On Switching Aware Synthesis for Combinational Circuits

Jan Lanik[1] and Oded Maler[1]

VERIMAG
CNRS AND UNIVERSITY OF GRENOBLE
FRANCE
{jan.lanik,oded.maler}@imag.fr

Abstract. We propose a synthesis algorithm for combinational circuits which optimizes the expected number of gate switchings induced by typical sequences of input vectors. Our algorithm, which is based on simple observations concerning AND gates, performs quite well on sequences produced by the same probabilistic models used to generate the training sequences.

1 Introduction

Digital circuit synthesis [3,7,8,15] from higher level descriptions to technology dependent standard cells is one of the core activities in Electronic Design Automation (EDA), well-studied in academic research and implemented in powerful commercial tools. This is the hardware analog of optimizing compilation, and indispensable tool in producing efficient chips. Traditionally, the major optimization objectives in synthesis have been area and speed, associated with the depth of the circuit from primary inputs to outputs. In the last decades, power consumption has become a no less important performance measure for reasons that need not be repeated here [1,2,4,14]. In this work we develop a new synthesis algorithm geared toward reducing the expected number of switchings in the circuit, an important ingredient in its power consumption. This work can be viewed as part of the trend to apply formal technology (abstract reasoning on Boolean functions and automata) outside the traditional scope of verification, namely handling *quantitative* properties such as timing and power consumption that were considered non-functional properties, and applying optimization/synthesis rather than evaluation/verification with respect to them.

Fig. 1 sketches a possible logic synthesis flow. Starting from a multi-level logic specification, the circuit is brought into a form of an And-Inverter Graph (AIG) consisting solely of AND and NOT gates. This representation is than mapped into a concrete technology of standard cells admitting physical properties such as size and electrical characteristics. Syntactically AIGs are composed from 2AND gates but by collapsing together all NOT-free "cones", we obtain a semantically-equivalent function constructed from AND gates of unbounded fan-in (arity). Part of the technology-dependent mapping can be viewed as decomposing those ANDs into networks of 2ANDs and this is the problem we address in this paper.

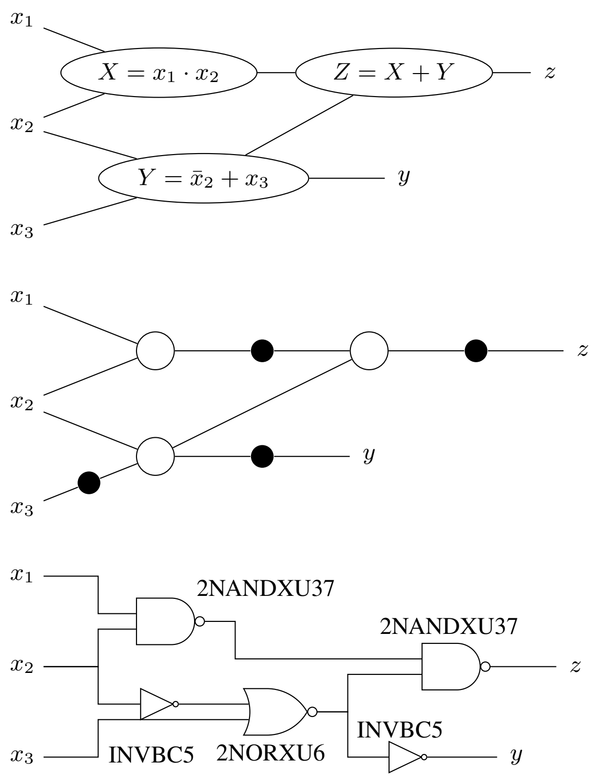

Fig. 1. A logic synthesis design flow: from multilevel logic specification to And-Inverter graphs to standard cells.

Dynamic power consumption of Boolean gates is associated essentially with their switchings between 0 and 1. In this work we consider synchronous combinational circuits that process sequences of input vectors. For each input vector, a circuit propagates values from input to output ports until it stabilizes and then reads the next input. The overall number of switchings associated with a pair of inputs is the number of gates whose stable value for one input is different from their value for the next input. For one such pair it is possible to steer the synthesis process and obtain a circuit with significantly less switchings compared to other arbitrary circuits that realize the same function. But of course, any circuit will process during its lifetime a long sequence consisting of diverse consecutive pairs of input vectors and optimizing synthesis with respect to all those is a challenging problem.

One natural approach is to define some probability function over sequences of input sequences, induced, for example, by a Markov chain which generates them. However, even the evaluation of the expected number of switchings in a *given* circuit is an intractable problem for non-trivial probabilistic generators with many input variables. As an alternative we develop in this paper a switching-aware synthesis procedure which optimizes the circuit relative to a *reference sequence* supposed to represent a typical input. In essence, the algorithm estimates the expected amount of switchings associated with a conjunction of any pair of input variables and then solves an optimal perfect matching problem to decide which variables to pair together as inputs to a 2AND gate. The procedure obtains quite a good switching reduction compared to arbitrary realizations of the same function by circuits of the same topology.

We then study the question of optimization with respect to inputs generated by Markov chains of small description size, that is, networks of sparsely-interacting 2-state probabilistic automata. We use such networks to generate the reference (training) sequences and then measure the performance gains on other sequences generated from the same model. We perform experiments on models of varying degree of variable dependencies and other assumptions on the inputs and we obtain significant reduction in switching activity. Finally, we introduce a reduced model of an instruction decoder and evaluate our procedure under probabilistic assumptions concerning the instruction stream.

2 Problem Statement

Our starting point is a Boolean circuit constructed from unbounded fan-in AND gates and NOT gates and our goal is to replace the AND gates by 2AND gates, yielding a semantically equivalent circuit C. Once we have a good solution for the AND-to-2AND problem we can apply it to any AND in the AIG and solve the synthesis problem for the whole circuit. From now on we consider a function $f : (x_1, \ldots, x_n) \mapsto x_1 \wedge \cdots \wedge x_n$ and a target circuit C which is a properly structured directed acyclic graph whose nodes are 2ANDs of the form $g : (x_1, x_2) \mapsto x_1 \wedge x_2$. We denote the input space $\mathbb{B}^n$ by X and the state-space of C, that is, the set of possible values in the output ports of all its gates, as $Y = \mathbb{B}^m$. The synthesized circuit C can be viewed as a memoryless transducer from X^* to Y^* such that for every t, $y[t]$ is the stable state of the circuit after processing $x[t]$. The amount of switching in C relative to input x and at time t is

$$S(C, x, t) = \Delta(y[t-1], y[t])$$

where Δ is the Hamming distance between Boolean vectors. The total amount of switching while reading a sequence $x \in X^*$ is

$$S(C, x) = \sum_{t=1}^{|x|} S(C, x, t).$$

A circuit C is better than C' relative to x if $S(C, x) < S(C', x)$. We want to build circuits which are optimal or reasonable in this sense. A major issue is what to assume about the set of inputs used to evaluate $S(C, .)$. One can think of two approaches.

1. Assume some probability function P on X^*, or more precisely a family of probabilities $P_k : X^k \to [0, 1]$, defined for example via a Markov chain, and then attempt to optimize the expected number of switchings per time step

$$S(C, P) = \lim_{k \to \infty} \sum_{x \in X^k} P_k(x) \cdot S(C, x)/k.$$

2. Use a long reference sequence $\underline{x}$ and evaluate C according to $S(C, \underline{x})$.

We will use a mixture of these two approaches. We optimize $S(C, \underline{x})$ for some training sequence $\underline{x}$ generated by a Markov chain and then evaluate the synthesized circuit according to the number of switchings that occur while processing other sequences generated from the same chain.

3 Input Pairing for AND Gates

The principle underlying switching reduction for AND gates is simple. Among all elements of X only a single vector $\mathbf{1} = (1, \ldots, 1)$ satisfies $f(x) = 1$. Input transitions of the form $x \to x'$ such that $x \neq \mathbf{1}$ and $x' \neq \mathbf{1}$ will not change the primary output. They can change, however, the values of intermediate gates that realize a conjunction of a *subset* of the input variables. We call such transitions *useless* and our goal is to "abort" them as soon as possible.

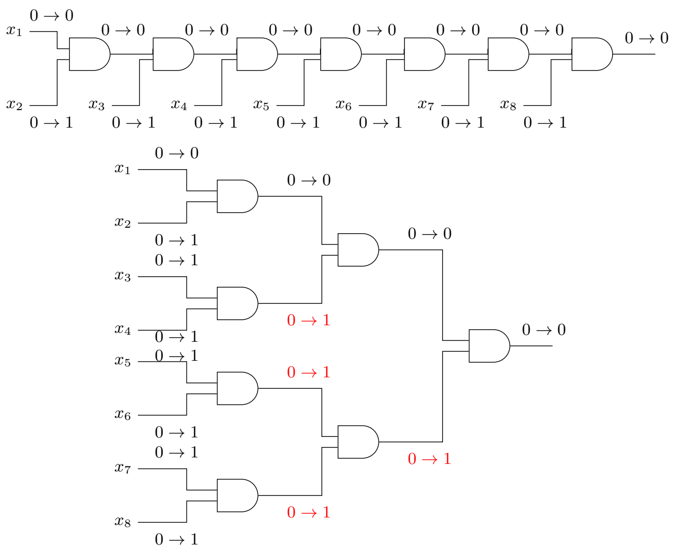

Fig. 2. For an input transition $(0, 0, 0, 0, 0, 0, 0, 0) \to (0, 1, 1, 1, 1, 1, 1, 1)$ a chain realizations can abort all switchings but a tree cannot.

There are numerous realizations of f by 2ANDs, all using $n-1$ gates. Among those, one can single out two extreme topologies, the sequential chain, whose depth is $n-1$ and the balanced tree with depth $d = \log n$. Since circuit depth determines propagation delay from input the output, speed considerations favor balanced trees and we will focus in this paper on those. Balanced trees bring some regularity to the problem, allowing us to work recursively on the levels of the tree from 0 (primary inputs) to $d-1$. Note, however, that chains tend to be more efficient in switching reduction in AND gates because they can abort useless switchings earlier as shown in Fig. 2. We have implemented also a version of our procedure, not reported here, which does not commit a priori to the circuit topology and which can be applied when power consumption is considered much more important than latency.

For the fixed balanced tree topology, the synthesis problem reduces to mapping input variables to the circuit input ports. The problem can be phrased recursively as follows. At level i of the tree, 2^{d-i} inputs should be partitioned into pairs to be mapped into 2^{d-i-1} 2AND gates. To understand which input signals should be paired together, let us look at Table 1-(A) which shows which transitions are taken by the output as a function of the transitions taken by the inputs. Table 1-(B) shows the number of output switchings in each case while Table 1-(C) shows the net switching reduction effect, namely, the number of input switchings minus the number of output switchings. It is intuitively clear that for one consecutive pair of inputs, we should pair together variables taking respective transitions $1 \to 0$ and $0 \to 1$. Such transitions cancel each other and send as inputs to the next level a variable doing $0 \to 0$ which will not trigger further switching with any other input it will be paired with. Fig. 3 shows two circuits and their performance differences with respect to a single consecutive pair of input vectors.

	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
$0 \to 0$	$0 \to 0$	$0 \to 0$	$0 \to 0$	$0 \to 0$
$0 \to 1$	$0 \to 0$	$0 \to 1$	$0 \to 0$	$0 \to 1$
$1 \to 0$	$0 \to 0$	$0 \to 0$	$1 \to 0$	$1 \to 0$
$1 \to 1$	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$

(A)

	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
$0 \to 0$	0	0	0	0
$0 \to 1$	0	1	0	1
$1 \to 0$	0	0	1	1
$1 \to 1$	0	1	1	0

(B)

	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
$0 \to 0$	0	1	1	0
$0 \to 1$	1	1	2	0
$1 \to 0$	1	2	1	0
$1 \to 1$	0	0	0	0

(C)

Table 1. (A) The output transitions of an AND-gate as a function of the input transitions; (B) The number of switchings associated with every pair $(u \to u', v \to v')$ of input transitions; (C) the net switching reduction: number of input switchings minus output switching.

Let $R_{jk}(u, u', v, v')$ be the probability that a pair (x_j, x_k) of input variables takes the joint transition $(u \to u', v \to v')$. When the inputs are generated by a Markov chain, these probabilities can be derived from the steady state of the chain which is, however, typically too hard to compute. Given a reference sequence $\underline{x}$, we can approx-

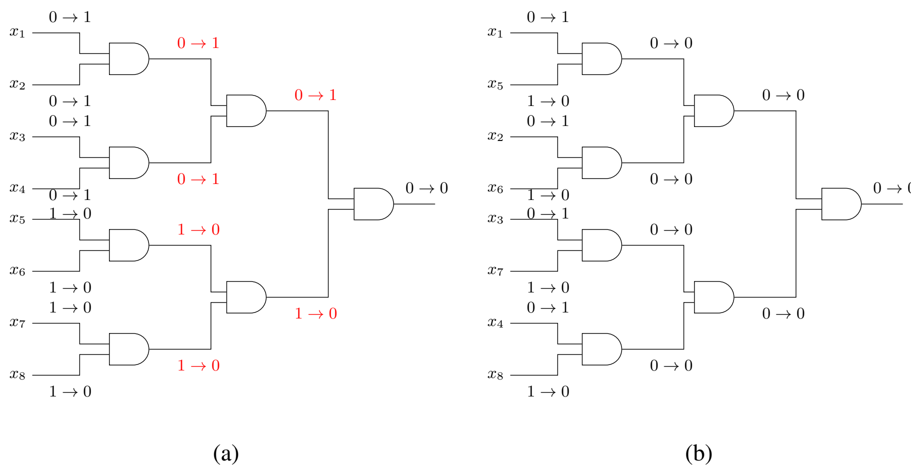

Fig. 3. Two pairings for input transition $(0,1,0,1,0,1,0,1) \to (1,0,1,0,1,0,1,0)$: (a) a bad pairing with 6 switchings; (b) a good pairing with no switchings.

imate $R_{jk}(u,u',v,v')$ by computing the number of occurences of the given transition in the sequence. Denoting the number of switchings associated with a pair of transition $u \to u'$ and $v \to v'$ (Table 1-(B)) by $s(u,u',v,v')$ (always 0 or 1), the expected number of switchings in $x_j \wedge x_k$ is

$$\mu_{jk} = \sum_{u,u',v,v'} R_{jk}(u,u',v,v') \cdot s(u,u',v,v').$$

Let $G = (V, E, \mu)$ be a complete graph with n nodes where each edge (j,k) is labeled by μ_{jk}. For the first level of the tree, the problem of finding input pairing which is optimal in terms of expected total number of switching is equivalent to the optimization problem known as *minimal-weight perfect matching* [12] for G. Once such an optimal pairing is found for level i, the outputs of the gates at this level serve as inputs for the pairing problem of the next level as summarized in Algorithm 1. The first polynomial algorithm for the optimal matching problem dates back to [5] using linear programming. The complexity of the algorithm has been improved in [10] from $O(n^4)$ to $O(n^3)$. Thus, together with the computation of μ from the training sequence the complexity of our procedure is $O(n^2 \cdot |\underline{x}| + n^3)$.

The results of the algorithm may deviate from the optimal expected number of switchings for three reasons. First, it is not based on the real value of μ but on its approximation from the training sequence. Secondly, it works by levels in a level-greedy fashion and hence, in principle, it is not guaranteed to produce the optimal among all circuits. Finally it does only statistics for pairs of variables and ignores more complex dependencies between three or more variables that may influence the outcome. However, as the experimental results show, it constitutes a very effective heuristics. We have implemented the algorithm and evaluated it empirically on purely synthetic examples

Algorithm 1 Synthesizing a balanced-tree circuit for a conjunction of n variables.

```
procedure Synthesize($\underline{x}$)
Input: A Boolean sequence $\underline{x}$ of dimension $n = 2^d$
Output: A balanced-tree circuit $C$ realizing $x_1 \wedge \cdots \wedge x_n$

$i := 0$
while $i < d - 1$ do
  $\underline{x}$ :=Reduce($\underline{x}, d - i$)
  $i := i + 1$
end

function Reduce($x, i$)
Input: A Boolean sequence $x$ of dimension $m = 2^i$
Output: An optimal pairing and a Boolean sequence $y$
          of dimension $2^{i-1}$

forall $j \neq k \in [1..i]$ compute $\mu_{jk}$
let $G = (N, E, \mu)$ be the corrsponding weighted graph
$M$ :=optimal_match($G$) $= \{(x_{r_1}, x_{r_2}), \ldots, (x_{r_{m-1}}, x_{r_m})\}$
$y := (x_{r_1} \wedge x_{r_2}, \ldots, x_{r_{m-1}} \wedge x_{r_m})$
return($y$)
```

and then on a realization of an instruction decoder. In the current implementation, since we limit the evaluation to $n = 16$, we find the optimal matching by enumeration.

4 Evaluation: Synthetic Boolean Models

We evaluate our algorithm against different classes of probabilistic 16-dimensional input generators. To define probabilities over X^* using arbitrary Markov chains we need to handle transition matrices of size at least $2^n \times 2^n$. For large n even writing down such a matrix is infeasible, not to mention computing its steady state probability. As is common in domains such as probabilistic verification and performance analysis, we use a compositional model consisting of a network of sparsely-interacting probabilistic automata. A probabilistic automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is an input-dependent Markov chain where every input letter $\sigma \in \Sigma$ induces a different transition matrix over state-space Q. The probabilistic transition function is thus of the form $\delta : Q \times \Sigma \times Q \to [0, 1]$ satisfying

$$\sum_{q' \in Q} \delta(q, \sigma, q') = 1$$

for every q and σ. A Markov chain can be viewed as a degenerate probabilistic automaton without an alphabet and a transition function of the form $\delta : Q \times Q \to [0, 1]$.

Let $N = \{1, \ldots, n\}$. A network of n interacting probabilistic automata is given as $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n, h)$ where $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i)$ and $h : N \to 2^N$ is an *influence function* such that $h(i)$ is the set of the other automata (besides itself) whose states are observed by $\mathcal{A}_i$ and influence its transitions. In our network each automaton has a state-space encoded by one bit, $Q_i = \mathbb{B}$, and an input alphabet $\Sigma_i = \mathbb{B}^{|h(i)|}$ which is the state-space of the influencing automata. The composition of the automata yields a global Markov chain (Q, δ) with $Q = Q_1 \times \ldots Q_n = \mathbb{B}^n$. The local input letter read by automaton $\mathcal{A}_i$ in a global state q is the projection of q on the variables in $h(i)$ that we denote by $\pi_i(q)$. The transition function of the global Markov chain is defined as

$$\begin{gathered}\delta((q_1, \ldots, q_n), (q'_1, \ldots, q'_n)) \\ = \\ \delta_1(q_1, \pi_1(q), q'_1) \cdot \delta_2(q_2, \pi_2(q), q'_2) \cdots \delta_n(q_n, \pi_n(q), q'_n).\end{gathered}$$

The structure of $h(i)$ can be used to classify models according to variable interaction. When the maximum of $|h(i)|$ is small, the system admits a small description from which random sequences for training and evaluation can be generated.

For each class of models we draw model instances randomly and measure the reduction obtained by our algorithm with respect to inputs generated by the model. All model classes share a tuning parameter $\alpha \in [0, 1]$ intended to quantify the degree of regularity in the input sequences which can be exploited to come up with good input pairing. Whenever we need to fix a probability while defining a model instance, we draw it from I_α defined as

$$I_\alpha = \begin{cases} [0, \alpha] \cup [1 - \alpha, 1] & \text{when } \alpha \leq \frac{1}{2} \\ \\ [\alpha - \frac{1}{2}, 1 - (\alpha - \frac{1}{2})] & \text{when } \alpha \geq \frac{1}{2} \end{cases}$$

The regularity in the inputs (and the potential effectiveness of our procedure) is monotone decreasing with α. When $\alpha = 0$ the probabilities are taken from $\{0, 1\}$ and the resulting model is deterministic. When $\alpha = 1/2$ the probabilities are drawn from the whole interval $[0, 1]$ and when $\alpha = 1$ all probabilities in the model instances are equal to $1/2$. In this case there is no regularity in the input, all sequences of states and transitions are uniformly distributed and no switching reduction is expected because any input pairing would be as good as another.

The whole experimental protocol is summarized in Algorithm 2. For each model class and value of α, we draw randomly a set $\{M_1, \ldots, M_{50}\}$ of model instances. For each instance M_i we generate a training sequence $\underline{x}_i$ of length 10000, apply our algorithm and synthesize an optimized circuit C_i. We generate an evaluation sequence x_i of length 10000 and let $\underline{S}_i$ be the number of switchings it induces in C_i. Then we draw a set $\{C_{i1}, \ldots C_{i20}\}$ of arbitrary circuits, let S_i be the average number of switchings induced by x_i in these circuits and let R_i be the relative improvement in $\underline{S}_i$ relative to S_i. Finally R is the average reduction over all model instances of the same class.

Independent Inputs We start by evaluating the switching reduction for two simple cases where the input variables are independent of each other. The first is the case where the value of each x_i is drawn according to a stateless Bernoulli process with parameter a_i while in the second model each bit is generated by an independent Markov chain with parameters a_i and b_i. The respective transition matrices are:

$$\begin{pmatrix} a_i & 1 - a_i \\ a_i & 1 - a_i \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} a_i & 1 - a_i \\ 1 - b_i & b_i \end{pmatrix}$$

Algorithm 2 Average switching reduction evaluation for a class of probabilistic input generators.

```
Input: A class of probabilistic input generators
Output: An estimation $R$ of the average switching
        reduction obtained by our algorithm
for $i := 1$ to $50$
  draw a model $M_i$
  generate a training sequence $\underline{x}_i$ of length $10000$
  $C_i :=$ Synthesize$(\underline{x}_i)$
  generate an evaluation sequence $x_i$ of length $10000$
  $\underline{S}_i := S(C_i, x)$
  for $j = 1$ to $20$
    draw a circuit $C_{ij}$
    $S_{ij} := S(C_{ij}, x)$
  end
  $S_i :=$ average$_j$ $S_{ij}$
  $R_i := (S_i - \underline{S}_i)/S_i$
end
$R :=$ average$_i$ $R_i$
```

For these models μ_{jk} is computed analytically (see Table 2) without a training sequence. Fig. 4-(a) shows for these two model classes the average reduction obtained by our algorithm as a function of α. In both cases the reduction is around 70% when the system is close to deterministic and 30% when probabilities are taken from $[0, 1]$.

Cascades Next we explore the class of cascade structures where the automata are ordered and each automaton observes the state of some of its predecessors. A network is a cascade of depth k if $h(i) = \{i-k, \ldots, i-1\}$ and the number transition matrices for each automaton is 2^k. The results for cascades of depth 1 and 2 are plotted in Fig. 4-(b). For depth 1 the reduction ranges from 70% for close to deterministic inputs to 15% for $\alpha = 1/2$ while for depth 2 the range is from 50% to 10%.

Partitioned Variables Next we applied our procedure to a network where the variables are partitioned into clusters of size 2 and 4 and each automaton observes only the states of the automata in its cluster. The results are plotted in Fig. 4-(c). For 2-clusters the range or reduction is between 65% for almost deterministic inputs and 15% for $\alpha = 0.5$, while for 4-clusters the corresponding reductions are less than 50% and 10%.

Arbitrary Sparse Network In the last class of examples we consider arbitrary networks where each automaton observes the states of k randomly chosen other automata. Fig. 4-(d) shows the results obtained for $k = 2$ and 4. In the former case we obtain 45% for $\alpha = 0.05$ and around 5% for $\alpha = 0.5$, while for the latter we obtain the worst results: less than 10% for quasi-deterministic inputs and less than 5% when probabilities are drawn anywhere in $[0, 1]$.

	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
$0 \to 0$	$(1-a_j)^2(1-a_k)^2$	$(1-a_j)^2 a_k(1-a_k)$	$(1-a_j)^2$	$(1-a_j)^2 a_k^2$
$0 \to 1$	$a_j(1-a_j)(1-a_k)^2$	$a_j(1-a_j)a_k(1-a_k)$	$a_j(1-a_j)a_k(1-a_k)$	$a_j(1-a_j)a_k^2$
$1 \to 0$	$a_j(1-a_j)(1-a_k)^2$	$a_j(1-a_j)a_k(1-a_k)$	$a_j(1-a_j)a_k(1-a_k)$	$a_j(1-a_j)a_k^2$
$1 \to 1$	$a_j^2(1-a_k)^2$	$a_j^2 a_k(1-a_k)$	$a_j^2 a_k(1-a_k)$	$a_j^2 a_k^2$

(a)

	$0 \to 0$	$0 \to 1$	$1 \to 0$	$1 \to 1$
$0 \to 0$	$\frac{a_j a_k(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$\frac{a_j(1-a_k)(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$-\frac{a_j(a_k-1)(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$-\frac{a_j(a_k-1)(1-b_j)b_k}{(a_j+b_j-2)(a_k+b_k-2)}$
$0 \to 1$	$\frac{(1-a_j)a_k(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$\frac{(1-a_j)(1-a_k)(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$\frac{(a_j-1)(a_k-1)(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$\frac{(a_j-1)(a_k-1)(1-b_j)b_k}{(a_j+b_j-2)(a_k+b_k-2)}$
$1 \to 0$	$-\frac{(a_j-1)a_k(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$-\frac{(a_j-1)(1-a_k)(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$\frac{(a_j-1)(a_k-1)b_j(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$\frac{(a_j-1)(a_k-1)b_j b_k}{(a_j+b_j-2)(a_k+b_k-2)}$
$1 \to 1$	$-\frac{(a_j-1)a_k b_j(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$-\frac{(a_j-1)(1-a_k)b_j(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$-\frac{(1-a_j)(a_k-1)(1-b_j)(1-b_k)}{(a_j+b_j-2)(a_k+b_k-2)}$	$-\frac{(1-a_j)(a_k-1)(1-b_j)b_k}{(a_j+b_j-2)(a_k+b_k-2)}$

(b)

Table 2. (a) The probabilities of transition pairs for two sequences generated by: (a) Bernoulli processes with parameters a_j and a_k; (b) independent Markov chains with parameters a_j, b_j and a_k, b_k.

Table 3 shows the average number of *absolute* switching elimination per gate in one time step. Upon closer inspection we observe that the results become consistently worse as the number of variables observed by an automaton becomes larger, quite independently of the interaction pattern. This may be an artifact of the way we generate model instances. The reason is that when an automaton has several transition matrices, the values of an entry (u, v) in different matrices may be taken from opposite sides of I_α, cancel each other an render the behavior of the variables more random and less regular.

α	Bern	iMar	casc1	casc2	part2	part4	spar2	spar4
0.05	0.115	0.110	0.117	0.102	0.118	0.060	0.093	0.020
0.10	0.106	0.104	0.105	0.076	0.091	0.041	0.075	0.017
0.15	0.095	0.097	0.089	0.060	0.081	0.037	0.057	0.015
0.20	0.093	0.091	0.079	0.050	0.074	0.029	0.047	0.013
0.25	0.084	0.088	0.066	0.041	0.061	0.023	0.040	0.011
0.30	0.084	0.081	0.063	0.032	0.055	0.019	0.032	0.009
0.35	0.071	0.071	0.048	0.029	0.049	0.016	0.027	0.008
0.40	0.065	0.067	0.040	0.022	0.043	0.013	0.023	0.007
0.45	0.063	0.061	0.037	0.021	0.036	0.012	0.021	0.006
0.50	0.054	0.057	0.036	0.019	0.031	0.011	0.018	0.005
0.55	0.040	0.044	0.026	0.013	0.024	0.008	0.014	0.004
0.60	0.031	0.031	0.018	0.010	0.017	0.006	0.010	0.002
0.65	0.023	0.024	0.013	0.007	0.013	0.004	0.006	0.002
0.70	0.016	0.017	0.009	0.005	0.009	0.002	0.004	0.001
0.75	0.010	0.011	0.006	0.003	0.005	0.001	0.003	0.001
0.80	0.007	0.007	0.003	0.002	0.003	0.000	0.001	0.000
0.85	0.003	0.003	0.001	0.000	0.001	0.000	0.000	0.000
0.90	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000
0.95	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 3. The *absolute* reduction in number of switching per gate per time step for all the models.

5 Evaluation: a Mini Instruction Decoder

Finally, we synthesize a mini instruction decoder, where we apply our procedure to a full AIG. We consider a very simple hand-held calculator whose instructions are listed in Table 4. The instruction are encoded using 4 bits although 3 bits would suffice, to reflect the fact that in a real application often not all the possible input combinations are used.

We assume that the typical use of the calculator will be just to perform an operation (add, subtract, multiply, divide) on two numbers entered from the numeric keypad. More sophisticated users might perform more complex operations, say add three numbers at once, but with a lower probability. The Markov model for instruction sequences is depicted in Fig. 5 and explained below:

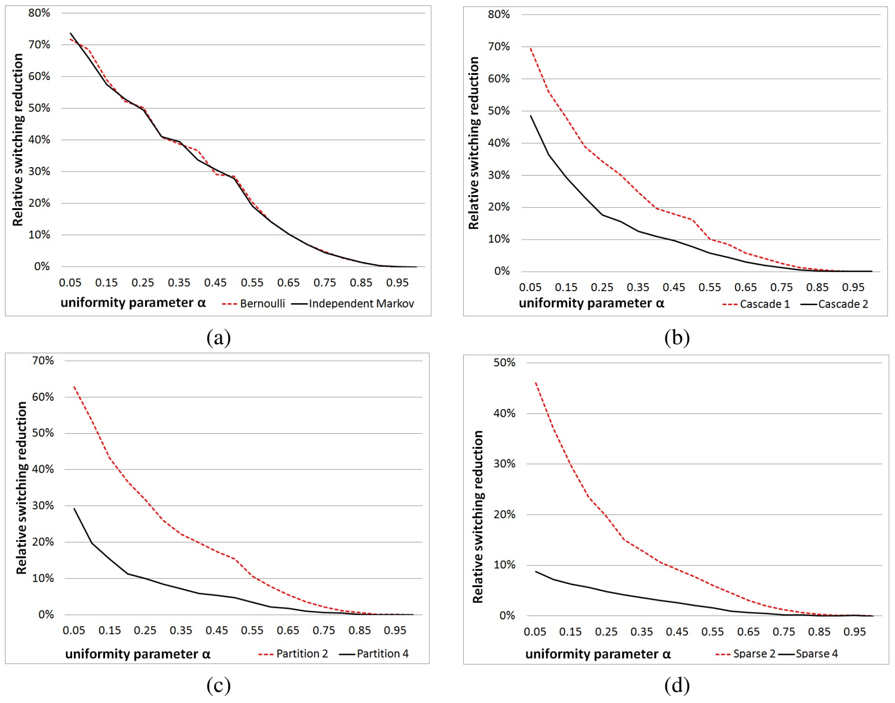

Fig. 4. The average switching reduction as a function of the uniformity parameter α for different input models: (a): Independent inputs – Bernoulli (dashed red) and Markov processes. (b): Variables are arranged in a cascade structure of depth 1 (dashed red) and 2 (c): Variables are partitioned into mutually-dependent clusters of size 2 (dashed red) and 4 (d): Each variable depends on 2 (dashed red) and 4 other arbitrary variables.

1. With probability p_{lm} load an argument previously stored in memory, otherwise just type in some number as the first argument.
2. Press one of $\{+, -, \times, \div\}$ with respective probabilities $\{p_{add}, p_{sub}, p_{mul}, p_{div}\}$.
3. Load the second argument either from memory (probability p_{lm}) or by typing the number.
4. Evaluate by pressing '$=$' and then with probability P_{sm} store the result in memory.

For the experiment we set the parameters of the model as follows:

$$p_{lm} = 0.1 \quad p_{add} = 0.4 \; p_{sub} = 0.3$$
$$p_{mul} = 0.2 \; p_{div} = 0.1 \; p_{sm} = 0.1$$

We generate from the model a training sequence of size 20000 and use it to synthesize an optimized circuit denoted by OC. For evaluation purposes we generate an input sequence of length 100000 and compare the number of switchings it induces in OC with 20 randomly drawn implementations of the decoder. The results are shown in Figure 6. Note that there is a large variation in the number of switchings among the different

instruction	code	meaning
LOAD	1001	loading from numerical keys
LOADM	1010	loading from memory
SET_ADD	1100	pressing '$+$'
SET_SUB	1101	pressing '$-$'
SET_MUL	1110	pressing '$\times$'
SET_DIV	1111	pressing '$\div$'
EVAL	0000	pressing '$=$'
STORE	0101	saving result to memory

Table 4. The instruction set of the calculator.

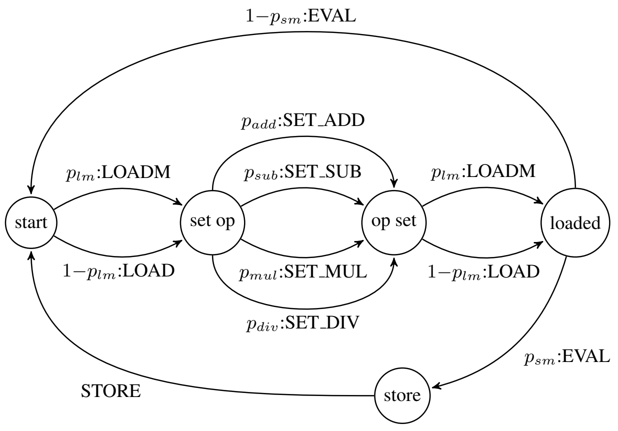

Fig. 5. The probabilistic model of the instruction generator.

realizations. Circuit OC was always better than any of the other circuits and on the average achieved a reduction of 16.49%. Naturally these results are also sensitive to the uniformity of the probabilities. For example when we set $p_{lm} = 0.25$ and $P_{sm} = 0.2$ we obtained a smaller reduction of 12.53%.

6 Discussion

The interest in switching reduction and in the evaluation of circuit behavior against probabilistic models in general [6] is not new. Concerning switching reduction we can distinguish between an abstract approach like ours which focuses only on the number of transitions as an approximate indicator of power consumption and more physical approaches that map abstract circuits onto a concrete technology where power consumption can be measured more accurately. The work of [16] which belongs to the second category, mentions the abstract problem that we solve here as a suggestion for future work that could be plugged upstream to their own work on power-aware mapping

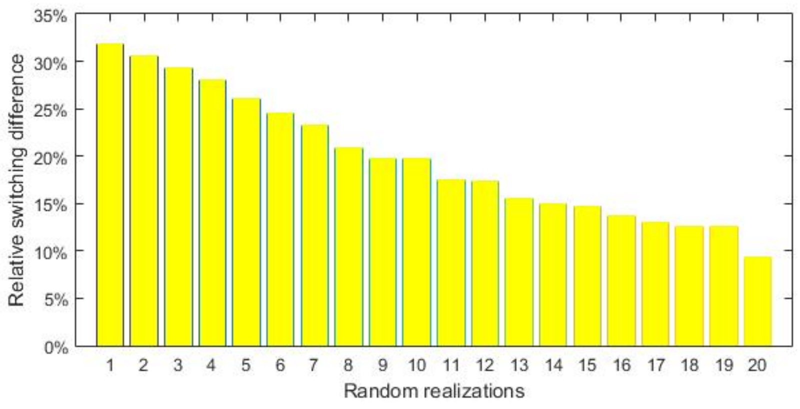

Fig. 6. A comparison of the number of switchings in the optimized instruction compared to 20 other arbitrary realizations. The height of bars shows how much switching can be saved using the optimized circuit compared to that realization.

using a real technology library. The work of [19] is also of this type, mapping abstract AIGs to real gates. The input is specified as a set of input vectors (patterns) and simulation with these patterns is used to estimate power consumption for different mappings alternatives onto real gates from a library.

The work of [17, 18] applies a similar reasoning concerning input pairing for 2AND gates and uses a variant of Huffman's algorithm for constructing a binary tree with minimal average weighted path length [9]. However, this work is restricted to the case were variables are assumed to be generated by independent Bernoulli processes while our approach is applicable to any small-description Markov process or any user-provided training sequence. Moreover, they use a greedy pairing algorithm such that at each step of the algorithm one pair of variables, the one which induces the least expected number of switching is selected as an input to an AND gate. Experiments show that our scheme which treats at once a complete level of the tree via optimal matching is significantly more efficient.

The work of [11] also uses Huffman's algorithm but in a different way that seems to yield a random balanced tree. They do not give any explicit probabilistic model but introduce some delay assumptions and claim their algorithm to be optimal in terms of reducing only the switching activity which is due to glitches. This is the place to mention that as we do not model gate delays, we cannot detect glitches but one may argue that their importance in balanced trees structures is less pronounced. The work of [18] is extended significantly in [20] who give an optimal algorithm for unbounded depth 2AND synthesis, restricted to a Bernoulli input model. Their algorithm tends to produce deep circuits with long delays.

To summarize, we devised a novel procedure for an early step in the synthesis flow for digital circuits/functions. The major novelty of the algorithm is its ability to approximate in a tractable manner, polynomial in the number of inputs to an AND gate, the minimal average-case number of switchings, based on a training input sequence.

The approach can be applied, in principle to any probabilistic model of the input but, of course, formal guarantees of approximation quality can be given only in restricted cases.

For synthetic empirical evaluation we developed an original framework based on sparsely interacting networks of probabilistic automata and ran extensive experiments under various probabilistic models of the input. The reduction obtained on these synthetic examples were quite impressive, reaching, in some cases, dozens of percents. Two major open questions remain concerning their transfer to real life:

1. Can such reduction be pushed downstream to the more physical steps of synthesis? This question has two versions: can it be done using existing commercial tools that carry a lot of legacy, or can it done in principle by new tools if this type of optimization criterion is considered important.
2. How do real applications look like in terms of circuit structure and input model?

We made a preliminary exploration of the second question using the instruction decoder model and the results seem encouraging. We believe the behavior of real circuits is much more regular than arbitrary Markov chains. In the future we intend to attack larger industrial-scale examples and follow them, as far as possible, down to technology-dependent mapping, being able to detect timing effects and measure real power consumption. It has already been observed that synthesis is an old technology and the outcome of commercial synthesis tools is sensitive to many syntactic features [13] and we hope that this work will bring a fresh look on the topic.

On the theoretical side we intend see under what assumptions our level-greedy algorithm is optimal and to give bounds on its deviation from the optimum when it is not. Another potential direction for exploration is to present trade-offs between speed and switching reduction by being less committed to the circuit topology. Although typically the number of inputs to a single AND cone need not be very large, it would be interesting to explore how far we can go with the number of inputs using the polynomial algorithm for optimal matching. Finally we intend to extend this work to sequential machines and to explore the application of switching-oriented reasoning to the encoding of states and symbolic inputs.

References

1. A Bellaouar and Mohamed I Elmasry. *Low-power digital VLSI design: Circuits and systems*. Kluwer, 1995.
2. Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI*, 8(3):299–316, 2000.
3. Robert K Brayton, Gary D Hachtel, McMullen C, and Alberto Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*.
4. Anantha P Chandrakasan and Robert W Brodersen. *Low power digital CMOS design*. Kluwer, 1995.
5. Jack Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *J. Res. Nat. Bur. Standards B*, 69:125–130, 1965.

6. Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Markovian analysis of large finite state machines. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(12):1479–1493, 1996.
7. Gary D Hachtel and Fabio Somenzi. *Logic synthesis and verification algorithms*. Springer, 2006.
8. Zvi Kohavi and Niraj K Jha. *Switching and finite automata theory*. Cambridge University Press, 2010.
9. Lawrence L Larmore and Daniel S Hirschberg. A fast algorithm for optimal length-limited huffman codes. *Journal of the ACM (JACM)*, 37(3):464–473, 1990.
10. Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Dover Publications, 1976.
11. Rajeev Murgai, Robert K Brayton, and Alberto Sangiovanni-Vincentelli. Decomposition of logic functions for minimum transition activity. In *Proceedings of the 1995 European conference on Design and Test*, page 404. IEEE Computer Society, 1995.
12. Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.
13. A. Puggelli, T. Welp, A. Kuehlmann, and A. Sangiovanni-Vincentelli. Are logic synthesis tools robust? In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 633–638, June 2011.
14. Jan M Rabaey and Massoud Pedram. *Low power design methodologies*. Springer, 1996.
15. Tsutomu Sasao. *Switching theory for logic synthesis*, volume 1. Springer, 1999.
16. Vivek Tiwari, Pranav Ashar, and Sharad Malik. Technology mapping for low power. In *Design Automation, 1993. 30th Conference on*, pages 74–79. IEEE, 1993.
17. Chi-Ying Tsui, Massoud Pedram, and Alvin M Despain. Technology decomposition and mapping targeting low power dissipation. In *Proceedings of the 30th international Design Automation Conference*, pages 68–73. ACM, 1993.
18. Chi-Ying Tsui, Massoud Pedram, and Alvin M Despain. Power efficient technology decomposition and mapping under an extended power consumption model. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(9):1110–1122, 1994.
19. Chingwei Yeh, Chin-Chao Chang, and Jinn-Shyan Wang. Technology mapping for low power. In *Design Automation Conference, 1999. Proceedings of the ASP-DAC'99. Asia and South Pacific*, pages 145–148. IEEE, 1999.
20. Hai Zhou and DF Wong. An exact gate decomposition algorithm for low-power technology mapping. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 575–580. IEEE Computer Society, 1997.