computers*, 17(1):81–86, 1968.

[14] F. Koushanfar and M. Potkonjak. CAD-based security, cryptography, and digital rights management. In *DAC*, pages 268–269, 2007.

[15] A. Oliveira. Techniques for the creation of digital watermarks in sequential circuit designs. *IEEE Trans. on CAD*, 20(9):1101–1117, 2001.

[16] S. Pontarelli, G. Cardarilli, A. Malvoni, M. Ottavi, M. Re, and A. Salsano. System-on-chip oriented fault-tolerant sequential systemsimplementation methodology. In *DFT*, pages 455–460, 2001.

[17] G. Qu and M. Potkonjak. *Intellectual Property Protection in VLSI Design*. Kluwer, 2003.

[18] B. Randell. System structure for software fault tolerance. *Software Engineering*, 1(2):221–232, 1975.

[19] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady. Security in embedded systems: Design challenges. *ACM Trans. on Embedded Computing Systems*, 3(3):461–491, 2004.

[20] L. Yuan and G. Qu. Information hiding in finite state machine. In *Information Hiding*, pages 340–354, 2004.