VLSI DESIGN
2001, Vol. 12, No. 2, pp. 275–300
Reprints available directly from the publisher
Photocopying permitted by license only

Integrated Area-power Optimal State Assignment

AKHILESH TYAGI*

Department of Electrical and Computer Engineering, Coover Hall, Iowa State University, Ames, IA 50011-3060

(Received 20 June 2000; In final form 3 August 2000)

This paper presents a state assignment algorithm with the objective of lower energy along with area comparable to the area-targeting state assignments such as JEDI. The underlying framework is MUSTANG's complete weighted graph with weights representing state affinity. The weight computation phase estimates the computation energy of potential common cubes using steady state probabilities for transitions. The weight computation phase also identifies a large set of potential state cliques, which are incorporated into a recursive bipartitioning based state assignment procedure. Reuse of cliques identified by the weight computation phase results in a faster and efficient state assignment. The energy targeting weights result in $\approx 9\%$ lower area and 18% lower power than area targeting weights in JEDI over 29 MCNC Logic Synthesis '93 benchmarks. The clique based state assignment performs almost as well as the annealing based state assignment in JEDI, and takes only about half as much time.

Keywords: Finite-state-machine; State assignment; Low power; Low-energy; Multi-level logic synthesis

1. INTRODUCTION

There have been several research studies dealing with state assignment problem for low power [11, 4, 14, 18, 16]. Most of these state assignments target a reduction in average switching per transition (or average Hamming distance between states). This reduction in Hamming distance often results in lower power, but usually at a cost of additional area. Some (Olson and Kang [10], and Benini and De Micheli [1]) of them incorporate objective functions involving both area and switching (power). However these efforts have been limited to redefining an objective function to be a linear combination of area and power based objective functions with the user choosing a relative weight for the area and power optimization. This paper develops a state assignment approach which is a variant of MUSTANG's [2] area optimization weights to incorporate switching activity. MUSTANG attempts to increase the number and size of shared cubes in the resulting multi-level implementation. It does not take into account the switching activity and expected fanout

*Tel.: (515) 294-4396, Fax: (515) 294-8432, e-mail: tyagi@iastate.edu

of the shared cube. The weights in our method try to maximize the expected fanout and the switching activity of shared cubes. Tsui *et al.* [14] also had a similar objective with a different approach.

The objective function in MUSTANG is the minimization of $\sum_{\text{over all edges}} (h_{i,j} w_{i,j})$ so that pairs of states s_i, s_j with high weight (or high affinity) $w_{i,j}$ receive a state assignment with low Hamming distance $h_{i,j}$. Ideally, one would like to seek out all the cliques with high weight in this complete graph and assign least Hamming distance state codes to all the members of the same clique. MUSTANG accomplishes this through a heuristic called *wedge clustering*. However, the constructive procedure followed to derive the weights for the complete graph already builds a large set of potential cliques from the FSM structure. Consider a state set for a state s_k built by the fanout based algorithm. This is a vector of size N for an FSM with N states. All the states that have a transition to state s_k have a nonzero entry in this vector. All the states with nonzero entries in this state set form a potential clique. Similarly, each output set in the fanout based algorithm is a potential clique. Fanin based algorithm provides another $2*n+N$ potential cliques for an *FSM* with N states and n inputs. Note that *clique* in this context means a *weighted clique*, *i.e.*, a set of vertices with the existence of pairwise edges with a high total edge weight.

We use a bipartitioning based approach to assign state codes. Veeramachaneni [17] and Hachtel *et al.* [4] have independently used bipartitioning on the state transition graph with edges labeled by the steady-state transition probabilities. We recursively bipartition the complete weighted graph generated by fanin, fanout or fanin-fanout based weights. The bipartitioning is applied to an equivalent graph whose vertices are the cliques generated by the weight-assignment algorithm (appropriately pre-processed). This bipartitioning stops when each leaf in the recursion contains exactly one clique. At that point the state transition graph for each clique is bipartitioned until each individual state is assigned a code. This method has an advantage in that it keeps all the states within a clique together for as late in the state assignment process as possible. Hence the codes for all the states in a clique share a large prefix resulting in low Hamming distance codes. The main advantage of this method over MUSTANG and JEDI is efficiency in deriving state assignments from a given weighted graph. The quality of solution is comparable to that of simulated annealing based assignment in JEDI, while the running time is only about half that of annealing.

There are many variants of this basic framework. We have experimented with a large set of these variants (such as break cliques at each level so that the state sets are balanced, break a clique only when need be, and break the one with the smallest clique weight; keep cliques disjoint by pruning them right in the beginning; retain overlapping cliques; compute weights only with fanout, only with fanin, or with both fanin and fanout; and many combinations of these factors). The summary results indicate that this method has approximately 9% area advantage and 18% power advantage simultaneously over JEDI [8] for MCNC Logic Synthesis '93 benchmarks with uniform distribution of primary inputs. Note that most of the algorithms reported in literature save power at the cost of adding more area. Hence the simultaneous reduction in area and power from our algorithm is noteworthy. Recall that JEDI incorporates MUSTANG's area affinity based weight framework. Section 2 gives the notation and the energy model. We provide the theoretical basis for weight computation in Section 3. The algorithms are described in Section 4 and the results are presented in Section 5. Section 6 concludes the paper.

2. NOTATION AND ENERGY FRAMEWORK

A finite state machine is represented by a state transition graph $M = (Q, \Sigma, \Delta, \delta, q_o)$ where Q, Σ, Δ are sets of states, input and output respectively. δ is the transition function and q_0 is the initial state. This graph could be provided in KISS (state

transition table) format where each line represents one implicant x_i, s_i, s_i', y_i with input x_i, current state s_i, next state s_i', and output y_i. Figure 1 shows one of the MCNC logic synthesis benchmark FSM (`bbtas`) in KISS format. It has 2 input bits, 2 output bits, 6 states and 24 product terms or specified transitions. This FSM is also shown in Figure 2 in state transition diagram form. In the following, N is the number of states, N_o is the number of output bits, and n is the number of input bits.

Each transition (s_i, s_j) has a conditional probability derived from the probability of the input

```
.i 2
.o 2
.p 24
.s 6
00 st0 st0 00
01 st0 st1 00
10 st0 st1 00
11 st0 st1 00
00 st1 st0 00
01 st1 st2 00
10 st1 st2 00
11 st1 st2 00
00 st2 st1 00
01 st2 st3 00
10 st2 st3 00
11 st2 st3 00
00 st3 st4 00
01 st3 st3 01
10 st3 st3 10
11 st3 st3 11
00 st4 st5 00
01 st4 st4 00
10 st4 st4 00
11 st4 st4 00
00 st5 st0 00
01 st5 st5 00
10 st5 st5 00
11 st5 st5 00
```

FIGURE 1 MCNC logic synthesis benchmark `bbtas` in KISS format.

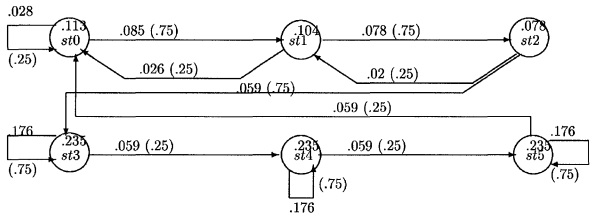

FIGURE 2 State diagram for bbtas with steady-state probabilities.

combination causing this transition. We label this conditional probability $q(s_i, s_j)$ (also referred to as $q_{i,j}$). Figure 2 shows $q(s_i, s_j)$ as the number in parenthesis labeling the corresponding transition. An FSM can be viewed as a Markov process and hence one can compute the steady-state probability of an FSM being in state s_i. Let $p(s_i)$ denote the steady-state probability of state s_i. Each state s_i in Figure 2 is labeled by its steady-state probability $p(s_i)$. The steady-state probability of a transition (s_i, s_j) is denoted by $p(s_i, s_j)$ which is the product $p(s_i)q(s_i, s_j)$. The number not in parenthesis labeling each transition in Figure 2 is $p(s_i, s_j)$. For a detailed discussion of these concepts and of methods for computing these probabilities, the reader is referred to [17, 9, 5]. A state assignment algorithm assigns an m-bit code $c(s_i)$ to each state s_i, for $m \geq \log N$. The *Hamming distance* between two states s_i and s_j (denoted by $h(c(s_i), c(s_j))$ or by $h_{i,j}$) under a state assignment c is the number of bit positions where codes $c(s_i)$ and $c(s_j)$ have different values.

In the rest of the paper, we assume that a state assignment algorithm c assigns state codes of length $[\log N]$. In order to reduce the clutter of notation, we will write $\log N$ instead of $[\log N]$. A state assignment c assigns a state code $s_{i,\log N} s_{i,\log N-1} \cdots s_{i,2} s_{i,1}$ for each state s_i where $s_{i,\log N} \in \{0, 1\}$ is the most significant bit and $s_{i,1} \in \{0, 1\}$ is the least significant bit of the state code $c(s_i)$.

Let us consider the classical FSM implementation where $\log N$ bits representing next state are cycled back to the present state input through a latch. Let the present state bits be $ps_{\log N}, ps_{\log N-1}, \ldots, ps_2, ps_1$ and the next state bits be $ns_{\log N}, ns_{\log N-1}, \ldots, ns_2, ns_1$ (shown in Fig. 3). Note that the computation of a state s_i requires computing $ps_{1_{s_{i,1}}} \wedge ps_{2_{s_{i,2}}} \wedge \cdots \wedge ps_{\log N_{s_{i,\log N}}}$ where $ps_{j_{s_{i,j}}}$ is the literal representing the value of the jth bit of $c(s_i)$, *i.e.*, $ps_{j_{s_{i,j}}} = ps_j$ for $s_{i,j} = 1$ and $ps_{j_{s_{i,j}}} = \overline{ps_j}$ for $s_{i,j} = 0$. Sometimes, during the weight computation phase, energy for computing the product term of a subset of literals $ps_{1_{s_{i,1}}} \wedge ps_{2_{s_{i,2}}} \wedge \cdots \wedge ps_{\log N_{s_{i,\log N}}}$ needs to be estimated. However, switching energy in these cases depends on the switching probability of the present state bits $ps_1, ps_2, \ldots, ps_{\log N}$. The problem is that these bit-level switching probabilities are not known until after the state assignment is done. The only information available before assignment is at state machine abstraction level: steady state probabilities of states $p(s_i)$ and transitions $p(s_i, s_j)$. We use an entropy based estimate of switching derived in [16].

The distributed lower bound from Theorem 3 of [16] considers the immediate neighborhood of a state s_i. For instance, the immediate neighborhood of state $st2$ in Figure 2 consists of states $st1$ and $st3$. The conditional distribution of outgoing transitions from state s_i is denoted by $\{q_{i,j}\}$, *e.g.*, for bbtas in Figure 2 the conditional distribution for state $st2$ is $\{q_{2,1} = .25, q_{2,3} = .75\}$. Theorem 3 in [16] proves that the expected total switching conditional upon present state being s_i (given by $\sum_j (q_{i,j} * h(c(s_i), c(s_j)))$) is at least $H(q_{i,j}) - k \log d_i$ where $H(q_{i,j})$ is the entropy of the $\{q_{i,j}\}$ distribution, k is a constant and d_i is the number of states in the immediate neighborhood of s_i. All the logarithms (log) in the following are to the base

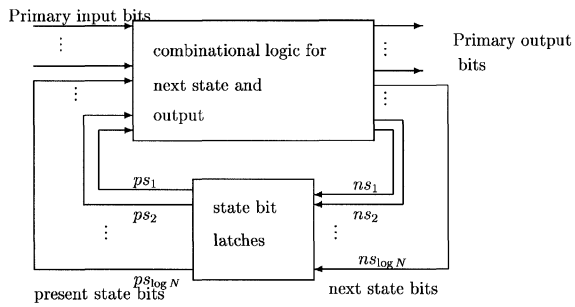

FIGURE 3 Typical FSM implementation scheme.

two unless otherwise stated. We simplify this result to assume that the total switching incurred for a transition from state s_i to one of its neighbors is approximately $H(q_{i,j})$. The `bbtas` state assignments given in Figure 7 have expected switching from $st2$ as 1.75 $(.25 * h(001,000 + .75 * h(001, 100))$ and 1.25 for clique and annealing based assignments respectively. This energy model predicts the expected switching for $st2$ to be $H(q_{2,j}) = .25 * \log 4 + .75 * \log(4/3) = .81$. Hence the formal energy model is:

Energy model 1 The expected switching per state bit for an outgoing transition from state s_i given that the present state is s_i is $H(q_{i,j})/\log N$. $H(q_{i,j})$ is the entropy of the conditional transition probability distribution for state s_i and N is the number of states.

We used another intuitive model to derive expected switching per state bit, which does not have a theoretical justification. We use the steady-state probability of state s_i as an indicator of expected switching for its outgoing transitions.

Energy model 2 The expected switching per state bit for an outgoing transition from state s_i given that the present state is s_i is $p(s_i)/\log N$.

In practice, Energy model 1 performs better than the Energy model 2, however Energy model 2 is easier to compute.

Note that in these energy models, we have assumed that the switching energy is directly proportional to the switching probability. This is a reasonable energy estimate per clock cycle for CMOS technology assuming that there are no routing capacitances and that all gates have unit capacitance.

3. WEIGHT COMPUTATION

We assume some familiarity with the MUSTANG approach [2] in this discussion. MUSTANG [2] and JEDI [8] build a complete graph from the state machine M, $G = (V, E, w)$ with $w(s_i, s_j)$ providing a weight to each edge (s_i, s_j) for $s_i, s_j \in V$. Note that $V = Q$, the set of states of M and $(s_i, s_j) \in \delta$ represents a transition from state s_i to s_j. Since G is a complete graph, $E = V \times V$. Figure 4 shows a complete weighted graph derived from `bbtas`. These FIFO weights were computed using Eqs. (5) and (7) presented later. We summarize MUSTANG's weight computation for area optimization in the next section. This is followed by energy modeling weight computation.

3.1. Area-based Weight Computation

The weights in the complete graph $G = (V, E, w)$ are designed to be used with a minimization of an objective function of the form $\sum_{s_i,s_j} w(s_i, s_j) * h(c(s_i), c(s_j))$. Hence, these weights $w(s_i, s_j)$ reflect the potential savings in the literal count based measure of area of a multi-level logic implementation of G if $h(c(s_i), c(s_j))$ were to decrease by one. In other words, $\delta A = w(s_i, s_j) * \delta h(c(s_i), c(s_j))$ where A is the area (literal count) of the implementation. These weights could be computed from the output (next state and primary outputs) perspective (termed *fanout based weights* in MUSTANG and *output dominant algorithm* in JEDI) or from the input (present state and primary inputs) perspective (termed *fanin based weights* in MUSTANG and *input dominant algorithm* in JEDI).

3.1.1. Fanout Based Weight Computation

Consider fanout based weight computation for $w(s_i, s_j)$. For a given output $y_k \in \Delta$ and state $s_k \in Q$:

$$O_{i,k} = \{(s_i, s) \in \delta | y_k = 1 \quad \text{for transition } (s_i, s)\}$$
$$S_{i,k} = \{(s_i, s_k) \in \delta\}$$

$O_{i,k}$ is the set of all transitions from state s_i that assert output y_k and $S_{i,k}$ is the set of all transitions from s_i to s_k. Let states s_i and s_j be assigned codes with Hamming distance t, *i.e.*, $h(c(s_i), c(s_j)) = t$. Assume that state codes are $\log N$ bits long. Then, under certain assumptions, a common cube of $t' = \log N - t$ literals can be extracted from s_i and s_j to be shared by the logic for y_k and s_k. Let us

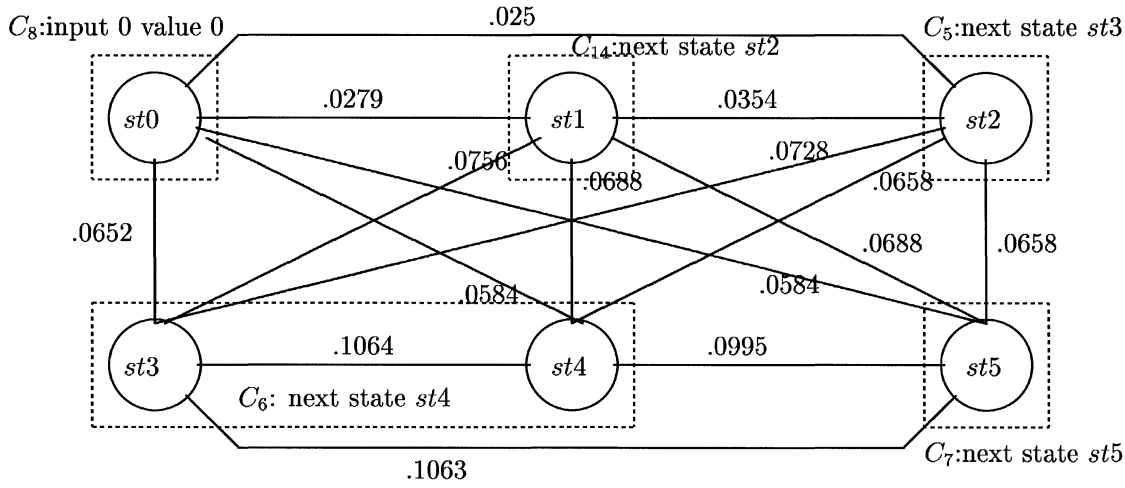

FIGURE 4 Weighted graph for `bbtas` with energy modeling weights.

assume that $s_{i,l_1} = s_{j,l_1}, s_{i,l_2} = s_{j,l_2}, \ldots, s_{i,l_r} = s_{j,l_r}$ and $s_{i,l'_1} \neq s_{j,l'_1}, \ldots, s_{i,l'_t} \neq s_{j,l'_t}$. Instead of computing $s_i \vee s_j$ as $s_{i,l_1}s_{i,l_2}\ldots s_{i,l_r}s_{i,l'_1}\ldots s_{i,l'_t} \vee s_{j,l_1}s_{j,l_2}\ldots s_{j,l_r}s_{j,l'_1}\ldots s_{j,l'_t}$, compute them in the multi-level form $CC_{i,j} = s_{ij,l_1}s_{ij,l_2}\ldots s_{ij,l_r}, s_i \vee s_j = CC_{i,j}(s_{i,l'_1}\ldots s_{i,l'_t} \vee s_{j,l'_1}\ldots s_{j,l'_t})$ where $s_{ij,l_r} = s_{i,l_r} = s_{j,l_r}$ for $1 \leq r \leq t'$ are the common literals. This common cube computation is illustrated in Figure 5. The $|O_{i,k}| + |O_{j,k}|$ fanins of $s_i \vee s_j$ benefit from the common cube $CC_{i,j}$. The literal count gain from extracting $CC_{i,j}$ from output perspective is $[\sum_{k=1}^{N_o}(|O_{i,k}| + |O_{j,k}|) * (t' - 1)] - t'$. Hence the marginal change in the literal count for a delta of 1 in $h(c(s_i), c(s_j))$ is approximately $\sum_{k=1}^{N_o} |O_{i,k}| + |O_{j,k}|$. Similarly, the logic for next state s_k benefits as many times as the number of l's in the code for s_k ($c(s_k)$). Since the state codes are not known *a priori*, assuming that each state code has half ones or log $N/2$ ones, the expected marginal literal count gain is $\sum_{k=1}^{N}(|S_{i,k}| + |S_{j,k}|) * \log N/2$. These two factors contribute to give:

$$w(s_i, s_j) = \sum_{k=1}^{N_o} |O_{i,k}| + |O_{j,k}| + \sum_{k=1}^{N} (|S_{i,k}| + |S_{j,k}|) * \log N/2$$

3.1.2. *Fanin Based Weight Computation*

The fanout based weight computation rewarded large cubes. On the other hand, fanin based weight

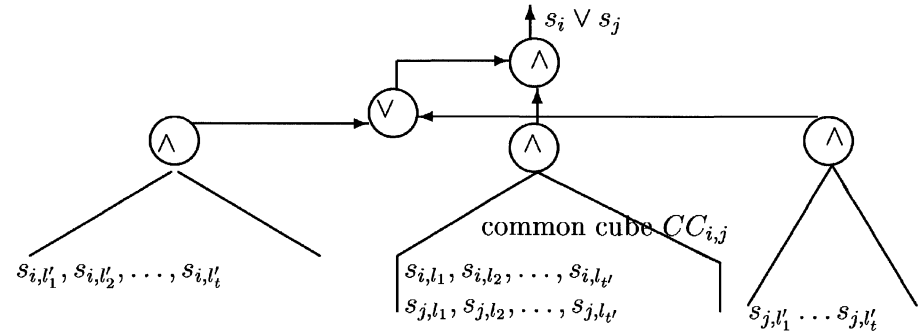

FIGURE 5 Illustration of common cube $CC_{i,j}$ between s_i and s_j.

computation rewards the amount of sharing for the common cubes derived from the input, primary input cubes or present state cubes. Let us define $I^0_{j,k}$ and $I^1_{j,k}$ to model sharing of input literals $\bar{x_k}$ and x_k for $1 \leq k \leq n$. Similarly, Set $PS_{i,k}$ models the sharing of cube for state s_k.

$$I^0 j,k = \{[(x_k = 0) \times s \rightarrow s_j] \in \delta\}$$
$$I^1 j,k = \{[(x_k = 1) \times s \rightarrow s_j] \in \delta\}$$
$$PS_{i,k} = \{(s_k, s_i) \in \delta\}$$

$I^0_{j,k}$ is the set of all transitions to s_j, on an input value with $x_k = 0$, and $I^1_{j,k}$ is similarly defined. Note that the notation $[(x_k = 0) \times s \rightarrow s_j]$ denotes a transition from a state s to s_j on an input with $x_k = 0$. $PS_{i,k}$ is the set of all transitions from state s_k to state s_i. Once again, let $h(c(s_i), c(s_j)) = t$ which implies that $t' = \log N - t$ of $\log N$ bits for s_i and s_j have the same value. All of these t' bits can use the common cube consisting of $\log N$ bits of state code for s_k. Hence the literal count savings are $[\sum_{k=1}^{N}(|PS_{i,k}| + |PS_{j,k}|) * \log N * (t' - 1)] - t'$. This results in marginal literal count savings for $\delta t' = \delta h(c(s_i), c(s_j)) = 1$ as $\sum_{k=1}^{N}(|PS_{i,k}| + |PS_{j,k}|) * \log N$. Similarly, the t' common literals in s_i and s_j can share the 1-literal cube x_k with $|I^1_{i,k}| + |I^1_{j,k}|$ frequency. Hence the marginal improvement in input literal count savings are $\sum_{k=1}^{n}[(|I^1_{i,k}| + |I^1_{j,k}|) + (|I^0_{i,k}| + |I^0_{j,k}|)]$. The fanin based weight $w(s_i, s_j)$ then is:

$$w(s_i, s_j) = \sum_{k=1}^{N}(|PS_{i,k}| + |PS_{j,k}|) * \log N + \sum_{k=1}^{n}[(|I^1_{i,k}| + |I^1_{j,k}|) + (|I^0_{i,k}| + |I^0_{j,k}|)]$$

Sometimes, $|O_{i,k}|$ and $|O_{j,k}|$ (similarly $|S_{i,k}|$ and $|S_{j,k}|$ or $|I^0 i,k|$ and $|I^0 j,k|$) are multiplied rather than added. The state assignment phase attempts to minimize the objective function:

$$\sum_{s_i, s_j \in Q} h(c(s_i), c(s_j)) * w(s_i, s_j)$$

The rationale for this is as follows. Recall that $(\delta A / \delta h(c(s_i), c(s_j))) = w(s_i, s_j)$. Hence $\delta A = \sum_{s_i, s_j \in Q} [\delta h(c(s_i), c(s_j)) * w(s_i, s_j)]$ which results in $A = \sum_{s_i, s_j \in Q} h(c(s_i), c(s_j)) * w(s_i, s_j)$ as the objective function.

3.2. Energy-based Weight Computation

Let us extend this framework to compute weights modeling energy savings instead of literal count savings. As we discussed in Section 2, this energy model ignores the routing capacitances and considers all gate capacitances to be unit capacitances.

3.2.1. Fanout Based Weights

If the common cube $CC_{i,j}$ were to be extracted from $s_i \vee s_j$, what are the resulting energy savings? The common cube extraction saves on the energy of computing $CC_{i,j}$ for all its fanouts.

3.2.2. Energy for Computing a Common Cube

The energy for computing $CC_{i,j} = s_{ij,l_1} s_{ij,l_2} \ldots s_{ij,l_{t'}}$ $(E(CC_{i,j}))$ depends on the steady-state and conditional switching probabilities of state bits s_{ij,l_q} for $1 \leq q \leq t'$. Recall that $p(s_i)$ denotes the steady-state probability of FSM M being in state s_i. Let $p(s_{i,l_q})$ be the switching probability of state bit s_{i,l_q} for $1 \leq q \leq t'$. In particular (ignoring routing capacitances), based on a constant fanin (binary) **and** tree, where d is the number of levels in the tree and r is the number of terms at each level: $E(CC_{i,j}) = \sum_{1 \leq d \leq \log t'} [\sum_{1 \leq r \leq t'/2^d} (\prod_{(r-1)*2^d + 1 \leq k \leq r*2^d} p(s_{ij,l_k}))]$. This is illustrated in Figure 6.

Note, however, that the state code bit probabilities, $p(s_{ij,l_q})$ for $1 \leq q \leq t'$, are not known *a priori*. We would be conservative and estimate a lower bound on $E(CC_{i,j})$. The factor with the minimum probability in the expression for $E(CC_{i,j})$ is $p(s_{ij,l_1}) p(s_{ij,l_2}) \ldots p(s_{ij,l_{t'}})$, which is the intersection of the cubes for states s_i and s_j. In the simplistic *Energy Model 2*, this probability should be $p(s_i) + p(s_j)$. The value for $p(s_i) + p(s_j)$ is known before state assignment. Level d of the fanin tree for

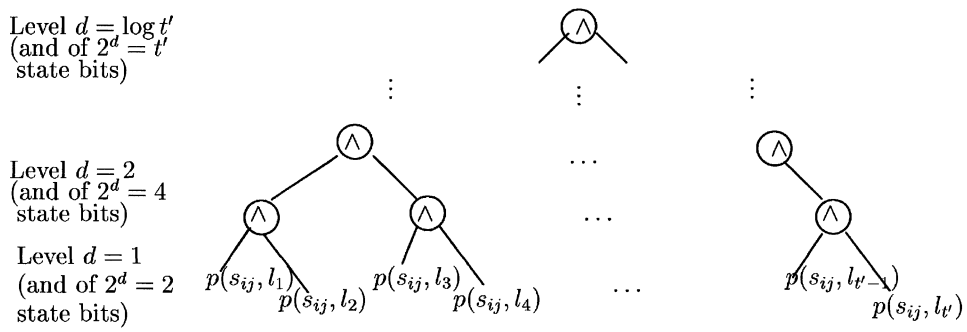

FIGURE 6 Illustration of energy needs in and **and** Tree.

$CC_{i,j}$ contributes $\sum_{1 \le r \le t'/2^d} (\prod_{(r-1)*2^d+1 \le k \le r*2^d} p(s_{ij,l_k})) \ge p(s_i)p(s_j)$ towards $E(CC_{i,j})$ for $1 \le d \le \log t'$. Hence a computable lower bound on $E(CC_{i,j})$ is:

$$E(CC_{i,j}) \ge \log t' * [p(s_i) + p(s_j)] \qquad (1)$$

This form is still not applicable in our framework as the size of the common cube or log $N - h(c(s_i), c(s_j))$ (t' in this case) is not known *a priori*. In fact, as for the area case, we would like to estimate change in energy for every unit of change in $h(c(s_i), c(s_j))$. Let $E(s_i, s_j, t')$ be the energy of computing a common-cube of size $1 \le t' \le \log N$ between state encodings of s_i and s_j. Let $h(c(s_i), c(s_j)) = t = \log N - t'$. We wish to estimate $\Delta E(s_i, s_j, t') = E(s_i, s_j, t'+1) - E(s_i, s_j, t')$ which is the resulting energy gain for $\Delta h(c(s_i), c(s_j)) = 1$. In other words, a change in codes for s_i and s_j such that their Hamming distance decreases by one results in a larger common cube such that the energy for computing the larger cube is $\Delta E(s_i, s_j, \log N - h(c(s_i), c(s_j)))$ units higher. By Eq. (1), we get:

$$\begin{aligned} \Delta E(s_i, s_j, t') &= E(s_i, s_j, t'+1) - E(s_i, s_j, t') \\ &= (\log(t'+1) - \log t') * [p(s_i) + p(s_j)] \end{aligned} \qquad (2)$$

Using Taylor's series expansion, we can use the identity $\ln(N+1) - \ln N = (1/2N+1) + (1/3(2N+1)^3) + (1/5(2N+1)^5) + \cdots \approx (1/2N+1)$ to simplify Eq.(2) to:

$$\Delta E(s_i, s_j, t') = \left(\frac{(\ln e) * 2}{2t'+1}\right) * [p(s_i) + p(s_j)] \qquad (3)$$

However, we do not know the value for t' *a priori*. We could approximate the average size of a shared cube derived from $s_i \vee s_j$. A simple assumption is to set t' at $\log N/2$. An alternative is to assume a perfect state assignment algorithm which generates large common cubes between pairs of states (s_i, s_j) with high steady-state probability (and hence with high computation energy for the common cubes) and with a high probability of common cube's use. We elaborate on this alternative in the following discussion.

State	FIFO-CLIQUES assignment	Annealing assignment
st0	010	000
st1	000	001
st2	001	010
st3	100	011
st4	101	100
st5	110	101

FIGURE 7 FIFO-CLIQUES and annealing state assignments for `bbtas`.

3.2.3. *Probability of Energy Savings for a Common Cube*

The other side of the coin in the total energy saved by extracting the common cube $CC_{i,j}$ is the

probability with which the computation of $CC_{i,j}$ is avoided. Recall that $p(s_i, s_j)$ is the steady-state probability of a transition (s_i, s_j).

The probability of benefitting from the common cube can be computed by adding up the steady-state probability of all the transitions in sets $Q_{i,k}$, $O_{j,k}$, $S_{i,k}$, $S_{j,k}$. Let $OP_{i,k} = \sum_{(s_i,s) \in O_{i,k}} p(s_i, s)$. Similarly, let $SP_{i,k} = \sum_{(s_i,s) \in S_{i,k}} p(s_i, s)$. The expected fanout count for the shared cube $CC_{i,j}$ then is $\sum_{k=1}^{N_o}(OP_{i,k}+OP_{j,k})+\sum_{k=1}^{N}(SP_{i,k}+SP_{j,k})*\log N/2$. This along with Eq. (3) gives a lower bound on marginal energy saved for every unit increment (or unit change in $h(c(s_i), c(s_j))$) in the size of common cube as (which also is $w(s_i, s_j)$ in our formulation):

$$w(s_i,s_j) = \left[\sum_{k=1}^{N_o}(OP_{i,k}+OP_{j,k}) + \sum_{k=1}^{N}(SP_{i,k}+SP_{j,k}) * \log N/2\right] * [p(s_i)+p(s_j)] * (1.39/\log N) \tag{4}$$

The preceding value for $w(s_i, s_j)$ assumes that the size of the common cube between s_i and s_j is $\log N/2$ literals. The total number of literals in the common cubes over all distinct state pairs then is $(N^2/2) * (\log N/2)$. We will multiply this number by the probability of benefitting from the common cube $(\sum_{k=1}^{N_o}(OP_{i,k} + OP_{j,k}) + \sum_{k=1}^{N}(SP_{i,k} + SP_{j,k}) * \log N/2)$ to get an estimate of t'. With this modification:

$$w(s_i,s_j) = \left[\sum_{k=1}^{N_o}(OP_{i,k} + OP_{j,k}) + \sum_{k=1}^{N}(SP_{i,k} + SP_{j,k}) * \log N/2\right]^2 * [p(s_i) + p(s_j)] * (2.78/N^2 * \log N) \tag{5}$$

3.2.4. *Energy Model 1: Fanout*

In energy model 1, the energy for computing the common cube $E(CC_{i,j})$ is different. In this energy model (Section 2), the average switching of each state bit for transitions originating at s_i is given by $H(q_i, k)/\log N$. Similarly, the average switching for transitions from state s_j is $H(q_j, k)/\log N$. What is the value for the average switching for the state bits in the common cube $CC_{i,j}s_{i,j\,l_1}, \ldots, s_{i,j\,l_r}$? We average the switching values for the two states with respect to their steady-state probabilities. Hence $p = p(s_{i,j,l_q}) = (p(s_i) * H(q_{i,k}) + p(s_j) * H(q_{j,k})/\log N * (p(s_i) + p(s_j)))$ for all $1 \le q \le t'$. Once again, assuming an and tree as in Figure 6, $E(CC_{i,j})$ is given by $pt' + p^2(t'/2) + p^4(t'/4) + \cdots + p^{t'}(t'/t')$. Hence the marginal energy for computing a common cube of size t' *versus* $t'+1$ is $\Delta E(s_i, s_j, t') = E(s_i, s_j, t'+1) - E(s_i, s_j, t') \approx p + p^2/2 + p^4/4 + \cdots + p^{t'}/t'$. Since $p < 1$, typically the first few terms of this expression matter and hence $\Delta E(s_i, s_j, t') \approx p + p^2/2 + p^4/4$, which is:

$$\Delta E(s_i,s_j,t') \approx \frac{p(s_i) * H(q_{i,k}) + p(s_j) * H(q_{j,k})}{\log N * (p(s_i) + p(s_j))} + \left(\frac{p(s_i) * H(q_{i,k}) + p(s_j) * H(q_{j,k})}{\log N * (p(s_i) + p(s_j))}\right)^2/2 + \left(\frac{p(s_i) * H(q_{i,k}) + p(s_j) * H(q_{j,k})}{\log N * (p(s_i) + p(s_j))}\right)^4/4$$

The weights then are given by:

$$w(s_i,s_j) = \left[\sum_{k=1}^{N_o}(OP_{i,k} + OP_{j,k}) + \sum_{k=1}^{N}(SP_{i,k} + SP_{j,k}) * \log N/2\right] * \left[\frac{p(s_i) * H(q_{i,k} + p(s_j) * H(q_{j,k})}{\log N * (p(s_i) + p(s_j))} + \left(\frac{p(s_i) * H(q_{i,k}) + p(s_j) * H(q_{j,k})}{\log N * (p(s_i) + p(s_j))}\right)^2/2 + \left(\frac{p(s_i) * H(q_{i,k}) + p(s_j) * H(q_{j,k})}{\log N * (p(s_i) + p(s_j))}\right)^4/4\right] \tag{6}$$

3.2.5. *Fanin Based Weights*

Recall that fanin based weights model the frequency of sharing for the common cubes derived

from the present state and primary input bits. Let us model the frequency of use for both present state and primary input bit cubes. Once again, we need to accumulate probabilities of transitions in sets $PS_{i,k}$, $I^0_{j,k}$ and $I^1_{j,k}$. Let $PSP_{i,k} = \sum_{(si,s) \in PS_{i,k}} p(s_i, s)$, and similarly $IP^0 j,k = \sum_{(s_j,s) \in I^0_{j,k}} p(s_j, s)$, $IP^1 j,k = \sum_{(s_j,s) \in I^1_{j,k}} p(s_j, s)$. The probability of use of a present state shared cube derived from state s_k is $\sum_{k=1}^{N}(PSP_{i,k} + PSP_{j,k})$. The primary input cubes $\overline{x_k}$ and x_k are used with probability $\sum_{k=1}^{n}(|I^0_{i,k}| + |I^0_{j,k}|)$ and $\sum_{k=1}^{n}(|IP^1_{i,k}| + I^1_{j,k}|)$ respectively.

Let us now estimate the energy for computing s_k, the cube $ps1_{s_{k,1}} \wedge ps2_{s_{k,2}} \wedge \ldots \wedge ps_{\log N_{s_k \log N}}$. In Energy Model 2, this energy is simply $p(s_k) * \log\log N$. Switching at each level is proportional to steady state probability $p(s_k)$ and there are $\log\log N$ logic levels. For Energy Model 1, the energy is given by $H(q_{k,j}) * \log\log N$ since expected switching per level is $H(q_{k,l})$. Energy for computing the input literals $\overline{x_k}$ and x_k is just the switching probability of the input bit x_k, $p(x_k)$.

Hence the expected marginal energy gain for every unit reduction in the Hamming distance of states s_i and s_j is given by (Energy Model 2):

$$w(s_i, s_j) = \sum_{k=1}^{N}[(PSP_{i,k} + PSP_{j,k}) * p(s_k) * \log\log N] + \sum_{k=1}^{N}[(|I^0_{i,k}| + |I^0_{j,k}|) + (|IP^1_{i,k}| + |I^1_{j,k}|)] * p(x_k) \quad (7)$$

For Energy Model 1, the fanin weights are:

$$w(s_i, s_j) = \sum_{k=1}^{N}[(PSP_{i,k} + PSP_{j,k}) * H(q_{k,l}) * \log\log N] + \sum_{k=1}^{N}[(|I^0_{i,k}| + |I^0_{j,k}|) + (|IP^1_{i,k}| + |I^1_{j,k}|)] * p(x_k) \quad (8)$$

Again, we sometimes use the multiplicative form for $w(s_i, s_j)$, *e.g.* $[[\sum_{k=1}^{N_o}(OP_{i,k} * OP_{j,k}) + \sum_{k=1}^{N}(SP_{i,k} * SP_{j,k}) * \log N/2] * \cdots]$. Once again, the objective function to minimize is $E = \sum_{s_i,s_j \in Q} h(c(s_i), c(s_j)) * w(s_i, s_j)$.

3.2.6. *FIFO Weights*

A hybrid approach to weight computation is FIFO based weights where fanout and fanin based weights are added together. Note that circuits with relatively large number of output (input) bits are modeled better with fanout (fanin) based weights. FIFO based weights usually perform better than either of fanin or fanout based weights.

4. STATE ASSIGNMENT ALGORITHMS

The state assignment algorithm takes the weighted graph $G = (V, E, w)$ as its input and generates a state assignment c attempting to minimize the objective function $\sum_{s_i,s_j \in Q} h(c(s_i), c(s_j)) * w(s_i, s_j)$. Several heuristics have been used for this purpose, specifically *wedge clustering* in MUSTANG [2] and simulated annealing in JEDI [8]. Note that it would be advantageous to assign state codes with low Hamming distances to the groups of states with relatively high weights with respect to each other (so that high values of $w(s_i, s_j)$ would have low coefficient values $h(c(s_i), c(s_j))$).

We have used two classes of state assignment algorithms – annealing based and *clique-based.*

Annealing JEDI uses annealing based minimization of $\sum_{s_i,s_j \in Q} h(c(s_i), c(s_j)) * w(s_i, s_j)$ as the default state assignment method. We provided energy based weights (computed with Eqs. 6 and 8) to JEDI annealing assignment phase.

Clique Wedge-clustering in MUSTANG detects the clusters of states $s, s_{y_1}, s_{y_2}, \ldots, s_{y_k}$ with $k \leq \log N$ such that $\sum_{i=1}^{k}(w(s, s_{y_i}))$ is a high value. The intuition is that up to $\log N$ neighbors of s can be potentially assigned unidistant codes from s. A cluster that appears to benefit most is chosen and smallest Hamming distance codes are assigned to all the yet unassigned states in the cluster. This

process is repeated until there are no unassigned states.

A more general notion of a cluster is a *weighted clique*, a set of states with higher than average pairwise weight. We could identify all the weighted cliques (which could also be singletons) in the weighted graph $G=(V,E,w)$. These cliques can also be rank-ordered on the basis of average weight (clique weight divided by number of states in the clique). Ideally, we would like to assign the best codes (lowest, average Hamming distance) to the states in the highest weight clique. The states in a higher weight clique get to choose their codes before the states in a lower weight clique.

In general, clique identification is a hard problem [3]. However, we observe that the weight computation steps in Section 3 form the *bricks* of clique formation. Hence, we already know candidates for potential cliques in G. Each component of the weight $w(s_i, s_j)$, affinity between states s_i and s_j, is attributable to either an output bit or a next-state in the fanout based weight computation; or to an input bit or a present state in fanin based weight computation. We can determine which of these potential cliques actually form a clique. These cliques can then be used by a state assignment method as additional information about the FSM structure. The cluster cover notion of Shi and Brzozowski [13] comes closest to our notion of weighted cliques. In the following, we describe several methods for clique based state assignment.

There are four major, well-identifiable phases in the state assignment.

Steady state probabilities Compute the steady state probabilities $p(s)$ for each state $s \in Q$ using classical Markov chain theory [17, 9]. This is used to compute steady state probabilities $p(s_i, s_j)$ of each transition from state s_i to state s_j.

Weight computation This phase computes the weights for the complete, undirected, weighted graph $G(V,E,w))$ where V is in 1-to-1 correspondence with the state set Q. Section 3 describes this procedure.

Clique computation There are potentially N_o output based cliques, $2n$ input based cliques, N next-state based cliques and N present state based cliques. This phase determines all the cliques from these $N_o+2N+2n$ potential cliques. We describe the notation and methods later in this section.

State assignment From the weighted graph G and the set of cliques C computed in previous step, another induced graph is derived: $G_C=(C,E_C,w_C)$ where C is the set of clique vertices, E_C usually induces a complete graph, and w_C aggregates the affinities between states into affinities between cliques. This graph is then bipartitioned recursively to assign state codes to each state in V. The objective function minimized in this phase still is $\sum_{s_i,s_j \in V}(w(s_i,s_j) * h(c(s_i),c(s_j)))$.

The first of these steps, steady-state probability computation is fairly standard [9, 17]. We describe the other three steps in detail in the following subsections.

4.1. Weight Computation

The weight computation follows the description in Section 3. We give the details of only the fanout oriented weight assignment. The fanin oriented weights can be computed in a very similar manner. The data structures maintained are: $STATE\_SET[N][N] - STATE\_SET[i][j] = p(s_j, s_i)$ if state s_j has a transition to s_i with steady-state probability $p(s_j, s_i)$; and $OUTPUT\_SET[N_o][N] - OUTPUT\_SET[i][j] = \sum_{y_i=1 \in (s_j,s)} p(s_j, s)$. The ith row of $STATE\_SET$ ($OUTPUT\_SET$) corresponds to the affinity set due to the state s_i (the output bit y_i) respectively. The program in Figure 8 computes the frequency of use of the potential common cubes into $STATE\_SET$ and $OUTPUT\_SET$. The steady-state probability computation phase computes $p(s_i)$ into $STATE\_PROBABILITY[i]$ and $H(q_{i,k})$ into $LENTROPY[i]$. The program in Figure 9 accumulates the weights into $w[i][j]$ using Energy Model 1 from Eq. (6).

The fanin based weight assignments can be done in an analogous fashion. The FIFO weights are

```
for i = 1 to N_o
  for each edge e_{j,k} = (s_j, s_k) ∈ G
    if (e_{j,k} asserts the output bit y_i)
      OUTPUT_SET[i][j] = OUTPUT_SET[i][j] + p(s_j, s_k);
for i = 1 to N
  for each edge e_{j,i} = (s_j, s_i) ∈ G
    STATE_SET[i][j] = STATE_SET[i][j] + p(s_j, s_i);
```

FIGURE 8 Fanout based weight frequency computation.

```
for i = 1 to N
  for j = 1 to N
    P = (STATE_PROBABILITY[i] * LENTROPY[i] + STATE_PROBABILITY[j] *
      LENTROPY[j])/[log N * (STATE_PROBABILITY[i] + STATE_PROBABILITY[j])];
    for k = 1 to N
      w[i][j] = w[i][j] + (STATE_SET[k][i] + STATE_SET[k][j]) * 0.5 * log N * (P + P^2/2);
    for k = 1 to N_o
      w[i][j] = w[i][j] + (OUTPUT_SET[k][i] + OUTPUT_SET[k][j]) * (P + P^2/2);
    w[j][i] = w[i][j];
```

FIGURE 9 Accumulation of fanout based affinity weights.

computed by running both fanin and fanout based weight assignments and adding them. The *experimental choices* offered by weight computation are weight styles: fanout based weights (FO), fanin based weights (FI), or FIFO weights (FIFO).

4.2. Clique Computation

A weighted clique C in $G=(V, E, w)$ is a set of states in V with higher than average weights. Note that the notation $|S|$ specifies the cardinality of the set S.

DEFINITION 1 The weight of a set of states $C \subseteq V$, $W(C)$, is given by $\sum_{(s_i,s_j) \in E \text{ and } s_i,s_j \in C} w(s_i, s_j)$.

DEFINITION 2 The normalized weight of a set of states $C \subseteq V$, $NW(C)$, is given by $(W(C)/0.5 * |C| * (|C|-1))$.

Normalized weight of a set is the total weight of that set divided by the number of edges in it.

DEFINITION 3 Let avg_weight $(G) = (\sum_{s_i,s_j \in V} w(s_i, s_j)/0.5 * |V| * (|V|-1))$. A set of states $C \subseteq V$ is a weighted clique in G if $NW(C) \geq$ avg_weight (G).

Note that the clusters in MUSTANG [2] are a special case of a weighted clique.

The intention is to assign state codes of low Hamming distance to states within weighted cliques. However, how do we identify all the weighted cliques in G? Recall that the problem of determining if a given graph G has a clique of size k is NP-complete [3]. Unfortunately, *weighted clique* problem, to determine if there exists a weighted clique of size k is also NP-complete through the following polynomial time, simple reduction. Given an instance of *CLIQUE*: a graph $G=(V, E)$ and k, reduce it to an instance of *WEIGHTED-CLIQUE*: $G'=(V, E, w)$ and k where $w(u, v)=1$ for all $u, v \in V$. It can be seen that G has a clique of size k iff G' has a weighted clique of size

k. This is because the average weight over G is 1 (since each pair of vertices has been assigned weight 1 even if it is not in E). However, for $W(C)$ we only count the weight of those vertex pairs that are in E.

4.2.1. *Potential Weighted Cliques*

This is where the following observation helps. The reasons for adding towards the weight between a pair of states (s_i, s_j) are their affinity within our weight assignment framework of Section 3. The fanout based weight assignment provides this affinity for one of $N+N_o$ reasons, *i.e.* (s_i, s_j) are close either because of one or more of the N next states s or because of one or more of the N_o output bits. Each of the $N+N_o$ sets, $\bigcup_i S_{i,k}$ for $1 \leq k \leq N$ and $\bigcup_i O_{i,k}$ for $1 \leq k \leq N_o$ defines a possible weighted clique or a subset of a weighted clique. All the states s in the union of $S_{i,k}$ for a fixed k: $\bigcup_{1 \leq i \leq N} S_{i,k}$, are potentially close together due to the next state s_k. Similarly, all the states in $\bigcup_{1 \leq i \leq N} O_{i,k}$ could potentially form a subset of a weighted clique due to their shared affinity with respect to the output bit y_k. Hence the ith row of *STATE_SET* and *OUTPUT_SET* data structures used in Figures 8 and 9 describes an affinity set. For instance, all the columns in *STATE_SET*[i] with nonzero entries correspond to potential states within a weighted clique. Most likely, many of these rows (sets) have very few non-zero entries. Figure 10 shows both fanout and fanin based weight assignment data structures for `bbtas`. Note that *INPUT_SET*0[i][j], *INPUT_SET*1[i][j] and *PREV_STATE_SET*[i][j] are used to store the affinity between state s_j and input $x_i=0$ or input $x_i=1$ or present state s_i respectively. The entries for *STATE_SET*[0] in Figure 10 imply that the next-state logic for state $st0$ benefits from common cubes derived from states $st0$, $st1$ and $st5$. We could also deduce that $\bigcup_{1 \leq i \leq N} S_{i,st0}$ is the set $\{st0, st1, st5\}$. Hence it is likely that the set $\{st0, st1, st5\}$ forms a potential weighted clique.

This simple observation leads to a set of $2N+2n+N_o$ potential weighted cliques with FIFO weights, to a set of $N+N_o$ potential cliques with fanout weights, and to a set of $N+2n$ potential

C_0	*OUTPUT_SET*[0]	0.000000	0.000000	0.000000	0.352174	0.000000	0.000000
C_1	*OUTPUT_SET*[1]	0.000000	0.000000	0.000000	0.352174	0.000000	0.000000
C_2	*STATE_SET*[0]	0.028261	0.026087	0.000000	0.000000	0.000000	0.058696
C_3	*STATE_SET*[1]	0.254348	0.000000	0.019565	0.000000	0.000000	0.000000
C_4	*STATE_SET*[2]	0.000000	0.234783	0.000000	0.000000	0.000000	0.000000
C_5	*STATE_SET*[3]	0.000000	0.000000	0.176087	0.528261	0.000000	0.000000
C_6	*STATE_SET*[4]	0.000000	0.000000	0.000000	0.058696	0.528261	0.000000
C_7	*STATE_SET*[5]	0.000000	0.000000	0.000000	0.000000	0.058696	0.528261
C_8	*INPUT_SET*0[0]	0.113043	0.104348	0.078261	0.234783	0.234783	0.234783
C_9	*INPUT_SET*0[1]	0.113043	0.104348	0.078261	0.234783	0.234783	0.234783
C_{10}	*INPUT_SET*1[0]	0.000000	0.169565	0.156522	0.469565	0.352174	0.352174
C_{11}	*INPUT_SET*1[1]	0.000000	0.169565	0.156522	0.469565	0.352174	0.352174
C_{12}	*PREV_STATE_SET*[0]	0.028261	0.254348	0.000000	0.000000	0.000000	0.000000
C_{13}	*PREV_STATE_SET*[1]	0.026087	0.000000	0.234783	0.000000	0.000000	0.000000
C_{14}	*PREV_STATE_SET*[2]	0.000000	0.019565	0.000000	0.176087	0.000000	0.000000
C_{15}	*PREV_STATE_SET*[3]	0.000000	0.000000	0.000000	0.528261	0.058696	0.000000
C_{16}	*PREV_STATE_SET*[4]	0.000000	0.000000	0.000000	0.000000	0.528261	0.058696
C_{17}	*PREV_STATE_SET*[5]	0.058696	0.000000	0.000000	0.000000	0.000000	0.528261

FIGURE 10 Candidate weighted cliques in `bbtas`.

cliques with fanin weights. From now on, we will assume FIFO weights unless stated otherwise. Since each weighted clique in $G=(V,E,w)$ is built due to the weight contributions of one or more of these $2N+2n+N_o$ sets, these sets provide us with a heuristic way around the NP-complete problem of determining all the weighted cliques from scratch. Of course, many weighted cliques are formed due to unions of some of these $2N+2n+N_o$ sets. We could merge some of these candidate weighted cliques as long as the result is still a weighted clique. However, determining a cohesive merger scheme for the best set of resulting weighted cliques seems to be a hard problem (potentially NP-complete). This is where the inaccuracy in this heuristic comes from.

4.2.2. *Pruning Weighted Cliques*

The set of potential weighted cliques for `bbtas` from Figure 10 is $\{C_2=\{st0,st1,st5\}$, $C_3=\{st0,st2\}$, $C_5=\{st2,st3\}$, $C_6=\{st3,st4\}$, $C_7=\{st4,st5\}$, $C_8=\{st0,st1,st2,st3,st4,st5\}$, $C_{10}=\{st1,st2,st3,st4,st5\}$, $C_{12}=\{st0,st1\}$, $C_{14}=\{st1,st3\}$, $C_{17}=\{st0,st5\}$. They are labeled by their row number in Figure 10, *i.e.*, set C_i is in ith row of Figure 10. We have eliminated duplicate sets and singletons. Not all of these sets are *weighted cliques*. Let us prune this set to include only the weighted cliques. We need to determine $avg\_weight(G)$ for the weighted graph in Figure 4. The weights in Figure 4 have been normalized so that they all add up to 1 and the weights for (s,s) pairs have been forced to be 0. Hence $avg\_weight(G)=1/15=.0667$, $NW(C_2)=.0517$, $NW(C_3)=.025$, $NW(C_5)=.0728$, $NW(C_6)=.1064$, $NW(C_7)=.0995$, $NW(C_8)=.0667$, $NW(C_{10})=.0765$, $NW(C_{12})=.0279$, $NW(C_{14})=.0756$, $NW(C_{17})=.0584$. Hence pruning results in the set of weighted cliques: $\{C_5, C_6, C_7, C_8, C_{10}, C_{14}\}$.

4.2.3. *Overlapping Weighted Cliques*

One problem with the weighted clique set $\{C_5, C_6, C_7, C_8, C_{10}, C_{14}\}$ is that some states belong to several cliques. As we will see later, ensuring that this set consists of disjoint sets often results in better and more efficient state assignment algorithms. In this set: $st0 \in C_8$; $st1 \in C_8, C_{10}, C_{14}$; $st2 \in C_5, C_8, C_{10}$; $st3 \in C_5, C_6, C_8, C_{10}, C_{14}$; $st4 \in C_6, C_7, C_8, C_{10}$; $st5 \in C_7, C_8, C_{10}$.

DEFINITION 4 Affinity of a state $s \in C$ to a clique C is defined as $a\ f\ finity\ (s,C)=(\sum_{s' \in C} w(s,s')/(|C|-1))$.

Affinity captures the *average attraction* of a state to a clique. It is designed to be a measure of how important is a clique to a state or *vice versa*. In order to generate a disjoint set of weighted cliques, we force a state s to a clique with maximum affinity. For example, state $st1$'s affinities are: $a\ f\ finity(st1, C_8)=.0553$, $a\ f\ finity(st1, C_{10})=.06215$, $a\ f\ finity(st1, C_{14})=.0756$. Hence $st1$ is forced into C_{14}. Similarly, $a\ f\ finity(st2, C_5)=.0728$, $a\ f\ finity(st2, C_8)=.053$, $a\ f\ finity(st2, C_{10})=.06$, and hence $st2$ is forced into C_5. Computing other affinities forces $st3$ to C_6, $st4$ to C_6, and $st5$ to C_7. This gives the set of disjoint weighted cliques C as: $\{C_8=\{st0\}, C_{14}=\{st1\}, C_5=\{st2\}, C_6=\{st3,st4\}, C_7=\{st5\}\}$.

Figure 11 summarizes these steps in clique computation.

We also maintain several other data structures to help the bipartitioning based state assignment. An array $CLIQUE\_MEMBER[2N+2n+N_o][N]$ maintains the membership function for all cliques. $CLIQUE\_MEMBER[i][j]=1$ iff the state s_j is member of clique C_i. For disjoint cliques case, array $CLIQUE[N]$ keeps track of clique assigned to each state, *i.e.*, $CLIQUE[j]=i$ iff s_j is in clique C_i. We also support state assignment with cliques that are not disjoint (or are overlapping cliques). For overlapping cliques case, an array $CLIQUE[N][2N+2n+N_o]$ keeps track of all the cliques containing a given state. If a state s_j belongs to cliques $C_{i_1}, C_{i_2}, \ldots, C_{i_l}$ then $CLIQUE[j][i_1]=CLIQUE[j][i_2]=\cdots=CLIQUE[j][i_l]=1$. We also keep the cliques sorted by sizes and weights. In addition, we keep all the members of a clique sorted by their affinities to that clique.

procedure CLIQUE_COMPUTATION($G = (V, E, w)$); **sets** $S_{i,k}, PS_{i,k}$, $1 \le i, k \le N$; $O_{i,k}$, $1 \le i \le N$, $1 \le k \le N_o$; $I^0_{i,k}, I^1_{i,k}$, $1 \le i \le N$, $1 \le k \le n$);

potential weighted cliques:

$C_i = \bigcup_{1 \le j \le N} O_{j,i}$, $0 \le i < N_o$;

$C_i = \bigcup_{1 \le j \le N} S_{j,i}$, $N_o \le i < N + N_o$;

$C_i = \bigcup_{1 \le j \le N} I^0_{j,i}$, $N + N_o \le i < N + N_o + n$;

$C_i = \bigcup_{1 \le j \le N} I^1_{j,i}$, $N + N_o + n \le i < N + N_o + 2n$;

$C_i = \bigcup_{1 \le j \le N} PS_{j,i}$, $N + N_o + 2n \le i < 2N + N_o + 2n$;

prune cliques:

$avg\_weight(G) = \frac{\sum_{s_i, s_j \in V} w(s_i, s_j)}{0.5 * |V| * (|V| - 1)}$;

$\forall i$, $0 \le i < 2N + 2n + N_o$, **if** $NW(C_i) < 2/(N * (N - 1))$ **then** discard C_i;

Let $C_{i_1}, C_{i_2}, \ldots, C_{i_l}$ be the weighted cliques for $0 \le i_1, i_2, \ldots, i_l < 2N + 2n + N_o$;

disjoint cliques:

compute $affinity(s_j, C_{i_q})$ for $1 \le j \le N$ and $1 \le q \le l$;

Let j_{max} be such that $affinity(s_j, C_{j_{max}}) = \max_{1 \le q \le l} affinity(s_j, C_{i_q})$;

Delete s_j from every weighted clique except $C_{j_{max}}$ to give the set of weighted cliques C;

FIGURE 11 Clique computation.

The *experimental choices* offered by the clique computation phase are with respect to disjoint or overlapping clique covers of G.

4.3. Recursive Bipartitioning Induced State Assignment

We use a recursive bipartitioning based scheme to generate state codes to minimize the objective function $\sum_{s_i, s_j \in V} w(s_i, s_j) * h(c(s_i), c(s_j))$. A bipartition based state assignment was used in [17] and [4]. The objective function minimized in the state assignments of [17] and [4] $(\sum_{s_i, s_j \in Q} p(s_i, s_j) * h(c(s_i), c(s_j)))$ is slightly different than our objective function. The bipartitioning based state assignment algorithms are simple adaptations of the well established mincut heuristics employed for circuit placement and layout synthesis. For a detailed description of the layout mincut heuristic, the reader is referred to [7, pp. 227–230]. Note that the objective functions for both layout synthesis and state assignment have similar forms: $\sum_{v_i, v_j \in V} w(v_i, v_j) * l(v_i, v_j)$ for layout synthesis *versus* $\sum_{s_i, s_j \in V} w(s_i, s_j) * h(c(s_i), c(s_j))$ for state assignment. v_i and v_j are gates in a circuit and $l(v_i, v_j)$ is an estimate of the complexity of wiring between them. In a layout, one is willing to accept long connections with low weights in order to make the lengths of the connections with high weights small. Similarly, we may let the Hamming distance of the low frequency transitions be high so that the high-frequency or high weight transitions can be assigned low Hamming distance codes. A mincut bipartition keeps the clusters of states with high affinity together (in the same set) as late in recursion as possible. This results in low hamming distance codes for the states in such a cluster. The generic bipartitioning based state assignment algorithm is presented in Figure 12. Note that the weights w can either be affinity weights as described in Section 3 or could be steady-state transition probabilities $p(s_i, s_j)$ depending on the objective function. Procedure **assign_codes** assigns state codes to $c(s_i) = c^{\log N}(s_i) \cdots c^2(s_i) c^1(s_i)$ where $c^{\log}$ $c^{\log N}(s_i)$ is the most significant bit of the code for s_i. It calls a recursive procedure **assign_state_bit** that assigns to the tth bit of $c(s_i)$ for all s_i. We use a variant of Kernighan-Lin mincut heuristic [6] for bipartitioning a given set of vertices V with cost matrix w in procedure *bipartition*. The procedure

```
procedure assign_codes(G = (V, E, w), c(s_i) for 1 ≤ i ≤ N);
  t = log N;
  assign_state_bit(V, w, c(s_i) for 1 ≤ i ≤ N, t);
end procedure assign_codes

procedure assign_state_bit(V, w, c(s_i) for 1 ≤ i ≤ N, t);
  A, B = ∅;
  bipartition(V, w, A, B);
  For all s_i ∈ A, c^t(s_i) = 0;
  For all s_i ∈ B, c^t(s_i) = 1;
  if (t = 1) then return
  else
    assign_state_bit(A, w, c(s_i) for 1 ≤ i ≤ N, t − 1);
    assign_state_bit(B, w, c(s_i) for 1 ≤ i ≤ N, t − 1);
  end if
end procedure assign_state_bit
```

FIGURE 12 Bipartitioning based state assignment algorithm.

bipartition returns a partition of set V into two disjoint sets A and B with $|A| = |B| \pm 1$ such that $cut(A, B) = \sum_{s \in A, s' \in B} w(s, s')$ is minimum over all balanced partitions (A', B') with $|A'| = |B'| \pm 1$.

The recursion in **assign_state_bit** goes through $\log N$ depth for an N state FSM. We illustrate bipartitioning based state assignment for `bbtas` in Figure 13. Note that the values for w are as given in Figure 4. At the top level in recursion, $V = \{st0, st1, st2, st3, st4, st5\}$ is partitioned into $A = \{st0, st1, st2\}$ and $B = \{st3, st4, st5\}$. One can verify that it is a mincut. Set $A(B)$ is assigned a 0 (1) for the most significant state code bit. The state codes at the leaf nodes of the recursion can be read as the labeling from the root, *e.g.*, $c(st1) = 010$.

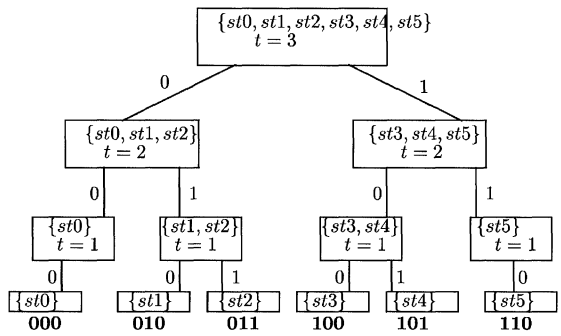

FIGURE 13 Illustration of bipartitioning based assignment for `bbtas`.

4.3.1. *Clique Bipartitioning*

The state assignment algorithm in Figure 12 can be adapted to use the additional information about the weighted cliques in the weighted graph $G = (V, E, w)$. In the process, one hopes that the result of the bipartitioning heuristic would improve.

4.3.2. *Weighted Clique Graph*

The first step is to build a clique weighted graph $G_C = (C, E_C, w_C)$ from $G(V, E, W)$ whose vertices represent the cliques from the set C provided by the clique computation in Figure 11. For instance, for `bbtas` we computed $C = \{C_8 = \{st0\}, C_{14} = \{st1\}, C_5 = \{st2\}, C_6 = \{st3, st4\}, C_7 = \{st5\}\}$. E_C represents a complete graph, *i.e.*, $E_C = C \times C$. The function w_C is derived from w so that inter-state weights can be translated into inter-clique weights. A simple definition for w_C is $w_C(C_i, C_j) = \sum_{s \in C_i, s' \in C_j} w(s, s')$. However, there is a problem with this definition. It rewards large cliques with

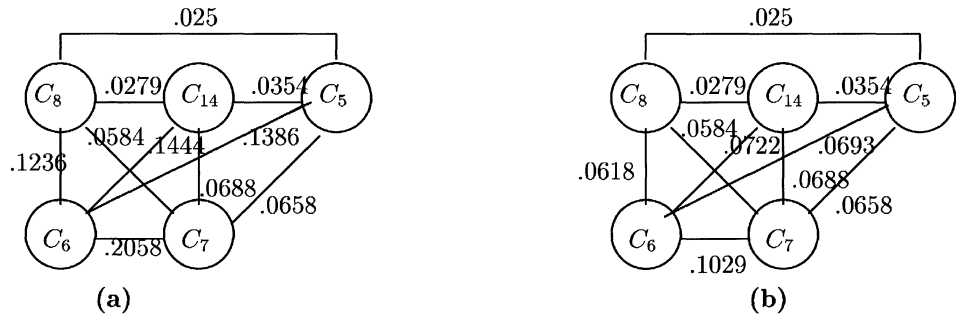

FIGURE 14 w_C values for `bbtas`, aggregate and normalized.

large weights. Figure 14(a) shows the weights computed according to this form. Note that the edges incident on C_6 have higher than average weights since they are the sum of two edge weights from Figure 4, *e.g.*, $w_C(C_8, C_6) = w(st0, st3) + w(st0, st4)$. A better model than aggregation of the edge weights from G would be to normalize these weights by the size of the clique. This puts a multi-state clique on the same footing as a singleton clique. The normalized form for w_C is:

$$w_C(C_i, C_j) = \frac{\sum_{s \in C_i, s' \in C_j} w(s, s')}{|C_i| * |C_j|}$$

Figure 14(b) gives the normalized weights for G_C. We work with the normalized w_C function unless otherwise specified.

4.3.3. Clique Bipartitioning

Figure 15 gives the state assignment algorithm based on bipartitioning of clique graph $G_C = (C, E_C, w_C)$. Procedure **clique_assign_codes** checks to see if the clique cover C has only one clique in it, *i.e.*, all the N states got clustered into a single clique by the clique computation phase. In this case, we resort to the state bipartitioning procedure **assign_codes** of Figure 12. Otherwise, the procedure **assign_clique_bit** is invoked. The set V is the set of all states to be assigned codes and set C is the clique cover of V, either disjoint or overlapping. The first step in **assign_clique_bit** is to see if there is only one clique left in C. For a single clique case, the recursive state based bipartitioning in **assign_state_bit** is used with the state set V. Otherwise, *clique_bipartition* is used to partition the set of cliques C into two disjoint sets C_A and C_B s.t. $C = C_A \cup C_B$. Once again, a variant of Kernighan-Lin mincut heuristic [6] is used. However, the balancing condition is different this time. Let us introduce some notation before discussing the balancing condition.

DEFINITION 5 Let C be a set of cliques over a set of states V, *i.e.*, $C \subseteq 2^V$, where 2^V is the power set of V. $|C|$ specifies the number of subsets of V in C. The number of elements of V in all the subsets of C are denoted by $\#C$, *i.e.*, $\#C = |\bigcup_{C_i \in C} C_i|$.

Note that the set C is bipartitioned such that some cliques from C are in C_A and some are in C_B. However, we wish to keep the number of states on the two sides as close to each other as possible. This means that $ABS(\#C_A - \#C_B)$ should be as small as possible, ABS represents absolute value. We also wish to allow enough flexibility to the bipartitioning procedure so that the largest clique in C can be moved between the two sides. Let the largest clique in C consist of $|C_{max}|$ states, *i.e.*, $|C_{max}| \geq \max_{C_i \in C} |C_i|$ for $C_{max} \in C$. We would allow all the bipartitions of C into (C_A, C_B) s.t. $ABS(\#C_A - \#C_B) \leq |C_{max}|$. The procedure *clique_bipartition* in Figure 15 finds such a balanced partition (C_A, C_B) with the minimum value for the $cut(C_A, C_B) = \sum_{C_i \in C_A, C_j \in C_B} w_C(C_i, C_j)$.

```
procedure clique_assign_codes(G = (V, E, w), G_C = (C, E_C, w_C), c(s_i) for 1 ≤ i ≤ N);
  if (|C| = 1) then assign_codes(G = (V, E, w), c(s_i) for 1 ≤ i ≤ N); return;
  t = log N;
  assign_clique_bit(V, w, C, w_C, c(s_i) for 1 ≤ i ≤ N, t);
end procedure clique_assign_codes

procedure assign_clique_bit(V, w, C, w_C, c(s_i) for 1 ≤ i ≤ N, t);
  if (|C| = 1) then assign_state_bit(V, w, c(s_i) for 1 ≤ i ≤ N, t); return;
  C_A, C_B, A, B = ∅;
  clique_bipartition(C, w_C, C_A, C_B);
  Generate A, B consistent with C_A, C_B:
   ∀s_i ∈ V, if (s_i ∈ C_j and C_j ∈ C_A) then A = A ∪ {s_i}; else B = B ∪ {s_i};
   for overlapped clique partitions:
    ∀s_i ∈ V, if (affinity(s_i, C_A) ≥ affinity(s_i, C_B)) then
      A = A ∪ {s_i};
      Delete s_i from every clique in C_B;
     else
      B = B ∪ {s_i};
      Delete s_i from every clique in C_A;
  if (balanced) rebalance(C_A, C_B, A, B, w, w_C);
  For all s_i ∈ A, c^t(s_i) = 0;
  For all s_i ∈ B, c^t(s_i) = 1;
  if (t = 1) then return;
  else
   assign_clique_bit(A, w, C_A, w_C, c(s_i) for 1 ≤ i ≤ N, t − 1);
   assign_state_bit(B, w, C_B, w_C, c(s_i) for 1 ≤ i ≤ N, t − 1);
  end if
end procedure assign_clique_bit
```

FIGURE 15 Clique bipartitioning based state assignment algorithm.

4.3.4. *Overlapped Cliques and Consistent State Sets*

The next step is to generate state sets A and B that are consistent with C_A and C_B, *i.e.*, $s \in A$ iff $C_i \in C_A$ and $s \in C_i$ and $s \in B$ iff $C_j \in C_B$ and $s \in C_j$. This step is somewhat more complicated if overlapping cliques were generated by the clique computation phase. Note that a state s could be in multiple cliques $C_{i_1}, C_{i_2}, \ldots, C_{ik1}, C_{j_1}, C_{j_2}, \ldots, C_{jk2}$ such that $C_{i_1}, C_{i_2}, \ldots, C_{ik1} \in C_A$ and $C_{j_1}, C_{j_2}, \ldots, C_{jk2} \in C_B$. The method of allocating s to either A or B described above would allocate it to both A and B! Let us define affinity of a state s to a set of cliques C:

DEFINITION 6 Let $G = (V, E, w)$ be a weighted graph. Let $C \subseteq 2^V$. For $s \in V$, $a\ f\ finity\ (s, C) = \sum_{C_i \in C} a\ f\ finity(s, C_i)$.

For overlapped cliques, s goes to A if $a\ f\ finity(s, C_A) \geq a\ f\ finity(s, C_B)$, otherwise s is allocated to B.

4.3.5. *Rebalancing*

Note that we have allowed the difference in sizes of A and B after each stage of clique bipartitioning to be as large as the number of states in the largest clique. This can be a very large imbalance and can

result in an imbalanced recursion tree. For example, consider the recursive bipartitioning tree for `bbtas` shown in Figure 17. Here the initial clique set is $C=\{\{st0\},\{st1,st2,st3,st4,st5\}\}$. The balancing factor allows the state sets at this level to be unbalanced by as many as 5 states. A mincut with $C_A=\{\{st0\}\}$ and $C_B=\{\{st1,st2,st3,st4\}\}$ results in $A=\{st0\}$ and $B=\{st1,st2,st3,st4,st5\}$. Here $ABS(|A|-|B|)=4$ which is within bounds. However, an unbalanced partition results in higher than necessary bits for state codes, 4 in this example rather than 3. Hence, one choice is to rebalance sets A and B (with necessary adjustments in C_A and C_B) such that $|A|=|B|\pm 1$. Procedure **rebalance** in Figure 16 does exactly this.

We first determine in **rebalance** which of the two sets A and B has surplus states. The variable ev is the number of states that need to be moved from the surplus set to the deficient set. We have assumed, without loss of generality, that A has surplus states. At this point, some of the cliques from C_A need to be moved to C_B. The clique in C_A that is least important (C_i), or that has the least normalized weight ($NW(C_i)$), is moved first. If C_i has fewer than ev states, we repeat this procedure. If $|C_i|<ev$, then we wish to move only ev states from C_i. This is done by choosing ev states in C_i that have least affinity to C_i, since they will perturb the objective function the least.

4.3.6. *Example Clique State Assignment*

Figure 18 shows the state assignment based on clique bipartitioning for `bbtas`. Note that the initial clique cover $C=\{C_5,C_6,C_7,C_8,C_{14}\}$ is from the clique computation phase. The corresponding weight function w_C is given in Figure 14(b). At first bi-partition stage, $|C_{max}|=2$, since C_6 has 2 states. The mincut partition is $C_A=\{C_5,C_8,C_{14}\}$ and $C_B=\{C_6,C_7\}$. Since $|A|=|B|=3$ for this partition, no rebalancing need be done. At the node where $C=\{\{C_6\}\}$, procedure **assign_state_bit** would be invoked to assign the least significant state bits for $st3$ and $st4$. The resulting state codes label the leaf nodes in Figure 7.

Note that clique based state assignment offered many experimental choices. Salient ones are as follows. Is the balance going to be maintained on the state sets at each recursion stage? Are the cliques disjoint? What should be done with the $2^{\lceil \log N\rceil}-N$ unused codes? We report on the results of these variations in the next section.

```
procedure rebalance(C_A, C_B, A, B, w, w_C);
  if (ABS(|A| - |B|) = ev ≤ 1) return;
  Without loss of generality, assume |A| - |B| = ev > 1;
  repeat
    Let C_A-min ∈ C_A be the clique with smallest normalized weight NW(C_A-min);
    if (|C_A-min| > ev) then
      Break C_A-min into C^1_A-min and C^2_A-min (C_A-min = C^1_A-min ∪ C^2_A-min) as follows;
      Move ev states s in C_A-min with smallest value for affinity(s, C_A-min) to a new set
        C_move = C^2_A-min; C^1_A-min = C_A-min - C^2_A-min; C_A-min = C^1_A-min;
    else
      C_move = C_A-min;
    end if;
    Move C_move from C_A to C_B and move all the states s ∈ C_move from A to B;
    ev = ev - |C_move|;
  until (ev ≤ 1);
end procedure rebalance;
```

FIGURE 16 Rebalancing for clique partitions.

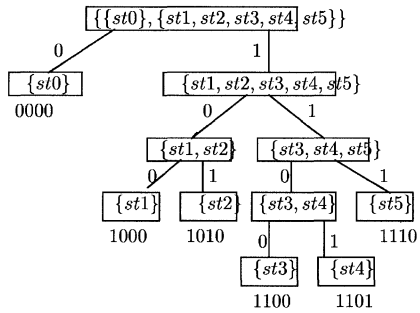

FIGURE 17 A hypothetical clique bipartitioning for `bbtas` with unbalanced state sets.

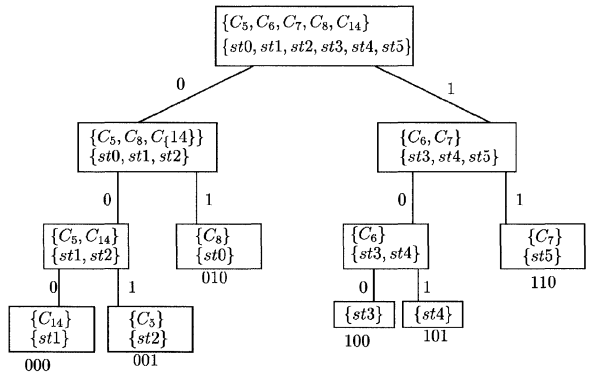

FIGURE 18 Clique bipartitioning based state assignment for `bbtas`.

5. EXPERIMENTAL RESULTS

We implemented the weight computation methods described in Section 3 for fanin, fanout, and FIFO weights for both Energy model 1 and 2. The state assignment algorithms from Section 4.3 were also implemented – plain state bipartitioning, clique computation, clique bipartitioning, balanced partitions at each stage *versus* no rebalancing, disjoint cliques *versus* overlapping cliques. We also modified `jedi` [8] to accept energy modeling weights for an annealing based state assignment. Table I outlines all the experiments conducted and their clique and weight, and bipartitioning parameters. Note that for the experiments where bipartitioning is not balanced at each stage (such as FOC-HLATCH and FOC-OVERLAP-HLATCH), the number of latches used in encoding is not necessarily minimal (log N). This is because the recursive bipartitioning tree is not necessarily balanced as shown in Figure 17, especially if the cliques are of uneven sizes. However, the number of bits used in encoding depends entirely on the clique size distribution in the FSM. Hence one hopes that the number of latches used (or information expansion) would result in lowering of average switching per transition which may also result in lower power. Tyagi [15] shows that when n bit of information is expanded into $m \geq n$ bits, the average switching is reduced to $(n/2 \log(2m/n))$. For three of the benchmark circuits, a larger number of latches resulted surprisingly in lower area and lower power than the minimal number of latches case. However, in most of the cases, these experiments did not yield competitive results.

All of the experimental cases in Table I were tested on 29 of the MCNC logic synthesis '93 benchmarks. Before we look at the area and power numbers for these experiments, let us look at the distribution of cliques in the disjoint and overlapping cliques cases. This is given in Table II. Note that some of the benchmarks end up having a single clique of N states in the disjoint cliques case. Hence clique information does not really help at all in these instances. Also notice that the overlapped clique case provides a much better structural information with many more cliques. Unfortunately, it is very hard to handle the overlapped clique case properly in bipartitioning. We are considering some other ways of clique pruning for disjoint clique case that would retain more of the information about cliques potentially resulting in better state assignments.

The state codes generated by these algorithms were used to generate a 2-level logic description with NOVA. This 2-level description in Berkeley Logic Intermediate Form (BLIF) [12] was optimized for a multilevel logic implementation in SIS [12]. We used an industrial strength technology library from Intel for technology mapping. The SIS optimization was done with the `rugged` script.

TABLE I Experimental parameters

Name	Weight, clique and st assignment
FI	Fanin based weights, no cliques, state based vanilla bisection
FO	Fanout based weights, no cliques, state based, vanilla bisection
FIFO	Fanin + Fanout based weights, no cliques, state based vanilla bisection
FI-CLIQUES	Fanin based weights, disjoint cliques, clique based balanced bisection
FO-CLIQUES	Fanout based weights, disjoint cliques, clique based balanced bisection
FIFO-CLIQUES	Fanin + Fanout based weights, disjoint cliques, clique based balanced bisection
FOC-HLATCH	Fanout based weights, disjoint cliques, clique based not balanced bisection
FOC-OVERLAP	Fanout based weights, overlapping cliques, clique based balanced bisection
FIFOC-OVERLAP	Fanin + Fanout based weights overlapping cliques, clique based balanced bisection
FOC-OVERLAP -HLATCH	Fanout based weights, overlapping cliques, clique based not balanced bisection
FIFOC-EM1	Same as FIFO-CLIQUES, Energy Model 1 instead of Energy Model 2
FIFO-ANNEAL	FIFO Energy Model 1 weights, JEDI annealing state assignment

Our comparisons are based on the area and power (5 V and 20 MHz assumption) numbers from SIS.

For the steady state probability computation, we assumed the 1-probability of all primary input bits to be 0.5. Second and third columns in Table III report the area-power numbers for JEDI using *coupled dominant* or FIFO based area modeling weights and annealing based state assignment. These results are compared to state bipartitioning FIFO (FIFO), clique bipartitioning FIFO (FIFO-CLIQUES), overlapped cliques FIFO (FIFOC-OVERLAP), clique bipartitioning FIFO with Energy model 1 (FIFOC-EM1) and Energy model 1 weights with annealing assignment (FIFO-ANNEAL) respectively. The last row in Table III shows the average values for area and power over the 29 MCNC benchmarks. We also report the relative aggregate area and power with respect to JEDI. For instance, FIFO-CLIQUES has 2.9% less aggregate area and 9.84% less aggregate power than JEDI, *i.e.* $(23541-22858/23541)=-2.9\%$. Table IV reports the area and power for each benchmark as a percentage reduction from JEDI. For example, for `bbara` FIFO-CLIQUES has 8.5% higher area and 9.5% less power than JEDI, *i.e.*, for FIFO-CLIQUES `bbara` area is $10027*(1+0.08507)$ and power is $168*(1-0.09524)$. The bottom row reports the average of these percentage reductions for a given state assignment method.

Note that among all these implementations, FIFO-ANNEAL performs the best with an aggregate area advantage of 8.86% and power advantage of 17.69% over JEDI. The area reduction accompanying the power reduction is a big surprise. The only other study targeting multilevel logic synthesis weights [14] reports a power advantage of 15.64% with an area increase (literal count increase) of 3.2% over 20 MCNC logic synthesis benchmarks (13 of them are also in our 29 benchmarks). This state assignment algorithm would be of use where one does not wish to give up any area, but still gain some power advantage over conventional area-optimizing state assignment algorithms. Note that FIFOC-EM1 performs almost as well as FIFO-ANNEAL. This is not surprising given that they both use FIFO Energy

TABLE II The distribution of clique sizes

Name	Disjoint (size: count)	Overlapping (size:count)
bbara	(2:2),(1:6)	(10:8),(9:2),(4:5),(3:3),(2:6),(1:2)
bbtas	(1:4), (2,1)	(6:2),(5:2),(3:1),(2:8),(1:3)
beecount	(2:1),(1:5)	(7:6),(5:2),(4:1),(3:9),(2:2),(1:1)
cse	(2:5),(1:6)	(16:7),(15:2),(14:1),(13:1),(12:1), (11:3),(6:1),(5:2),(4:6),(3:12),(2:10)
dk14	(2:1),(1:5)	(7:2),(6:3),(5:5),(4:10),(2:1),(1:1)
dk15	(1:4)	(4:10),(3:5),(1:1)
dk16	(3:1),(2:9),(1:6)	(23:1),(21:1),(17:1),(16:1),(15:1), (11:1),(7:4),(6:3),(5:6),(4:24), (3:2),(2:10),(1:4)
dk17	(2:1),(1:6)	(7:2),(6:4),(5:1),(4:4),(3:4), (2:3),(1:3)
dk27	(1:7)	(5:1),(3:4),(2:7),(1:5)
ex1	(2:2),(1:16)	(20:10),(19:5),(18:2),(17:3),(15:2), (14:1),(11:1),(9:1),(6:2),(5:7), (4:5),(3:8),(2:14),(1:7)
keyb	(3:1),(1:16)	(19:7),(18:3),(17:1),(15:1),(13:1), (11:2),(9:1),(8:1),(4:1),(3:11), (2:6),(1:12)
kirkman	(1:16)	(16:30),(2:4),(1:16)
lion	(2:1),(1:2)	(4:3),(3:4),(2:4)
mc	(2:1),(1:2)	(4:7),(2:7),(1:2)
opus	(2;2),(1:6)	(10:10),(9:2),(4:1),(3:5),(2:11),(1:2)
planet	(6:1),(4:1),(3:2), (2:1),(1:30)	(48:10),(47:3),(46:1),(45:1),(41:1), (39:1),(36:1),(34:2),(24:1),(22:1), (21:1),(20:2),(14:1),(13:1),(11:1), (4:5),(3:7),(2:25),(1:57)
pma	(5:1),(2:3),(1:13)	(24:4),(23:1),(22:3),(21:2),(19:2), (16:1),(14:3),(13:3),(12:2),(10:1), (8:1),(7:1),(4:2),(3:5),(2:13),(1:20)
s1	(2:5),(1:10)	(20:11),(18:3),(17:2),(16:2),(9:1), (8:4),(6:9),(5:1),(4:4), (3:7),(2:8),(1:2)
s1488	(2:4),(1:40)	(48:12),(47:3),(46:1),(44:1),(41:1), (38:1),(33:1),(21:1),(20:2),(19:1), (13:2),(8:1),(5:2),(4:8),(3:12), (2:41),(1:33)
s1494	(2:4),(1:40)	(48:12),(47:3),(46:1),(44:1),(41:1), (38:1),(33:1),(21:1),(20:2),(19:1), (13:2),(8:1),(5:2),(4:7),(3:16), (2:37),(1:34)
s27	(3:1),(2:1),(1:1)	(6:8),(5:1),(4:4),(3:3),(2:1)
s386	(2:1),(1:11)	(13:9),(12:5),(5:2),(4:2),(3:7), (2:10),(1:5)
s510	(10:1),(7:1),(1:30)	(47;24),(46:10),(45:2),(44:1),(43:1), (42:1),(28:1),(22:1),(8;1), (5;1),(3:4),(2:40),(1:33)
s820	(2:5),(1:15)	(25:30),(24:4),(17:1),(16:2),(15:1), (11:2),(5:2),(4:3),(3:6), (2:22),(1:14)
s832	(2:5),(1:15)	(25:30),(24:4),(17:1),(16:2),(15:1), (11:2),(5:2),(4:3),(3:5), (2:23),(1:14)
sand	(3:2),(2:10),(1:6)	(32:16),(31:5),(29:1),(27:1),(24:1), (22:1),(21:1),(15;1),(12:1),(11:2), (8:1),(7:2),(6:1),(5:3),(4:2), (3:17),(2:25),(1:3)
shiftreg	(2:4)	(4:3),(2:15)
styr	(3:4),(2:5),(1:8)	(30:11),(28:1),(27;2),(26:3),(25:1), (23:1),(22:1),(20:1),(19:2),(17:2), (12:2),(9:1),(8:1),(5:5),(4:9), (3:10),(2:18),(1:8)
tav	(1:4)	(4:12),(1:4)

model 1 weights. However, annealing based state assignment takes almost twice as much time on average as shown in Table V. This is the execution time of the assignment phase on an HP 715/33 workstation. The entries reporting 0 time were too small to be measured by the UNIX *time* utility. Note that FIFOC-EM1 is a polynomial time heuristic, *i.e.*, it takes time polynomial in $N+n+N_o$. This is the number of cliques handled in the worst case. The dominating time factor comes from the Kernighan-Lin bipartitioning of the clique graph with up to $2N+2n+N_o$ vertices. Kernighan-Lin heuristic takes time proportional to $m^2 \log m$ for an m-vertex graph. There are about 10 passes of this heuristic giving FIFOC-EM1 time as proportional to $(2N+2n+N_o)^2 \log(2N+2n+N_o)$. However, FIFO-ANNEAL can take time exponential in number of states (N). Hence, it is likely that the execution time gap may be even wider for larger state machines.

TABLE III Area-power comparison of some experiments with JEDI

	JEDI		FIFO		FIFO-CLIQUES		FIFOC-OVERLAP		FIFOC-EM1		FIFO-ANNEAL	
Name	JArea	JPower	Area	Power	Area	Power	Area	Power	Area	Power	Area	Power
bbara	10027	168	10708	188	10880	152	10815	168	10120	148	9912	143
bbtas	7470	108	5827	156	5666	112	5746	126	6047	104	6047	104
beecount	8784	183	7480	194	6927	184	6909	175	6288	192	6246	190
cse	23621	481	22398	411	20613	377	21966	376	21260	386	22458	406
dk14	15251	573	10516	496	10990	501	10474	508	10789	509	10789	509
dk15	11719	433	8558	419	8803	430	8446	443	7865	407	7865	407
dk16	27187	1275	25310	1143	24494	1175	25303	1113	23556	1050	23556	1050
dk17	10507	389	9115	418	8607	338	8326	343	8126	392	8126	392
dk27	7228	248	6169	284	5338	241	5247	230	5556	232	5556	232
ex1	27385	768	25773	656	26519	641	27567	578	25724	681	23849	637
keyb	26138	745	27471	667	23796	504	26502	579	23412	524	22594	546
kirkman	19526	667	20253	653	20547	736	20581	730	18202	558	18202	558
lion	4761	89	3366	86	3344	82	3366	86	3625	73	3747	68
mc	5904	120	4853	114	4559	102	4835	116	4559	102	4559	102
opus	11682	325	11087	232	12157	291	12461	296	10567	253	10567	253
planet	57317	2510	53382	2380	51885	2282	54272	2502	52328	2107	55577	2093
pma	26470	1120	24488	913	22238	833	22692	871	21991	741	21991	741
s1	23250	755	26290	828	21335	637	27215	782	21766	688	21766	688
s1488	57762	1222	60293	1098	63629	1122	62664	1119	57735	976	57735	976
s1494	58903	1156	61977	1055	61514	1113	63159	1129	59624	952	59352	909
s27	5930	105	4889	111	7113	229	7313	282	5116	143	5116	143
s386	16323	454	16514	450	15393	407	21162	400	16758	466	16758	466
s510	35902	911	44040	1357	41098	1192	36944	982	29123	786	29123	786
s820	31857	1026	31795	753	39184	883	38029	826	34580	776	34580	776
s832	32526	1023	38477	854	33748	784	33867	773	35056	816	35056	816
sand	53906	1541	62488	1872	56522	1583	64940	1690	50392	1411	49939	1371
shiftreg	5203	148	3384	96	3384	96	3581	99	3187	94	3187	94
styr	53906	1541	51959	1080	48063	1077	54858	1197	44868	1078	43555	1061
tav	6244	245	4559	237	4559	237	4559	237	4413	220	4413	220
AVG	23541	701	23566	662	22858	632	23924	646	21470	581	21455	577
%JEDI			0%	−5.56%	−2.9%	−9.84%	1.63%	−7.85%	−8.8%	−17.12%	−8.86%	−17.69%

TABLE IV Area-power of some experiments as a percentage of JEDI

	JEDI		FIFO		FIFO-CLIQUES		FIFOC-OVERLAP		FIFOC-EM1		FIFO-ANNEAL	
Name	Area	Power	%Area red.	%Pwr red.	%Area red.	%Pwr red.	%Area red.	%Pwr red.	%Area red.	%Pwr red.	%Area red.	%Pwr red.
bbara	10027	168	6.792	11.905	8.507	−9.524	7.859	0.000	0.927	−11.905	−1.147	−14.881
bbtas	7470	108	−21.995	44.444	−24.150	3.704	−23.079	16.667	−19.050	−3.704	−19.050	−3.704
beecount	8784	183	−14.845	6.011	−21.141	0.546	−21.346	−4.372	−28.415	4.918	−28.893	3.825
cse	23621	481	−5.178	−14.553	−12.734	−21.622	−7.006	−21.830	−9.995	−19.751	−4.924	−15.593
dk14	15251	573	−31.047	−13.438	−27.939	−12.565	−31.323	−11.344	−29.257	−11.169	−29.257	−11.169
dk15	11719	433	−26.973	−3.233	−24.883	−0.693	−27.929	2.309	−32.887	−6.005	−32.887	−6.005
dk16	27187	1275	−6.904	−10.353	−9.905	−7.843	−6.930	−12.706	−13.356	−17.647	−13.356	−17.647
dk17	10507	389	−13.248	7.455	−18.083	−13.111	−20.758	−11.825	−22.661	0.771	−22.661	0.771
dk27	7228	248	−14.651	14.516	−26.148	−2.823	−27.407	−7.258	−23.132	−6.452	−23.132	−6.452
ex1	27385	768	−5.886	−14.583	−3.162	−16.536	0.665	−24.740	−6.065	−11.328	−12.912	−17.057
keyb	26138	745	5.100	−10.470	−8.960	−32.349	1.393	−22.282	−10.429	−29.664	−13.559	−26.711
kirkman	19526	667	3.723	−2.099	5.229	10.345	5.403	9.445	−6.781	−16.342	−6.781	−16.342
lion	4761	89	−29.301	−3.371	−29.763	−7.865	−29.301	−3.371	−23.861	−17.978	−21.298	−23.596
mc	5904	120	−17.801	−5.000	−22.781	−15.000	−18.106	−3.333	−22.781	−15.000	−22.781	−15.000
opus	11682	325	−5.093	−28.615	4.066	−10.462	6.668	−8.923	−9.545	−22.154	−9.545	−22.154
planet	57317	2510	−6.865	−5.179	−9.477	−9.084	−5.313	−0.319	−8.704	−16.056	−3.036	−16.614
pma	26470	1120	−7.488	−18.482	−15.988	−25.625	−14.273	−22.232	−16.921	−33.839	−16.921	−33.839
s1	23250	755	13.075	9.669	−8.237	−15.629	17.054	3.576	−6.383	−8.874	−6.383	−8.874
s1488	57762	1222	4.382	−10.147	10.157	−8.183	8.487	−8.429	−0.047	−20.131	−0.047	−20.131
s1494	58903	1156	5.219	−8.737	4.433	−3.720	7.225	−2.336	1.224	−17.647	0.762	−21.367
s27	5930	105	−17.555	5.714	19.949	118.095	23.322	168.571	−13.727	36.190	−13.727	36.190
s386	16323	454	1.170	−0.881	−5.697	−10.352	29.645	−11.894	2.665	2.643	2.665	2.643
s510	35902	911	22.667	48.957	14.473	30.845	2.902	7.794	−18.882	−13.721	−18.882	−13.721
s820	31857	1026	−0.195	−26.608	23.000	−13.938	19.374	−19.493	8.548	−24.366	8.548	−24.366
s832	32526	1023	18.296	−16.520	3.757	−23.363	4.123	−24.438	7.778	−20.235	7.778	−20.235
sand	53906	1541	15.920	21.480	4.853	2.726	20.469	9.669	−6.519	−8.436	−7.359	−11.032
shiftreg	5203	148	−34.961	−35.135	−34.961	−35.135	−31.174	−33.108	−38.747	−36.486	−38.747	−36.486
styr	53906	1541	−3.612	−29.916	−10.839	−30.110	1.766	−22.323	−16.766	−30.045	−19.202	−31.149
tav	6244	245	−26.986	−3.265	−26.986	−3.265	−26.986	−3.265	−29.324	−10.204	−29.324	−10.204
AVG	0	0	−6.7	−3.12	−8.39	−5.61	−4.64	−2.13	−13.55	−13.26	−13.65	−13.82

TABLE V Run times of FIFO-ANNEAL and FIFOC-EM1

Name	FIFO-ANNEAL Extime (seconds)	FIFOC-EM1 Extime (seconds)
bbara	0.4	0.2
bbtas	0.1	0.0
beecount	0.2	0.1
cse	1.9	0.7
dk14	0.2	0.1
dk15	0.1	0.0
dk16	12.5	11.2
dk17	0.2	0.2
dk27	0.1	0.1
ex1	5.0	4.3
keyb	3.5	4.1
kirkman	3.0	1.7
opus	0.4	0.4
planet	113.4	64.8
pma	8.2	8.2
s1	4.2	2.5
s1488	111.8	53.6
s1494	111.6	67.2
s386	1.0	0.9
s510	100.4	82.9
s820	11.1	11.6
s832	11.5	10.9
sand	23.8	12.3
shiftreg	0.1	0.1
styr	18.9	8.4
tav	0.1	0.0
AVG	20.91	13.32

6. CONCLUSIONS

This paper presents two improvements within the framework developed by MUSTANG in order to derive state assignments to optimize area and power simultaneously. The weight computation in the state affinity graph takes into account computation energy of the common cubes estimated from the steady state probabilities for each state and transition. Additionally the cliques suggested by the weight computation are incorporated into the state assignment procedure itself so that the states within a clique can be assigned low Hamming distance codes. The existence of these cliques was overlooked in MUSTANG. In the process, we are able to reduce the area requirements of JEDI by 8–9% and simultaneously reduce the power requirements by 18% for MCNC logic synthesis '93 benchmarks. The clique based state assignment seems to be almost as good in quality as the annealing based state assignment, but takes only about half as much time.

References

[1] Benini, L. and De Micheli, G., State assignment for low-power dissipation. *IEEE Journal of Solid State Circuits*, **11**(4), 32–40, March, 1995.

[2] Devadas, S., Ma, K., Newton, A. R. and Vincentelli, A., MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations. *IEEE Trans. on CAD*, **7**(12), December, 1988.

[3] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

[4] Hachtel, G. D., Hermida De La Rica, M., Pardo, A., Poncino, M. and Somenzi, F. (1994). Re-encoding Sequential Circuits to Reduce Power Dissipation. In: *Proc. of ICCAD*, pp. 70–73, ACM/IEEE.

[5] Hachtel, G. D., Macii, E., Pardo, A. and Somenzi, F. (1994). Probabilistic Analysis of Large Finite State Machines. In: *Proc. of DAC*, pp. 270–275, ACM/IEEE.

[6] Brian W. Kernighan and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, **49**, 291–307.

[7] Lengauer, T. (1990). *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons.

[8] Lin, B. and Newton, A. R., Synthesis of Multiple-Level Logic from Symbolic High-Level Description Languages. In: *Proceedings of IFIP International Conference on VLSI*, pp. 187–196, August, 1989.

[9] Monteiro, J., Devadas, S., Lin, B., Tsui, C. Y., Pedram, M. and Despain, A. M. (1994). Exact and Approximate Methods of Switching Activity Estimation in Sequential Logic Circuits. In: *Proc. of International Workshop on Low Power Design*, pp. 117–122, ACM/IEEE.

[10] Olson, E. and Kang, S. M. (1994). Low-Power State Assignment for Finite State Machines. In: *Proc. of International Workshop on Low Power Design*, pp. 63–68, ACM/IEEE.

[11] Roy, K. and Prasad, S. C., Circuit Activity Based Logic Synthesis for Low Power Reliable Operations. *IEEE Trans. on VLSI Systems*, pp. 503–513, December, 1993.

[12] Sentovich, E. M., Singh, K. J., Lavango, L., Moon, C., Muragi, R., Saldhana, A., Savoj, H., Stephen, P., Brayton, R. and Sangiovanni-Vincentelli, A., *SIS: A System for Sequential Circuit Synthesis*. Technical Report Memorandum Number UCB/ERL M92/41, Electronics Research Laboratory, Dept. of EECS, University of California, Berkeley, 1992.

[13] Shi, C. J. and Brzozowski, J. A. (1998). Cluster-cover a theoretical framework for a class of vlsi-cad optimization problems. *ACM Trans. on Design Automation of Electronic Systems*, **3**, 76–107.

[14] Tsui, C. Y., Pedram, M., Chen, C. and Despain, A. M. (1994). Low Power State Assignment Targeting Two- and Multi-level Logic Implementations. In: *Proc. of ICCAD*, pp. 82–87, ACM/IEEE.

[15] Tyagi, A., Energy-Time Trade-Offs in VLSI Computations. In: *Proceedings of the Ninth Conference on Foundations of Software Technology & Theoretical Computer Science*, pp. 301–311, Lecture Notes in Computer Science #405, Springer-Verlag, 1989. A revised version to appear in IEEE Trans on Computers.

[16] Tyagi, A. (1996). Entropic Bounds on FSM Switching. In: *Proc. of International Symposium on Low Power Electro-*

nics and Design, pp. 323–328. ACM/IEEE. A revised version to appear in IEEE Transactions on VLSI Systems, December, 1997.
[17] Veeramachaneni, V. M., Low Power State Assignments of FSMs. *Master's Thesis*, Department of Computer Science, Iowa State University, Ames, Iowa, May, 1995.
[18] Veeramachneni, V., Rajgopal, S. and Tyagi, A. (1995). Re-encoding for Low Power State Assignment of FSMs. In: *Proc. of ACM/IEEE International Symposium on Low Power Design*, pp. 173–178, ACM/IEEE.

Authors' Biography

Akhilesh Tyagi received B.E. (Honors) in Electrical and Electronics Engineering from (1981) Birla Institute of Technology and Science, Pilani followed by M. Tech. in computer Engineering (1983) from Indian Institute of Technology, New Delhi, India. He received Ph.D. in Computer Science from University of Washington, Seattle in 1988. He was on the faculty of the Departments of Computer Science at the University of North Carolina at Chapel Hill and Iowa State University, Ames. He is with the Department of Electrical and Computer Engineering at Iowa State University, Ames, Iowa since August of 1999. His research interests induce VLSI: complexity theory, design, synthesis and Computer architecture.