Exact Computation of the Entropy of a Logic Circuit

Enrico Macii       Massimo Poncino

Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

Computing the entropy of a digital circuit has proved to be very useful for several applications in the area of VLSI system design. Recently, a method for entropy calculation has been used in the context of power estimation for logic circuits described at the register-transfer level. The technique has shown to be reasonably effective concerning the trade-off between the accuracy of the estimates produced and the execution time. However, the assumptions required to make the computation feasible are such that the obtained results are approximate. In this paper, we propose a symbolic algorithm for the exact calculation of the entropy of a logic circuit which is able to handle reasonably large examples without introducing any approximation. We present experimental data on standard benchmark designs in order to show the effectiveness of the new method; in addition, we compare our results to the ones obtained with the approximate approach. As a result, we observe a marginal penalty in the performance of the symbolic procedure; on the other hand, accuracy in the calculation increases significantly.

1 Introduction

The concept of entropy has been used extensively in the field of information theory to characterize the statistical behavior of random variables and processes. In particular, entropy represents a measure of the quantity of information carried by these objects [1].

Interesting applications of the entropy measure to the field of logic synthesis have been proposed in the past. Early work by Kellerman [2], Hellerman [3], Cook and Flynn [4], and Pippenger [5] employed the definition of entropy to estimate the area cost of the gate-level implementation of a single-output Boolean function. Later on, Cheng and Agrawal [6] generalized the entropy-based formulation of the logic complexity estimation problem to the case of circuits realizing multiple-output Boolean functions; in addition, the relationship between the delay of a single-output logic network and its entropy was investigated. Finally, other relevant applications of the entropy measure to digital design topics concerned the state assignment of sequential circuits [7, 8], the test pattern generation of combinational and sequential circuits [9], and the calculation of testability measures of designs described at both the gate-level [10] and the functional-level [11].

Very recently, independent works by Pedram et al. [12] and Najm [13] have found that the entropy well models the statistical behaviors of logic circuits described at the register-transfer (RT) level; this observation was made during the development of high-level power estimation tools, whose availability is becoming more and more important as the level of abstraction at which digital systems are designed increases. In both the proposed approaches, approximate calculation of the entropy of a logic circuit was used, in order to dominate the exponential complexity (in the number function outputs) of the problem. In spite of the approximation, power estimation results were found to be sufficiently sound and accurate; however, we believe that an exact method to calculate the entropy of a reasonably large Boolean function is needed to make the techniques of [12, 13] applicable in practice, also to circuits described at gate-level.

The observation above has been the motivation for the development of an algorithm, based on the symbolic manipulation of Boolean and pseudo-Boolean functions, represented as decision diagrams, that is able to handle real circuits without any approximation. (A pseudo-Boolean function is a function which maps the set \( \{0,1\}^n \) onto the set of the real numbers.) Experimental results, obtained on standard benchmark designs from the Mccn'91 [14] and the Iccas'89 [15] suites, are very promising, and seem to point to the conclusion that the use of the proposed entropy calculation technique inside a power estimation tool similar to the ones of [12, 13] will improve the accuracy of the results without heavily affecting the total execution time.

The rest of this paper is organized as follows. Section 2 presents some background material on binary and algebraic decision diagrams (BDDs and ADDs). Section 3 recalls the definition of the entropy of a multiple-output Boolean function, and discusses some associated properties. Section 4 summarises the approximate approach, and describes the symbolic algorithm for exact entropy calculation. Section 5 reports and gives comments on the experimental results. Finally, Section 6 is devoted to concluding remarks and indicates directions for future work; in particular, it outlines a possible application of the entropy calculation algorithm of this paper to extend and to improve the performance of the power estimation technique of [13].
2 Decision Diagrams

2.1 Binary Decision Diagrams (BDDs)

A BDD is a graph representation of a logic function. Under some conditions, the BDDs are canonical, that is, there is one unique BDD for a given logic function. For example, let us consider the function

\[ f(x, y, z) = xy + z'x + y'z. \]

The BDD of \( f \) is shown in Figure 1.

![Figure 1: The BDD of \( f(x, y, z) = xy + z'x + y'z \).](image)

To obtain the value of the function corresponding to a given variable assignment, it suffices to follow a path in the BDD from the root node to a leaf by taking the dotted branch when the value of the variable associated with the node is 0, and the dot-free branch otherwise. The value of the leaf gives the value of the function.

Sophisticated algorithms exist for efficiently constructing and manipulating BDDs. For a more detailed treatment of this matter, the reader can refer to [16, 17].

2.2 Algebraic Decision Diagrams (ADDs)

Algebraic Decision Diagrams (ADDs) [18] can be viewed as a form of multi-terminal BDDs that support algebraic and arithmetic operations on their terminal nodes. Terminal nodes can hold objects drawn from an arbitrary set, for instance, real numbers. An ADD is then a directed acyclic graph representing a set of pseudo-Boolean functions \( f_i : \{0, 1\}^n \rightarrow S \), where \( S \) is the carrier of the algebraic structure over which the ADD is defined.

ADDs are particularly suitable for representing vectors and matrices. For example, the matrix of Figure 2-a can be represented by the ADD of Figure 2-b. \( x \) variables encode row variables, and \( y \) variables encode column variables.

![Figure 2: A Matrix and its Corresponding ADD.](image)

Notice that, as in the case of BDDs, the value of the function for a specific assignment of the variables is given by the value of the leaf reachable from the root of the ADD by taking the dotted branch when the value of the variable associated with the node is 0, and the dot-free branch otherwise.

As shown in [19], the use of ADDs has made it possible to realize computer programs for the manipulation of very large vectors and matrices (in the case of Walsh matrices, arrays of up to \( 2^{1000} \) rows and columns have been handled successfully with very limited memory consumption); in fact, the ADD data structure has appeared to be much more memory efficient than traditional storage techniques that exploit structure sparsity.

3 Entropy of a Boolean Function

Let \( z \) be a random Boolean variable, and let \( p \) be the probability of \( z \) to be at the logic value 1. The entropy of variable \( z \), denoted in the sequel by \( H(z) \), gives a measure of the total information that \( z \) is carrying. Its definition is the following:

\[
H(z) = p \log_2 \frac{1}{p} + (1 - p) \log_2 \frac{1}{1 - p}. \tag{1}
\]

\( H(z) \) has a maximum value of 1 at \( p = 0.5 \), and a minimum value of 0 at \( p = 0 \) and \( p = 1 \), as shown by the diagram in Figure 3.

![Figure 3: Entropy of a Random Boolean Variable.](image)

The definition introduced by Equation 1 can be generalised very easily to the case of a vector of \( n \) random Boolean variables. Let \( X = (x_1, \ldots, x_n) \) be such a vector; clearly, \( X \) can take on \( 2^n \) distinct values. Let \( p_i \) be the probability of vector \( X \) to assume the value \( X_i \); then, the entropy of \( X \) is given by the following expression:

\[
H(X) = \sum_{i=1}^{2^n} p_i \log_2 \frac{1}{p_i}. \tag{2}
\]

In this case, the maximum value that \( H(X) \) can assume is \( n \) (in the case of \( p_1 = p_2 = \ldots = p_{2^n - 1}, p_{2^n} = 1/2^n \)), while the minimum value is 0 (in the case of \( p_i = 1 \) and \( p_{i_1, i_2} = 0 \)).
Now, let us consider a $n$-input, $m$-output Boolean function, $F(f_1, f_2, \ldots, f_m)$, where $f_k = f_k(X) = f_k(x_1, x_2, \ldots, x_n)$, $k = 1, 2, \ldots, m$.

The input entropy of function $F$, denoted as $H_I(F)$, is given by:

$$H_I(F) = \sum_{i=1}^{2^n} p_i \log_2 \frac{1}{p_i}, \quad (3)$$

where $p_i$ indicates the probability of the input vector $X$ to take on the value $X_i$. If we assume the input variables to be equiprobable, we have that $p_i = 1/2^n$, $\forall i$, and therefore $H_I(F) = n$. The vector $P = (p_1, \ldots, p_{2^n})$ is called the input probability vector.

The output entropy of function $F$, denoted as $H_O(F)$, is given by:

$$H_O(F) = \sum_{i=1}^{2^n} q_i \log_2 \frac{1}{q_i}, \quad (4)$$

where $q_i$ indicates the probability of the output vector to assume the value $O_i$. Obviously, $q_i = n_{O_i}/2^n$, where $n_{O_i}$ denotes the number of occurrences of the vector $O_i$ in the truth table of $F$. The vector $Q = (q_1, \ldots, q_{2^m})$ is called the output probability vector.

As a simple example of entropy computation, let us consider the 3-input, 2-output Boolean function $F = (f_1, f_2)$, whose truth table is shown in Figure 4.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f_1$</th>
<th>$f_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4: The Truth Table of a Multiple-Output Function.

If we assume the primary input variables to be equiprobable, we have that $p_i = 1/2^3 = 0.125$, $i = 1, \ldots, 2^3$; then, from Equation 3 we get a value of the input entropy of $H_I(F) = 3$. Furthermore, we have the following values of the output probabilities: $q_1 = 2/8$, $q_2 = 2/8$, $q_3 = 4/8$, and $q_4 = 0$. The value of the output entropy, $H_O(F)$, is then obtained by the straight application of Equation 4; we obtain:

$$H_O(F) = \frac{1}{4} \log_2 4 + \frac{1}{4} \log_2 4 + \frac{1}{2} \log_2 2 + 0 = 1.5.$$  

It can be noticed that $H_O(F) < H_I(F)$; this is true in general, that is, given any Boolean function, $F$, we have that $H_O(F) \leq H_I(F)$ [1].

4. Computing the Entropy of a Circuit

4.1 The Approximate Algorithm

Equations 3 and 4 give an exact way to compute the input and the output entropy of a multiple-output Boolean function and, therefore, of a multiple-output logic circuit realizing such function. Clearly, the computation has exponential complexity in the number of elements in the Boolean vector to be considered (i.e., the number of circuit inputs in the case of input entropy, and the number of circuit outputs in the case of output entropy). As a consequence, the direct application of Equations 3 and 4 does not allow us to determine the input and the output entropy but for very small circuits. In order to carry out the calculation for circuits of meaningful sizes, the following approximation has been introduced in [13].

Given a vector $V = (v_1, \ldots, v_n)$ of $n$ random Boolean variables, each variable $v_k$ is considered to be independent from all the others. Therefore, the actual entropy of vector $V$ (see Equation 2):

$$H(V) = \sum_{i=0}^{2^n} p_i \log_2 \frac{1}{p_i},$$

is approximated by its upper bound:

$$H'(V) = \sum_{i=0}^{n} H(v_i). \quad (5)$$

If we consider a logic circuit realizing a $n$-input, $m$-output Boolean function, $F(f_1, \ldots, f_m)$, where $f_k = f_k(X) = f_k(x_1, \ldots, x_n)$, $\forall k$, the approximation of Equation 5 translates to:

$$H'(F) = \sum_{i=0}^{2^m} H(x_i) \quad (6)$$

in the case of the input entropy, and to:

$$H'(F) = \sum_{i=0}^{m} H(f_i) \quad (7)$$

in the case of the output entropy.

Obviously, while the approximation in Equation 5 may be acceptable for the calculation of the input entropy, under the assumption that the input signals of the circuit are totally uncorrelated, it appears to be too simplistic for determining the output entropy of the circuit. This is because the assumption of the circuit outputs to be completely uncorrelated is far from being realistic.

4.2 The Symbolic Algorithm

In [13], it is made the claim that the approximation of Equation 5 is acceptable for the cases that have been considered. However, such a claim is not supported by any quantitative analysis, but only by experimental evidence. In this paper, we show that there exist several cases where the error in the calculation of the output entropy of a circuit introduced by Equation 7 is actually large.
In order to make the comparison, we have developed an algorithm, based on the symbolic manipulation of Boolean and pseudo-Boolean functions through ADDs, which allows us to perform the exact computation of the output entropy of reasonably large multiple-output circuits. The procedure, whose pseudo-code is shown in Figure 5, receives, as input parameter, the function \( F \) realized by the multiple-output circuit, and returns the exact value of the output entropy, \( H_0(F) \).

To emphasise the symbolic nature of the algorithm, we represent each symbol as a function of some variable sets. In particular, variable \( z \) and \( y \) are used to encode circuit inputs and outputs, respectively.

```plaintext
procedure Entropy(F) {
    1   R_F(x,y) = BuildRelation(F);  
    2   Q(y) = \( \frac{1}{2} \cdot \sum R_F(x,y) \);  
    3   LogQ(y) = AddApply(Q(y), Log);  
    4   Q'(y) = AddApply(Q(y), LogQ(y)*Times);  
    5   H_0(F) = \( \sum Q'(y) \);  
    6 return H_0(F);  
}
```

Figure 5: The Entropy Algorithm.

In line 1, the function \( F \) of the circuit is transformed into a relation, that is, into a Boolean function \( R_F(x,y) \) that evaluates to 1 for each \( y = F(x) \). In practice, \( R_F \) is built as follows:

\[
R_F(x,y) = \prod_{i=1}^{m} (y_i \equiv f_i).
\]

In line 2, the output probability vector, \( Q(y) \), is computed by algebraic existential quantification of the input variables \( z \) through the operator \( \sqcap \). This operator works conceptually as the conventional Boolean existential quantification \( \exists \), but it applies algebraic sum instead of Boolean sum (i.e., OR). For further details about algebraic operators and ADDs, the reader may refer to [19]. In lines 3 and 4, the vector \( LogQ \) containing the logarithms of the output probabilities is computed, and then multiplied by the output probability vector itself. Also in this case the \( Times \) operator is used since algebraic values are involved. Procedure \( AddApply \) simply applies the operator to the function on a minterm-by-minterm basis. Notice that in the current implementation of the algorithm, \( Q'(y) \) is obtained directly from \( Q(y) \), for efficiency purposes. In other words, while traversing the ADD of \( Q \), the representation of \( Q' \) is obtained directly by multiplying the values of the terminal nodes of \( Q \) by their logarithms. Finally, the value of the entropy is obtained by quantifying out the \( y \) variables, that is, summing over all the possible output values.

The Entropy algorithm is a generalisation of the well-known BDD-based procedure for computing the probability of a single-output Boolean function [20], where probabilities are computed by a post-order traversal of the BDD of the function. In our algorithm, a multiple-output function is viewed as an integer valued function, whose values are encoded by the output variables. The function is represented with ADDs, on which the probability computation is performed for all the circuit outputs simultaneously.

As discussed in [18], the smaller the size of the carrier \( S \) over which the ADDs are defined (i.e., the number of terminal nodes appearing in the DAG), the higher the chance of obtaining a compact data structure (thanks to the high "recombination" that may happen). On the other hand, when the cardinality of set \( S \) is large, the ADDs get closer to ordinary binary trees. Clearly, in the worst case, the ADDs involved in the calculation of the output entropy of a logic circuit fall into the latter situation. In fact, it can be proved that the number of distinct values of the output probability vector can be, in principle, \( O(2^{n/2}) \), where \( n \) is the number of circuit inputs. Fortunately, this is not what actually happens in most of the cases, where the ADDs involved in computation behave well with respect to their sizes. An intuitive explanation of this fact is that, typically, real circuits have some output values which never occur; from the entropy stand-point, all these "impossible" values are seen as 0 entries of the probability vector, thus contributing to promote node sharing in the ADDs.

5 Experimental Results

We have applied the symbolic algorithm for exact entropy calculation to some circuits taken from the Mecn'91 [14] and the Iscas'89 [15] benchmark suites. In the case of sequential designs, only the combinational logic has been considered, i.e., present state and next state variables have been treated as primary inputs and outputs, respectively. The results, reported in Table 1 and obtained on DECStation 5000/240 with 32M of memory, indicate that the symbolic method can be applied to circuits with over 30 primary inputs and 20 primary outputs; obviously, entropy calculation using the explicit definition of Equation 4 can not be performed on such large examples.

Concerning the comparison to the approximate method of [13], we have that, as expected, the new algorithm is more time consuming (CPU times are in seconds). On the other hand, results prove that the approximation of Equation 5 is unacceptable. In fact, the relative error, computed as:

\[
Error = \frac{H_0(F) - H_0(F)}{H_0(F)}
\]

is around 20% on average, and in some cases it exceeds 50%.

Column \textit{Leaves} gives the number of terminal nodes of the ADD representing the output probability vector. Interestingly, this number is small for most circuits, and it is always few below the theoretical limit. This result indicates that adopting an ADD package which provides dynamic variable ordering [21, 22] can be very beneficial for both the memory requirements and the execution times of the algorithm.
<table>
<thead>
<tr>
<th>Circuit</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Literals</th>
<th>Exact</th>
<th>Approximate</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$H_0(F)$</td>
<td>$H_0(F)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Leaves</td>
<td>Time</td>
<td>$H_1(F)$</td>
</tr>
<tr>
<td>5xp1</td>
<td>7</td>
<td>10</td>
<td>163</td>
<td>7.0</td>
<td>2</td>
<td>0.16</td>
</tr>
<tr>
<td>C432</td>
<td>16</td>
<td>7</td>
<td>372</td>
<td>5.5</td>
<td>121</td>
<td>70.33</td>
</tr>
<tr>
<td>alu2</td>
<td>10</td>
<td>6</td>
<td>453</td>
<td>4.7</td>
<td>30</td>
<td>0.18</td>
</tr>
<tr>
<td>alu4</td>
<td>14</td>
<td>8</td>
<td>855</td>
<td>6.2</td>
<td>101</td>
<td>3.82</td>
</tr>
<tr>
<td>bi2</td>
<td>15</td>
<td>9</td>
<td>112</td>
<td>4.6</td>
<td>32</td>
<td>0.14</td>
</tr>
<tr>
<td>cm163a</td>
<td>16</td>
<td>5</td>
<td>53</td>
<td>3.0</td>
<td>4</td>
<td>0.10</td>
</tr>
<tr>
<td>cordic</td>
<td>23</td>
<td>2</td>
<td>194</td>
<td>0.5</td>
<td>4</td>
<td>0.01</td>
</tr>
<tr>
<td>frg1</td>
<td>28</td>
<td>3</td>
<td>130</td>
<td>2.8</td>
<td>7</td>
<td>0.50</td>
</tr>
<tr>
<td>i1</td>
<td>25</td>
<td>15</td>
<td>60</td>
<td>9.0</td>
<td>87</td>
<td>10.59</td>
</tr>
<tr>
<td>ps1</td>
<td>16</td>
<td>13</td>
<td>85</td>
<td>6.3</td>
<td>17</td>
<td>0.34</td>
</tr>
<tr>
<td>misex3c</td>
<td>14</td>
<td>14</td>
<td>164</td>
<td>10.7</td>
<td>61</td>
<td>12.87</td>
</tr>
<tr>
<td>pcle</td>
<td>19</td>
<td>9</td>
<td>71</td>
<td>5.9</td>
<td>4</td>
<td>1.10</td>
</tr>
<tr>
<td>act</td>
<td>19</td>
<td>15</td>
<td>164</td>
<td>7.5</td>
<td>25</td>
<td>42.11</td>
</tr>
<tr>
<td>too_large</td>
<td>3</td>
<td>3</td>
<td>1038</td>
<td>1.0</td>
<td>8</td>
<td>39.21</td>
</tr>
<tr>
<td>x2</td>
<td>10</td>
<td>7</td>
<td>71</td>
<td>2.9</td>
<td>8</td>
<td>0.04</td>
</tr>
<tr>
<td>s1186</td>
<td>32</td>
<td>14</td>
<td>694</td>
<td>6.1</td>
<td>204</td>
<td>53.88</td>
</tr>
<tr>
<td>s298</td>
<td>17</td>
<td>20</td>
<td>244</td>
<td>12.3</td>
<td>20</td>
<td>60.72</td>
</tr>
<tr>
<td>s208</td>
<td>18</td>
<td>10</td>
<td>166</td>
<td>3.1</td>
<td>12</td>
<td>1.59</td>
</tr>
<tr>
<td>s544</td>
<td>34</td>
<td>11</td>
<td>20</td>
<td>9.0</td>
<td>3</td>
<td>1.09</td>
</tr>
<tr>
<td>s510</td>
<td>25</td>
<td>7</td>
<td>138</td>
<td>3.6</td>
<td>8</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 1: Experimental Results.

6 Conclusions and Future Work

Measuring the entropy of a logic circuit has appeared to be useful for several digital design applications. Recently, approximate entropy calculation has been used in the context of power estimation for circuits described at the RT level. The technique is quite effective, but it suffers from accuracy problems due to the fact that all the nodes in the circuit are assumed to be totally uncorrelated. In this paper, we have proposed a symbolic algorithm for the exact calculation of the entropy of a circuit which is able to handle reasonably large examples without introducing any approximation. We have presented results on standard benchmarks in order to show the effectiveness of the new method, and we have compared them to the ones obtained using the approximate approach. The outcome of the comparison is that accuracy in the computation sensibly increases, while the execution times do not get too penalized.

The power estimation method proposed by Najm in [13] requires the computation of the average entropy, $\mathcal{H}$, of each node in the circuit. This quantity is determined as follows. First, the circuit is levelized so that its gates are labeled with level values indicating their distances from the primary inputs. For every level $j = 1, 2, \ldots, K$, the output nodes of the gates at level $j$ form the $j$-th cross-section of the circuit. Then, the entropy of each cross-section of the circuit is determined by calculating the input entropy, $H_1(F)$, and then by exploiting the property that the output entropy of a circuit decreases quadratically with the circuit depth.

The entropy of the $j$-th cross-section of the circuit is then given by:

$$H^j(F) = (H_1(F) - H_0(F)) \left(1 - \frac{j}{K}\right)^2 + H_0(F),$$

where $K$ is the total number of levels in the circuit.

The average node entropy is then given by:

$$\mathcal{H} = \frac{1}{N} \sum_{j=1}^{K} H^j(F),$$

where $N$ is the total number of gates in the circuit. Determining $H^j(F)$ may be computationally expensive, since it requires the calculation of the output entropy $H_0(F)$. The approximation of Equation 5 can then be applied, so that the expression of $\mathcal{H}$ can be re-written as:

$$\mathcal{H} = \frac{1}{N} \sum_{j=1}^{K} \left(\sum_{i=1}^{\mathcal{W}} H(w_i)\right),$$

where $w_i$ represents the output wire of the $i$-th gate in the $j$-th cross-section of the circuit, and $\mathcal{W}$ the total number of variables in such cross-section. Obviously, the smaller the number of variables in the $j$-th cross-section that are correlated, the more accurate the evaluation of $\mathcal{H}$.

It is straightforward to see that the symbolic algorithm that we have proposed in this paper can be used to determine the exact value of the entropy at the $j$-th cross-section of the circuit, thus eliminating the need of an approximate
computation. In addition, even when the exact calculation becomes infeasible (this may be the case for circuits whose cross-sections are particularly wide), our method can be applied to increase the degree of accuracy of the approximate power estimation. In fact, instead of considering each variable \( w_i \) of a given cross-section separately from all the others, it may be possible to group them by looking at their correlations. In other words, it may be possible to generate a partition \( \pi_j = \{W_1, \ldots, W_P\} \) of the variables of the \( j \)-th cross-section, and then computing the entropy \( H^j(F) \) as:

\[
H^j(F) = \sum_{i=1}^{P} H(W_i).
\]

There might be several ways of choosing the partition \( \pi_j \); however, a similar problem (i.e., partitioning the set of input variables of a multiple-output Boolean function such that the variables belonging to different partitions are as uncorrelated as possible) has been encountered in the context of approximate finite state machine traversal [23], and efficient algorithms for its solution have been proposed in the last few years [24, 25, 26]; therefore, we are planning to use such techniques to improve the accuracy of the power estimation tool of [13].

Acknowledgments

We wish to thank Fabio Somensi (University of Colorado at Boulder) for valuable suggestions on the realization of ADD-based algorithms.

References