Systematic Approximate Logic Optimization Using Don’t Care Conditions

Sahand Salamat¹, Mehrnaz Ahmadi¹, Bijan Alizadeh¹,², Masahiro Fujita³
¹School of Electrical and Computer Engineering, College of Engineering, University of Tehran, P.O.Box 14395-515, Tehran, Iran.
²School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
³VLSI Design and Education Center, the University of Tokyo, P.O.Box 113-8654, Tokyo, Japan.
E-mail: {s.salamat, m.ahmady, b.alizadeh}@ut.ac.ir, fujita@ee.t.u-tokyo.ac.jp

Abstract

Approximate computing has emerged as a new design paradigm in wide variety error resilient applications to decrease the hardware cost in an acceptable deviation from the nominal output values. In this paper, a logic approximation technique is proposed which takes the advantage of error budget in error tolerant applications to achieve an approximation circuit with smaller area and latency. Moreover, a heuristic is proposed to prune the search space, which increases the scalability of the proposed technique. The results show 38.3% reduction in area and 26.6× speed up compared to the state-of-the-art approximation techniques.

Keywords

Approximate computing, error tolerable, latency.

1. Introduction

Several applications including image processing and audio processing, to name but two, use approximate computing in their underlying computations [1]. Since these applications are able to tolerate erroneous values at their output in case the error rate remains in the acceptable range. Approximate computing improves the energy consumption as well as performance.

Hardware faults can cause errors at the output of the circuit. In [2], these errors are classified into acceptable and unacceptable for image and video compression applications. It is shown that a large percentage of stuck at faults on interconnects lead to negligible erroneous outputs. Two main quantitative metrics have been previously proposed to measure severity of errors: (I) Error Significance (ES) and (II) Error Rate (ER). ES is defined as the maximum amount by which the numerical value at the outputs of an imperfect circuit can deviate from the corresponding value for the perfect circuit. ER is the percentage of vectors for which values at a set of outputs deviate from the error-free response, during the normal operation. A combination of these two metrics has also been employed by [3].

A common technique for approximate computation is to design the approximate circuits by employing hardware techniques to make a trade-off between accuracy and efficiency [1]. These techniques can be classified into two categories: (I) timing approximation, where timing errors are introduced in the circuit [4], and (II) functional approximation, where the functionality of the circuit is slightly modified to generate a more efficient design [5]. In [4] the supply voltage is scaled to save energy consumption. The scaling is done until the resulting timing errors of the circuit do not violate an application-specified ER. There have been introduced several approaches for the functional approximation of the circuits. In [5], a functional approximation technique is presented based on complementing the minterms in Karnaugh map of the logic design. Since all possible minterms need to be evaluated to find the best choice, this technique is not applicable to large circuits. In [6] a new heuristic technique for functional approximation is proposed. This heuristic inserts a stuck-at fault at a node of the critical path that needs to be approximated, and, to simplify the path, explores the fault forward and backward propagation on the path. The signals in the path are replaced with fixed values which results in area reduction for a given ER. In [7] the isomorphism between error magnitude and Boolean relation is proved. Then an approximate logic synthesis under error magnitude is introduced. This approximation technique finds the optimal set of minterms on which the exact outputs must be enforced under the error magnitude constraint. Then the error frequency constraint is applied and the best approximation under this constrain is obtained. The method of [8] is applying prime implicant expansion and reduction in order to reduce the number of literals. Nonetheless, without pruning the search space this method suffers from tremendous amount of possible expander and reducer. A BDD-based approximation technique is presented in [9] where some operators are applied to derive approximated functions. Furthermore, some algorithms are introduced to compute the error metrics directly on the BDD representation. This technique, however, is not applicable to large designs due to BDD space explosion. Authors of [10] approximate the logic value of critical paths using the gates inside each cone of the critical path. Although this technique boasts a very low overhead, it is not effective in 57% of benchmark circuits, mainly because of dependency on the existing gates in the cone of the critical path. Moreover, since the logic gates do not have the same inputs as the original path, the probability of inequality is high.

Approximating the behavior of a circuit is addressed in this paper in order to reduce the area and runtime complexity under a user-defined ER. In other words, a gate-level circuit with less area compared to the original circuit is generated. Hence, our main contributions in this paper are as follows:

- Reducing the area of the circuit with respect to the predefined ER using the don’t care injection to the special minterms of a given Boolean logic.
- Reducing the runtime complexity with pruning the search space for the approximation circuit. Note that, such a pruning mechanism makes our heuristic technique applicable to industrial large-size designs due to its independency from the circuit size as well as ER.

The rest of this paper is organized as follows. Section 2 gives the preliminaries to our approximation technique. Section 3 presents our circuit approximation techniques including random and heuristic techniques. Section 4 reports the results and Section 5 concludes the paper.
2. Preliminaries

2.1. Basic Definitions

**Definition 1**: Minterm complement threshold (C_t): C_t is the maximum number of minterms of a Boolean function that can be complemented under a maximum allowed ER. For an n-input Boolean function with maximum allowed ER, up to C_t = ER×2^n minterms are allowed to be complemented.

**Definition 2**: Adjacent minterms: two minterms are considered adjacent to each other if their hamming distance is one and they only differ in one bit. So, the number of adjacent minterms of each minterm in a Karnaugh map equals the number of Karnaugh map’s inputs. For example, in a three input function the minterm $x_1x_2x_3$ has 3 adjacent minterms $\overline{x_1}x_2x_3$, $x_1\overline{x_2}x_3$, and $x_1x_2\overline{x_3}$.

a) **Adjacent 1**: minterm $m_1$ is adjacent 1 of minterm $m_2$ only if $m_1$ is adjacent to the $m_2$ and the Boolean function of $m_1$ is one.

b) **Adjacent 0**: minterm $m_1$ is adjacent 0 of minterm $m_2$ only if $m_1$ is adjacent to the $m_2$ and the Boolean function of $m_1$ is zero.

**Definition 3**: Karnaugh map Error set (KE): KE consists of a set of minterms in a given Karnaugh map whose values should be changed to don't care value. This set injects errors which are supposed to simplify the circuit.

**Definition 4**: Approximated Karnaugh maps (App-Kmaps): App-Kmaps includes a set of Karnaugh maps that each member of the set is in a correspondence with a unique KE. Therefore, for any specific member of App-Kmaps, namely App-Kmaps, there exists a KE member (KEi) such that:

\[
\forall K\text{-map}_i, K\text{-map}_j \in K\text{-maps} : K\text{-map}_i \neq K\text{-map}_j, KE_i \neq KE_j
\]

2.2. Basic Idea

In order to clarify our main contributions, we explain our ideas for the multi-output Boolean function approximation through an example. Let us consider the example shown in Fig. 1, which represents a two-output circuit. To begin with, the most critical path of the circuit (i.e. the path with the highest latency) is extracted which in our case is the cone of out1. The logic of out1 is represented bellow.

\[
\text{out1} = ab\overline{c} + ab\overline{d} + \overline{b}cd + cda
\]

The relation between the output whose latency is more than expected latency for the design (critical cone) and its inputs is depicted in Karnaugh map of Fig. 2(a), in which the related implicants are highlighted. In the next step, errors should be injected to the Karnaugh map so as to simplify the logic. The simplifications should lead to area reduction while do not exceed ER. One solution is considering all the possible combinations of minterms in Karnaugh map in which the error rate retains less than ER. Then change the minterms in to Don’t care values, and choose the circuit considering both area and latency. Another solution is to randomly choose some of the possible combinations. The last but not the least significant solution is a smart selection, which chooses the minterms assisting more expanding primary implicants in the original minimum cover. Since the zero minterms with larger number of adjacent ones have higher chance to merge with their adjacent

![Figure 1: A two-output Boolean logic](image)

![Figure 2: Simple example, (a) original circuit Karnaugh map, (b) approximated Karnaugh map](image)

implicants when they are complemented, they are more likely to reduce the number of literals in the approximated logic. For instance, in our example the 1101 minterm is surrounded by more one minterms, i.e. three one minterms. To investigate the effect of individual minterm approximations, first let us consider complementing the 1101 minterm. The approximated logic related to this newly modified minterm is as follows:

\[
\text{out1} = cd + bd + b\overline{c}d
\]

The obtained logic has two fewer implicants compared to the original logic and both area and circuit latency decrease. The newly generated logic approximates the original one for $16/15 = 93.75\%$. In addition, if higher error rate is acceptable, complementing the minterm 0001 (shown in Fig. 2(b)) leads to even more simplified logic. The Boolean function of Fig. 2(b) is:

\[
\text{out1} = bc + cd
\]

This circuit has less area and latency as compared to the related circuit of (1). However, the similarity between (3) and the original circuit is decreased to $14/16 = 87.5\%$. As illustrated in this example, there is a trade-off between similarity and the area, the more the similarity, the less reduction in the area in general.

3. Proposed Approximation Techniques

In this section, we propose three approximation techniques by considering some of the minterms as don’t cares in the Karnaugh maps of circuits. These techniques approximate the functionality of Boolean logics in order to reduce the area and latency of the original circuit such that accurate enough results are delivered. ALGORITHM 1 introduces the approximation
process and Line (3) represents our proposed techniques which are elaborated upon subsections 3.1, 3.2 and 3.3.

Our first approximation technique called **Exhaustive Don’t Care (EDC)** exhaustively searches the all minterms of Karnaugh map to find the best possible KE so that making the minterms of KE don’t care resulted to an approximated circuit with minimum area under given ER. This technique is precise; nonetheless, it is not applicable for all circuits but small-size. Our second approximation technique called **Random Don’t Care (RDC)** is our random solution to perform the approximation. In this technique, a set of KE is selected randomly and the corresponding App-Kmaps are evaluated. In this technique, the runtime is improved significantly and the technique is applicable for even large-size designs. The heuristic approximation technique called **Heuristic Don’t Care (HDC)** is our improved approximation technique in which the efficiency of random technique is increased through pruning the search space.

![Figure 3: The approximated circuit of logic in Fig.1.](image)

### ALGORITHM 1: Approximation Process

*Inputs:* (LOC, expected characteristics, Maximum allowed ER)  
*Output:* LAC: list of approximated cones

1. For each original cone $c \in$ LOC do  
2.   EC=Approximated_Cone_Initialization ($c$)  
3.   App-Kmaps= output of Approximation Technique -EDC, RDC, HDC- (EC)  
4.   AC= Best_Cone_Selection(App-Kmaps)  
5.   LAC= LAC $\cup$ AC  
6. End  

Return output list

- **LOC:** list of cones to be approximated (generated in LOC-detector of ALGORITHM 2.)  
- **Expected characteristics** and **Maximum allowed ER** are set by designer and based on application.  
- **EC:** the exact cone which is under approximation process.  
- **AC:** Approximated Cone.  
- **Approximated Cone Initialization ($c$):** This function accepts the gate level description of a cone as an input and extracts the related Karnaugh map. To extract the Karnaugh map, all possible input vectors are applied to the circuit and the corresponding outputs are stored in a file.  
- **Best Cone Selection(App-Kmaps) function:** The input of this function is App-Kmaps which are defined in blif format and whose characteristics—for each App-Kmap—are evaluated. This function looks for the best circuit, which has the smallest area and latency. If the constraints are not met the algorithm is considered inefficient. Otherwise, the circuit with the best characteristics is output of the function.

#### 3.1. Exhaustive Don’t Care (EDC) Approximation

Exhaustive Don’t Care technique is a straightforward technique to find the best possible approximated solution. We start with the list of cones to be approximated (LOC). In order to detect the LOCs, ALGORITHM 2, is applied to a gate level description of the circuit. In ALGORITHM 2, all paths inside the circuit, which have larger area than the expected ones, are determined and their corresponding cones are inserted in a LOC list. Additionally, this algorithm is able to extract the cones with respect to their latency. Thus, by approximating the critical paths, a logic with higher performance is achieved. This algorithm can be applied to any node in the circuit in general and any output in particular, thereby extracting the corresponding cone. Thus, this algorithm supports multi-output logics. ALGORITHM 3 describes our EDC technique. It starts with one of the cones in the LOC, namely *original cone*. First of all, the Karnaugh map of the *original cone* is extracted using **Approximated_Cone_Initialization** function and the exact cone (EC) is initialized with the *original cone* Karnaugh map (line 2 of ALGORITHM 1). Next, EDC approximation algorithm is applied to EC.

#### ALGORITHM 2: LOC-detector

*Input:* gate level description of circuit  
*Output:* LOC: list of cones to be approximated  
1. Extract paths with large area and their corresponding cones  
2. For each path do  
3.   While (cone inputs > maximum number of inputs) do  
4.     Explore the path backward and find the feeding nodes  
5.     Extract the cones of the feeding nodes  
6.   End  
7.   Add the cones to the LOC  
8. End  

Return LOC

- Maximum number of inputs is a predefined variable which is set due to the system limitations.

In this technique a unique KE is generated for each minterm by selecting a minterm value in the Karnaugh map and changing its value to a Don't care value (lines 3-6 of ALGORITHM 3). In line 5, the corresponding App-Kmap, is generated in a blif format, although any other data structure can be used to represent the modified Karnaugh map of the approximated logic. Each member of App-Kmaps is then synthesized and its area as well as its latency are evaluated and finally, among all App-Kmaps, the logic with the smallest area is chosen (line 7) If the area and latency of the logic is less than our limits EDC set the App-Kmaps as its output. Otherwise, if the current ER is less than the maximum allowed ER, the above steps are repeated on the App-Kmap of the best recent approximated circuit (line 2) to reach a new approximated logic with the required area. The best circuit among output of EDC function is selected by **Best Cone Selection** function and it is added to Approximated Cone List of output (line 5 of ALGORITHM 1). This iterative process is continued (line 1 of ALGORITHM 1) until all the cones become approximated. Since in this technique, inserting one error in the circuit requires calling synthesis tool for all minterms in the Karnaugh map, with the growth of the input number of logic, the size of Karnaugh map and consequently the runtime complexity grows exponentially.
By changing the value of minterm into don’t care both zero and one—complemented value and original value—is considered for that minterms. Hence, inserting a Don’t care value would cause no changes in the Karnaugh map when complementing the minterm introduces new primary implicant. Otherwise the complemented value is considered, causing reduction in terms of area. The size of EDC calculations and synthesis tool calls grows exponentially (i.e., \(2^n\) number of inputs). Our experimental results reveals that this technique is not applicable to even a medium-size benchmarks, because of immense runtime. Therefore, random and heuristic algorithms are proposed to reduce the amount of calculation, and increase effectiveness of technique.

**ALGORITHM 3: EDC Approximation technique**

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>EC: exact cone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>App-Kmaps and characteristics</td>
</tr>
</tbody>
</table>

1. \(AC = EC\)
2. **While** ((AC’s area > expected area) & (current ER< Maximum allowed ER)) do
3. **For each** minterm \(m \in AC\) do
4. \(KE = m\)
5. Generate App-Kmap
6. **End**
7. \(AC = Best\) Cone Selection (LSC)
8. **End**

**Return** App-Kmaps

**ALGORITHM 4: RDC Approximation**

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>EC: exact cone, number of sets, (C_i), time constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>App-Kmaps and characteristics</td>
</tr>
</tbody>
</table>

1. \(AC = EC\)
2. **While** ((\(i < \) number of sets)) do
3. \(KE = \) randomly \(C_i\) selected minterms
4. Generate App-Kmap
5. **END**

**Return** App-Kmaps

### 3.2. Random Don’t Care (RDC) Approximation

According to the fact that the complexity of EDC technique grows exponentially with the number of inputs, and it is only applicable to small-size circuits. RDC technique is, therefore, proposed to randomly select \(C_i\) minterms among all minterms of Karnaugh map instead of choosing the minterms one by one in a sequence manner (EDC technique). This random selection speeds up RDC technique and makes it applicable to larger designs. In this technique, likewise EDC, ALGORITHM 2 is employed to find the LOC with the maximum number of inputs equal to 20 which is a limiting factor of the synthesis tool, since for more than 20 inputs the synthesis time becomes dominant. Then RDC approximation technique (ALGORITHM 4) is applied to the LOC. The algorithm for each original cone in LOC is repeated. First, the “number of sets” (#set) unique KE which has \(C_i\) randomly selected members among all minterms are chosen (line 2-5). Note that in most of designs there are inputs for which the output of logic must be exact. For instance, activating the reset input for each logic must change its output to the exact initial value rather than the approximate initial value. In this case, the minterms with the constraint on their inputs (high value for reset) are omitted from the list of random selection. In line 4, the corresponding App-Kmap, is generated in a blif format. The App-Kmaps is returned as an output of function. Line 4 of ALGORITHM 1 determines the Approximation Cone (AC) with smallest area considering the latency requirements using the Best_Cone_Selection function. Finally, if the approximated circuit does not meet the constraints, the technique is reported as ineffective. Otherwise, the AC is added to approximated cones list. RDC runtime complexity is linearly proportional to the number of sets. Although the higher value for number of sets results in a better approximated circuit, the runtime is a limiting factor, and this parameter is set by the designer based on the runtime as well as performance.

### 3.3. Heuristic Don’t Care (HDC) Approximation

Although RDC technique reduces the runtime complexity, it suffers from tremendous search space of selecting minterms. Therefore, finding a set which leads to satisfying the area and latency constraints needs an immense amount of time. Omitting minterm candidates whose don’t care value have lower probability for area reduction during generating App-Kmaps phase, prunes the search space. To accelerate the approximation process and pruning the search space, we consider two issues in Heuristic Don’t Care Approximation (HDC) technique. First, since complementing a minterm of value zero has more chance of merging [6] and consequently the more reduction in literal numbers, the complementing of the zero minterms is only considered. Furthermore, the zero minterms with larger number of adjacent 1 in Karnaugh map have a higher chance to introduce a new implicant which can be merge with adjacent implicants, thereby making a new bigger implicant. These minterms are more willing to become don’t care in HDC. It is axiomatic that merging the implicants results in bigger primary implicants and almost reduces the area and the latency of the design. As illustrated in Fig. 4, the zero minterm with highest number of adjacent 1 is not always the best candidate for becoming don’t care. However, this minterm is still one of the candidates which results to reduction in terms of literals. The minterm 0101 have three adjacent 1 however, not only does the 0000 minterm have lower adjacent 1 than 0101 minterm, but its complementing is resulted to the maximum reduction as well.

**Figure 4:** The effect of number of adjacent on performance of approximation.

Changing a zero minterm into one does not always turn into the merging the implicants. Therefore, using the pruning techniques can strongly improve the efficiency of approximation technique as well as make the technique more scalable. The number of outputs for each circuit is limited to a fix value and therefore the circuit has bounded number of critical cones; additionally, each cone is approximated in a finite time, for our techniques runtime are dependent to the
number of inputs rather than size of the cone or number of gates in the cone. The maximum number of inputs for each cone is set to 20, and for logics with more inputs, the ALGORITHM 2 is applied which extracts the cones that satisfy the maximum number of inputs. Due to these facts the HDC algorithm is applicable for even large circuits, because all circuits consist of cones with inputs less than maximum number of inputs on which our techniques are applied.

The algorithm of HDC is presented in ALGORITHM 5. In this algorithm, first, the number of adjacent 1 for each zero minterm in the extracted Karnaugh map of the original cone is counted. Note that likewise ALGORITHM 4 the minterms with constraint in their inputs are not taken into account from the list. The zero minterms are then sorted out according to the number of their adjacent 1 in a descending order in the List of Zeros (line 2). If the number of adjacent 1 are equal for two or more minterms, their order is selected randomly. Next, we need to select C_i Zeros from the List of Zeros to create KE. According to the aforementioned pruning technique, although the zeros with larger amount of adjacent 1 are not always the best choice, they can be considered as candidates of changing to don’t care value. Accordingly, we choose C_i Zeros from the top #Zero Candidates (#ZC) of the List of Zeros which leaves us with (#ZC, C_i) choices to select a group of C_i minterms. Evaluating all these selected groups of minterms is very time consuming and it is impractical to large circuits. To make the algorithm applicable, we only choose #set groups of C_i minterms randomly from the top #ZC of list of zeros and each group forms a KE_i (lines 3-6). Like previous techniques best circuit is chosen among #set App-Kmapi as the approximated cone. In our approximation technique the values for #ZC and #set are determined due to the size of original cone. To find the best value for #ZC and #set, we performed HDC technique on benchmark circuits. Fig.5 depicts the area of the approximation circuit for different values of #ZC and #set for 9sym benchmark circuit which has 9 inputs (n=9). In this figure, the area of the approximated circuit decreases sharply for #set < 2^#ZC. While, for larger values of #set, the graph shows a slight decrease and the slope of the graph is almost negligible. Additionally, according to Fig. 6 increasing the #set causes the processing time to grow linearly. So, we set #set equal to 2^#ZC in our HDC technique. Moreover, it can be inferred from Fig. 5 that the best value for #ZC is equal or smaller than 2^#ZC. As illustrated in Fig. 5. Choosing the value 2^#ZC for #ZC results in the minimum area for the approximated circuit.

3.4. Example of heuristic technique

In this example we apply our heuristic technique to the critical cone of Fig. 1. The Verilog gate level description of circuit and maximum allowed ER is given to HDC algorithm. In the first step, the Karnaugh map of this circuit is extracted from the gate level description of the original cone using Modelsim Altera. Next, the number of adjacent 1 for each zero minterm in the Karnaugh map is calculated. Fig. 7 represents the number of adjacent 1 for each zero minterm in the Karnaugh map. In this figure, the gray area represents the 1101 minterm and its adjacent. Zero minterms are then sorted in a descending order of their number of adjacent 1. Table 1 is an example of List of Zeros for Karnaugh map of Fig. 7. In this table minterms with equal number of adjacent 1 are sorted in a random order. The circuit of this example has 4 inputs and according to our simulations the best value for #set and #ZC is equal to 2^4 = 16. Next, we select #ZC minterms from the top of the List of Zeros which includes 1101, 0100,1010, and 0110 minterms. The minterms with equal number of adjacent 1 are sorted in a random order. Therefore, the chance of being in a top #ZC of the list for all these minterms is equal. If C_i equals to one, four sets are selected. Since ∀ KE_i ∈ KE are Non-repetitive the probability of selecting 1101 in one of our random sets is 100% which makes the results of HDC technique similar to those of EDC technique. If C_i equals two, then we have (#ZC, C_i) = 6 choices. By choosing four different sets, four unique KE_i and their corresponding App-Kmap, is generated for each set in blif format. Finally, each blif file is synthesized into the gate level description and the circuit with minimum area is selected as an approximated circuit. If one of the sets includes both 1110 and 0001 minterms, the result of this technique is the same as that of the exhaustive technique.

![Figure 5: Impact of #set and #ZC on performance for 9sym circuit.](image)

![Figure 6: Impact of #set and #ZC on CPU time for 9sym circuit.](image)

---

**ALGORITHM 5: HDC**

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>EC: exact cone, number of sets, C_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>App-Kmaps and characteristics</td>
</tr>
</tbody>
</table>

1. AC= EC
2. List of zeros= Create Zero Sorted List (c)
3. While (i< #set) do
4. KE_i= randomly C_i selected minterms from the top #ZC of List of zeros
5. Generate App-Kmap
6. END

Return App-Kmaps and characteristics

- Create Zero Sorted List (AC) function: cone AC is the input of this function. It calculates the number of adjacent 1 for each zero minterm in the Karnaugh map of AC. Then, zero minterms are sorted in a List of Zeros in a descending order based on their adjacent 1. Minterms with equal number of adjacent 1 are sorted in a random order in this list which is the output of this function.
Figure 7: number of adjacent 1 for each zero minterm. The cross represents the adjacent 1 minterms of 1101 minterm.

Table 1: Example of list of zeros

<table>
<thead>
<tr>
<th>Minterm</th>
<th>11</th>
<th>01</th>
<th>00</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>01</th>
<th>00</th>
<th>00</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacent 1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

4. Experimental Results

We have implemented our proposed techniques in C++, and the experiments are carried out on an Intel CORE i5 3210m CPU, 4 GB memory running Linux 3.14. In order to evaluate the effectiveness of our proposed approximation techniques we applied our techniques to benchmarks from ISCAS99 and MCNC89 [11]. All commercial and academic synthesis tools can be used to achieve the approximated circuit from the modified Karnaugh map. In our experiments we used ABC [12] tool to synthesize the benchmarks from blif format into the gate level Verilog description and extracting their cones. To have a precise evaluation on the area and latency of the critical paths in benchmark circuits for average area and latency of our circuits decreases for 27% and 22% respectively.

In another experiment we compared our heuristic technique with approximate logic synthesis (ALS) of [5]. Table 3 compares the area reduction and the CPU runtime of our HDC technique and ALS of [5] for $C_t = 2$. The first two columns of Table 3 contain the benchmark circuit information including the circuit description (Benchmark circuit column) and their area (Original circuit area column). The Next column (#Appr. paths) gives the number of approximated critical paths using our HDC technique. The next two columns give the information of area reduction (Area reduction% sub column) and the CPU time needed to perform approximation technique (CPU time sub column) related to approximation techniques of ALS [5] (ALS [5] column) and our HDC (HDC column). As reported in Table 3, our approximation technique shows to be 7.4 times more effective in area reduction. Moreover, our HDC technique is 26.6 times faster than ALS technique. Not to mention that since our HDC technique choses $C_t$ random number of minterms from the pruned search space to create KE, incrementing $C_t$ does not increase runtime complexity while in [5] the runtime is reported to grow linearly with $C_t$.

Table 2: Experimental results (area and latency reductions) of our two approximation techniques

<table>
<thead>
<tr>
<th>Benchmark circuit</th>
<th>Original circuit Area</th>
<th>#Appr. paths</th>
<th>Reduction of RDC</th>
<th>Reduction of HDC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Area%</td>
<td>Latency%</td>
</tr>
<tr>
<td>9sym</td>
<td>702</td>
<td>1</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>ALU4</td>
<td>1545</td>
<td>3</td>
<td>0</td>
<td>43.5</td>
</tr>
<tr>
<td>b12</td>
<td>1970</td>
<td>3</td>
<td>25</td>
<td>36.3</td>
</tr>
<tr>
<td>Pdc</td>
<td>11578</td>
<td>1</td>
<td>41.6</td>
<td>18.46</td>
</tr>
<tr>
<td>Spla</td>
<td>27669</td>
<td>3</td>
<td>27</td>
<td>12.1</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>9367</strong></td>
<td><strong>2.6</strong></td>
<td><strong>27</strong></td>
<td><strong>22.0</strong></td>
</tr>
</tbody>
</table>

Table 3: Comparison of ALS technique [5] and our heuristic approximation technique

<table>
<thead>
<tr>
<th>Benchmark circuit</th>
<th>Original circuit area</th>
<th>#Appr. paths</th>
<th>ALS [5]</th>
<th>HDC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Area reduction%</td>
<td>CPU time(sec)</td>
</tr>
<tr>
<td>9sym</td>
<td>702</td>
<td>1</td>
<td>4.3</td>
<td>0.165</td>
</tr>
<tr>
<td>clip</td>
<td>1106</td>
<td>3</td>
<td>4.09</td>
<td>1.8</td>
</tr>
<tr>
<td>rd73</td>
<td>794</td>
<td>1</td>
<td>3</td>
<td>0.0793</td>
</tr>
<tr>
<td>sao2</td>
<td>516</td>
<td>3</td>
<td>9.5</td>
<td>0.066</td>
</tr>
<tr>
<td>5xp1</td>
<td>414</td>
<td>4</td>
<td>5.77</td>
<td>0.580</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>706</strong></td>
<td><strong>2.6</strong></td>
<td><strong>5.77</strong></td>
<td><strong>0.580</strong></td>
</tr>
</tbody>
</table>
5. Conclusion and Future Work

In this paper, we have presented approximation logic synthesis techniques for error resilient applications by setting zero minterms with larger adjacent 1 in the Karnaugh map of the circuit to Don’t care value. Our heuristic approximation technique shows reduction in area of our benchmark circuits for 38.34% on average. Moreover, our heuristic technique prunes the size of our search space, which makes it 26.6× faster and 7.4× more effective than the approximation logic synthesis techniques introduced in [5].

One possible avenue for future work is to combine logic optimization techniques with polynomial optimization methods such as [14] in order to improve the quality of the circuits in terms of the area as well as performance at different abstraction levels. We are also going to formally verify such approximate circuits by comparing their functionalities with a higher level of abstraction [15][16].

6. References


