# SALSA: Systematic Logic Synthesis of Approximate Circuits

Swagath Venkataramani, Amit Sabne, Vivek Kozhikkottu, Kaushik Roy and Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
{venkata0,asabne,vkozhikk,kaushik,raghunathan}@purdue.edu

## ABSTRACT

Approximate computing has emerged as a new design paradigm that exploits the inherent error resilience of a wide range of application domains by allowing hardware implementations to forsake exact Boolean equivalence with algorithmic specifications. A slew of manual design techniques for approximate computing have been proposed in recent years, but very little effort has been devoted to design automation.

We propose SALSA, a **S**ystematic methodology for **A**utomatic **L**ogic **S**ynthesis of **A**pproximate circuits. Given a golden RTL specification of a circuit and a quality constraint that defines the amount of error that may be introduced in the implementation, SALSA synthesizes an approximate version of the circuit that adheres to the pre-specified quality bounds. We make two key contributions: (i) the rigorous formulation of the problem of approximate logic synthesis, enabling the generation of circuits that are correct by construction, and (ii) mapping the problem of approximate synthesis into an equivalent traditional logic synthesis problem, thereby allowing the capabilities of existing synthesis tools to be fully utilized for approximate logic synthesis. In order to achieve these benefits, SALSA encodes the quality constraints using logic functions called *Q-functions*, and captures the flexibility that they engender as *Approximation Don't Cares* (ADCs), which are used for circuit simplification using traditional don't care based optimization techniques. We have implemented SALSA using two off-the-shelf logic synthesis tools - SIS and Synopsys Design Compiler. We automatically synthesize approximate circuits ranging from arithmetic building blocks (adders, multipliers, MAC) to entire datapaths (DCT, FIR, IIR, SAD, FFT Butterfly, Euclidean distance), demonstrating scalability and significant improvements in area (1.1X to 1.85X for tight error constraints, and 1.2X to 4.75X for relaxed error constraints) and power (1.15X to 1.75X for tight error constraints, and 1.3X to 5.25X for relaxed error constraints).

## Categories and Subject Descriptors

B.7.1 [**INTEGRATED CIRCUITS**]: VLSI (Very large scale integration)

## General Terms

Algorithms, Design, Synthesis

## Keywords

Logic Synthesis, Approximate Computing, Low Power Design, Error Resilience


*DAC 2012,* June 3-7, 2012, San Francisco, California, USA.
 978-1-4503-1199-1/12/06 ...$10.00.

## 1. INTRODUCTION

*Error resilience* can be broadly defined as the characteristic of an application to produce acceptable outputs despite its constituent computations being performed imperfectly (with errors). A plethora of emerging application domains, in both embedded and general purpose computing, exhibit this intriguing trait. For example, applications in machine learning, recognition and data mining [1] demonstrate significant *algorithmic resilience* to errors. This tolerance can be attributed to several factors like redundancies in large input data-sets; non-existence of a unique golden result; aggregating nature of the algorithms leading to errors averaging out, *etc.* [2]. *Perceptual resilience* to errors is exhibited by applications that involve a human interface. These systems could tolerate errors provided that they are not perceivable by the end user. Such applications abound in speech, video and graphics processing domains [3]. In general, most of these applications are highly compute intensive and their hardware implementations expend significant amounts of energy. By forsaking the convention of designing precise circuits and by harnessing the error tolerance provided by these application domains, significant savings in power and performance can be realized [2–8].

When it comes to the design of approximate circuits, there have been two major schools of thought. The first class of *over-scaling based approximation* methods induce timing errors in circuits by subjecting them to voltage over-scaling [8]. In contrast, the other class of *functional approximation* techniques approximate the logic functions computed by the circuits so as to reduce their implementation complexity, leading to area and energy benefits.

Initial attempts to design functionally approximate circuits focused on manual re-design of common arithmetic building blocks [9–11]. However, these techniques are confined to specific well-studied circuits such as adders and multipliers and for larger and more complex circuits, an automated synthesis procedure is indispensable.

In this work, we present SALSA, a novel systematic methodology for logic synthesis of functionally approximate circuits. Starting with an RTL description of the exact circuit and an error constraint that specifies the type and amount of error that the implementation can accommodate, SALSA automatically synthesizes a functionally approximate version of the circuit that adheres to the pre-specified error constraints. The proposed methodology rigorously reformulates the problem of Approximate Logic Synthesis (ALS) and maps it into a traditional logic synthesis problem. The problem formulation and the solution approach adopted in SALSA beget the following advantages:

- The proposed methodology provides an inherent guarantee that the specified bounds are never transgressed, thus enabling synthesis of correct-by-construction approximate circuits.
- The transformations are completely independent of the target error metric as well as the circuit considered for approximation. In essence, this decouples the synthesis procedure from the error metric, making this approach

flexible and general.

- Additionally, by virtue of transforming and mapping ALS to a traditional logic synthesis problem, existing off-the-shelf logic synthesis tools could just be re-used for approximate circuit synthesis. This obviates the need for developing a custom tool for ALS, thus lowering the barrier to adoption. Further, this widens the scope of approximations that can be effected on the circuits, since the entire power of existing logic optimization algorithms can be leveraged.

We have prototyped SALSA using two different logic synthesis tools to demonstrate its generality and used it to synthesize a range of arithmetic circuits and datapaths. We demonstrate that the circuits synthesized by SALSA achieve significant reductions in area and power.

The rest of the paper is organized as follows. Section 2 overviews prior work pertaining to approximate circuit design and synthesis. The essence of the SALSA approach and the required preliminaries are elaborated in Section 3. The details of the proposed SALSA methodology are explained in Section 4. The experimental methodology and results are subsequently presented in Sections 5 and 6. Section 7 provides a brief summary and concludes the paper.

## 2. RELATED WORK

Previous research efforts have exploited the error resilience of applications at various levels of design abstraction. Compile time software techniques like [6], architecture level approaches like [5] and cross-layer methodologies like [7] are representative examples at higher levels of design abstraction. In this section, we provide a survey of techniques that apply approximations at the logic and transistor levels of abstraction.

A number of previous works have focused on manually approximating specific circuits like adders [9, 10] and multipliers [11] by taking advantage of their structural properties and the difference in the significance of their output bits. However, all these design techniques are confined to the specific circuits that they target. Automation becomes necessary as circuits grow functionally complex and the approximations that can be performed on them become non-intuitive.

One class of automation techniques target synthesizing circuits that trade-off accuracy for power through voltage over-scaling. Traditional synthesis optimizations result in circuits that contain a large number of near-critical paths and impede aggressive voltage scaling. To ensure a graceful degradation in the number of timing violations under over-scaling, the path delay distribution of the circuit is reshaped by increasing the slack of frequently exercised paths through cell sizing [12]. Techniques are proposed in [13] to estimate and analyze the errors caused due to such approximations.

Improvement in power and performance could be alternatively achieved by simplifying the logic functions to reduce their implementation complexity. The first automation effort in this direction focused on two-level circuits, by complementing the output for selected minterms to reduce the sum-of-products implementation [14]. For multi-level circuits, [15] proposes a scheme where a node in the circuit is assumed to have a stuck-at-fault and the circuit is simplified by propagating this redundancy. The resultant errors are then estimated using simulation and a modified automatic test pattern generation (ATPG) algorithm. This process is iterated until the pre-specified bounds are violated. A similar iterative approach is adopted in [16], however, pruning is instead carried out on paths with lowest path activation probabilities.

A common attribute of the above techniques is that they require design of a custom tool to perform the required approximations. In both cases, the quality metric is, in essence, hardwired into their synthesis procedures *i.e.,* the synthesis tools need to be substantially modified for using them with different error metrics. Also, these techniques do not perform any structural modifications to circuits but rather simplify circuits only through redundancy propagation or by pruning gates exclusive to a path. Lastly, these techniques mostly rely on simulations to testify if the approximate circuit adheres to the quality bounds.

In contrast, SALSA takes a systematic approach to approximate logic synthesis. The problem is reformulated using circuit transformations and cast in such a manner that existing logic synthesis tools could be leveraged for approximate logic synthesis. This vastly enhances the extent of approximations applied, since the full suite of techniques used in logic synthesis tools can be utilized. In addition to pruning/removing gates, this approach provides the capability to transform the functionality of circuit nodes. Also, SALSA decouples the synthesis procedure from the target error metric, which makes the approach more generic and easily adaptable. Finally, SALSA provides an inherent guarantee that the synthesized approximate circuit adheres to the pre-specified quality bounds. We believe that the above distinguishing traits make SALSA a promising approach to approximate logic synthesis.

## 3. SALSA: PRELIMINARIES AND APPROACH

The problem statement for approximate logic synthesis could be articulated as follows. Given the description of a logic circuit and a constraint on the errors that could be tolerated, the synthesis procedure should identify avenues for logic simplification and generate a functionally approximate version of the circuit that satisfies the pre-defined error bounds. This section describes the approach used in SALSA to accomplish this objective.

### 3.1 Quality Constraint Circuit

Figure 1 shows the Quality Constraint Circuit (QCC) that is used in SALSA to formulate the problem of approximate synthesis. The QCC is composed of three major blocks *viz.* the *Original circuit*, the *Approximate circuit* and the *Quality function* (Q-function). The original circuit block contains a structural description of the circuit that needs to be approximated and the error constraints that are to be satisfied are encoded into the Q-function. From the problem definition, both these blocks are available as inputs to SALSA. The task of SALSA is to synthesize the approximate circuit, so that the constraints set in the Q-function are never violated.

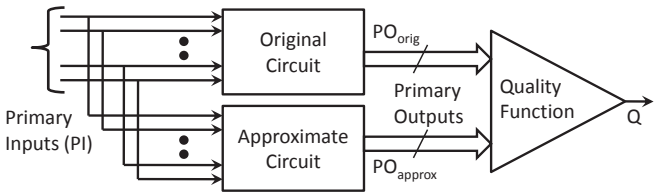



**Figure 1: Quality constraint circuit**

The inputs to the QCC are the primary inputs of the circuit considered for approximation. The output of the QCC is a single bit $Q$ that indicates whether the constraints encoded into the Q-function are satisfied. The Q-function takes outputs from both the original circuit $PO_{orig}$ and approximate circuit $PO_{approx}$ and decides if the quality constraints

are satisfied. A $Q$ output of logic '1' means that the approximate circuit conforms to the imposed quality bounds whereas a logic '0' output indicates a transgression. Thus, the QCC determines the legitimacy of the approximate circuit. From a functional viewpoint, for the approximate circuit to be valid, we need to ensure that $Q$ evaluates to '1' for all possible input combinations. Stated otherwise, the QCC with the synthesized approximate circuit should evaluate to a tautology. At all times during the approximate synthesis process, SALSA preserves this invariant.

## 3.2 Quality Function

As mentioned earlier, the Q-function takes in outputs from the original and approximate circuits and generates a single bit output indicating quality. In a circuit with M primary outputs, the Q-function maps 2M inputs into a one bit output. Thus, in SALSA, any error metric that could be expressed as a Boolean function of the original and approximate circuit output bits could be specified as the Q-function.

For our experiments, we use two different quality metrics. The first is error magnitude, where the approximate output can differ from the correct output by no more than a specified value. The other is relative error, in which the ratio of the original and approximate values is constrained to differ from 1 by at most a certain margin. The Q-functions for these quality metrics are provided in equations 1 and 2 and can be easily encoded as logic functions.

$$Q = \left( |PO_{orig} - PO_{approx}| \leq K \right) \quad ? \quad 1 : 0 \tag{1}$$

$$Q = \left( 1 - K \leq \frac{PO_{approx}}{PO_{orig}} \leq 1 + K \right) \quad ? \quad 1 : 0 \tag{2}$$

The results obtained by applying SALSA on a wide range of circuits using these metrics are described in Section 6.

## 3.3 Approximation Don't Cares

We next describe the strategy used in SALSA to transform the ALS problem into a traditional logic synthesis problem. In the QCC, the primary outputs of the original and approximate circuit represent internal nodes. We know that the outputs of the approximate circuit $PO_{approx}$ are valid provided that they do not cause the value at $Q$ to evaluate to '0' for any input value. In other words, we could functionally modify the approximate circuit if the change can never affect the value of $Q$.

In multi-level logic synthesis, the Observability Don't Cares (ODCs) of a node in a logic circuit can be defined as the set of input values for which the primary outputs of the circuit remain insensitive to the node's output [17]. These input combinations can be used to simplify the node because they do not affect the primary outputs of the circuit.

Applying this concept in our scenario, finding the observability don't cares at a bit of $PO_{approx}$ (which is an internal signal in the QCC) gives us the set of primary input values for which $Q$ is insensitive to an output of the approximate circuit. SALSA uses this information to aid in approximating the circuit, and by virtue of their special significance these ODCs are termed as *Approximation Don't Cares* (ADCs) of the circuit.

The question that remains is how we could make use of the ADCs to approximate the circuit. We know that External Don't Cares (EXDCs) of an output in a circuit are the set of primary input combinations for which that primary output is a don't care. In our case, if the approximate circuit block is looked at in isolation, the ADCs for a given bit of $PO_{approx}$ could be considered as the external don't cares for that output. Therefore, by setting these input combinations (ADCs) as EXDCs of an output in the approximate circuit, we could legally simplify or (in our context) approximate the cone of logic generating that output using standard don't care based synthesis techniques [18–20].

## 3.4 Iterative Simplification

In the above method, it is important to note that when one output is being approximated, the functionality of all other outputs remain unaffected. This is because the ADCs, by definition, are specific to a given output bit and do not influence other outputs in any way. However, there could be avenues for approximation in the cones of logic of other output bits and hence this process of approximation should be repeated for all output bits. After each approximation, the QCC setup is updated with the latest available approximate circuit before computing the ADCs for the next output bit.

In summary, the key steps in SALSA are as follows

- Compute ODCs at an output bit of the approximate circuit to derive the set of input combinations for which $Q$ is insensitive to that output.
- Set these ADCs as EXDCs for that output bit and simplify the circuit under this condition.
- Update the QCC with the latest available approximate circuit and iterate this process over all output bits.

The above procedure ensures that the approximations carried out never violate the specified error bounds. Also, the intermediate circuit produced after each iteration is legal and synthesis can be stopped at any point to yield a valid approximate circuit. As shown in later sections, these steps can be realized using conventional logic synthesis tools, which vastly increases the scope of optimization techniques used in ALS.

# 4. SALSA METHODOLOGY

This section describes the methodology that we use to realize the approach proposed in the previous section. The potential challenges in such an implementation and the speedup techniques and heuristics to overcome the same are also described.

## 4.1 SALSA Algorithm

**Algorithm 1** SALSA

**Inputs:** $O$ : Original Circuit
$Q$ : Quality Function
**Output :** $A$ : Approximate Circuit
**Begin**
Initialize $A \Leftarrow O$
**for** each $PO_i \epsilon PO_{approx}$ **do**
  ## STEP 1: Obtain ADCs as f($PO_{orig} \cup PO_{approx}$ - $PO_i$) ##
  $ADC\_PO_i \Leftarrow$ Get_ADC_PO ($Q$, $PO_i$)
  ## STEP 2: Obtain ADCs as f(PI) ##
  $ADC_i \Leftarrow$ Get_ADC ($O$, $A$, $ADC\_PO_i$)
  ## STEP 3: Approximate $PO_i$ using ADCs ##
  $A_i \Leftarrow$ Approx_PO ($A$, $PO_i$, $ADC_i$)
  Update $A \Leftarrow A_i$
**end for**
Return $A$
**End**

Algorithm 1 provides an overview of the steps involved in SALSA. For each output bit, SALSA computes the ADCs and

uses them to approximate the logic cone that generates it. The process of finding the ADCs for a given output bit is carried out in two steps. First, the ADCs are computed as a function of other inputs to the Q-function in the QCC *i.e.,* $PO_{orig}$ and $PO_{approx}$ with the exception of the output bit being processed. Next, using the original and the approximate circuits, these ADCs are expressed in terms of the primary inputs. After this, the computed ADCs are specified as EXDCs for the output bit under consideration and used to simplify its logic. The approximate circuit thus obtained is retained as the starting point for subsequent iterations. We describe below how these steps can be implemented using off-the-shelf logic synthesis tools.

**STEP 1**: In order to compute the ADCs of a primary output bit $PO_i$ of the approximate circuit, we should perform ODC analysis at that node in the QCC. Finding ODCs of an internal node in a circuit, as shown in Figure 2, involves co-factoring the output with respect to the internal node and finding the set of input combinations for which both the positive and negative co-factors are equal. The resultant circuit contains a description of the ADCs of $PO_i$ in terms of all outputs in $PO_{orig}$ and all outputs in $PO_{approx}$ except $PO_i$. We call this the ADC-PO circuit.

In this step, we have essentially extracted the information about the sensitivity of $Q$ to the primary output of interest. This step is performed only with the Q-function and does not involve the original circuit in any way. Once we have extracted this information, the Q-function is not required any further in the algorithm.

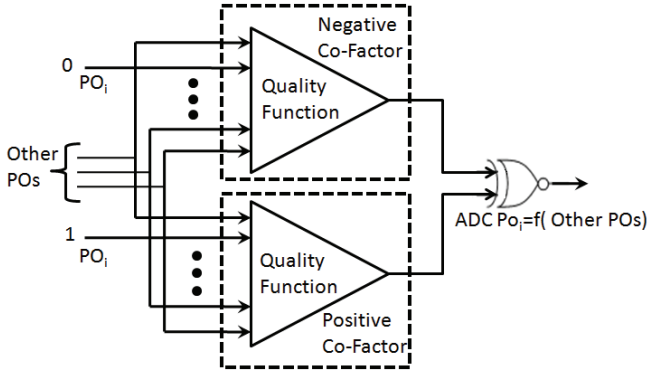



**Figure 2: STEP1 - Obtaining ADCs of a primary output in terms of other original and approximate circuit outputs**

**STEP 2**: After STEP 1, the ADCs for $PO_i$ are available as a function of other primary outputs in the ADC-PO circuit. In this step, we express the ADCs in terms of primary inputs of the circuit. As shown in Figure 3, we connect the approximate and original circuits to the ADC-PO circuit obtained in the previous step. This concatenated circuit is simplified and the required ADCs for $PO_i$ are thus obtained.

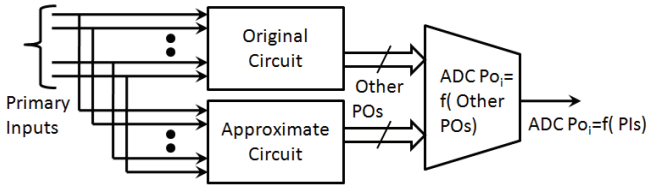



**Figure 3: STEP2 - Obtaining ADCs of a primary output in terms of primary inputs**

**STEP 3**: Given a set of ADCs for an output bit, approximating its logic can be done in a fairly straight forward manner. The computed ADCs are specified as External Don't Cares in the appropriate format required by the logic synthesis tool and conventional don't care based optimization techniques are invoked to simplify the logic cone that generates the output bit. The resultant circuit is used as the approximate circuit in the next iteration.

Thus, SALSA efficiently implements the approach described in Section 3 by reformulating the ALS problem using traditional logic synthesis operations. In each iteration of the algorithm, the synthesis tool is called thrice — once to perform each of the three steps in the algorithm.

## 4.2 Speedup Techniques and Other Heuristics

We next describe some optimizations that could be used to enhance the scalability of the SALSA methodology. The challenges to the above methodology could stem from two different sources - the quality function and the original circuit. If the Q-function is complex, the run-times of steps 1 and 2 of the algorithm are impacted. Also, if the original circuit has a large number of inputs or outputs, then forming the ADCs in step 2 could be a time consuming process. Speed up techniques and heuristics to overcome these challenges are discussed below.

### *4.2.1 Equating Un-approximated Output Bits*

In SALSA, each iteration of the algorithm approximates the logic cone that generates one output bit. The hitherto unprocessed output bits should have their logic to be same as the original circuit. Therefore, while calculating the ADCs, we need not specify the entire approximate circuit, but only the logic cones that generate the output bits that have been previously approximated.

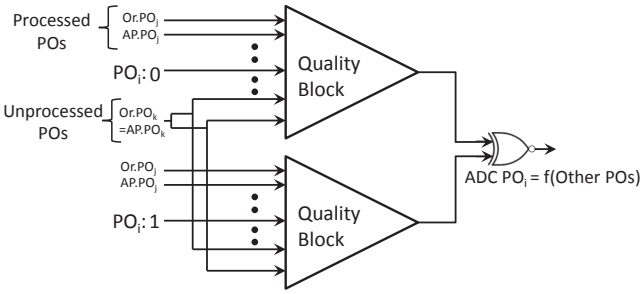



**Figure 4: Equating un-approximated output bits**

Using this observation, as shown in Figure 4, the unprocessed output bits of the approximate circuit are set equal to the corresponding output bits of the original circuit. This vastly simplifies the logic for ADC generation (STEP 1 and STEP 2), especially for initial output bits processed, and does not result in loss of any optimality in the approximations.

### *4.2.2 Quality Function Decomposition*

When the number of outputs present in the original circuit is large, the complexity of the Q-function eventually grows and STEP 1 and STEP 2 of the algorithm consume significant time. A divide-and-conquer heuristic, shown in Figure 5, could be used to tackle this bottleneck. The idea is to decompose the Q-function into stages, with each stage only considering a subset of outputs from the original circuit. For the first stage, the functionality of the Q-function does not change because none of the bits have been approximated. However, for subsequent stages, the maximum error that could occur in the previously approximated bits should be considered while

designing the Q-function. For example, if an error magnitude based metric is used and the Q-function is decomposed in chunks of 8 bits from the LSB to MSB, then while designing the Q-function for the second set of 8 bits, the maximum error that could accrue from the lower order 8 bits should be subtracted from the actual error magnitude threshold.

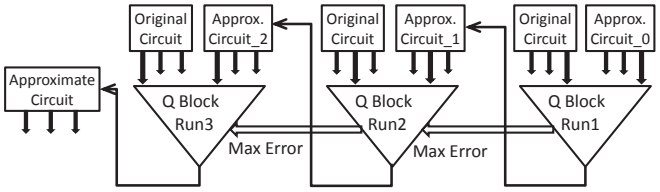



**Figure 5: Quality function decomposition**

This heuristic is very powerful because Q-functions of any arbitrary size can be handled by appropriately decomposing them into stages. However, we do lose some optimality in this procedure because we propagate the worst possible error across stages.

### 4.2.3 Exploiting Input-Output Dependencies

The previous speedup techniques were targeted at addressing the challenge of the Q-function being complex or having a large number of inputs. However, challenges may arise in finding the ADCs when the circuit to be approximated itself is large. We present two techniques to directly address this issue.

We know that, for a given output bit, not all primary inputs lie in its cone of logic. Hence, when finding the ADCs (in STEP 2) for a given output bit, we just need to define the circuit in terms of primary inputs in its transitive fan-in and generate the ADCs only in terms of these inputs. It is important to note that, this technique is exploited when generating the ADCs (STEP 2) and not when simplifying the circuit using these ADCs (STEP 3). This is because we would like to preserve the logic sharing between the output bits. In the implementation, we use the IO dependencies for ADC generation but do not extract cones of logic during logic simplification.

### 4.2.4 Calculating Subset of ADCs

Although the above technique is efficient in many cases, it is not effective when a primary output depends on most of the primary inputs. This scenario happens in the output MSB bits of arithmetic circuits, where the output depends on all less significant input bits. To tackle this, we resort to computing only a subset of the ADCs and use them for circuit approximation. In the implementation, we set certain dependent inputs in the cone of logic of an output to zero and then calculate its ADCs using the usual procedure. In the calculated ADC set, the condition for the dependent inputs that were set to zero is appropriately added before using them for circuit approximation. The above techniques allow us to use SALSA on larger circuits and more complex Q-functions.

## 5. EXPERIMENTAL METHODOLOGY

In order to demonstrate the proposed approach, we tested it on a wide range of circuits for two different error metrics *viz.* error magnitude and relative error. To demonstrate generality, the methodology was implemented using two different off-the-shelf synthesis tools, namely SIS [21] and Synopsys Design Compiler [22]. The circuits used in the experiments, listed in Table 1, range from simple arithmetic circuits to complex datapaths. The complexity of the circuits in terms of number of inputs, outputs, and gates is also listed. The circuits were approximated for a range of error values. The original and approximate circuits were mapped to the IBM 45nm technology library using Design Compiler for iso-delay and evaluated for area and power.

**Table 1: Circuits used in experiments**

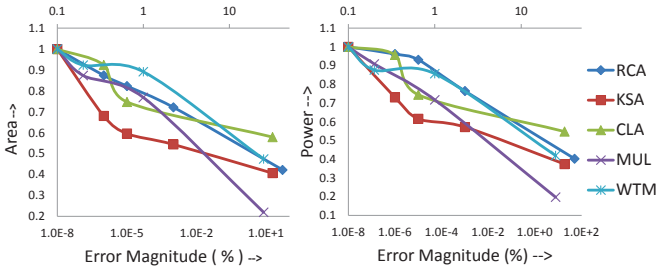
| Name | Function | Bit Width | Gate Count | I/O |
|---|---|---|---|---|
| RCA | Ripple Carry Adder | 32 | 1012 | 64/33 |
| KSA | Kogge Stone Adder | 32 | 1361 | 64/33 |
| CLA | Carry Look-ahead Adder | 32 | 926 | 64/33 |
| MUL | Array Multiplier | 8 | 1055 | 16/16 |
| WTM | Wallace Tree Multiplier | 8 | 1132 | 16/16 |
| MAC | Multiply and Accumulate with 32-bit accumulator | 8 | 1910 | 48/33 |
| SAD | Sum of Absolute Differences (Used in Motion Estimation) | 8 | 1241 | 48/33 |
| EU_DIST | 2D-Euclidean Distance Unit (sans square root) | 8 | 1668 | 32/16 |
| BUT | Butterfly structure (Used in FFT computation) | 8 | 496 | 16/18 |
| FIR | 4-tap FIR filter | 8 | 1719 | 32/16 |
| IIR | 4-tap IIR filter | 8 | 2135 | 56/16 |
| DCT | 8-input Discrete Cosine Transform Block | 8 | 10817 | 64/72 |

## 6. RESULTS

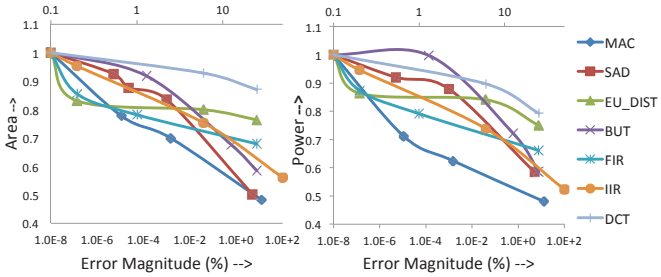
In this section, we present the results of experiments that evaluate the approximate circuits generated by SALSA.

Figure 6 shows the relative area (ratio of approximate circuit to original circuit) *vs.* error magnitude and relative power (ratio of approximate to original) *vs.* error magnitude plots for the benchmark circuits. The error magnitude is shown as a percentage of the maximum output value because the circuits possess different numbers of output bits and thus errors of the same magnitude have varying significance. The dynamic ranges of feasible errors are accordingly different, prompting the use of 2 different error ranges in the graphs. For 32-bit circuits like adders, MAC, and SAD, the lower X axis scale is used while other circuits, whose individual outputs have fewer bits (9 for DCT, BUT and 16 for the rest), follow the upper X axis. From the results, we see an exponential decrease (Note: X axis is in log scale) in area and power initially, which then commences to taper out as we move towards larger error values. This is explained by the fact that, in any arithmetic circuit, adjacent output bits have an exponential difference in their significance. So, for the same increase in error magnitude, the incremental potential for approximation is less as the actual value of the error increases. Also, for a given error magnitude, the cone of logic generating the LSB bits, that have exponentially lower significance compared to their MSB counterparts, have a large set of ADCs and hence have a better chance of being approximated. From Figure 6, we see that SALSA yields area savings in the range of 1.1X-1.85X for tight error constraints (less than 1%) and up to 4.75X for relaxed error constraints (upto 20%). Power benefits range from 1.15X-1.75X and 1.3X-5.25X for similar tight and relaxed error constraints respectively.

The next set of graphs, in Figure 7, show the results obtained for the relative error metric. The relative error is defined as the ratio of the approximate output to the original output. Similar trends with savings up to 1.7X in area and 1.65X in power are observed for this metric. We also observed that the ADCs derived by SALSA for the relative error metric and the error magnitude metric differed significantly. In case of the relative error metric, for small actual values of output, even a small change in the logic would prompt the output



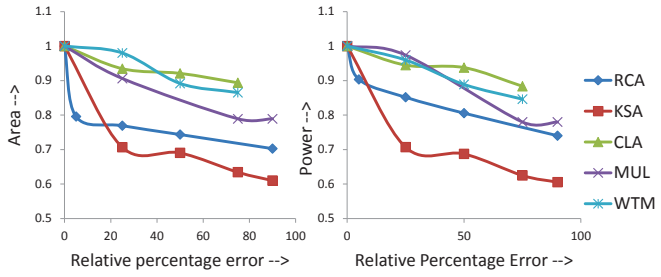

(a) Area and power savings for arithmetic circuits



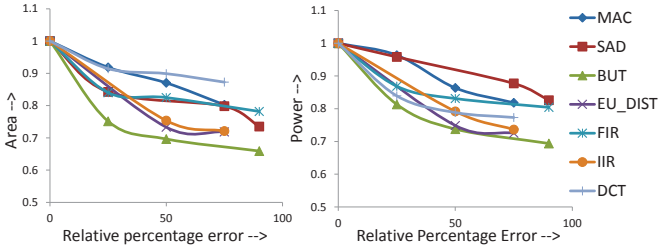

(b) Area and power savings for complex blocks and complete datapaths

**Figure 6: Results for error magnitude metric**

to deviate by a large percentage relative to the golden value. Moreover, the ADCs for the LSBs cannot depend on the MSB inputs. Therefore, we get a comparatively larger ADC set for the MSB output bits.




(a) Area and power savings of arithmetic circuits




(b) Area and power savings of functional blocks and datapath modules

**Figure 7: Results for relative error metric**

The execution times of SALSA, on a server with an AMD Opteron 6176 (2.29 GHz) processor and 198 GB RAM, ranged from 4 minutes for smaller circuits (adders etc.) to 2.5 hours for larger datapath units (DCT).

## 7. CONCLUSION

Error resilient applications provide designers with a unique dimension for optimizing power consumption and area of a circuit. Paradigms like approximate computing have enabled a vast scope of implementation strategies for error resilient circuits. In our work, we have developed SALSA, a framework to automatically synthesize approximate circuits for a given error constraint. This framework, by virtue of transforming approximate circuit synthesis into a well studied logic synthesis problem, can make use of any underlying synthesis tool to effect these approximations, thereby largely widening its scope for application. Since SALSA completely separates the notion of quality, or the error metric, from the actual synthesis procedure, it is adaptable across different error metrics. Further, it provides a guarantee that the error constraints are respected. We demonstrated the utility of SALSA by approximating various arithmetic circuits, complex blocks and entire datapaths and evaluated the benefits in terms of power and area savings.

**Acknowledgment:** This work was supported in part by the National Science Foundation under grant no. 1018621.

## 8. REFERENCES

[1] Y. K. Chen, J. Chhugani, P. Dubey, C.J. Hughes, D. Kim, S. Kumar, V.W. Lee, A.D. Nguyen, and M. Smelyanskiy. Convergence of recognition, mining, and synthesis workloads and its implications. *Proc. IEEE*, 96(5):790 –807, May 2008.

[2] S. T. Chakradhar and A. Raghunathan. Best-effort computing: Re-thinking parallel software and hardware. In *Proc. DAC*, pages 865 –870, June 2010.

[3] M.A. Breuer. Multi-media applications and imprecise computation. In *Proc. Euromicro Conf. on Digital System Design*, pages 2 – 7, Aug.- Sept. 2005.

[4] K. Palem et. al. Sustaining moore's law in embedded computing through probabilistic and approximate design: Retrospects and prospects. In *Proc. CASES*, pages 1–10, 2009.

[5] L. Leem, H. Cho, J. Bau, Q.A. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In *Proc. DATE '10*, pages 1560 –1565, Mar. 2010.

[6] H. Hoffmann et. al. Dynamic knobs for responsive power-aware computing. In *Proc. ASPLOS*, pages 199–212, 2011.

[7] V.K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S.T. Chakradhar. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Proc. DAC*, pages 555 –560, June 2010.

[8] R. Hegde and N. R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proc. ISLPED*, pages 30–35, 1999.

[9] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan, and K. Roy. IMPACT: Imprecise adders for low-power approximate computing. In *Proc. ISLPED 2011*, pages 409 –414, Aug. 2011.

[10] D. Shin and S.K. Gupta. A re-design technique for datapath modules in error tolerant applications. In *Proc. ATS*, pages 431 –437, Nov. 2008.

[11] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *Proc. VLSI Design*, pages 346 –351, Jan. 2011.

[12] A.B. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *Proc. ASP-DAC*, pages 825 –831, Jan. 2010.

[13] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. MACACO: Modeling and analysis of circuits for approximate computing. In *Proc. ICCAD*, pages 667 –673, Nov. 2011.

[14] D. Shin and S.K. Gupta. Approximate logic synthesis for error tolerant applications. In *Proc. DATE*, pages 957–960, Mar. 2010.

[15] D. Shin and S.K. Gupta. A new circuit simplification method for error tolerant applications. In *Proc. DATE*, Mar. 2011.

[16] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet. Energy parsimonious circuit design through probabilistic pruning. In *Proc. DATE, 2011*, Mar. 2011.

[17] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.

[18] H. Savoj and R. K. Brayton. The use of observability and external don't cares for the simplification of multi-level networks. In *Proc. DAC*, pages 297–301, 1990.

[19] K. H. Chang, V. Bertacco, I. L. Markov, and A. Mishchenko. Logic synthesis and circuit customization using extensive external don't-cares. *ACM TODAES*, 15:26:1–26:24, June 2010.

[20] S.C. Chang and M. M. Sadowska. Perturb and simplify: optimizing circuits with external don't cares. In *Proc. ED TC*, pages 402 –406, mar 1996.

[21] E.M. Sentovich and K.J. Singh. SIS: A system for sequential circuit synthesis. Technical report, EECS, UCB, 1992.

[22] Design Compiler. Synopsys inc.