A Fast Pruning Technique for Low-Power Inexact Circuit Design

Johan Broc², Pierre-Emmanuel Gaillardon¹, Luca Amarti¹, Jaume Joven Murillo¹, Krishna Palem², Giovanni De Micheli³
Integrated Systems Laboratory (LSI), EPFL, Switzerland¹
Department of Computer Science, Rice University, TX, USA²

Abstract—Inexact Circuits are circuits in which the accuracy of the output can be traded for cost savings (energy, area and/or delay). In the context of advanced technology scaling and power density increase, inexact circuits appear to be very promising as a solution. In this paper, we present a novel pruning technique developed as a logic level method to select and prune parts of a digital circuit. The error is computed at each pruning step using probabilistic error propagation and Hamming distance computation, making the evaluation possible at runtime. The technique was validated on several parallel adder architectures. Experimental results proved the efficiency of the technique with Energy-Delay-Area product reduction of 1.8× for less than 10^{-4} % of relative error on the considered benchmarks at 45-nm technology node.

I. INTRODUCTION

In the context of advanced technology scaling and power density increase, circuit design innovations are required to keep the power budget under control. Inexact or approximate circuit design is a computing solution that trades circuit precision for cost reduction. Cost can be energy, area and/or delay. Approximate solutions are already heavily used in the field of computer science, especially in the multimedia domain where systems can tolerate varying amounts of error and still realize useful computations. For example, the need for exact results is not justified in applications that interact with human perception, such as vision and audition [1]. Thereby, exploiting approximate computing at the hardware level seems to be a relevant solution that can be applied to a range of applications in exchange for a higher energy efficiency.

Initial attempts of inexact circuit design focused on manual re-design of common arithmetic building blocks [2] or the use of logic synthesis to generate approximate circuits [3] by selecting portions of a circuit and applying logic simplifications that do not impact more than one circuit output at a time. Other synthesis techniques achieving power reduction are based on selectively stopping the clock in portions of the circuit where active and exact computation is not required [4], or use precomputation based on sequential logic optimization like in [5]. Lately, a concrete chip prototype [6] was developed employing a pruning technique. The chip implements an inexact 64-bit Kogge-Stone adder [6] and demonstrates a cost saving quantified through the Energy-Delay-Area Product (EDAP) and Hamming distance computation, making the evaluation possible at runtime. The technique was validated on several parallel adder architectures implemented using 45-nm and 180-nm technology nodes. Experimental results proved the efficiency of the technique with an EDAP reduction of 1.8× for less than 10^{-4} % of relative error at both technology nodes and an average execution time of 1.5 seconds on the considered benchmarks.

The remainder of this paper is organized as follows. Section II provides the required background on logic network manipulation. Section III introduces the novel pruning technique. Section IV validates, through experimental results, the efficiency of the technique. Section V concludes the paper.

II. BACKGROUND

All digital integrated circuits can be represented using Boolean logic networks [11]. A Boolean network is a Direct Acyclic Graph (DAG) whose nodes are components, such as gates, inputs, or outputs and whose edges represent wires. The direction of the edges follows the natural computation flow from inputs to outputs. The terms logic network, Boolean network, and logic circuit are used interchangeably in this paper. In this work, Boolean networks employ AND, OR, INV and XOR operators as basic Boolean primitives. Every Boolean primitive is represented by a Truth Table (TT) of size 2^m (where m is the number of inputs), allowing us for easy network manipulation, simulation and computation.

In addition to its TT, a node in a logic network is associated with three main functional parameters: the Significance (S), the Switching Activity (α) and the Fanout. The significance S measures the value of the information carried by a node through its impact onto the network outputs. For example, we consider an adder. A node impacting the Most Significant Bit (MSB) will have a higher S as compared to a node impacting only the Less Significant Bit (LSB). The fanout of a node corresponds to the number of nodes that it drives. The power consumption of a logic gate can be estimated early on the logic network by considering the α and the fanout of its corresponding logic node. Indeed, according to [12], the dynamic power of a logic gate can be expressed as:

\[ P_{\text{dyn}} = \frac{1}{2} \alpha f C_L V_{dd}^2 \] (1)

where \( V_{dd} \) the gate supply voltage, \( C_L \) the load capacitance and \( f \) the signal frequency (usually the clock frequency). The fanout is directly linked to the load capacitance (also called fanout capacitance). The computation of the switching activity at a circuit node \( n \) involves the estimation of the signal probability \( p_1(n) \), which is the probability that the signal value at the considered node \( n \) is ‘1’. Assuming temporal independence of the inputs and a zero-delay model [12], the switching activity of a node \( n \) is given by:

\[ \alpha = p_1(n)(1 - p_1(n)) \] (2)

A factor two appears in [12] but, for practical reasons, we deal with the normalized formula in Eq. 2. In addition, we assume the
estimation formulate to remain a good approximation when the inputs are not statistically independent. The signal probability of the output of a logic gate $p_1(out)$ can be estimated from the knowledge of the signal probability of its inputs ($p_{in}$) and the Boolean function that it computes. This can be expressed as:

$$p_{1(out)} = \sum_{k=1}^{2^N} (TT(k) \cdot p_{1(in_k)})$$

where $TT(k)$ is the truth table value obtained for an input combination $k$ and $p_{1(in_k)}$ is the probability that the cube corresponding to the input combination $k$ evaluates to '1'. In this paper, this method is only used on nodes representing 2-input Boolean primitives. On a global Boolean network, it is possible to inherit the probability to have '1' from circuits input(s) till output(s) using a depth-first propagation of Eq. (3) and Eq. (2). Note that we do not claim this approximation to be a good estimation of the real power consumption (as many parameters have been neglected, e.g., glitching, signal correlations, impact of technology mapping), but rather a good comparison estimator to quickly identify the largest power contributors in a logic network.

III. PRUNING TECHNIQUE FOR ENERGY/ERROR TRADEOFF IN INEXACT CIRCUIT

In this section, we introduce our novel fast pruning technique.

A. Pruning Algorithm

The pseudo-code of the pruning algorithm is given in Alg. 1. We introduce it through the example given in Fig. 1. We considered the logic network that describes $X = (A \cdot B), (A \cdot B \oplus C)$ and $Y = (A \cdot B \oplus C) + A \cdot B + C$ (Fig. 1-a). In Fig. 1-b, a cost value $C$, derived from $S$ and $\alpha$ has been assigned to all the nodes in the network (Alg. 1-A). The computation of $S$, $\alpha$ and $C$ is fully discussed in III-B. The node with the highest cost is selected for pruning. In our example, node 4 is selected (Fig. 1-c). The node is removed and its initial output is re-routed to either one of its inputs or to the logic 0/1 (Alg. 1-B). The re-routing minimizes the introduced local error by considering the Hamming distance $H$ between the truth table of the pruned node and the nodes driving its inputs. A full discussion is given in III-C. The Hamming distance measures the minimum number of substitutions required for changing bits between two binary numbers. In Fig. 1-c, the output of node 4 is re-routed to the output of node 2. Hence, we obtain a new Inexact circuit (Fig. 1-d) that minimizes the Boolean circuit error.

Before the next pruning step, we compute the circuit logic error (Alg. 1-C) using probabilistic error propagation [9], [10] and Hamming distance computation. Pruning steps are repeated until the Error Threshold is reached. Finally, for every modification, i.e., at each pruning step, a decision is made to keep the modified node or not depending on the final circuit error. If the error threshold is reached, the last modification is canceled to strictly stay below the set point.

B. Node Selection and Associated Cost Function

The presented pruning technique uses a cost function to identify the most relevant node to be pruned. The function considers three parameters: the Significance ($S$), the Switching Activity ($\alpha$) and the Fanout. A relevant node candidate is identified by a low significance, i.e., a low impact on the output results, but a high Switching Activity - Fanout product, i.e., a high contribution to the dynamic power consumption. Therefore, we introduce the following cost function:

$$Cost = \left( \frac{S_{MAX}}{S} \right) \cdot \beta + \left( \frac{Fanout.\alpha}{Fanout.\alpha_{MAX}} \right) \cdot (1 - \beta)$$

Alg. 1 Pruning Algorithm Flow and Illustrative Example

INPUT: Boolean network flattened on basic primitives

USER INPUT:
- Circuit output significance
- Circuit input probabilities $p_1$
- Ratio
- Error threshold
- Circuit input error distribution

while CircuitError $\leq$ Error Threshold do
  for (All circuit nodes) do
    COMPUTE & ASSIGN $\rightarrow$ Significance
    COMPUTE $\rightarrow$ Cost
  end for
  SELECT node with highest cost $\rightarrow$ node
  MODIFY (node)
  COMPUTE CIRCUIT ERROR $\rightarrow$ CircuitError
end while

UNDO last modification

OUTPUT: Boolean network pruned flattened on basic primitives
where $S_{MAX}$ is the maximum significance value over the logic circuit, $(\text{Fanout} . \alpha)_{MAX}$ is the maximum value over the circuit for the product between fanout and switching activity and $\beta$ is a ratio between the two terms. The left term of the cost function characterizes the impact on the output error while the right terms shows the contribution to their power consumption. Both terms are normalized and have a value between 0 and 1 included. The normalization allows us to easily tune the technique towards either more energy efficiency or lower error, depending on the application.

The significance $S$ and the switching activity $\alpha$ are required to compute the cost function. They are assigned to the node by the following methodologies:

a) Significance $S$: First, we assume that the significance of the primary outputs of a circuit is given as a design parameter. The parameter is called Circuit output significance in Alg. 1. Then, we define the significance of a node as the sum of the significance of the children nodes connected to its output(s). Therefore, the significance for every node in the logic circuit is computed by depth-first search propagation, going from primary outputs to primary inputs. This computation is illustrated in Fig. 1-b. For example, the gate 3 that is connected to gates 5 ($S(5) = 2$) and 6 ($S(6) = 1$) has a significance $S(3) = 3$.

b) Switching Activity $\alpha$: The switching activity in a logic circuit is computed bottom-up, starting from the primary inputs. First, the signal probabilities (of being ‘1’) are computed for all nodes, in a recursive manner, according to Eq. (3). As boundary conditions, the probabilities for the primary inputs are defined by the designer and denoted Circuit input probabilities $p_i$ in Alg. 1. After that all probabilities are computed, the switching activity for each node is computed as $p_i(1 - p_i)$.

C. Node Modification Strategy for Error Minimization

When a node is pruned, its output has to be re-routed. We expect the re-routing to minimize the introduced local error. We evaluate the introduced local error by the Hamming distance between the initial truth table of the pruned node and truth tables of the nodes driving its inputs. Three possible re-routings can be identified: (i) re-routing of the output(s) to one of the $n$ gate inputs; (ii) re-routing of the output(s) to Power supply $V_{CC}$ (logic 1); and (iii) re-routing of the output(s) to the ground $GND$ (logic 0). When different solutions have the same Hamming distance, we arbitrary select one. In the example in Fig. 1-c, we show the re-routing of the node 4. We can either connect its output to the node 2, the input $C$ or logic 0/1. We discard the re-routing to logic 0 because of its higher Hamming distance. We then select the first solution as the re-routing strategy, and connect the output to node 2.

D. Fast Error Evaluation of the Pruned Circuit

In order to quickly evaluate the error introduced during the pruning operation, we present a fast error computation method. In previous works, the circuit error is computed at each steps using time-consuming logic simulations. Here, after every pruning steps, we evaluate the circuit error using the following relation:

$$p_e(\text{circuit}) = \sum_{q=1}^{M} S.p_e(\text{out}_q)$$  \hspace{1cm} (5)

where $M$ is the number of primary outputs and $p_e(\text{out}_q)$ the error probability at the output $q$. Such a relation takes into account the impact of the circuit output significance $S$ to identify the outputs that tolerate a low error probability.

The error probability $p_e(\text{out}_q)$ of an output $q$ is computed using the following relation:

$$p_e(\text{out}) = \frac{1}{2^n} \sum_{i=1}^{2^n} p_e(f_i)$$  \hspace{1cm} (6)

where $n$ is the number of inputs of the node and $p_e(f_i)$ is the error probability associated to each line of the truth table. The contribution of each lines $p_e(f_i)$ is evaluated under the following two cases:

CASE 1 - If the node is intact, i.e., has not been pruned, we consider the impact of having a non-null error probability at the inputs that can be propagated to the output:

$$p_e(f_i) = \frac{1}{2^N} \sum_{i=1}^{2^N} p_e(f_{ij})*|\text{TT}(i) - \text{TT}(j)|$$  \hspace{1cm} (7)

where the $i$-th truth table line error probability is the sum of the error probability that line $i$-th is transformed in the $j$-th line multiplied by the Hamming distance between the line $i$-th and $j$-th of the truth table ($|\text{TT}(i) - \text{TT}(j)|$). Intuitively, it corresponds to the probability of jumping from one line of the truth table to another due to an input error.

CASE 2 - If the node has been pruned, we consider its modified truth table, i.e., accounting for the re-routings:

$$p_e(f_i) = \frac{1}{2^N} \sum_{i=1}^{2^N} p_e(f_{ij})*|\text{TT}_m(i) - \text{TT}(j)|$$  \hspace{1cm} (8)

where the $i$-th truth table line error probability is the sum of the error probability that line $i$-th is transformed in the $j$-th line multiplied by the Hamming distance between the correct truth table $\text{TT}(j)$ and the modified one $\text{TT}_m(i)$.

The overall circuit error is therefore computed by a depth-first propagation from primary inputs to primary outputs. The pseudo-code of the error evaluation function is shown in Alg. 2. By default, we assume the the error probability of the inputs is null, but a non-null error can be given as parameter, called Circuit input error distribution in Alg. 1. Such a method operates at the logic network level and is much faster than a simulation-based approach.

Algorithm 2 Fast error estimation using Hamming distance

GRAPH ERROR PROPAGATION (CurrentNode)
if (CurrentNode is a Primary Input OR CurrentNode was already visited) then
RETURN CurrentNode → Error : $p_e(\text{out})$
else
if CurrentNode → TruthTable in not modified then
$p_e(f_i) = \sum_{k=1}^{2^n} p_{ij} |\text{TT}(i) - \text{TT}(j)|$
else
CurrentNode → TruthTable is modified
$p_e(f_i) = \sum_{k=1}^{2^n} p_{ij} |\text{TT}_m(i) - \text{TT}(j)|$
end if
$p_e(\text{out}) = \frac{1}{2^n} \sum_{i=1}^{2^n} p_e(f_i)$
RETURN CurrentNode → Error : $p_e(\text{out})$
end if

IV. EXPERIMENTAL RESULTS

In this section, we illustrate the interest of the proposed pruning technique by applying it to various adder circuits.

A. Methodology

The proposed pruning technique is implemented using C language. Synopsys Design Compiler is used to pre-synthesize an HDL description into Boolean networks of basic primitives. After pruning, Design Compiler is used to map the circuit onto a standard cell library and to evaluate the circuit-level performance. We consider as evaluation metrics the final area, the delay and the energy of the resulting circuit, as well as energy-delay product and Energy-Delay-Area Product (EDAP) as meaningful figure-of-merits. We consider two benchmarks, a 64-bit Brent-Kung and a 64-bit Han-Carlson adder, both implemented into 45-nm and 180-nm CMOS technology.
For the pruning setup, we set the input switching activities to 0.25 because of the equiprobability to have 0 or 1 at the inputs \( p_i = 0.5 \). Likewise, the input error is considered null. The output significance is assigned according to the bit position in the output word, i.e., \( S(i) = 2^i \). For our experimental results, we observed better result for the adders by choosing a \( \beta \) parameter benefiting the significance, i.e., \( \beta = 1 \). Other values can be used to reach other optimization points. A detailed discussion about the \( \beta \) parameter is out of the scope of this paper.

### B. Post-pruning Experimental Results

The normalized gains on the two 64-bit adder architectures and for the two considered technologies are summarized in Fig. 2. The curves show the gains for an increasing target error threshold, i.e., that we increase the number of node that can be prune over the circuit. For each pruned circuit, we evaluate the relative error computation.

Our pruning technique results in large energy and area saving with an EDAP of \( 1.8 \times \) for less than \( 10^{-5}\% \) of relative error for the Brent-Kung adder and \( 1.7 \times \) for less than an average \( 10^{-2}\% \) for the Han-Carlson adder for both technology nodes compared to their conventional non-pruned counterparts. Those results are showing similar performances than in current state-of-the-art pruning techniques [2, 3]. The pruning technique has an average time of execution of 1.5 seconds.

### V. Conclusions

We proposed a novel pruning technique developed to generate approximate circuits. The introduced pruning error is computed at each step using a probabilistic error propagation and Hamming distance computation making the evaluation possible at runtime. The technique was validated on several parallel adder architectures implemented using both 45-nm and 180-nm technology nodes. Experimental results proved the efficiency of the technique with Energy-Delay-Area product reduction of \( 1.8 \times \) for less than \( 10^{-5}\% \) of relative error on the considered benchmarks. This technique provides the designers with a general purpose CAD tool capable to trade the computation quality for large power consumption, area and delay of a circuit.

### ACKNOWLEDGEMENTS

This research is supported by the grant ERC-2009-AdG-246810.

### REFERENCES