Multi-Level Approximate Logic Synthesis under General Error Constraints

Jin Miao, Andreas Gerstlauer, and Michael Orshansky
Department of Electrical & Computer Engineering, The University of Texas at Austin
{jinmiao, gerstl, orshansky}@utexas.edu

Abstract

We address the problem of multi-level approximate logic synthesis. Our strategy assumes existence of an optimized exact Boolean network, which is critical in practice since arithmetic blocks are rarely synthesized from 2-level representation automatically. The goal is to produce minimum cost circuits whose logic function deviates in a controlled manner from the exact function with deviations quantified by the magnitude and frequency of errors.

We rely on network simplifications allowed by external don't cares (EXDCs). We formulate the error-magnitude constrained problem by using Boolean relations to capture the allowed error behavior in a more general manner compared to incompletely specified functions. Our key contribution is in finding sets of external don't cares that maximally approach the Boolean relation. The algorithm starts with an EXDC set that is overly relaxed and iteratively, and in a greedy fashion, identifies a feasible EXDC set by solving a series of conventional EXDC-based network optimizations. The algorithm then ensures compliance to error frequency constraints by recovering the correct outputs on the sought number of error-producing inputs while aiming to minimize the network cost increase.

We applied the algorithm to several well-known adder and multiplier designs of varying bit-width. Even for small error magnitudes, the algorithm produces networks with gate count reduced by 30-50%, when the error frequency constraint is loose. This is up to 20% fewer gates than a naive EXDC-based approach.

1. Introduction

Trading off computation accuracy for improved energy efficiency has attracted significant attention in recent years. Many studies have demonstrated the possibility of approximating computations for digital systems that are computationally expensive but naturally tolerate errors, such as in signal processing, machine learning and data mining applications. Existing approaches range from the algorithm [6, 16] and architecture [9, 10, 12, 19] to the logic [7, 11, 22] and transistor levels [8].

In hardware, many efforts have been focused on addressing this problem by deriving approximate versions of basic combinational logic circuits that can be implemented with reduced logic complexity and hence smaller area, delay and energy. Initial work in this domain has been concerned with hand-crafting approximate version of fundamental arithmetic building blocks, such as adders and multipliers [1, 7, 11, 15, 22, 23]. However, manually re-designing approximate circuits for each new type of logic block and application is not feasible. As previous results have demonstrated, there exists a large design space with non-obvious trade-offs between acceptable accuracy and energy, which are in turn specific to a particular application. In general, the error tolerance of an application is determined by the frequency (rate) or magnitude of produced errors. In reality, most application-level quality metrics, such as Peak Signal-to-Noise Ratios (PSNRs) used in image and video processing applications, depend on a combination of both.

Overall, there is a need to develop effective techniques for systematic and automatic approximate logic synthesis with the ability of constraining general error types. Several approximate two-level and multi-level logic synthesis approaches have recently been proposed. Most of the techniques thus far have focused on constraining a single error type only. In [18], a heuristic algorithm was presented for synthesizing a minimized approximate Boolean function under error frequency constraints. In [5, 13], the authors propose approaches for pruning a multi-level Boolean network according to input vector statistics given by an application context. Distinct solutions are presented for dealing with error frequency or error magnitude constraints, but those two aspects are never jointly explored. Furthermore, their approach can only perform redundancy pruning rather than structural optimizations. In [20], the authors mapped the multi-level approximate logic synthesis problem into a conventional logic synthesis with external don't care (EXDC) sets. The original circuit is combined with a wrapper that models the constraints. The allowed EXDC set used for optimizations at the original circuit's outputs is then extracted as the internal Observability Don't Care (ODC) set of this combined circuit. Although the authors claim the ability to flexibly model various error constraints, only error magnitude-type constraints are actually permitted, and error frequency constraints can not be handled in their framework. Furthermore, EXDC sets cannot capture correlated error patterns across multiple outputs. By contrast, we aim to support definition of flexible, general and precise error constraints directly as part of the problem specification itself.

In [14], we demonstrated synthesis of approximate logic circuits under both magnitude and frequency types of constraints. Our previous work was aimed at solving the problem in two-level form, i.e. to find the simplest approximate Boolean function. However, an optimal two-level solution will not necessarily lead to an optimal multi-level Boolean implementation. This is especially important for complex logic blocks, such as arithmetic units, that are typically realized through carefully tuned macro libraries instead of being synthesized from their original Boolean function.

In this paper, we address the problem of multi-level approximate logic synthesis (MALS) under arbitrary error magnitude and error frequency constraints. We develop a heuristic that effectively synthesizes approximate Boolean networks with reduced gate count whose errors deviate from an exact network in a constrained way. We make two major contributions:

(1) We formulate the magnitude constrained MALS using Boolean relations to capture the allowed error behavior. This formulation is more general than relying on incompletely specified functions and leads to better solutions. Our strategy uses network simplifications allowed by EXDCs. Our core contribution is an algorithm that identifies an EXDC set that maximally approaches the Boolean relation. It starts with an EXDC set that is overly relaxed and iteratively, and

in a greedy fashion, identifies an optimal EXDC set by solving a series of conventional EXDC-based network optimizations.

(2) The algorithm then ensures compliance to error frequency constraints by recovering the correct outputs on the sought number of error-producing inputs while aiming to minimize the network cost increase. We introduce a novel network cost minimization principle for dealing with the multi-output case. It is based on the observation that in many networks there is a variation in the degree to which network outputs are dependent on the rest of the network. It is reasonable to expect that enforcing correctness on outputs with lower *embeddedness* leads to a lower network cost increase as it requires modifications to a smaller region of the network.

We applied the algorithm to several well-known adder and multiplier designs of varying bit-width. Even for small error magnitudes, the algorithm produces networks with gate count reduced by 30-50%, when the error frequency constraint is loose. This is up to 20% fewer gates than a naive EXDC-based approach.

The rest of this paper is organized as follows: Section 2 presents the formulation of the MALS problem. Section 3 describes our algorithm for dealing with the MALS problem under general error constraints. Section 4 shows the experimental results, while Section 5 concludes the paper with a summary.

2. MALS Formulation

Consider an n-input, k-output combinational logic network G realizing a Boolean function F: $B^n \rightarrow \{0, 1, -\}^k$, where $-$ refers to a don't care. A multi-level approximate logic synthesis problem is concerned with formally synthesizing a minimum-cost (gate count) network whose behavior deviates in a controlled manner from the specified exact Boolean function F. The deviations can be specified in terms of error magnitude and error frequency. The error magnitude constraint specifies the outputs that the approximate circuit is allowed to produce for each input $x_i \in B^n$. The total number of inputs that produce approximate outputs is described by the *error frequency* constraint. We denote $G_{m,r}$ to be an approximate version of G with r inputs in error and with the largest magnitude of error being m. Let R be the maximum number of inputs allowed to be in error and let M be the maximum allowed deviation for a given output. We assume that a circuit produces a multi-bit output for which an arithmetic distance metric can be used to establish the degree of difference between the outputs. We use the notation $|G_{m,r}(x_i) - G(x_i)|$ to represent the absolute value of the arithmetic difference of the outputs produced by an exact and an approximate network. With this, the full multi-level approximate logic synthesis problem is:

$$\begin{aligned} \min \quad & C(G_{m,r}) \\ \text{s.t.} \quad & |G_{m,r}(x_i) - G(x_i)| \leq M, \quad \forall x_i \in B^n \\ & r \leq R \end{aligned} \tag{1}$$

where $C(G_{m,r})$ is the cost function, taken to be the gate count of $G_{m,r}$.

The problem defined in (1) captures the general constraints of both error types. The primary goal in problem (1) is to utilize the alternative outputs to simplify the circuit complexity when the outputs are constrained to take on *specific* patterns.

3. MALS under Error Magnitude and Frequency Constraints

We first discuss solving the problem of (1) when the error frequency constraint is not imposed. As shown in [14], the error magnitude constraint in (1) can be viewed as implicitly defining a *Boolean relation* (BR), where each input can be mapped to multiple outputs. A Boolean relation is a generalization of an incompletely specified Boolean function (ISF). It allows capturing a wider class of constraints on the outputs and thus allows for better solutions. Boolean relation minimization has been previously studied in the context of two-level optimization with the goal of identifying the minimum-cost two-level realization of a Boolean function compatible with a given Boolean relation [2, 4, 21]. In this paper, we focus on simplifying an initial multi-level Boolean network by exploiting the flexibility captured by the Boolean relation. Such optimizations on an existing Boolean network are especially important for arithmetic blocks, which are often available directly in canonical form (e.g., various prefix adders).

There are currently no effective techniques to *directly synthesize* a Boolean network that satisfies a given Boolean relation. Therefore, we adopt the well-known principle of using external don't cares to simplify Boolean networks. In the context of approximate synthesis, such an approach has been proposed in [20]. In [20], the external don't care (EXDC) sets are based on conventional single output don't care extraction. However, a single-output approach does not exploit the full flexibility that may be permitted by the error specification. For instance, a function that allows two outputs $\{11, 00\}$ on some input can never be captured via single-output don't cares.

The contribution of our paper is in demonstrating that it is possible to find *non-trivial better* sets of external don't cares to drive multi-level optimization. The essence of the algorithm is that *it identifies an EXDC set that maximally approaches the Boolean relation.* Specifically, the algorithm starts with an EXDC set that is overly relaxed and iteratively, and in a greedy fashion, identifies an optimal EXDC set by solving a *series* of conventional EXDC-based network optimizations. The original Boolean network is ultimately minimized by using the optimal EXDC. To drive the algorithm, we use the multi-level synthesis tool as a black box and rely on its ability to exploit the flexibility offered by EXDC. (As detailed later, we use SIS [17] for network optimization.)

The central challenge is to identify an EXDC set that maximally approaches the Boolean relation. Such a set has the following properties. First, it maximally shares the flexibility for network simplification described by the Boolean relation. Because the sought EXDC set is based on single-output don't cares, we use the difference between a *relaxation of a BR into a multi-output ISF* and a candidate EXDC as a measure of such flexibility. Second, it defines a function compatible with BR. We call this EXDC set a Relation-Aware EXDC (RA-EXDC) set.

We give an outline of the algorithm first. The algorithm to find RA-EXDC is based on a relax-and-recover strategy: we first relax the error constraints of the original problem, captured via a BR, to produce an EXDC set that permits more than the original deviations specified by BR. This EXDC is an upper bound for RA-EXDC. Because the network simplified by this EXDC violates the original error constraints, we use an iterative recovery procedure to minimally refine EXDC until violations are removed. Along with the upper bound, we show that a lower bound on the sought EXDC is available and can also be extracted from BR. Both bounds are used during the recovery phase. Next we discuss a strategy to compute both bounds.

3.1 Extracting Lower and Upper Bounds

We are interested in finding the least upper bound and the greatest lower bound. The least upper bound EXDC is the minimum superset of BR that can be expressed using ISFs. As the lower bound for the extraction of the optimum EXDC we utilize the maximum subset of BR that can be expressed as an ISF and that contains the original function. In doing this, we assume that such an ISF provides the largest flexibility for network simplification *among all possible ISFs*. We note that the stated principle for finding the "best" ISF was used earlier in [4]. The choice of this ISF as a *useful* bound is justified via an assumption of a monotonic relation between the size of the

EXDC and the potential complexity reduction of a network under this EXDC.

Some preliminary definitions are first provided. Hereafter, we indistinctly use the incompletely specified function (ISF) and the corresponding don't care set described by this ISF.

A Boolean relation can also be specified by a characteristic function $R : B^n \times B^m \rightarrow B$, such that $(x, y) \in R \iff R(x, y) = 1$. Here, the B^n and B^m are the input and output sets of R, respectively. We use the characteristic function R for the following discussion.

Definition 3.1. MISF. A multi-output ISF (MISF) is a function $f : B^n \rightarrow (B \cup \{-\})^m$, which is a vector of ISFs $f = (f_1, f_2, ..., f_m)$.

Definition 3.2. Natural join [2]. The natural join over the input set X between two relations R and S is defined as

$$R(X, Y) \bowtie_X S(X, Z) = \{(x, y, z) | (x, y) \in R \wedge (x, z) \in S\} \quad (2)$$

where we use $X = (x_1, x_2, \ldots, x_n)$ to denote the set of inputs and $Y = (y_1, y_2, \ldots, y_m)$ and $Z = (z_1, z_2, \ldots, z_m)$ for outputs of two relations, respectively.

Definition 3.3. Projection of a Boolean relation [2]. The projection of a relation $R(X, Y)$ onto an output y_i is another function $(R \downarrow y_i)$ such that

$$\begin{aligned}(R \downarrow y_i) = &\{(X, z) | \exists y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m \\ &\text{such that } (X, y_1, \ldots, y_{i-1}, z, y_{i+1}, \ldots, y_m) \in R\}.\end{aligned} \quad (3)$$

The projection of a relation R onto an output y_i defines an ISF for that output.

Definition 3.4. MISF covering a Boolean Relation [2]. For a given Boolean relation R, an MISF covering R can be obtained as follows:

$$MISF_R(X, Y) = \bowtie_{i \in \{1,\ldots,m\}}^{X} (R \downarrow y_i) \quad (4)$$

It can be proved [2] that the MISF$_R$ is the minimum superset of Boolean relation R that can be expressed by ISFs, i.e. an MISF. Hence, the least upper bound EXDC is MISF$_R$. We refer to MISF$_R$ as an over-approximated EXDC (O-EXDC).

We now illustrate the extraction of O-EXDC from the Boolean relation table via a simple example. Consider a 2-input, 3-output multi-level Boolean network with error magnitudes as specified in Table 1. The first output in each row refers to the correct output of the network.

Table 1: Boolean relation

Inputs	Outputs
x_1, x_0	y_2, y_1, y_0
00	{000, 001}
01	{010, 011, 100}
10	{100, 101, 010}
11	{110, 100, 101}

It takes two steps to derive O-EXDC.

1) Project the Boolean relation onto each output $y_i, i = 0, 1, 2$. (see Table 2).

Table 2: Projections of BR on y_i

x_1, x_0	$R \downarrow y_2$	$R \downarrow y_1$	$R \downarrow y_0$
00	{0}	{0}	{0,1}
01	{0,1}	{0,1}	{0,1}
10	{0,1}	{0,1}	{0,1}
11	{1}	{0,1}	{0,1}

2) Perform a natural join on each projection of $(R \downarrow y_i)$ to form MISF$_R$. The {-} implies a don't care condition. We encode a don't care condition by a "1" and a care condition by a "0" to form O-EXDC, as shown in Table 3. For example, O-EXDC for y_2 is $\{x_1^{'}x_0, x_1x_0^{'}\}$, O-EXDC for y_1 is $\{x_1x_0, x_1^{'}x_0, x_1x_0^{'}\}$, and O-EXDC for y_0 is the full set of inputs.

Table 3: MISF$_R$ and corresponding O-EXDC

Inputs	MISF$_R$			O-EXDC		
x_1, x_0	$R \downarrow y_2$	$R \downarrow y_1$	$R \downarrow y_0$	y_2	y_1	y_0
00	0	0	-	0	0	1
01	-	-	-	1	1	1
10	-	-	-	1	1	1
11	1	-	-	0	1	1

Note that O-EXDC allows greater flexibility than BR. In Table 3, since all output bits are insensitive to input 01, the output of, for instance, 111 is also allowed by O-EXDC but not by Boolean relation in Table 1.

We now discuss the greatest lower-bound EXDC that can be extracted from BR. As stated above, we use the maximum subset of BR that can be expressed by an ISF and that contains the original exact function. It can be acquired using the following procedure:

1) For an input x, we have an output set $Y : (x, Y) \in R$. Identify the maximum prime implicant p_x of set Y that contains the original output. For instance, consider an input x and Y={001, 011, 111}: we have p_x={0-1}. The prime p_x provides the maximum number of output bits insensitive to x.
2) Repeat the above step for all $x \in B^n$. Hence, we acquire a set of independent ISFs, each for one output bit. These ISFs together form an MISF f.

Theorem 3.1. *f is a maximum subset of BR that can be expressed as an MISF.*

Proof. We first prove that f is contained by BR. Since $p_x \subseteq Y, \forall x \in B^n$, we have $R(x, p_x) = 1, \forall x \in B^n$. By definition, we have MISF $f : x \rightarrow p_x, \forall x \in B^n$, therefore $R(X, f) = 1$.

We now prove by contradiction that f is the maximum subset of BR. Assume there is another MISF h with a larger cardinality than f, i.e., $|f| < |h|$. Here, the cardinality refers to the number of don't cares contained by the MISF h. We denote q_x as the prime implicant of set Y that contains the original output for h. By definition, we have $|p_x| \geq |q_x|, \forall x \in B^n$. Hence, we have $\prod_{x \in B^n} |p_x| \geq \prod_{x \in B^n} |q_x|$. Consider $|f| = \prod_{x \in B^n} |p_x|$ and $|h| = \prod_{x \in B^n} |q_x|$, we have a contradiction against the assumption. Therefore, f is the maximum subset of BR. □

We denote the DC-set of f as a conservative EXDC (C-EXDC). We summarize the relation between C-EXDC, RA-EXDC, and O-EXDC as:

$$\text{C-EXDC} \subseteq \text{RA-EXDC} \subseteq \text{O-EXDC} \quad (5)$$

Note that there is no obvious containment relation between RA-EXDC and the original Boolean relation. Table 4 illustrates the relation between C-EXDC, RA-EXDC, and O-EXDC through an example.

3.2 Recovering Magnitude Conflicts

Optimizing the initial network with O-EXDC results in a minimal complexity network for the magnitude-only constrained MALS problem. However, the resulting network may produce outputs not allowed by the original Boolean relation. We refer to inputs for

Table 4: Boolean relation, C-EXDC, and O-EXDC

Inputs	Boolean Relation	C-EXDC	O-EXDC
x_1, x_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0
00	{000, 001}	001	001
01	{010, 011, 100}	001	111
10	{100, 101, 010}	001	111
11	{110, 100, 101}	010	011

which the error constraint is violated as *conflict inputs*. Notice that the network simplified by C-EXDC is guaranteed to be free of conflicts. This indicates that all conflicts fall in the *complement of C-EXDC in O-EXDC*, i.e. *O-EXDC*$\backslash$*C-EXDC*. We introduce a notion of *candidate inputs* to denote those inputs that can potentially cause conflicts.

Definition 3.5. Candidate inputs. A candidate input x is defined as $\{x | x \in$ *O-EXDC* $\setminus$ *C-EXDC*$\}$.

Each candidate input x may correspond to more than one output with possible conflicts. These outputs together form the *candidate outputs* for input x. For example, in Table 4, y_1, y_2 are candidate outputs for input 10.

It is important to see that not every candidate input is a conflict input. In other words, a network simplified with an EXDC set that contains a candidate input will not necessarily produce a conflict on this input. That happens for two reasons. First, not every allowed flexibility is actually utilized for simplifying the network. Whether a flexibility is eventually utilized depends on the network structure. Second, even if the changes are made, the simplified network may produce outputs compatible with the Boolean relation and thus remain conflict-free. For example, in Table 4, candidate input 10 may be used to simplify the network and result in output 010 which is compatible with the original Boolean relation.

We now discuss how we modify O-EXDC to remove conflicts. Let us first consider the example in Table 5. Suppose that after simplifying the original Boolean network using O-EXDC, the shaded two inputs produce outputs beyond the error magnitude constraint. In order to remove these conflicts, we need to remove the flexibility currently given to the candidate outputs of the shaded inputs; for brevity, we denote this operation as removing these candidate inputs from their candidate outputs. For instance, input 11 produces output 101, which conflicts with the error specification. We observe that O-EXDC and C-EXDC differ for y_1 and y_0. Therefore, outputs y_1 and y_0 (but not y_2) are the candidate outputs for this input. Input 11 is, in turn, considered for removal from the EXDC of each of these outputs (y_1 and y_0).

However, it may not be necessary to remove 11 from don't care sets of both y_1 and y_0. We assume that the gate count of the simplified network increases monotonically with a decrease in EXDC. The goal of the algorithm is to ensure that some allowed output is produced while removing the least flexibility from EXDC, e.g., by reducing EXDC minimally in each iteration. That means we need to change as few don't cares in EXDC for a candidate input as possible. This is achieved by aiming to match an allowed output that is least distinct from the output produced by the current network. The example of Table 5 illustrates this by showing that the number of conflicting outputs for a given input, say input 11, depends on the allowed output being considered. The current approximate output 101 has a difference in only one output when compared with 111 but it has two erroneous outputs when compared with 011 or 110.

Hamming distance provides the appropriate metric. We only disallow (remove) the don't cares from the candidate input of those erroneous outputs that *intersect* with the candidate outputs. This guarantees the search space is restricted to lie within the given lower and upper bound, i.e., C-EXDC and O-EXDC. Since C-EXDC contains the original network outputs, there is always at least one allowed output, i.e., the correct output, that has a non-empty intersection with candidate outputs (in this case, the intersection contains all candidate outputs).

We summarize the strategy for selecting the candidates for removal from EXDC. For each input with conflicting outputs:

Algorithm 1: MALS under general error constraints

Input: NL: Original Boolean network, BR: Error magnitude constraint, R: Error frequency constraint, N: number of input bits
Output: Minimized Boolean network with constrained error magnitude and error frequency

$EXDC_n = EXDC_o = \mathrm{MISF_R}(BR)$; $EXDC_c = \mathrm{ISF}(BR)$;
$NL_n = \mathrm{Optimize}(NL, EXDC_o)$;
$Conf = \{x_i | NL_n(x_i) \notin BR(x_i), x_i \in B^n\}$;
$r = |\{x_i | NL_n(x_i) \neq NL(x_i), x_i \in B^n\}|/2^n$;
$j = 0$;
while *(*$Conf \neq \Phi$*) or (*$r > R$*)* **do**
  while $Conf \neq \Phi$ **do**
    $y_{approx} = NL_n(x_i)$;
    foreach $x_i : NL_n(x_i) \notin BR(x_i)$ **do**
      $a = BR(x_i)$;
      do
        $y_{allow} = \mathrm{argmin}_a \,\mathrm{Hamming}(y_{approx}, a)$;
        $y_{remov} = (y_{allow} \oplus y_{approx}) \wedge \mathrm{Candidate}(EXDC_o, EXDC_c)$;
        $a = a - y_{allow}$;
      while $y_{remov} \neq 0$;
      foreach non-zero bit b of y_{remov} **do**
        $EXDC_n \leftarrow$ Remove input x_i from output b;
      end
    end
    $NL_n = \mathrm{Optimize}(NL, EXDC_n)$;
    $Conf = \{x_i | NL_n(x_i) \notin BR(x_i)\}$;
  end
  $r = |\{x_i | NL_n(x_i) \neq NL(x_i)\}|/2^n$;
  $k =$ Difference inputs # for output bit j;
  while $r > R$ **do**
    $c = 0$; Flag = True;
    foreach $x_i : (NL_n(x_i)[j] \neq NL(x_i)[j])$ **do**
      $EXDC_n \leftarrow$ Remove input x_i from the j^{th} bit;
      $c = c + 1$;
      if $c > \alpha k$ **then**
        Flag = Flase; Break;
      end
    end
    if *Flag* **then**
      $j = j + 1$;
    end
    $NL_n = \mathrm{Optimize}(NL, EXDC_n)$;
    $Conf = \{x_i | NL_n(x_i) \notin BR(x_i)\}$;
    if $Conf \neq \Phi$ **then**
      Break;
    end
    $r = |\{x_i | NL_n(x_i) \neq NL(x_i)\}|/2^n$;
  end
end
return NL_n;

Table 5: Error magnitude recovery example

Inputs	Outputs	C-EXDC	O-EXDC	Approx. Outputs	Candidate Outputs	Updated EXDC
x_1, x_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0
00	{000, 001}	001	001	001	000	001
01	{010, 011, 100}	001	111	101	110	001
10	{110, 101, 010}	100	111	101	011	111
11	{011, 111, 110}	100	111	101	011	101

1) Find the Hamming distance between the approximate output and each allowed output.
2) Identify as target output the output with the minimum Hamming distance for which at least one output bit is contained in the output candidate list (this is the output whose bitwise XOR difference from the approximate output also intersects with the candidate output set.)
3) Change all the output bits that are in the intersection from 2) to be sensitive to the input, i.e. remove the current input from the EXDC of each such output. Repeat this process for all inputs that produce conflicts.

Note that there can be multiple allowed outputs satisfying the strategy described in 2) and 3). We select the one that is first found among the allowed outputs which are listed in an arithmetic ascending order. In Table 5, by comparing the approximate output 101 for input 01 with all allowed outputs $\{010, 011, 100\}$, we know output 100 has the minimum Hamming distance with a difference only on y_0. However, y_0 is not a candidate output for this input. Therefore, we select 011 as the targeted allowed output: it has the next minimal Hamming distance but it has outputs that are candidate outputs. In this case, the allowed output 011 differs from the current approximated output 101 in y_1 and y_2 both of which are in the candidate list. Therefore, we modify y_1 and y_2 to be sensitive to input 01, i.e., we remove input 01 from O-EXDC of y_1 and y_2. Similarly, we remove input 11 from O-EXDC of y_1. The non-shaded elements of EXDC remain the same as there are no other conflicts in this iteration.

Applying the updated EXDC set to the original Boolean network produces another version of the approximate Boolean network. However, while the conflict outputs found in the previous iteration have been removed, new conflict outputs may be produced. We follow the above steps to iteratively correct those conflicts. The algorithm clearly implements a greedy approach: in every iteration, we correct *all conflicts manifest at this iteration* by making the minimum possible changes to EXDC. This heuristic solution strategy is observed to behave reasonably well. On the benchmarks that we utilized, we find that the number of iterations is typically small (3 to 4). We also find that the behavior is monotonic: with each iteration, as the number of conflict inputs is reduced, the network gate count monotonically increases. The algorithm stops when no conflict output exists. In the theoretical worst case, it stops when there are no candidate outputs left, i.e., O-EXDC is reduced to C-EXDC. On our benchmarks, we find that the algorithm stops earlier and produces a substantial improvement over using C-EXDC. Section4 provides more details of experiments and results. The above algorithm is summarized in Algorithm 1 up to Line 22.

An approximate network produced by the above algorithm is compatible with the original Boolean relation and satisfies the error magnitude constraint. Next, we show how to extend the above algorithm to solve the general MALS problem (1) that jointly considers error magnitude and frequency constraints.

3.3 Resolving Frequency Violations

The approximate Boolean network resulting from the above algorithm has an arbitrary error frequency (often close to 100%), and thus, typically, violates the constraint. We now develop a recovery procedure to produce a feasible solution with respect to error frequency.

Consider an input x for which the simplified Boolean network produces an output different from the exact output. We call such an input a *difference input* and the corresponding output a *difference output*. The error frequency reduction is based on recovering the correct outputs on some or all of the difference inputs while aiming to minimize the network cost increase. Corrections are achieved by updating EXDC to enforce the correct outputs on selected inputs.

We start by discussing the strategy to deal with a single-output case. Suppose that the number of difference inputs is N and the target error frequency is R. The goal is to identify at least $k = N - R$ difference inputs to correct out of the set of N possible inputs. The fact that N is typically very large and R is small makes identifying a *good* set of k inputs to correct very difficult. It is infeasible to try all possible sub-sets. We adopted a heuristic strategy that minimizes the dependence on the choice of a sub-set by picking an arbitrary smaller sub-set of αk inputs, where α is, typically, 0.2 to 0.5. Notice that aiming to correct k inputs does not mean that the resulting network has exactly $R - k$ difference outputs since new errors may appear. Thus, depending on the error frequency constraint R, the procedure requires several, typically 6 to 10 iterations to remove all difference inputs.

The network cost minimization principle is further extended to handle multiple outputs. This is crucial, since an input remains a difference input as long as one or more outputs are in error. In dealing with the multi-output case, we observe that *in some cases*, forcing correctness on different outputs may predictably lead to different network cost increases. This is based on the fact that in many networks, there is a variation in the degree to which network outputs are dependent on the rest of the network. In other words, there is variation in the degree of *embeddedness* of an output in the network. This can be quantified by finding the ratio of the gates that are in the fan-in cone of an output over the overall network gate count. A higher ratio indicates a higher value of embeddedness. It is reasonable to expect that enforcing correctness on outputs with *lower* embeddedness leads to a lower network cost increase as it requires modifications to a smaller region of the network. Our work is primarily concerned with arithmetic circuits where the degree of embeddedness is easily ascertained from basic arguments and therefore does not require explicit extraction. In arithmetic circuits, the outputs corresponding to the lower-significance bits (LSBs) naturally have a lower degree of embeddedness. Therefore, we prioritize enforcing correctness on network outputs corresponding to LSBs, which should lead to the smallest gate increase. The algorithm iteratively corrects difference inputs on outputs moving from LSB to MSB until the entire network satisfies the error frequency constraint. We summarize the strategy for error frequency recovery:

1) For an output i, identify all of its k_i difference inputs;
2) Remove αk_i fraction of difference inputs from the EXDC;
3) Repeat 2) until error frequency is satisfied or $k_i = 0$;
4) Repeat for output $i = i + 1$ from LSB to MSB until error frequency constraint is met.

See the example in Table 6 for an illustration of a single iteration of the algorithm. We find that altogether there are two difference inputs (00 and 10). Suppose the error frequency constraint is 25%. Therefore, in this iteration we would aim to correct one input. Because the difference output for 00 is the LSB, we choose this input to be corrected and modify y_0 to be sensitive to input 00.

We summarize the complete approach in Algorithm 1.

Table 6: Error frequency recovery example

Inputs	Exact Output	Old EXDC	Approx. Outputs	Updated EXDC
x_1, x_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0	y_2, y_1, y_0
00	{000}	001	001	000
01	{010}	100	010	100
10	{110}	110	010	010
11	{011}	101	011	101

4. Experimental Results

We have implemented the MALS algorithm in a C++ environment using ABC [3], SIS [17] and Design Compiler as the synthesis tools. To evaluate the capability of the proposed algorithm for significant gate count reduction under general error magnitude and frequency constraints, we used it to generate a range of approximate solutions of different types of adders and multipliers with different bitwidth. All experiments were performed on an Intel 3.4GHz workstation. Table 7 shows the circuit-specific information for the adders and multipliers we used.

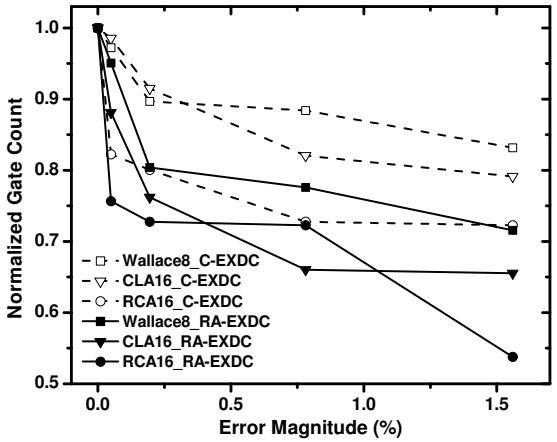
Table 7: Circuits used for MALS algorithm

Name	Function	I/O	Gates
RCA8	8-bit Ripple Carry Adder	16/9	323
RCA16	16-bit Ripple Carry Adder	32/17	411
CLA16	16-bit Carry Lookahead Adder	32/17	412
KS16	16-bit Kogge Stone Adder	32/17	465
RCA32	32-bit Ripple Carry Adder	64/33	834
Wallace8	8-bit Wallace Multiplier	16/16	1259
Dadda8	8-bit Dadda Multiplier	16/16	1128

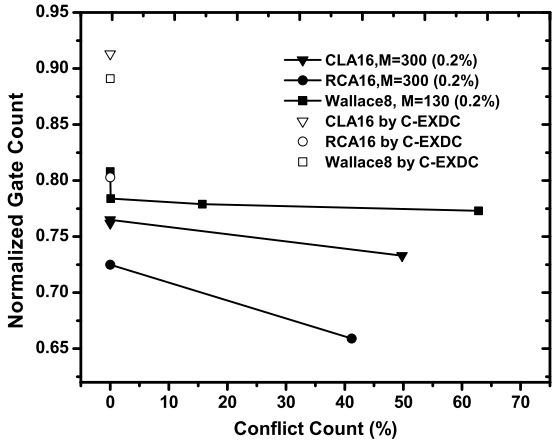
We first applied the MALS algorithm to several types of adders and multipliers with different bitwidths (Table 7). For the 32-bit RCA, we applied our algorithm to the lower 18 bits. The runtime varies from a few seconds for a small adder to more than 20 hours for large adders and multipliers.

We first demonstrate the effectiveness of the proposed MALS algorithm compared to synthesizing networks using lower bound C-EXDC when only the magnitude of error is constrained. Figure 1(a) shows two types of 16-bit adders and an 8-bit Wallace multiplier synthesized by both the C-EXDC and RA-EXDC identified by our algorithm for several magnitudes of allowed error. The error magnitude is shown as the percentage of the maximum output value (since circuits have different bitwidths, errors of the same absolute magnitude indicate varying relative significance). We find that the network simplified by the proposed RA-EXDC outperforms the approach using C-EXDC by up to 20% in terms of achieved gate count reduction. Figure 1(b) shows the gate count and conflict inputs changes over each iterations. Each curve refers to the iteration history of one network, where each point refers to one iteration. All iterations start from the O-EXDC (the point with the largest conflicts number) and monotonically reduce to zero conflicts. For the given benchmark circuits, our algorithm converged within 4 iterations in all cases. We also mark the solution identified by C-EXDC in this figure.

In order to give an intuition of how the network evolves over iterations, we show successive optimizations performed on the fan-in cone of the sum bit 4 of a simple 8-bit RCA adder (Figure 2). The



(a) Error magnitude sweep



(b) Gate count and conflicts over iterations

Figure 1: Networks simplified by C-EXDC and RA-EXDC

process converges in iteration 3. Compared to an C-EXDC based approach, 4 gates are saved by using RA-EXDC.

In Figure 3, we show the results of synthesizing approximate networks jointly under both types of constraints. We first synthesized each network with error magnitude constraints equal to 300 and 1000 (corresponding to different relative error magnitudes). We further perform an error frequency sweep by running the algorithm and recording every possible frequency achieved during the error frequency recovery phase. Results show that depending on the error magnitude and circuit, gate count reductions ranging from 5% to 50% can be achieved if frequency is unconstrained. Achievable gate count reductions decrease with stricter error frequency constraints. The results indicate that in some cases, the space of solutions is sparse in terms of achievable error frequencies. Note that this sparsity reduces with increased flexibility offered by larger error magnitude constraints.

5. Summary and Conclusions

In this paper, we address the multi-level approximate logic synthesis (MALS) problem under general error constraints. We formulate the error magnitude constrained MALS using Boolean relations to capture the allowed error behaviors. This formulation is more general, grants better flexibilities on error constraints and hence leads to better solutions than approaches based on incompletely specified functions. We further presented an algorithm to solve the MALS problem under general error magnitude and frequency constraints. The algo-

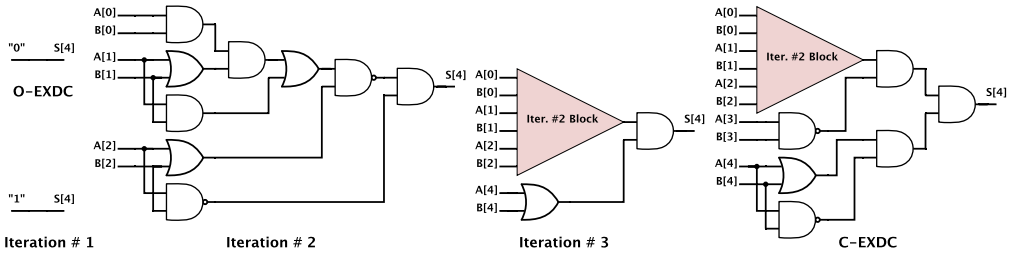
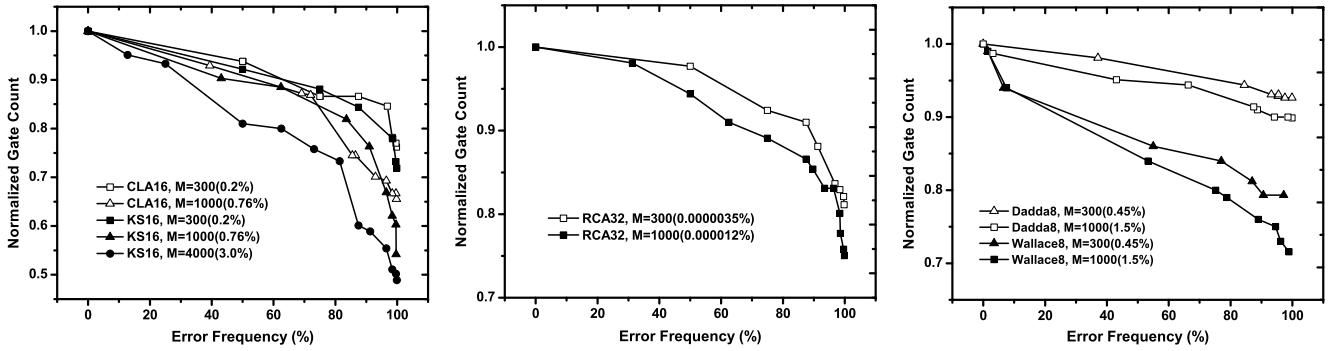


Figure 2: Logic cone of sum bit 4 in an 8-bit RCA as it changes over algorithm iterations



(a) 16-bit adders (b) 32-bit ripple carry adder (c) 8-bit multipliers

Figure 3: Error magnitude and frequency constrained solutions for adders and multipliers

rithm starts with a solution that is overly relaxed. In an iterative and greedy fashion, it then first identifies a solution satisfying the magnitude constraint by successively applying a series of less and less relaxed conventional multi-level network optimizations. The algorithm further ensures compliance to the error frequency constraint by recovering the correct outputs on error-producing inputs to minimize the network cost increase until the frequency constraint is met. Experiments on a range of arithmetic circuit blocks demonstrated the effectiveness in achieving large gate count reductions across flexible error magnitude and frequency constraints.

Acknowledgements

This work was supported by NSF grant CCF-1018075.

References

[1] P. Albicocco, G. C. Cardarilli, A. Nannarelli, M. Petricca, and M. Re. Imprecise arithmetic for low power image processing. In *Signals, Systems and Computers (ASILOMAR), 2012*.

[2] D. Baneres, J. Cortadella, and M. Kishinevsky. A recursive paradigm to solve boolean relations. *IEEE Trans. Comput.*, 2009.

[3] R. Brayton and A. Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification*, 2010.

[4] R. Brayton and F. Somenzi. An exact minimizer for boolean relations. In *ICCAD*, 1989.

[5] L. Chakrapani and K. Palem. A probabilistic boolean logic for energy efficient circuit and system design. In *ASP-DAC*, 2010.

[6] V. Chippa, A. Raghunathan, K. Roy, and S. Chakradhar. Dynamic effort scaling: Managing the quality-efficiency tradeoff. *DAC*, 2011.

[7] A. A. Del Barrio, R. Hermida, and S. O. Memik. Exploring the energy efficiency of multispeculative adders. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pages 309–315. IEEE, 2013.

[8] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. IMPACT: imprecise adders for low-power approximate computing. In *ISLPED*, 2011.

[9] K. He, A. Gerstlauer, and M. Orshansky. Controlled timing-error acceptance for low energy idct design. In *DATE*, 2011.

[10] R. Hegde and N. Shanbhag. Soft digital signal processing. *TVLSI01*.

[11] Y. Kim, Y. Zhang, and P. Li. An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems. In *ICCAD*, 2013.

[12] F. Kurdahi, A. Eltawil, K. Yi, S. Cheng, and A. Khajeh. Low-power multimedia system design by aggressive voltage scaling. *TVLSI10*.

[13] A. Lingamneni, C. Enz, J. L. Nagel, K. Palem, and C. Piguet. Energy parsimonious circuit design through probabilistic pruning. In *DATE2011*.

[14] J. Miao, A. Gerstlauer, and M. Orshansky. Approximate logic synthesis under general error magnitude and frequency constraints. In *ICCAD*, 2013.

[15] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *ICCAD12*.

[16] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig. Approximate signal processing. *VLSI Signal Processing*, 15, 1997.

[17] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit analysis. Technical report, Tech. Report No. UCB/ERL M92, 1992.

[18] D. Shin and S. K. Gupta. Approximate logic synthesis for error tolerant applications. In *DATE*, 2010.

[19] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[20] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: systematic logic synthesis of approximate circuits. In *DAC*, 2012.

[21] Y. Watanabe and R. Brayton. Heuristic minimization of multiple-valued relations. *TCAD*, 1993.

[22] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2013.

[23] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong. Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing. *TVLSI*, 2010.