A Strong FSM Watermarking propose for IP Security of Sequential Circuit Design

Kalikai Shivasankar and Prof.A. Balaji Nehru

Abstract—Finite state machines (FSMs) are the backbone ofsequential circuit design. In this document, A new-fangled FSM watermarking design is anticipated by building the authorship in sequence of a non redundant property of the FSM. To overcome the vulnerability to state removal attack and minimize the design overhead, the Water mark bits are effortlessly interweave into the outputs of the active and liberated transition of state transition graph (STG). Unlike other switch-based STG watermarking, pseudo input variables have been compact and prepared functionally indiscernible by the notion of reserved gratis accurate The obligation of held in reserve literals is oppressed to curtail the transparency of watermarking also compose the watermarked FSM frail leading subtraction of any pseudo key variable. A direct and convenient detection scheme is also planned to agree to the watermark on the FSM to be publicly detectable. Experimental results on the water marked circuit from the ISCAS'89 and IWLS'93 standard sets show minor or suitably short spending with upper rig resilience and stronger authorship substantiation in similarity with related watermarking schemes for sequential functions.

I. Introduction

The very large scale integration (VLSI) design industry is confronted with the increasing threat of intellectual property (IP) infringement. IP providers are in pressing need of a well-situated means to track the illegal redeployment of the sold IPs. An active approach to protect a VLSI design against IP intrusion is by embedding a cross that can only be distinctively generate by the IP writer into the propose during the process of its creation. When a forgery is suspected, the autograph can be improved from the misappropriated IP to provide as patent authorship attestation in front of a court. Such a patent safety method is generally known as water marking. It is cheaper and more effective than patenting or copyrighting by law to deter IP piracy In this document, a new active watermarking scheme is proposed. The watermark is embedded in the state transitions of FSM at the behavioral level. As a FSM design is usually specified by a STG or other behavioral descriptions that can be simply convertd into STG, the watermark is rooted into the STG of some size and residue a possessions of FSM after the watermarked aim is synthesize and optimized into circuit netlist. The authorship can be directly verified even after the downstream included route invent process by running the watermarked FSM with a detailed code progression Finite State Machine Watermarking: A formal definition of a FSM is given in [19] as follows.

Definition 1: A FSM is a tuple M = (,, Q, s0, δ, λ),where ∑ and Δ are finite, non-empty sets of the input and output alphabets, respectively. Q is a finite, non-empty set of states and s0 □ Q represents a unique reset state. δ(s,X): Q×∑ → Q□{Ø} is the state transition function and λ(s,X): Q ×∑ = Δ □ {τ} is the output function, where Ø denotes an unspecified state and τ denotes an unspecified output. For si, sj □ Q, sj is said to be the next state of si if □X □ s.t. sj = δ(si, X). The application of X on si also produces an output, Y = λ(si, X) □ . For a FSM with n input and k output variables, each input alphabet, X = x1 x2 . . . xn, is a string of n bits and each output alphabet,Y= y1 y2 . . . yk, is a string of k bits. Each bit of X and Y, xi, yi□ {0, 1, −}, where "0" and "1" are the binary constants, and "–" denotes a "don't care" value. To avoid unnecessary notational complexity, we use an upper case letter to denote an input or output alphabet in ∑ andΔ , a lower case letter to denote an input or output variable in {0, 1, −}, and $y_{i,j}$ to address the jth bit of the ith alphabet, Y_i.

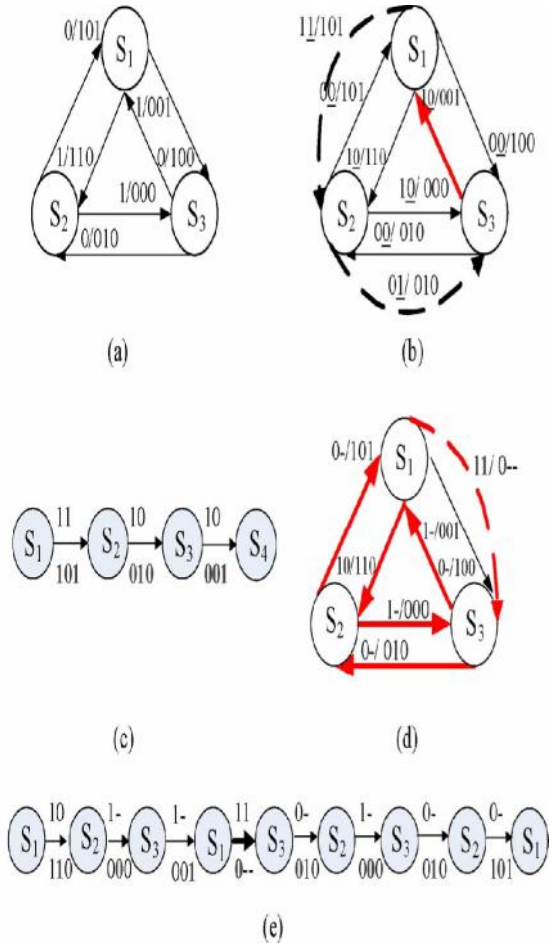

Fig. 1. Watermark embedding on transitions of STG. (a) Original STG. (b) Watermarked STG by the scheme in [14]. (c) Excitation of watermarked transitions of STG in (b). (d)

Watermarked STG by proposed scheme. (e) Excitation of watermarked transitions of STG in (d).

FSMs are usually designed with their STG. A STG, STG(M)= G(V, E), is a labeled directed graph of a machine M of V nodes and E edges. Each symbolic state, s Q, is represented by a node in V. A state transition t from a source node S(t) to a destination node D(t) is represented by a directed edge, eij E, connecting S(t) to D(t). Each edge is tagged with aninput/output label, I(t)/O(t), to encapsulate the relations, D(t)= δ[S(t), I(t)] and O(t) = λ[S(t), I(t)]. Thus, a state transition t can be represented by a quadruple [S(t), D(t), I(t), O(t)]. The input combinations that are absent from all transitions of a source state in a STG are called the free (or unspecified) input combinations of that state, and a transition that can be created from the free input combinations is called an unspecified transition. Unlike [12], as the number of states in a FSM is a dominant factor of the implementation complexity, we modify only the properties of the edge set to synthesize the watermarked design in order to preserve the nodes in STG (M).

II. Watermarking Insertion

The watermark W is inserted into STG(M) by modifying some of its edges without changing the operational behaviour of M to find a sequence of N consecutive transitions, ˆti = ˆsi, ˆsi+1, ˆXi, ˆYi, i = 1, 2, . . . , N, such that each watermark bit, wl W, l [1, m], will be randomly mapped to one bit in the sequence, ˆY = ˆY 1ˆY2 · · · ˆYN = ˆy1,1 · · · ˆy1,k ˆy2,1 · · · ˆy2,k · · · ˆyN,1 · · · ˆyN,k. The mapping from W to ˆY is injective but not surjective. The value of each bit ˆyi,j in ˆY can be determined as follows: if (i–1)k + j = bl, then ˆyi,j = wl, else ˆyi,j = "–," as shown in Fig. 2.Given an output ˆYi and a source state ˆsi, the destination state ˆsi+1 of watermarked transition ˆti will be determined by an output compatibility check. Two bits, x, y {0, 1, −}, are compatible if they are of equal value or one of them has a don't care value, i.e., x ∩ y = Ø. This intersection of two ternary variables is defined in Table I. Likewise, two alphabets, X and Y are compatible, denoted by X ≡ Y, if none of the elements in X ∩ Y = {xi ∩ yi} has a null value. Starting with i = 1, an arbitrary state, ˆs1 Q, is selected. Let T(ˆsi) be the set of transitions emanating from a state, ˆsi. A set of transitions C(ˆsi) that is output compatible with ˆYi is sought, i.e., C(ˆsi) = {ti T (ˆsi)|O(ti) ≡ ˆY1}. To avoid entering into a deadlock, transitions terminated at a deadlock state (i.e., state with no fanout) are excluded from C(ˆsi). Four distinct scenarios are considered for the determination of ˆti.

1) Case 1: there is only one output compatible transition, |C(ˆsi)| = 1, then ˆti = C(ˆsi) and ˆsi+1 = D(ˆti).

2) Case 2: if more than one output compatible transition are found, i.e., |C(ˆsi)|> 1, then a transition from C(ˆsi), with the next state having the highest number of free input combinations, will be selected asˆti. Its output will be modified to O(ˆti) = O(ˆti) ˆYi and ˆsi+1 = D(ˆti).

3) Case 3: if |C(ˆsi)| = 0, then the free input combinations of ˆsi will be considered. Let F(ˆsi) = {X |δ(ˆsi,X) = Ø} be the set of free input combinations of ˆsi. For F(ˆsi) = Ø, let D(ˆsi) = {ˆsj Q|ˆsj = D(ˆti) ˆti T (ˆsi)} be the set of all destination states of ˆsi. ˆsi+1 is set to the state with the highest number of free input combinations in D(ˆsi) (excluding the deadlock states) unless D(ˆsi) = Ø. When D(ˆsi) = Ø, ˆsi+1 is set to the state with the highest number of free input combinations in STG(M). If there exists an edge connecting ˆsi to ˆsi+1 in STG(M), a new input/output pair, I(ˆti)/O(ˆti), is added for the transitionˆti. Otherwise, a new edge directed from ˆsi to ˆsi+1 labeled with I(ˆti)/O(ˆti) will be created in STG(M) for ˆti, and O(ˆti) = ˆYi. The determination of I(ˆti) will be explained later.

4) Case 4: if |C(ˆsi)| = 0 and F(ˆsi) = Ø, then a pseudo input variable xn+1 needs to be introduced in M and the number of input variables n is incremented by 1. xn+1 is set to an unspecified logic value "*" for all existing transitions. A new edge directed from ˆsi to ˆsi+1 labeled with I(ˆti)/O(ˆti) will be created for ˆti. ˆsi+1 is set to the state with the highest number of free inputs in D(ˆsi) or in STG(M) if D(ˆsi) = Ø, and O(ˆti) = ˆYi. Both symbols "*" and "–" can assume either a logic "0" or a logic "1" value but there is a subtle difference. "–" is meant for the currently used input combinations whereas "*" can be associated with either the used or free input combinations. A "*" can be construed as a reserved free input literal as its logic state ("0" or "1") will only be defined at the time when some input combinations subsumed by it are freed to become I(ˆti). The pseudo codes for the determination of watermarked transitions are shown in Fig. 3. The input alphabets for the watermarked transitions found in Cases 3 and 4 are determined by the subroutine.

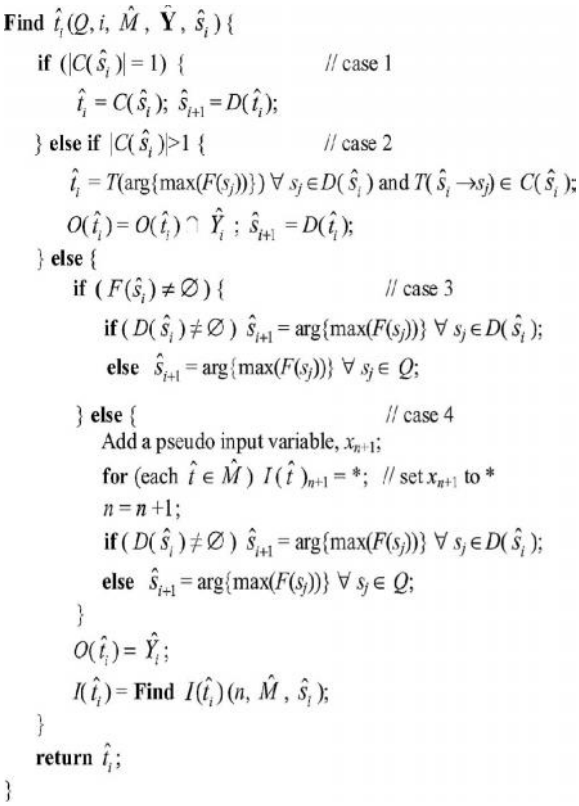

Fig. 2. Determination of watermarked transition.

FSM_watermarking $(M, MSG, m, k, N, Q, K_e)$ {

$W = \{w_i\}_{i=1}^{m} = \mathbf{MD}(\mathbf{encrypt}(MSG, K_e))$;

$B = \{b_i\}_{i=1}^{m} = \mathrm{PNG}(K_e, N \times k)$, $b_i \in [1, N \times k]$;

$\hat{\mathbf{Y}} = \mathbf{Generate}\ \hat{\mathbf{Y}}\ (W, B)$;

$\hat{s}_1 \in Q$; $\hat{M} = M$;

for $(i = 1$ **to** $N)$ {

Find $\hat{t}_i(Q, i, \hat{M}, \hat{\mathbf{Y}}, \hat{s}_i)$;

$\hat{s}_i = D(\hat{t}_i)$; }

for (each transition $\hat{t} \in \hat{M}$) {

replace * in $I(\hat{t})$ by $-$; }

return $\hat{M}$;

}

Fig. 3. Algorithm for FSM watermarking.

N_adaptation $(A, \varepsilon, N_{max}, m)$ {

$i = 1$; $N_i \approx m$; $A_{wm_0} = A$; $N = N_i$;

repeat {

$\hat{M}$ = **FSM_watermarking** $(M, MSG, m, k, N_i, Q, K_e)$;

Synthesize $\hat{M}$ and obtain A_{wm_i};

if $(A_{wm_i} > A_{wm_{i-1}})$

$N_{i+1} = N_i + \delta_i$;

else

$N_{i+1} = N_i - \delta_i$;

Compute $\overline{A_{wm}}$ and σ_i;

$N = N_i$; $i = i + 1$;

} **until** $\sigma_i / A \le \varepsilon$ **or** $N \ge N_{max}$;

return $\hat{M}$ with minimum area;

}

Fig.4. Minimization of FSM watermarking overhead by adaptation of *N*.

Step 1: Insertion of the STG.

Step 2: Finding the minimal straddling tree from the STG.

Step 3: Calculation of the number of states from the tree, starting from a terminal state and ending at another terminal state.

Step 4: If number of state is n, calculate i such that i= log2n

Step 5: Check whether 2i=n, if yes go to step 6, else go to step 7

Step 6: If 2i=n, all the states are occupied and to insert watermarked state we need to add one bit extra which is logic high only at the watermarked state. Set i=i+1 and n=n+1.

Step 7: If 2i ≠n insert a watermark in an unused state and set n=n+1;

Step 8: After each insertion verify, test whether all the required watermarked states are fixed or not, if yes, stopif not go to step 5

This algorithm will only make the watermarked states, but transition between the states will be created in transition algorithm .Transition between states:

Step 1: Marked the states in the STG.

Step 2: For a scrupulous state, check whether there is any unused transition.

Step 3: If yes, use this shift as a changeover to first watermarked state.

Step 4: If no, make a transition by totalling an extra bit which will be logic high only for the transition for the first watermarked state.

Step 5: do again this process for each discrete state.

Step 6: From the W1 (first watermarked state) create a transition to W2 (second

Water marked state) with no i/p dependence.

Step 7: Repeat this for W2 to W3, W3 toW4…… awaiting one reaches the last Water marked condition

Step 8: From the last watermarked state, build a transition to any of the separate state with no i/p hope.

The algorithms are illustrated using the example given below. A simple FSM of figure 1 is considered to be a given FSM and both the algorithms are applied on it to create the watermark states and transition between them. In this FSM there are 5 states and the requirement is to add 4 watermarked states based on the calculation from minimal spanning tree. From insertion of the watermark algorithm the value of i is 3 and 2i >n . So there are free states which can be utilized as watermarked state. Figure 6 shows the FSM after the addition of first watermarked state

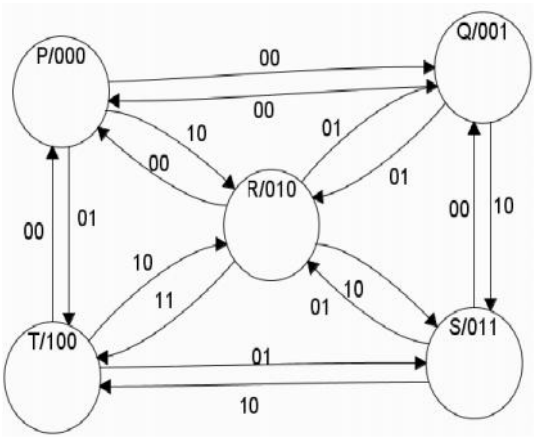

Fig.5. Simple FSM.

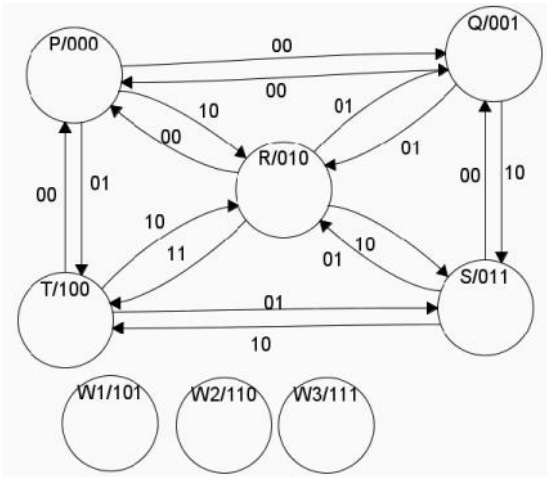

Fig.6. FSM after the addition of first watermarked state

After addition of three watermarked states, n=8 and 2i=n. Still one watermarked state is required to be added. So another extra bit is added which is at logic high only for the last watermarked state and at logic low for other states. Figure 8 shows the FSM after adding all states.

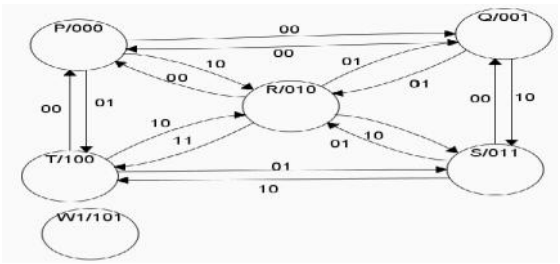

Fig.7. FSM after the addition of first watermarked state

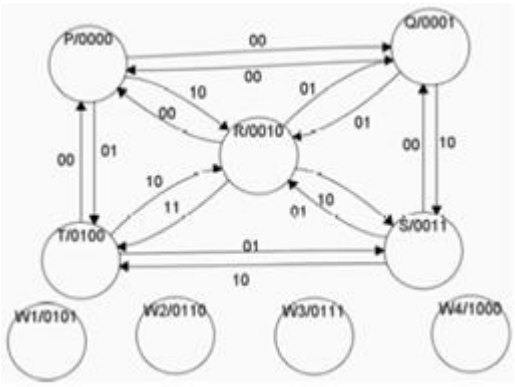

Fig.8. FSM after addition of all water marked states

Now it is needed to add transition to the watermarked states. It can be observed that for state P, Q, S, T, three transitions are used. Hence there is an unused transition for these states which will be utilized for the transition to the first watermarked state. Figure 5 represent the FSM after having first watermark transition. For the state R, there is no unused transition, hence an extra bit which will be at logic high only for that transition and at logic low for the other transition need to add. Finally it is just needed to add transition to the other watermarked states with no input dependence and the transition from the last watermarked state

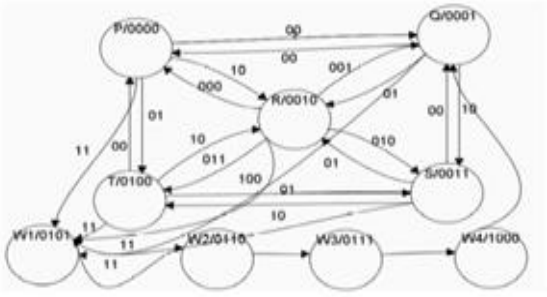

Fig.9. Final FSM after adding all transitions

to any discrete states. So finally the FSM will look like Figure 9.

III. Implementation of Proposed Method

For FPGA implementation of the proposed technique we started with a FSM which shown in Figure. For every input sequence in FSM results in a unique output sequence. Both algorithms are applied on FSM to generate the watermark states and transition connecting them. The watermarked FSM is shown in figure 8. The RTL model for the FSM before and after watermark been implemented by means of synthesis, translate, map, place and route on Xilinx (ISE version 13.1i). The behavioral simulations done with Xilinx ISim and test benches were written in VHDL to give the input vectors for the simulated programs. For function verification it is tested on Virtex series of FPGA.

To produce the circuits for FSM and watermarked FSM both the state diagram translated into state table and from there on state assignment table which help to generate

sequential circuits by the help of state variables . Watermark state

$$A = a'de' + a'df' + c'de'f + b'de'f + c'def' + b'def' + ab'd'e'f' + ac'd'e'f'; \quad (1)$$

$$B = ab'df' + a'def' + c'd'ef + a'd'ef + b'd'ef + a'b'd'e + abc'e'f' + a'bd'e'f'; \quad (2)$$

$$C = a'de'f - ab'c'de' + abc'df' + ac'd'ef' + a'bc'd'e + a'b'd'ef + a'b'c'd'f + ab'ce'f' + ab'cd'f' + a'bcd'f' + a'cd'e'f'; \quad (3)$$

$$Z_2 = a; \quad (4)$$

$$Z_1 = b; \quad (5)$$

$$Z_0 = c; \quad (6)$$

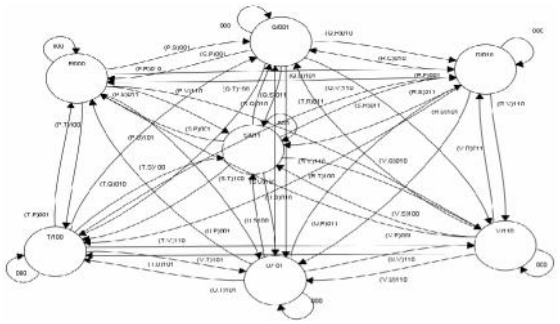

Fig.10.The watermarked FSM after including states and transition

machine having 9 water marked state and 7 discrete states (total 16).

IV. Experimental Results

The experimentation is performed on the circuits, which are described in KISS2 format [23], from the IWLS'93 benchmark set and some FSM designs from the ISCAS'89 benchmark set. The FSM watermarking scheme is implemented using the C++ language. 64 bit and 128 bit watermarks were embedded into each FSM design. Using the SIS [23] tool, state minimization and state assignment are carried out on the original and watermarked designs. The optimized FSM designs are synthesized using the algebraic script from SIS and technology mapped to the Mississippi State University (MSU) standard cell library. All experiments were run on a 750MHz Sun Ultra SPARC-III with Solaris operating system and 2 GB of memory. Table I summarizes experimental results conducted on ISCAS'89 and IWLS'93 benchmark designs. The columns "$|Q|$," "n," and "k" are the numbers of states, input variables, and output variables of each FSM design, respectively. "A" and "D" are the area and delay, respectively, of the optimized design as reported by SIS [23] before watermarking. "P" is the estimated power in μW obtained by using GENERAL delay model [23] with 20MHz clock and 5V supply. Each design is watermarked with the first 64 and 128 bits of SHA-1 hash values of the ownership information. Different lengths of verification code sequence, N have been experimented with $N1 = 80$ and $Nmax = 600$. $\delta i = 20$ when $Ni < 100$ and $di = 100$ when $Ni \geq 100$.Typically, $\sigma i/A$ converges to $\varepsilon = 0.05$ in less than five trials. The value of "N" indicated is the one that produces the least area overhead watermarked FSM design. For most designs tested, only one pseudo input variable is introduced in the watermarking

TABLE I
Statistics for ISCAS'89 and IWLS'93 Benchmarks

Circuit	$\|Q\|$	n	k	A	D	P	m	N	n_a	ΔA	ΔD	ΔP
s27	6	4	1	824	7	307	64	600	2	3.9	5.7	−6.2
							128	300	2	3.9	5.7	−6.2
s208	18	11	2	1912	13	672	64	100	4	3.8	−9.2	6.0
							128	100	4	4.2	−9.2	10.3
s386	13	7	7	2512	13	842	64	100	5	20.7	7.7	9.7
							128	600	5	22.6	10.8	15.2
s832	25	18	19	5144	19.8	1714	64	40	4	10.0	2.0	6.5
							128	200	6	18.2	−4.0	16.2
s510	47	19	7	5528	18.4	2021	64	100	4	6.5	−3.3	2.8
							128	200	7	13.5	−3.3	10.1
s820	25	18	19	6112	20.6	2089	64	200	3	−10.0	−4.9	−17.3
							128	300	4	3.8	−2.9	−3.0
s1488	48	8	19	105	23	3391	64	200	2	3.0	5.2	10.8
							128	300	4	5.9	1.7	12.8
s1494	48	8	19	109	24.8	3746	64	300	4	−0.4	−2.4	−6.5
							128	300	5	8.3	−1.6	4.4
bbara	10	4	2	1112	10.4	433	64	600	3	8.6	−11.5	−4.7
							128	600	5	17.3	−1.9	1.4
dk15	8	3	5	1440	10	556	64	500	1	0.6	4.0	4.3
							128	500	1	8.9	12.0	2.7
ex4	14	6	9	1584	10.8	553	64	300	11	30.8	3.7	36.7
							128	600	9	22.7	1.9	26.9
opus	10	5	6	1768	10.8	612	64	400	4	10.0	11.0	4.9
							128	400	9	21.7	18.5	16.5
sse	16	7	7	2560	12.2	905	64	400	3	16.3	11.5	15.1
							128	500	6	13.1	16.4	−2.5
ex1	20	9	19	4360	16	1439	64	300	5	18.7	16.3	15.0
							128	200	10	32.5	20.0	22.2
s1	20	8	6	5112	16.6	1844	64	200	4	−7.5	1.2	−8.0
							128	500	7	9.4	21.7	4.7
tbk	32	6	3	5352	19.6	1870	64	400	2	−2.4	3.1	−13.9
							128	200	6	0.0	−0.01	−14.7
styr	30	9	10	8408	22.2	3028	64	60	5	7.4	0.0	−4.2
							128	200	5	3.0	−11.7	−1.2
sand	32	11	9	9096	24.8	3128	64	400	4	1.0	−13.7	−1.4
							128	400	5	3.3	0.0	−6.5
planet	48	7	19	9784	21	3454	64	100	5	1.4	0.95	−3.5
							128	100	6	12.1	−2.9	7.9
ram_test	72	16	24	9840	23.8	3563	64	600	3	−7.0	−16.7	−16.0
							128	600	4	−7.0	−10.1	−13.2
scf	121	27	56	12640	23.8	3705	64	400	13	11.8	9.2	0.8
							128	400	15	17.3	10.1	6.5

process while no pseudo input variable is needed for the designs, "ex4," "ex1," and "sand." "*na*" denotes the number of new transitions added onto the watermarked STG. _A, _D, and _P are the percentage area, delay, and power overheads, respectively. A negative percentage implies that watermarking has actually improved the performance. In general, more new transitions have been added onto the designs with 128 bit watermark than with 64 bit watermark.

TABLE II
Comparison with FSM Watermarking Method

Circuit	m	Area			Delay			Power		
		[14]	Prop.	ΔA	[14]	Prop.	ΔD	[14]	Prop.	ΔP
s27	64	1064	856	19.6	7.8	7.4	5.1	411	288	30
	128	1064	856	19.6	7.8	7.4	5.1	411	288	30
s208	64	3680	1984	46.1	15.8	11.8	25.3	1314	712	45.8
	128	5248	1992	62.0	19.8	11.8	40.4	1968	741	62.3
s386	64	3976	3032	23.7	17.8	14.0	12.5	1257	923	26.6
	128	4592	3080	32.9	16.2	14.4	11.1	1479	970	34.4
s832	64	6256	5656	9.6	21.0	20.2	3.8	1939	1826	5.83
	128	6904	6080	11.9	21.6	19.0	12.0	2026	1991	1.73
s510	64	7272	5888	19.0	21.2	17.8	16.0	2641	2077	21.4
	128	7816	6272	19.8	19.4	17.8	8.2	2805	2226	20.6
s820	64	6208	5504	11.3	20.6	19.6	4.9	1944	1727	11.2
	128	7352	6344	47.0	21.4	20.0	6.5	2183	2025	7.24
s1488	64	12 032	10 864	9.7	23.2	24.2	−4.3	3877	3758	3.07
	128	14 120	11 168	20.9	25.6	23.4	8.6	4399	3825	13.0
s1494	64	12 488	10 856	13.1	29.0	24.2	16.6	4166	3503	15.9
	128	12 816	11 808	7.9	25.2	24.4	3.2	4143	3912	5.58
bbara	64	2512	1208	51.9	13.2	9.2	30.3	888	412	53.6
	128	2512	1304	48.1	13.2	10.2	22.7	888	439	50.6
dk15	64	2104	1448	31.2	11.6	10.4	10.3	738	581	21.3
	128	2504	1568	37.4	13.6	11.2	17.6	946	571	39.6
ex4	64	2104	2072	1.5	11.4	11.2	1.8	756	756	0.0
	128	3008	1944	35.4	13.0	11.0	15.4	1077	702	34.8
sse	64	4216	2976	29.4	15.4	13.6	11.7	1474	1041	29.4
	128	4496	2896	35.6	15.2	14.2	6.6	1460	882	39.6
ex1	64	5632	5176	8.1	16.0	18.6	−16.3	1764	1654	6.24
	128	6056	5776	4.6	18.4	19.2	−4.3	1880	1758	6.49
s1	64	5760	4728	17.9	16.8	16.8	0.0	2109	1695	19.6
	128	7376	5592	24.2	19.6	20.2	−3.1	2675	1935	27.7
Sand	64	8944	9184	−2.68	20.4	21.4	4.9	3013	3086	2.37
	128	10 952	9392	14.2	21.6	24.8	−14.8	3674	2925	20.4
Planet	64	11 552	9920	16.5	22.6	21.2	6.2	3787	3332	12.0
	128	11 712	10 968	6.35	25.2	20.4	19.0	4064	3725	8.34
styr	64	9240	9032	2.25	25.0	22.2	11.2	3014	2900	3.78
	128	10 216	8656	15.3	24.2	19.6	19.0	3407	2992	12.2
scf	64	13 568	14 128	−4.1	24.4	26	−6.6	3679	3733	−1.5
	128	13 880	14 824	−6.8	22.8	26.2	−14.9	3625	3945	−8.8

The performance overheads decrease as the size of FSM increases. For the six larger designs (12 watermarked designs), the average area has increased by 4.23% but the average timing and power have actually improved by 0.52% and 0.33%, respectively. It is conjectured that the watermarking overheads will become negligible for FSMs with many more states and input and output variables than those simulated.

We used the SIS tool and the same technology library to synthesize the designs watermarked by the method and compared their areas and delays with those of our proposed (abbreviated as Prop.) FSM watermarking method in Table II._A, _D, and _P are the percentage reductions of area, delay and power, respectively. It is evident that most designs watermarked by our method have lower area, timing, and power overheads.

V. Conclusion

This paper presented a new robust dynamic watermarking scheme by embedding the authorship information on the transitions of STG at the behavioral synthesis level. The proposed method offers a high degree of tamper resistance and provides an easy and noninvasive copy detection. The FSM watermark is highly resilient to all conceivable watermark removal attacks. The redundancy in the FSM has been effectively utilized to minimize the embedding overhead. By increasing the length of input code sequence for watermark retrieval and allowing the output compatible transitions to be revisited to embed different watermark bits, the watermarks are more randomly dispersed and better concealed in the existing transitions of FSM. The new approach to the logic state assignments of pseudo input variables also makes it infeasible to attack the watermarked FSM by removing the pseudo inputs. Without compromising the watermark strength, the length of verification code sequence can be adapted to reduce the area overhead of watermarked design to a reasonable bound within a preset number of iterations. Our experimental results show that the water marking incurs acceptably low performance overheads and possesses very low possibility of coincidence and false positive rate. Similar to other FSM watermarking schemes [12]–[14], this
method is not applicable to some ultrahigh speed designs that do not have a FSM. Fortunately, regular sequential functions are omnipresent in industrial designs [13], making FSM watermarking a key research focus for dynamic watermarking. One recommendation to overcome such limitation is to augment it with combinational watermarking scheme [5] applied simultaneously or on different levels of design abstraction to realize hierarchical watermarking [9], [10]. The watermarked FSM can be fortified by a scan chain watermarking [16], [17] to enable the authorship to be easily verified even after the protected IP has been packaged. While the robustness of the authorship proof lies mainly on the watermarked FSM, the auxiliary post-synthesis scan-chain reordering serves as an intruder-alert for the misappropriation of sequential design under test and increases the effort level required to successfully forge a testable IP without being detected. Even if the scan chain is removed or deranged by the aggressor, the more robust FSM watermark remains intact and detectable on chip.

References

[1]. Intellectual Property Protection Development Working Group, *Intellectual Property Protection: Schemes, Alternatives and Discussion*. VSI Alliance, Aug. 2001, white paper, version 1.1.

[2]. I. Hong and M. Potkonjak, "Techniques for intellectual property protection of DSP designs," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 5. May 1998, pp. 3133–3136.

[3]. A. T. Abdel-Hamid, S. Tahar, and E. M. Aboulhamid, "A survey on IP watermarking techniques," in *Design Automation for Embedded Systems*, vol. 10. Berlin, Germany: Springer-Verlag, Jul. 2005, pp. 1–17.

[4]. D. Kirovski, Y. Y. Hwang, M. Potkonjak, and J. Cong, "Protecting combinational logic synthesis solutions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2687–2696, Dec. 2006.

[5]. A. Cui, C. H. Chang, and S. Tahar, "IP watermarking using incremental technology mapping at logic synthesis level," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 9, pp. 1565–1570, Sep. 2008.

FIRST AUTHOR.
KALIKAI .SHIVASHANKAR
M.tech (VLSI)
CMR INSITTUTE OF TECHNOLOGY,
KANDLAKOYA (Vill)
MEDCHAL (MD)
R.R DISTICT
EMAIL ID:K.SIHVA121@GMAIL.COM

SECOND AUTHOR.
Prof.A.BALAJI NEHRU M.E, (ph.D)
CMR INSITTUTE OF TECHNOLOGY,
KANDLAKOYA (Vill)
MEDCHAL (MD)
R.R DISTICT