

# Information Hiding in Finite State Machine

Lin Yuan and Gang Qu

Department of Electrical and Computer Engineering  
and Institute for Advanced Computer Studies  
University of Maryland, College Park, MD 20742  
{yuanl, gangqu@eng.umd.edu}

**Abstract.** In this paper, we consider how to hide information into finite state machine (FSM), one of the popular computation models. The key advantage of hiding information in FSM is that the hidden information becomes inexpensive to retrieve, yet still hard to remove or delete. This is due to the fact that verifying certain FSM properties is easy, but changing them requires efforts equivalent to redoing all the design and implementation stages after FSM synthesis.

We first observe that not all the FSM specifications (or transitions) are needed during the state minimization phase. We then develop a Boolean Satisfiability (SAT) based algorithm to discover, for a given minimized FSM, a maximal set of redundant specifications. Manipulating such redundancy enables us to hide information into the FSM without changing the given minimized FSM. Moreover, when the original FSM does not possess sufficient redundancy to accommodate the information to be embedded, we propose a state duplication technique to introduce additional redundancy. We analyze these methods in terms of correctness, capacity of hiding data, overhead, and robustness against possible attacks. We take sequential circuit design benchmarks, which adopt the FSM model, as the simulation testbed to demonstrate the strength of the proposed information hiding techniques.

## 1 Introduction

Finite state machine (FSM) is a powerful computation model. It consists of a finite set of states, a start state, an input alphabet, and a transition function that defines the next state based on the current state and input symbols. FSM may also have outputs associated with the transition. The outputs are functions of the current state and/or input symbols. Figure 1 is the standard state transition graph representation for an FSM with eight states. Each transition is represented by a weighted directed edge. For example, the edge from state 4 to state 1 labeled 0/1 corresponds to the fact that on input 0, there is a transition from state 4 to state 1 that produces an output 1.

FSM is the core of modern computability theory (for example, the Turing machine), formal languages and automata theory. It has also found numerous applications such as hardware verification and natural language processing. In this paper, we study the problem of how to hide information into an FSM.

Because FSM design and specification is normally the starting point for most of these applications, the information embedded in this early stage will be inherited throughout the implementation and hence will be robust.

Comparing to the existing information hiding or watermarking practice, FSM information hiding is different from traditional multimedia watermarking and shares a lot of similarity with the VLSI design intellectual property (IP) protection in the sense that the stego-FSM needs to be functionally equivalent to the original FSM. The constraint-based watermarking approach, state-of-the-art IP protection technique [7,14,15], embeds IP owner's digital signature into the design as additional design constraints such that the design will become rather unique. These embedded constraints can be revealed later for the proof of authorship. The correct functionality of the watermarked IP is guaranteed as none of the original design constraints will be altered.

The constraint-based watermarking technique is applicable to FSM information hiding, where the constraints are the transitions among the states. It is possible to add new edges in the state transition graph to hide information, but this will never be as easy as adding edges to a graph for the graph coloring problem [14] due to the following reasons. First, it is non-trivial to alter or add transitions while still maintain the FSM's functionality. Second, the watermarked FSM needs to be synthesized and the synthesis process (the FSM state minimization in particular) could remove the watermark. Finally, watermarking's impact to later development and implementation is hard to control and may be unacceptably high. For instance, the design overhead (in terms of performance degradation, increased area, power, and design cost) in VLSI IP watermarking is inevitable and the pseudo-randomness of the signature-based constraints makes these overhead unpredictable. Lach et al. [7,8,9] watermark FPGA by hiding information in unused LUTs. They experiment resource overhead from 0.005% to 33.86% and timing overhead from -25.93% to 11.95% for various techniques. Oliveira [11] proposes a technique to watermark sequential circuit designs by changing topology of FSMs. Their area and delay overhead can be negligible for large designs (due to the small size of the signature), but are as high as 2747% and 273%, respectively, for small designs.

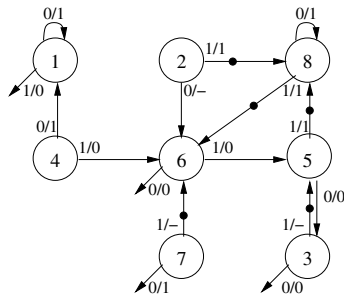
In this paper, we challenge the fundamental assumption in constraint-based watermarking – “original design constraints cannot be touched in order to keep the correct functionality” – by manipulating ‘redundant’ original constraints to hide information. In the first approach, we introduce the concept of ‘redundant’ original constraints in FSM and propose a SAT based approach to identify the maximal set of redundant constraints in Section 2. In Section 3, we present our second approach, a state duplication technique, to create redundancy in the minimized FSM for information hiding. In Section 4, we empirically demonstrate the rich redundancy in the original FSM specification and show the state duplication technique's impact to the design quality on sequential circuit benchmarks. Section 5 concludes the paper.

## 2 Finding Redundant Specifications in FSM State Minimization

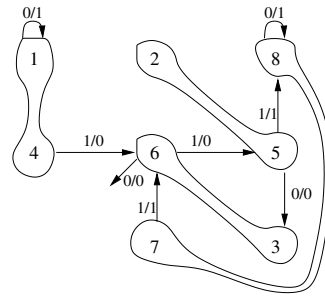
In constraint-based watermarking techniques, information is hidden in the additional constraints enforced by the designer. The original specifications of design is untouched in order to keep the correct functionality. However, we observe that not all the original specifications or constraints are necessary for achieving the design solution. We can manipulate some of the ‘redundant’ constraints to embed secret information. In this section, we will first show by a motivational example in FSM minimization the existence of redundant constraints. And then, in the framework of FSM state minimization, we formally define the problem of finding redundant constraints for information hiding. At last, we show that it can be converted to the problem of finding a truth assignment to a SAT formula with maximal number of 1s.

### 2.1 Motivational Example

Consider an incompletely specified finite state machine (FSM) with eight states in Figure 1. Each node in the graph represents one state. Each (directed) edge indicates a state transition and the attributes carried by the edge specify the input/output associated with the transition. For example, the edge from node 4 to node 6 with “1/0” means that on input “1”, the system moves from state 4 to state 6 and outputs “0”. An edge without ending state represent the case when the next state of that transition is a *don't care*.



**Fig. 1.** State transition graph for the FSM.



**Fig. 2.** State transition graph for the minimized FSM.  $A = \{1,4\}$ ,  $B = \{2,5\}$ ,  $C = \{3,6\}$ , and  $D = \{7,8\}$ .

The state minimization problem seeks to find another FSM which (1) always gives the same output as the original FSM whenever it is specified, and (2) the number of states is minimized. The FSM in Figure 1 can be minimized to one with only four states, and there are two solutions:  $\{\{1, 4\}, \{2, 5\}, \{3, 6\}, \{7, 8\}\}$ ,

and  $\{\{1, 4, 7\}, \{2, 5\}, \{3, 6\}, \{8\}\}$ . The state transition graph for the first solution is shown in Figure 2. For example, if we start with state 5 in Figure 1, on input string “011”, the output will be “0-1”. In the minimized FSM in Figure 2, state 5 corresponds to state B, and if we start with this state, on the same input string “011”, we have output “001” that differs from “0-1” only on the second bit which is not specified in the original FSM.

Surprisingly, if we keep all the output values and only five transitions (the edges with a dot) in Figure 1, the solution remains the same. In another word, this implies that these conditions are sufficient to obtain the above solution(s). The other transitions specified in the original FSM are ‘*redundant*’ constraints in the sense that their absence (i.e., changing the next states of these transitions from *specific states* to *don’t care states*) will have no impact on the final state minimization solutions.

We can leverage such redundant constraints to hide information in the FSM. Since the existence of any redundant constraint will not change the solution, we can embed one bit information on each redundant constraint by either specifying its next state as in the original FSM or replacing its next state by a don’t care. Apparently, the length of the hidden information depends on the number of redundant constraints and the problem remains is whether we can find all the redundant constraints systematically.

## 2.2 Definitions and Problem Formulations

A finite state machine(FSM) is defined as a 6-tuple  $\langle I, S, \delta, S_0, O, \lambda \rangle$  where:

- $I$  is the input alphabet;
- $S$  is the set of states;
- $\delta : S \times I \rightarrow S$  is the next-state function;
- $S_0 \subseteq S$  is the set of initial states;
- $O$  is the output alphabet;
- $\lambda : S \times I \rightarrow O$  is the output function;

Finding an equivalent FSM with minimal number of states is generally referred as *state minimization* or *state reduction*(SR) problem. State minimization is an effective approach in logic synthesis to optimize sequential circuit design in terms of area and power. Our purpose is to identify and hide information in the redundant constraints of FSM such that the state minimization solution remains the same.

Given an FSM and a solution to the state minimization problem, we define a transition in the original FSM **redundant** if the given solution can still be obtained after we replace the next state of this transition by a don’t care (we call this the *removal* of this transition). A *maximal redundant set* (MRS) is a set of redundant transitions that can be removed without affecting the given state minimization solution, but removing one more transition will not preserve this solution.

Finding the MRS in FSM is non-trivial. First of all, the state minimization in incompletely specified FSM is NP-complete. The solution space grows exponentially large in the size of the FSM; removing a transition can make other

transitions indispensable to achieve the same minimized FSM. Second, we can convert the problem of finding MRS to a MaxONEs SAT problem which is defined as: finding a SAT solution with the maximum number of variables assigned one. MaxONEs SAT is also NP-complete [2]. We will show this formulation in the following text.

### 2.3 Finding Maximal Redundant Set

Our approach takes the original and minimized FSM as input. By comparing them, it identifies all the possible redundant constraints in the original FSM. To extract the maximal set of them, it assigns a boolean variable to each of these constraints and generate a Boolean SAT formula. The variables assigned to be 1 in the formula will be redundant; the maximal number of “1” variables correspond to the MRS.

Figure 3 depicts the state transition table, another representation of FSM, for the same FSM given in Figure 1. Each table entry specifies the next state and the output given the current state (which row this entry is in) and the input (which column this entry is in). For each entry with a specific next state (i.e., not don’t care), we introduce a Boolean variable  $x_i$  and stipulate that  $x_i = 1$  means the entry is redundant. The clauses in the SAT formula are created as follows.

	In=0	In=1	In=0	In=1
1	1 ( $x_1$ )	-	1	0
2	6 ( $x_2$ )	8 ( $x_3$ )	-	1
3	-	5 ( $x_4$ )	0	-
4	1 ( $x_5$ )	6 ( $x_6$ )	1	0
5	3 ( $x_7$ )	8 ( $x_8$ )	0	1
6	-	5 ( $x_9$ )	0	0
7	-	6 ( $x_{10}$ )	1	-
8	8 ( $x_{11}$ )	6 ( $x_{12}$ )	1	1
	Next State		Output	

**Fig. 3.** State Transition Table for the FSM.

First, we compute all the compatible sets of the given FSM. Recall that i) two states are *compatible* if they have the same output values whenever they are both specified and their next states are also compatible whenever they are both specified; and ii) a *compatible set* is a set of states that are compatible pairwise. The essence of the state minimization problem is to include all the states using the minimal number of compatible sets.

Next, we examine each pair of states that are not compatible according to the given state minimization solution to make sure that we include sufficient constraints in the FSM specification to distinguish them. If there exists some input value on which the two states output different values (e.g., states  $s_1$  and

$s_2$  on input  $i = 1$ ), then they are not compatible and no information on their next states is required. If, however, when the two states, for every input value, either have the same output value or at least one has a don't care as its output, then we need the information on their next states to distinguish them. Take states  $s_2$  and  $s_3$  for instance, the only way to distinguish them is to make both transitions  $x_3$  and  $x_4$  non-redundant, which can be done by including the expression  $x'_3x'_4$  into the SAT formula.

A more complicated example is the pair of states  $s_2$  and  $s_8$ . Their respective next states  $s_6$  and  $s_8$  are not compatible and we need the presence of both transitions  $x_2$  and  $x_{11}$  or  $x_3$  and  $x_{12}$  to distinguish them. This can be conveniently enforced by the following Boolean expression (in CNF format)

$$\begin{aligned} & x'_2x'_{11} + x'_3x'_{12} \\ (DeMorgan) &= ((x_2 + x_{11})(x_3 + x_{12}))' \\ (Distributive) &= (x_2x_3 + x_2x_{12} + x_{11}x_3 + x_{11}x_{12})' \\ (DeMorgan) &= (x'_2 + x'_3)(x'_2 + x'_{12})(x'_{11} + x'_3)(x'_{11} + x'_{12}) \end{aligned}$$

As a result, for the FSM in Figure 1 and its state minimization solution in Figure 2, we have the following SAT instance:

$$\mathcal{F} = x'_3x'_4x'_{10}(x'_2 + x'_3)(x'_3 + x'_{11})(x'_2 + x'_{12})(x'_{11} + x'_{12})x'_8$$

For variables that do not appear in this formula (such as  $x_1$ ), we can safely assume that their corresponding transitions are redundant. Clearly, finding the MRS becomes equivalent to finding a solution to the corresponding SAT formula such that the number of variables assigned to be '1' is maximized. An exact algorithm is to formulate it as an integer linear programming (ILP) problem which has been discussed in [2]. Practically, one can also simply solve the SAT formula multiple times, each solution represents one MRS, and pick the one with the maximal 1s.

## 2.4 Analysis

Unlike the previous constraint based watermarking approaches, our information hiding technique conceals information in the original redundant constraints. We will now analyze this method in terms of correctness, information hiding capacity, overhead and robustness.

*Correctness* In our algorithm, we identify the redundant constraints by comparing two states that are uncompatible in the minimized FSM; we keep all the necessary constraints to distinguish them. Therefore, in any state minimization solution, these two states must be in different compatible sets. On the other hand, if two states are reduced to one state in minimized FSM, they are also compatible in the original FSM, because no other constraints have been modified to distinct them. As a result, the same minimized FSM can still be achieved even with the removal of redundant constraints.

*Information hiding capacity* For a pair of incompatible states in minimized FSM, there is at most one pair of transition constraints under one input symbol needed in original FSM to separate them; the rest of transitions under other input symbols (if not *don't cares*) can be used to hide information. In a FSM with  $l$  input bits (i.e.,  $2^l$  symbols) at each state, a pair of incompatible states can have up to  $2^l$  of redundant constraints. Suppose there are  $k$  reduced states in minimized FSM corresponding to a compatible set of  $n_1 \dots n_k$  states in the original FSM, we can embed up to

$$2^l \cdot \sum_{1 \leq i < j \leq k} n_i n_j$$

bits of information in such FSM.

*Overhead* Since no additional constraints are attached to original specification, the optimal solution will not be affected. As we have seen in the FSM minimization example, we can always achieve the same minimized solution even if we embed information by removing some of the redundant constraints.

*Robustness* Information is embedded in the original redundant constraints, so there is no such way of “removing” them. On the other hand, the rest of constraints are necessary; removal of them will definitely affect the design solution. Similarly, in order to change or fake the watermark, the attacker has to know the original FSM transition graph. However, it is private to designers.

### 3 Creating Redundancy in Minimized FSM

The previous technique discovers a maximal set of redundant transitions with respect to a minimized FSM. Information can be hidden then by the way how we manipulate these redundant transitions. Therefore, its information hiding capacity is limited by the size of the maximal set of redundant transitions. In this section, we overcome this limitation by a state duplication technique which creates redundancy in the minimized FSM to facilitate information hiding.

#### 3.1 An Illustrative Example

We first illustrate the idea of state duplication by the following example Figure 4(a) shows the state transition graph of a 2-input 2-output FSM with five states  $\{S1, S2, S3, S4, S5\}$ . The FSM has already been minimized. We reconstruct this FSM by introducing a new state  $S6$  as shown in Figure 4(b). One can easily verify that these two STGs are functionally equivalent. In fact, state  $S6$  is an equivalent state of  $S1$ . The 3-bit number next to each state is the code assigned to that state by a state encoding tool.

Considering the continued development from the two STGs in Figure 4 (a) and (b), we observe the following:

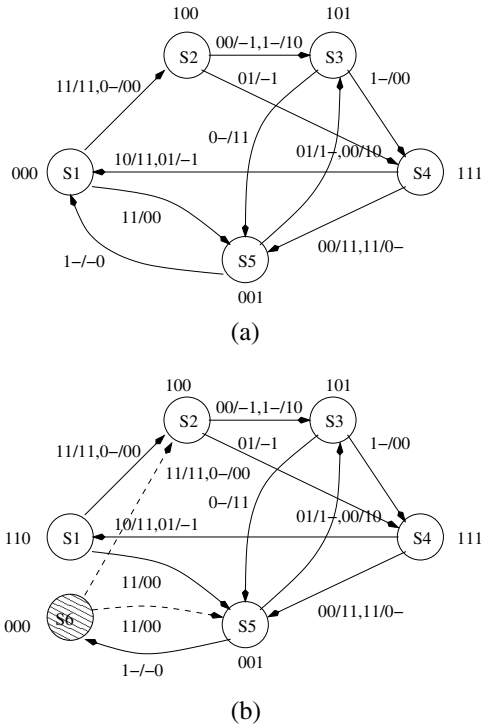


Fig. 4. A 5-state FSM and a functionally equivalent 6-state FSM.

- they result in functionally equivalent designs.
- the quality of the two designs in general has little difference.
- information can be hidden by the way we introduce state  $S6$ , namely, which state we want to duplicate, how we will duplicate it, and how we assign codes to the newly duplicated state.

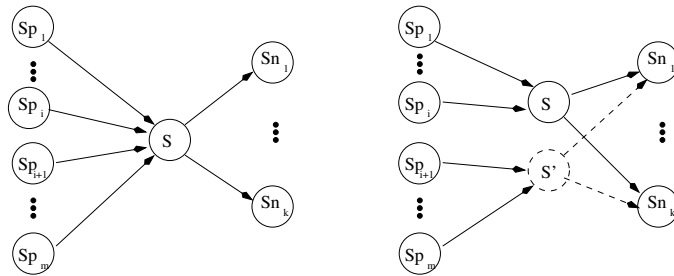
The first observation guarantees the correctness and the second one reveals that design overhead due to watermarking or information hiding is limited. Finally, the last one implies the flexibility of this technique in hiding information. In the rest of this section, we formally discuss these issues.

### 3.2 State Duplication

We consider a minimized (and encoded) FSM with  $n$  states and an  $m$ -bit message. Our goal is to embed the message into the FSM. The proposed state duplication method consists of three steps 1) select a state for duplication. 2) duplicate the selected state. 3) encode the duplicated state. Information can be hidden in each of these three steps and they will be repeated until the entire message is embedded.

Recall that two states  $S$  and  $S'$  are equivalent if and only if on every possible input symbol, they produce the same output and move to the same state or equivalent states. Equivalent states can be collapsed to one single state to simplify the FSM without changing its functionality. This is the basis for FSM state minimization, which is important for later FSM implementation because fewer states normally lead to state encoding with smaller code size. An FSM is minimized if it does not contain any equivalent states. In the state duplication approach, we reverse the state minimization process by introducing equivalent states into the minimized FSM. This is guided by the message to be embedded and creates redundancy in the FSM.

Figure 5 illustrates the basic idea of state duplication. We see that a new state,  $S'$ , is added as a duplicate of state  $S$  as follows:  $S'$  goes to the same next state under the same transition condition as state  $S$ ; the transitions from other states to state  $S$  in the original STG will be split such that some of them still go to state  $S$  while the rests go to the new state  $S'$ .



**Fig. 5.** A simple way to duplicate a state in an FSM.

*Selection state for duplication* Technically, every state can be duplicated. If we restrict to deterministic FSMs, only states that are reachable from the starting state through different sequences of transitions can be considered as candidates for duplication. To see this, we assume that the FSM in Figure 4 (a) has state  $S1$  as its starting state. Clearly state  $S3$  is duplicable because there are two paths  $S1 - S2 - S3$  and  $S1 - S5 - S3$  to reach  $S3$  from  $S1$ . State  $S2$  can also be duplicated despite of the fact that it is only reachable from  $S1$  directly. This is because the direct edge from  $S1$  to  $S2$  actually represents three transitions: moving to state  $S2$  from  $S1$  when the inputs are 00, 01, or 10. However, if this edge only carries the label “10/11”, then state  $S2$  cannot be duplicated.

This suggests us that we can simply sort all the candidate states by, for example, their codes and then select the one based on the message to be hidden. To limit the impact of state duplication to the performance of later development, we propose to consider only the states with the following properties if the candidate pool is rich: (1) states with more than one previous states, to simplify the duplication process; (2) states with large average Hamming distance (the Hamming distance between two states is the number of different bits they have in their

codes) from all their previous states; (3) states with fewer next states. The last two properties in general will help us to find a better code for the duplicated state and/or the state being duplicated.

*Duplicating the selected state* As we have described in Figure 5, the duplicated state  $S'$  needs to have the same next states as the original state  $S$  to maintain the functional correctness. However, the previous states of state  $S$  can go to either  $S$  or  $S'$ . This flexibility gives us the opportunity to hide information. For example, a simple watermark encoding scheme can be defined as: *for each transition from a previous state  $S_p$  of  $S$  to  $S$ , define its next state to be  $S$  to embed a bit 0 and choose its next state to be  $S'$  to embed a bit 1.*

*Encoding the duplicated state* When the original FSM is encoded, we need to give the duplicated state a code too. Suppose that the FSM has  $n$  states and each state has a  $k$ -bit code, where  $k \geq \lceil \log_2 n \rceil$ . The newly introduced state can take any one from the  $2^k - n$  unused codes as its code and this code selection can again embed information.

Finally, we mention that multiple states can be duplicated by repeating this process if the size of the message is large.

### 3.3 Analysis

We now analyze the proposed state duplication information hiding technique.

*Correctness* Clearly, the stego-FSM has the same functionality as the original FSM because we only introduce states that are equivalent to existing states. Further development from the stego-FSM rather than the original FSM guarantees the functional correctness.

*Information hiding capacity* Suppose that the original FSM has  $n$  states and is encoded with  $k$  bits, where  $k \geq \lceil \log_2 n \rceil$ . To select a state for duplication, we can hide  $\lfloor \log_2 n \rfloor$  bits of information; to duplicate a selected state with  $p$  previous states, we can hide  $p$  bits of information; to assign the duplicated state a new  $k$ -bit code, we can hide  $\lfloor \log_2(2^k - n) \rfloor$  bits of information. Let  $l$  be the number of input bits, then there are  $2^l$  transitions from each state. The average number of previous states (count duplicates if there are multiple transition from the same previous state) a state has is  $2^l$ . Furthermore, when minimal code length encoding is assumed, we have  $k = \lceil \log_2 n \rceil$ . The average number of bits being embedded in the final encoding step can be estimated as follows:

$$\begin{aligned} \sum_{n=2^{k-1}+1}^{2^k-1} \frac{1}{2^{k-1}} \lfloor \log_2(2^k - n) \rfloor &= \frac{1}{2^{k-1}} \sum_{n=1}^{2^{k-1}-1} \lfloor \log_2 n \rfloor \\ &\approx \frac{1}{2^{k-1}} \log_2(2^{k-1} - 1)! \\ &\approx \frac{1}{2^{k-1}} \log_2(\sqrt{2\pi}(2^{k-1})^{2^{k-1}-\frac{1}{2}} \cdot e^{-2^{k-1}}) \\ &\approx k - 2 \end{aligned}$$

In sum, duplicating one state can hide approximately  $(2k + 2^l - 2)$  bits of information, where  $k = \lceil \log_2 n \rceil$  is the length of the (minimal length) encoding scheme and  $l$  is the number of input bits. This number will be multiplied when we duplicate more than one state.

*Overhead* The general goal of state minimization is to reduce the number of states such that the encoding length (or hardware implementation of the FSM) is minimized. From this point of view, assuming the minimal length encoding scheme is applied, we conclude that the state duplication technique will not introduce any overhead as long as we keep the number of duplicated states less than  $2^{\lceil \log_2 n \rceil} - n$ . The impact of duplicated states to other design and implementation objectives are hard to analyze before we have the final design. In next section, we consider a large set of sequential circuit design benchmark to demonstrate this impact in terms of area and power consumption.

*Robustness* The robustness of state duplication watermarking approach relies on the fact that FSM design and synthesis occurs at the early stage of the underlying application. Given a synthesized FSM (after state minimization and state encoding), the possible attacks include: 1) Removing the hidden information by identifying and deleting duplicated states in the STG. 2) Tampering the watermark by duplicating additional states. In the first case, removing or changing duplicated states (and thus delete or alter the hidden information) will affect the synthesis solution and the following design implementation stages, which eventually result in re-sign. In the second case, the attacker can only infringe a small part of watermark, if possible (e.g., changing the previous states of duplicated states); most of the hidden information will remain intact. To tamper more hidden information, the attacker has to duplicate a large number of states, which is not always feasible and will cause serious design quality degradation.

*Detectability* The easy identification of duplicated states provides an inexpensive mechanism for revealing the hidden information, which is in general referred as copy detection problem and considered as a problem harder than watermark embedding [13,15].

## 4 Experimental Results

In this section, we will first show how many redundant constraints for state minimization are there in each FSM benchmark. Next, for a minimized FSM, we demonstrate how much redundancy we can create to hide information via state duplication. With the knowledge of these redundant information, we evaluate the possible impact on FSM design by hiding additional information in these constraints. In experiment, we use the standard KISS format as representation of FSMs from MCNC benchmark suite [17]. The FSM minimization tool we use is *stamina* from the logic synthesis design package SIS [18]. And we use a power-driven state encoding algorithm *pow3* [3] to encode the FSM.

Table 1 reports the number of next-state transitions in the original FSMs that are ‘redundant’ for state minimization. We first extract the redundant state transitions and generate a SAT formula in the way as we explained in section 2. Solving the MaxONES SAT via a ILP solver CPLEX [16], we find the maximal number of redundant transitions. The fourth column in the table lists the number of constraints in original FSM. The maximized redundant constraints and the redundancy ratio are given in the fifth and sixth columns. One can observe that 5 to 176 state transitions are redundant for state minimization in the FSMs, which accounts for 20% to 100% of the original constraints in the FSMs. Interestingly, in 5 of these 16 benchmarks, all of the original next-state transitions are redundant for state minimization. This is because either the original FSM is reduced to a single-state machine or every pair of incompatible states in the original FSM can be distinguished purely by the different output bits. This redundancy provides us with a large space to hide information in the transitions; on the other hand, it ensures us that there will be no design overhead caused by embedding additional information. In the last two columns in the table, we use a SAT solver *zchaff* [10] to solve the SAT formula multiple times and choose among random solutions the one with maximal number of ones. Reported data show, for some benchmarks, the redundancy ratio obtained are very close to the maximum one. This tells us that in the case where the SAT formulas are too large for ILP solvers, we can use a SAT solver to find a random solution and still extract a considerable amount of redundancy.

**Table 1.** Number of redundant next-state constraints for FSM state minimization.

benchmark	states	input bits	orig. constr.	max. redundant		rand. redundant	
				constr.	ratio	constr.	ratio
donfile	24	2	96	96	100%	96	100%
ex2	19	2	72	27	38%	15	21%
ex3	10	2	36	19	53%	10	28%
ex5	9	2	32	19	59%	9	28%
ex7	10	2	36	20	56%	14	39%
example	6	2	24	24	100%	24	100%
example2	7	2	28	28	100%	28	100%
lion9	9	2	25	12	48%	12	48%
modulo12	12	1	12	12	100%	12	100%
s27	6	4	96	88	92%	85	89%
s8	5	4	80	80	100%	80	100%
train11	11	2	25	5	20%	3	12%
opus	10	5	176	176	100%	176	100%
beecount	7	3	51	36	71%	34	67%
bbara	10	4	160	36	23%	33	21%
mark1	15	5	240	129	54%	128	53%

**Table 2.** Adding maximum number of redundant states and resulted design overhead

Circuit	regs.	states	add. states	orig. area	incr	orig. power	incr
lin2	3	5	<b>3</b>	43616	12%	280.4	10%
mex3	3	5	<b>3</b>	46400	12%	314	-2%
ex5	4	9	<b>7</b>	70528	29%	405.2	44%
lion9	4	9	<b>7</b>	38976	62%	178.3	93%
ex7	4	10	<b>6</b>	78416	1%	405.8	-4%
train11	4	11	<b>5</b>	47792	17%	212.3	-3%
mmark1	4	12	<b>4</b>	94656	8%	280.7	3%
dk512	4	15	<b>1</b>	79344	2%	430.1	-5%
s1	5	20	<b>12</b>	321088	-1%	1388.7	-8%
ex1	5	20	<b>12</b>	234784	25%	744.9	21%
dk16	5	27	<b>5</b>	282112	-2%	1547.3	3%
styr	5	30	<b>2</b>	407856	0%	1347.6	1%
s510	6	47	<b>17</b>	302064	19%	923.1	44%
planet	6	48	<b>16</b>	504832	2%	2042.1	11%
				Avg. incr.	13%	Avg. incr.	15%

Next, we demonstrate that we can create redundancy by adding redundant states in the state encoding stage when there are not enough redundant constraints to hide information. We also show that this come at the costs of design overhead. We run experiments on 14 MCNC benchmarks, some of which are state minimized. To measure the design quality change, we map these FSMs after state encoding to sequential circuits using the SIS library. We then compare the design quality before and after embedding information in terms of area and power. In Table 2 we first create maximum number of redundant states in each benchmark. For simplicity, we constrain the number of total states to be less than  $2^k$  such that the encoding bits remain minimal. Column 4 lists the number states we added. After adding these states, the FSM can still be encoded using the same number of state bits and there are no space to add more states. In this case, the design overhead is considerable ranging from -2% to 62% in area and -8% to 93% in power. Note that, once a redundant state is added, we can hide information in encoding of the duplicate state, and the partition of its previous states transitions as well. In a STG, where each state has multiple previous state, this means a huge space for information hiding. Thus we consider reducing the number of redundant states added. This can greatly reduce the design overhead as shown in Table 3. In this Table, for each benchmark, we change the number of redundant states added and the way to partition its previous states. We selectively duplicate states in the way mentioned in section 3 and report the results with the least design overhead. The average area increase drops from 13% in Table 2 to 1.3% and interestingly, the average power increase reduces from 15% in Table 2 to -9.4%. This means instead of increasing, the circuits now consume less power with the redundant states in FSM. The main reason of this

is because by adding redundant states, we change the topology the FSM while maintaining its functionality such that the state encoding on these FSMs could give a smaller total switching activity and this eventually leads to the dynamic power reduction in sequential circuits.

**Table 3.** Adding less redundant state and the reduced design overhead

Circuit	regs.	states	added states	area incr	power incr
example	3	5	<b>3</b>	2.1%	2.6%
ex3	3	5	<b>3</b>	8%	-10%
ex5	4	9	<b>2</b>	13.8%	-32.9%
lion9	4	9	<b>2</b>	16.7%	-7.1%
ex7	4	10	<b>1</b>	-10.7%	-29.1%
train11	4	11	<b>1</b>	2.9%	-19%
mark1	4	12	<b>1</b>	5.4%	12.9%
dk512	4	15	<b>1</b>	2.3%	-5.1%
s1	5	20	<b>1</b>	-2.3%	-12.9%
ex1	5	20	<b>2</b>	-8.9%	-13.6%
dk16	5	27	<b>2</b>	-9.7%	-13.3%
styr	5	30	<b>2</b>	3.3%	0.4%
s510	6	47	<b>1</b>	-6.3%	-6.9%
planet	6	48	<b>1</b>	1.8%	1.9%
Average increase				1.3%	-9.4%

## 5 Conclusions

We study the information hiding problem in the context of finite state machine. It is an important problem because of the numerous applications of FSM. Hiding information in FSM provides a unique feature that combines robustness and detectability. We analyze the redundancy naturally in the FSM specification and develop a state duplication based method to introduce additional redundancy for a minimized FSM. We then discuss how to leverage such redundancy to hide information. Simulation on benchmark sequential circuit design demonstrates the correctness and low-cost of the proposed methods.

## References

1. A. Adelsbach, B. Pfitzmann, and A. Sadeghi. "Proving Ownership of Digital Content". *The 3rd International Information Hiding Workshop*, pp. 126-141, September 1999.
2. F. Aloul, A. Ramani, I. L. Markov, K. A. Sakallah, "Generic ILP versus 0-1 Specialized ILP", *IEEE/ACM International Conference on Computer Aided Design*, pp. 450-457, June 2002.

3. L. Benini and G. D. Micheli, "State Assignment for Low Power Dissipation," *IEEE Journal of Solid-State Circuits*, Vol.30, pp.258-268, March 1995.
4. S. Craver. "Zero Knowledge Watermark Detection". *The 3rd International Information Hiding Workshop*, pp. 102-115, September 1999.
5. F. Hartung and B. Girod. "Fast Public-Key Watermarking of Compressed Video". *IEEE International Conference on Image Processing*, pp. 528-531, October 1997.
6. A.B. Kahng, et al. "Watermarking Techniques for Intellectual Property Protection". *35th Design Automation Conference Proceedings*, pp. 776-781, 1998.
7. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. "Fingerprinting Digital Circuits on Programmable Hardware". *The 2nd International Information Hiding Workshop*, pp. 16-31, April 1998.
8. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. "Signature Hiding Techniques for FPGA Intellectual Property Protection", *IEEE/ACM International Conference on Computer Aided Design*, pp. 186-189, November 1998.
9. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. "Robust FPGA Intellectual Property Protection Through Multiply Small Watermarks", *36th ACM/IEEE Design Automation Conference Proceedings*, pp. 831-836, June 1999.
10. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver", *The 38th ACM/IEEE Design Automation Conference*, pp. 530-535, June 2001
11. A.L. Oliveira. "Robust Techniques for Watermarking Sequential Circuit Designs", *36th ACM/IEEE Design Automation Conference Proceedings*, pp. 837-842, June 1999.
12. B. Pfitzmann. "Information Hiding Terminology", *The 1st International Information Hiding Workshop*, pp. 347-350, May 1996.
13. G. Qu. "Keyless Public Watermarking for Intellectual Property Authentication". *The 4th International Information Hiding Workshop*, pp. 96-111, April 2001.
14. G. Qu and M. Potkonjak. "Hiding Signatures in Graph Coloring Solutions". *The 3rd International Information Hiding Workshop*, pp. 391-408, September 1999.
15. G. Qu and M. Potkonjak, "Intellectual Property Protection in VLSI Designs: Theory and Practice", Kluwer Academic Publishers, January 2003.
16. ILOG Inc. "ILOG AMPL CPLEX System Version 8.0 Use Guide", 2002
17. Saeyang Yang, "Synthesis and Optimization Benchmarks User Guide", 2002, <ftp://menc.menc.org>.
18. E. Sentovich, et al., "SIS: A System for Sequential Circuit Synthesis," *Electronics Research Laboratory Memorandum, U.C.Berkeley*, No. UCB/ERL M92/41.