A Fast Pruning Technique for Low-Power Inexact Circuit Design

Johan Broc[1], Pierre-Emmanuel Gaillardon[1], Luca Amarú[1], Jaume Joven Murillo[1], Krishna Palem[2], Giovanni De Micheli[1]
Integrated Systems Laboratory (LSI), EPFL, Switzerland[1]
Department of Computer Science, Rice University, TX, USA[2]

***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) of 1.2-1.6× with an acceptable error bound of less than 0.1% of relative error. Thus, inexact circuit proved their operational efficiency when result quality is not purely needed. Commonly used design techniques for inexact circuits prune, i.e., delete components of an exact logic circuit [7], [8]. At every pruning steps, critical components are selected and pruned in order to minimize the output error while maximizing the power reduction. This selection is typically based on two parameters: the *significance*, which measures the components impact on the output results and the *activity*, which is directly related to the component power consumption. The limit of the actual pruning techniques is the computation time required to prune a logic circuit. Indeed, for every pruning step, the introduced error in the circuit is evaluated through logic simulation using representative set of input vectors. The simulation complexity grows quadratically with the number of inputs and the number of input vectors making the pruning process hardly tractable for large circuits.

In this paper, we present a novel pruning technique that reduce runtime complexity, exploiting a logic level method to select and prune digital circuits. The error is computed at each pruning step using a probabilistic error propagation [9], [10] and Hamming distance computation making the evaluation tractable 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^n (where n is the number of inputs), allowing us for easy network manipulation, simulation and computation.

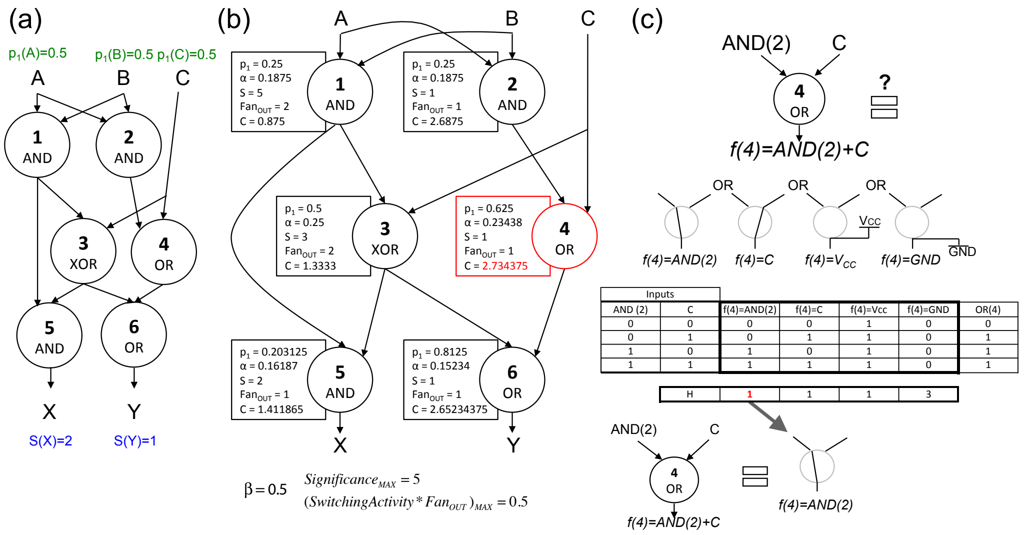
In addition to its TT, a node in a logic network is associated with three main functional parameters: the *Significance* (S), the *Switching Activity* (α) and the *Fanout*. The significance *S* measures the value of the information carried by a node through its impact onto the network outputs. For example, 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 :

$$P_{dyn} = \frac{1}{2}\alpha f C_L V_{dd}^2 \tag{1}$$

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:

$$\alpha = p_1(n).(1 - p_1(n)) \tag{2}$$

A factor two appears in [12] but, for practical reasons, we deal with the normalized formula in Eq. 2. In addition, we assume the


(a)
p1(A)=0.5
p1(B)=0.5 p1(C)=0.5
A
B
C
1 AND
2 AND
3 XOR
4 OR
5 AND
6 OR
X
Y
S(X)=2
S(Y)=1
(b)
p1 = 0.25, α = 0.1875, S = 5, FanOUT = 2, C = 0.875
p1 = 0.25, α = 0.1875, S = 1, FanOUT = 1, C = 2.6875
p1 = 0.5, α = 0.25, S = 3, FanOUT = 2, C = 1.3333
p1 = 0.625, α = 0.23438, S = 1, FanOUT = 1, C = 2.734375
p1 = 0.203125, α = 0.16187, S = 2, FanOUT = 1, C = 1.411865
p1 = 0.8125, α = 0.15234, S = 1, FanOUT = 1, C = 2.65234375
β=0.5
Significance_MAX = 5
(SwitchingActivity * FanOUT)_MAX = 0.5
(c)
AND(2)
C
?
f(4)=AND(2)+C
OR
f(4)=AND(2)
f(4)=C
f(4)=VCC
f(4)=GND
Vcc
GND
(d)

Inputs						
AND (2)	C	f(4)=AND(2)	f(4)=C	f(4)=Vcc	f(4)=GND	OR(4)
0	0	0	0	1	0	0
0	1	0	1	1	0	1
1	0	1	0	1	0	1
1	1	1	1	1	0	1
	H	1	1	1	3	

Fig. 1. Illustration of a pruning step applied on a 2-output Boolean network (a) Considered network with $X = (A.B).(A.B \oplus C)$ and $Y = (A.B \oplus C) + A.B + C$ (b) Assignment of cost parameters and relevant node selection. (c) Modification strategy of the selected node. Modification is performed according to the Hamming error introduced by the transformation. (d) Final Boolean network after one pruning step ($X = (A.B).(A.B \oplus C), Y = (A.B \oplus C) + A.B, RelativeError = 0\%$).

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 $p_1(in_k)$ and the Boolean function that it computes.This can be expressed as:

$$p_1(out) = \sum_{k=1}^{2^N} (TT(k).p_1(in_k)) \tag{3}$$

where $TT(k)$ is the truth table value obtained for an input combination k and p_1 (in_k) 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 $X = (A.B).(A.B \oplus C)$ and $Y = (A.B \oplus C) + A.B + C$ (Fig. 1-a). In Fig. 1-b, a cost value C, derived from S and α has been assigned to all the nodes in the network (Alg. 1-A). The computation of S, α 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-routeed 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-routeed 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* (α) 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:

$$Cost = \left(\frac{S_{MAX}}{S}\right).\beta + \left(\frac{Fanout.\alpha}{(Fanout.\alpha)_{MAX}}\right).(1-\beta) \tag{4}$$

Algorithm 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 $\rightarrow$ $Significance$, $SwitchingActivity$   } A
      COMPUTE $\rightarrow$ $Cost$
    end for
    SELECT node with highest cost $\rightarrow$ $node$   } B
    MODIFY($node$)
    COMPUTE CIRCUIT ERROR $\rightarrow$ $CircuitError$   } C
  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, $(Fanout.\alpha)_{MAX}$ is the maximum value over the circuit for the product between fanout and switching activity and β 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 α 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 α: 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_1 in Alg. 1. After that all probabilities are computed, the switching activity for each node is computed as $p_1.(1 - p_1)$.

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:

$$p_e(circuit) = \sum_{q=1}^{M} S.p_e(out_q) \tag{5}$$

where M is the number of primary outputs and $p_e(out_q)$ 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 $p_e(out_q)$ of an output q is computed using the following relation:

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

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:

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

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 ($|TT(i) - TT(j)|$). 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:

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

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 $TT(j)$ and the modified one $TT_m(i)$.

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 $CurrentNode \rightarrow Error : p_e(out)$
else
    if $CurrentNode \rightarrow TruthTable$ in not modified then
        $p_e(f_i) = \sum_{k=1}^{2^N} p_{f_{ij}}.|TT(i) - TT(j)|$
    else
        $CurrentNode \rightarrow TruthTable$ is modified
        $p_e(f_i) = \sum_{k=1}^{2^N} p_{f_{ij}}.|TT_m(i) - TT(j)|$
    end if
    $p_e(out) = \frac{1}{2^N} \sum_{i=1}^{2^N} p_e(f_i)$
    RETURN $CurrentNode \rightarrow Error : p_e(out)$
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.

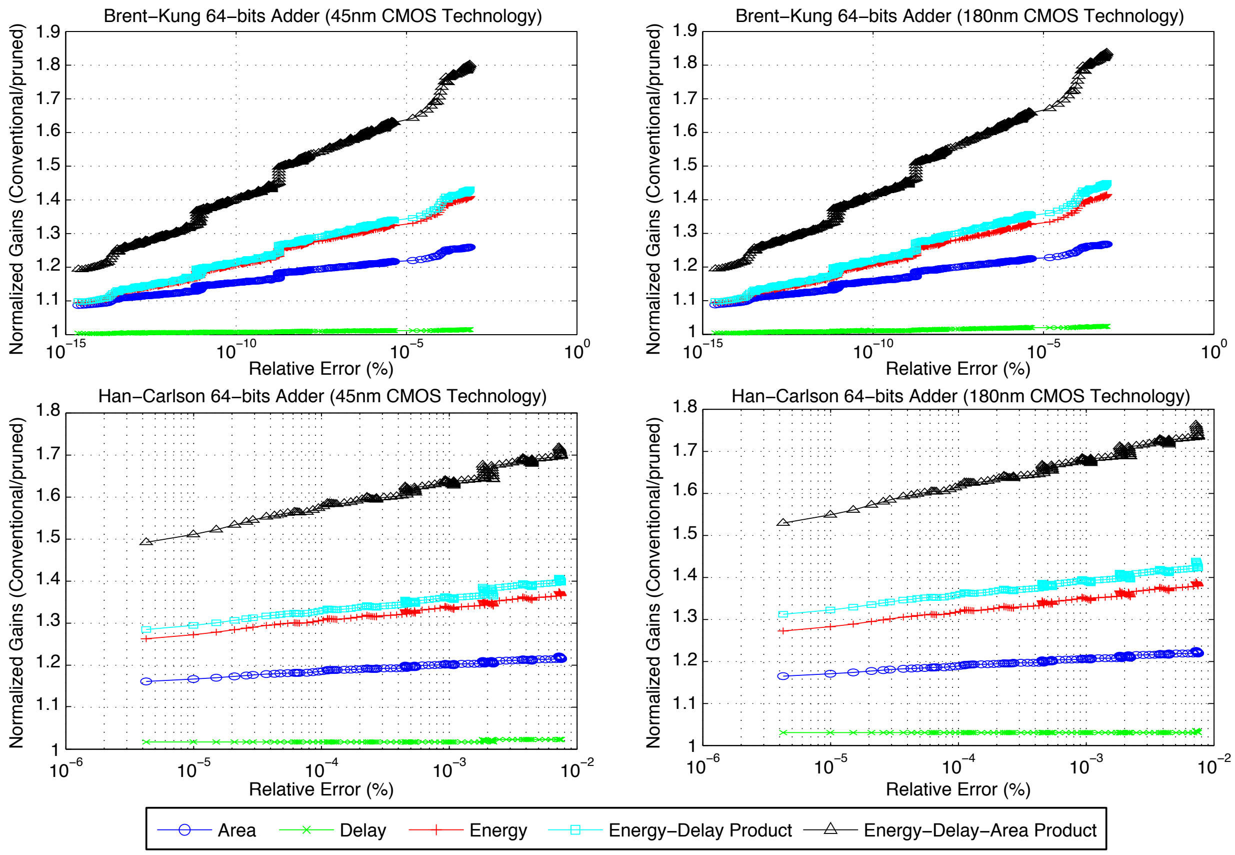


Fig. 2. Normalized gains vs. relative error percentage of various probabilistic pruned 64-bit adders

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 ($p_1(in) = 0.5$). 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 β parameter benefiting the significance, i.e., $\beta = 1$. Other values can be used to reach other optimization points. A detailed discussion about the β 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× for less than 10^{-4}% of relative error for the Brent-Kung adder and 1.7× 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 [8]. The pruning technique has an average time of execution of 1.5 seconds[1].

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× for less than 10^{-4}% 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.

[1]No comparisons with previous work are presented due to the lack of data available in literature.

Acknowledgements

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

References

[1] M. A. Breuer, *Multi-media Applications and Imprecise Computation*, DSD Tech. Dig., 2005.
[2] V. Gupta, *et al.*, *IMPACT: Imprecise adders for low-power approximate computing*, ISLPED Tech. Dig., 2011.
[3] S. Venkataramani, *et al.*, *SALSA: systematic logic synthesis of approximate circuits*, DAC Tech. Dig., 2012.
[4] L. Benini and G. De Micheli, *Transformation and synthesis of FSMs for low-power gated-clock implementation*, ISLPED Tech. Dig., 1995.
[5] M. Alidina, *et al.*, *Precomputation-Based Sequential Logic Optimization for Low Power*, IEEE TVLSI, 2(4):426-436, 1994.
[6] A. Lingamneni, *et al.*, *Designing Energy-Efficient Arithmetic Operators using Inexact Computing*, J. of Low Power Elec., 9(1):141-153, 2013.
[7] N. Pippenger, *et al.*, *Analysis of carry propagation in addition: An elementary approach*, J. of Algorithms, 42:317-313, 2002.
[8] A. Lingamneni, *et al.*, *Energy parsimonious circuit design through probabilistic pruning*, DATE Tech. Dig., 2011.
[9] N. Mohyuddin, E. Pakbaznia and M. Pedram, *Probabilistic error propagation in logic circuits using the Boolean difference calculus*, ICCD Tech. Dig., 2008.
[10] R.C. Ogus, *The Probability of a Correct Output from a Combinational Circuit*, IEEE TOC, 24(5):534-544, 1975.
[11] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.
[12] M. Pedram, *Power minimization in IC design: principles and applications*, ACM TODAES, 1(1):3-56, 1996.