AxLS: An Open-Source Framework for Netlist Transformation Approximate Logic Synthesis

Jorge Castro-Godínez*[†], Humberto Barrantes-García[†], Muhammad Shafique[‡], Jörg Henkel[†]

*Karlsruhe Institute of Technology (KIT), Germany

[†]Instituto Tecnológico de Costa Rica (TEC), Costa Rica

[‡]New York University Abu Dhabi (NYUAD), UAE

Corresponding Author: jorge.castro-godinez@kit.edu

***Abstract*—With the rise of Approximate Computing as an energy-efficient design paradigm for error-tolerant applications, different Approximate Logic Synthesis (ALS) approaches have been proposed to generate approximate circuits from accurate implementations automatically. In this paper, we present AxLS, our open-source framework for ALS techniques based on structural netlist transformations. We describe our framework and provide an experimental evaluation for arithmetic circuits with our current implementation. AxLS enables the study and test of existing ALS techniques based on netlist transformations and the proposition of new ones. AxLS is available to download at https://github.com/ECASLab/AxLS.**

***Index Terms*—Approximate computing, logic synthesis, design tools.**

I. Introduction

The need to improve power and energy efficiency in computing systems has motivated the emergence of new devices, architectures, and design techniques. *Approximate Computing* is one paradigm in which the required computational resources of error-tolerant applications can be reduced in exchange for some degradation in the accuracy of results [1]. Examples of these error-tolerant applications include image and video processing, computer vision, data mining, and machine learning [2], [3].

With the rise of the approximate computing paradigm, different Approximate Logic Synthesis (ALS) approaches have been proposed to generate approximate circuits from accurate implementations automatically. In the literature, three main approaches to exploit functional simplification have been reported [4]: netlist transformation [5], [6], Boolean rewriting [7], [8], and approximate high-level synthesis [9], [10].

For the case of Boolean rewriting, open-source contributions have been made [11]. However, to explore and implement current ALS netlist transformation techniques reported in the literature, develop methods for error modeling, and propose new approaches, an open-source framework that enables it is still missing.

Contribution: This paper presents AxLS, an open-source framework for ALS techniques based on structural netlist transformations. We describe our framework and provide an experimental evaluation for arithmetic circuits with our current implementation of AxLS. AxLS is available to the community at https://github.com/ECASLab/AxLS.

II. AxLS Framework

AxLS is a framework for ALS techniques based on the idea of structural netlist transformations. Figure 1 depicts the main components of our framework. As shown, AxLS requires the Verilog RTL description of the circuit to be approximated. A netlist is generated using a synthesis tool for a specific standard cell technology library. From the same technology library, particularly from the Verilog simulation models, a description of the cells is created and put into an XML file. A representation of the netlist is generated with a custom netlist to XML function (`v2xml`) and the cell description. XML files are easy to retrieve, manipulate, and save. Figure 2 presents a code snippet of a toy netlist representation with XML, using the NanGate 15nm technology library. Figure 3 shows a direct acyclic graph (DAG) representation for this netlist example, generated from the XML representation.

A post-synthesis simulation can be performed to obtain gate-level switching activity values (saif file). This information can be included in the XML netlist representation as part of the gate's properties, and it can be further used to guide the netlist transformation criteria [12].

With an XML representation of the netlist, different approximation criteria can be applied to transform the netlist. For instance, gates can be pruned one by one considering their impact on the output error and switching activity [5], genetic-based algorithms can be used to mutate the netlist into approximate versions by interchanging gates with wire connections [13], or output pruning can be applied by removing all logic gates that affect a specific primary output of the netlist.

Regardless of the netlist transformation technique followed, an accuracy threshold is required for a defined error metric. This accuracy target is used to assess if the approximations applied creates an approximate netlist that still produces acceptable results. According to the approximation criteria that can be implemented within AxLS, a Verilog file with the approximate netlist can be generated from the transformed netlist described with XML. Using a custom XML to netlist function (`xml2v`), this Verilog file can be generated every time the approximate version's accuracy is evaluated. The generated netlist is based on the gates available in the cell library used for the synthesis.

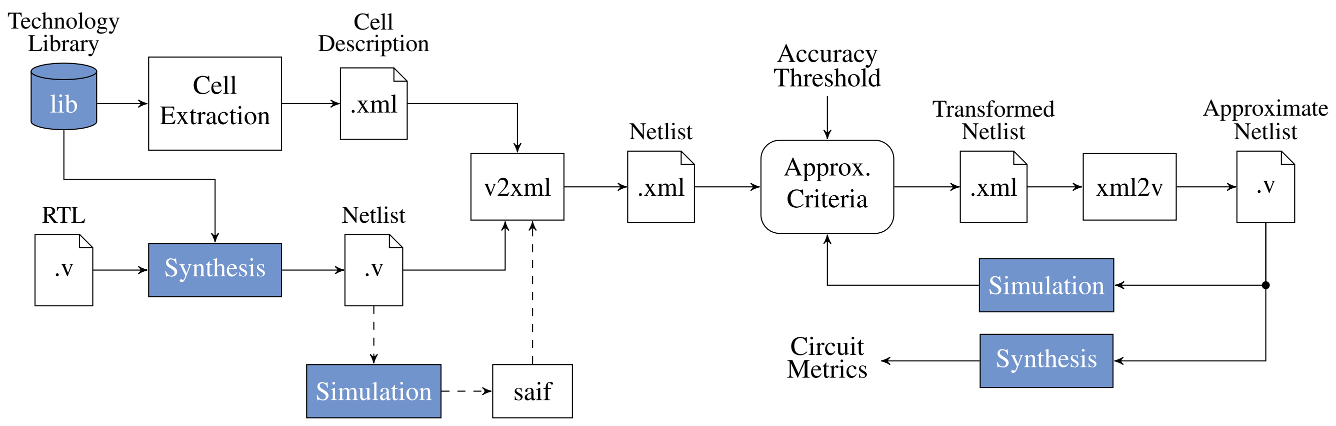


Fig. 1. An overview of our automated AxLS framework. An XML representation of the synthesized netlist is created to manipulate it according to the approximation criteria defined within AxLS. AxLS uses external tools for synthesis and simulation of the accurate and approximate versions of the netlist.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <node name="INV_X2" var="u19">
    <input name="A1" wire="n14" />
    <output name="ZN" wire="n16" />
  </node>
  <node name="OAI21_X1" var="u23">
    <input name="A1" wire="n14" />
    <input name="A2" wire="n20" />
    <output name="ZN" wire="n21" />
  </node>
  <node name="NAND2_X2" var="u33">
    <input name="A1" wire="n16" />
    <input name="A2" wire="n14" />
    <output name="ZN" wire="S[1]" />
  </node>
  [ ... ]
  <circuitinputs>
    <input var="in[0]"/>
    <input var="in[1]"/>
  </circuitinputs>
  <circuitoutputs>
    <output var="S[0]"/>
    <output var="S[1]"/>
  </circuitoutputs>
</root>
```

Fig. 2. XML code for a netlist description example.

The approximate netlist can be simulated to produce approximate outputs. AxLS provides functionality to compare the approximate results against accurate ones to obtain an error distribution in the form of a probability mass function, commonly use for accuracy estimations [14], and to generate values for different error metrics from it (for instance, those described in [13]). Although not depicted in Figure 1, the corresponding testbench is required for any simulation, which generates an output file with the approximate results. For this, representative test vectors are required for the input bit-width of the circuit to be approximated.

Once an approximate netlist is generated with satisfactory accuracy, the synthesis tool can be used to produce circuit metrics such as area, delay, and power consumption.

III. Evaluation

The current state of AxLS has been implemented using Python language. As depicted, AxLS relies on external tools for synthesis and simulation. Currently, AxLS uses the Yosys tool [15] for circuit synthesis and circuit area estimation, and Icarus Verilog [16] for netlist simulation. The results here presented were obtained using the NanGate 15nm technology library.

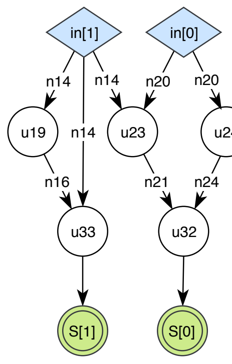
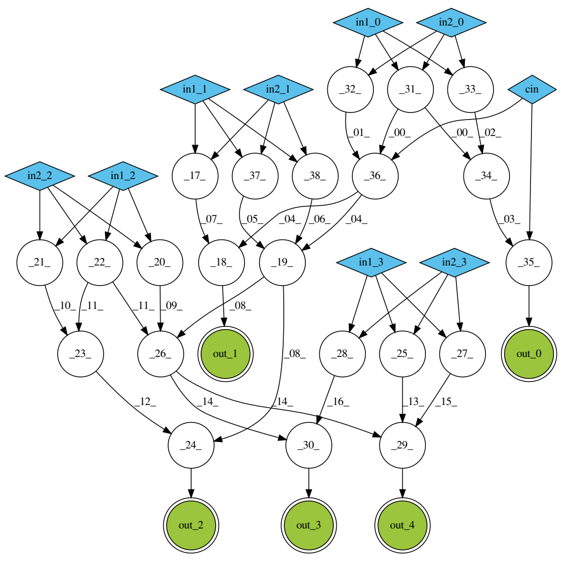


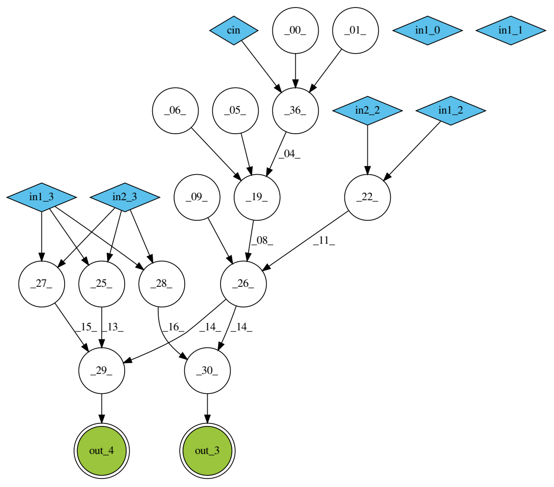
Fig. 3. DAG representing an XML netlist example.

We present an evaluation of AxLS for arithmetic circuits, particularly standard adders. As approximate criteria, the following steps have been applied:

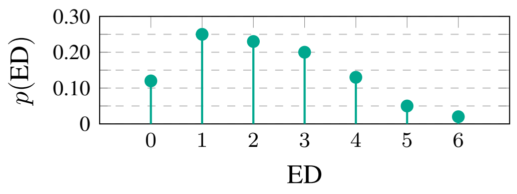
- Considering a primary input bit constant, all affected nodes (gates) are explored. A node is considered constant if all its inputs are constant, and then all dependencies of such node are explored. Each of the considered nodes is pruned, one by one, and accuracy is checked at each step. If the output error goes beyond the given threshold, the last pruning is reversed, and other nodes are further explored.
- After exploring from the perspective of the primary inputs, nodes are explored considering a primary output bit constant. All nodes affecting such output are considered, and a one by one pruning is also performed with accuracy evaluations after every step. Similar to before, if the output error goes beyond the given threshold, the last pruning step is reversed, and other nodes affecting such output are explored.



(a) Accurate netlist.



(b) Approximate netlist for WCE = 8.



(c) Error distribution for the approximate netlist generated for WCE = 8.

Fig. 4. Direct acyclic graph representation of an accurate and approximate 4-bit ripple-carry adder. The approximate version has been modified with a WCE constraint of 8.

- For those nodes and outputs removed, a 0 value is assigned. For instance, nodes depending on others pruned will receive a 0 as input instead of the expected result previously provided by the now missing node.
- For the scope of this evaluation, these steps are repeated for the half less significant bits (LSB). So, for instance, for a 16-bit adder, these approximation criteria are applied to the first 8 LSB of the inputs and then to the 8 LSB of the output.

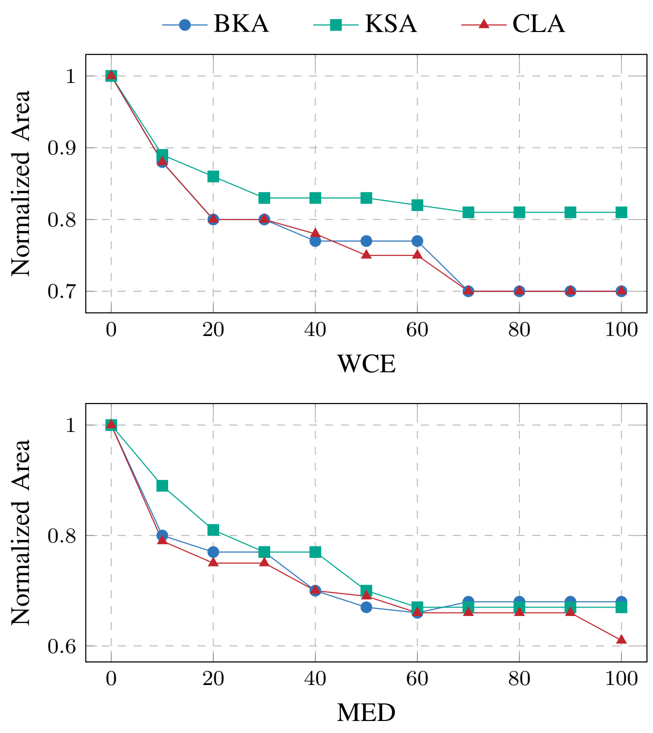


Fig. 5. Evaluation using AxLS for three 16-bit adders, BKA, KSA and, CLA, and two error metrics, WCE and MED.

Figure 4 depicts a simple example of a 4-bit ripple-carry adder. The approximate netlist (Figure 4b) has been generated following the previous steps described and for a worst-case error (WCE) of 8 as an accuracy threshold. WCE represents the maximum error tolerable for an approximate design, despite its probability of occurrence. As it can be noticed from the error distribution in Figure 4c, the higher error produced is 6, with a very low probability. From this error distribution, other error metrics can be calculated, such as the mean error distance (MED), which is 2.2 for this case.

As can be observed, the resulting netlist depends solely on the two most significant bits of the inputs and the carry-in signal. Just two of the output bits are calculated, while the others (not depicted in the diagram) are defined as 0. For nodes that had a dependency on pruned gates, for instance, `_036_`, the remaining wires `_00_` and `_01_` will drive a 0 value, as previously described for the netlist transformation steps followed as an example with AxLS.

Figure 5 presents the results for the approximate netlist generated for three 16-bit standard adders: Brent-Kung adder (BKA), Kogge-Stone adder (KSA), and carry-lookahead adder (CLA). These results correspond to circuit area reduction for different target WCE and MED. In general, as it can be observed, for a higher tolerable error, more savings are achieved. However, using the described steps, for some accuracy targets, the same area savings are obtained. This is the case for BKA and CLA for the WCE error metric, for which no further netlist modification can be applied without degrading the accuracy beyond the defined threshold.

IV. Conclusion

In this paper, we presented AxLS, an open-source framework for ALS techniques based on netlist transformation. We described the main components in our framework and presented an experimental evaluation performed with AxLS for arithmetic circuits (standard adders) and two accuracy metrics. AxLS demonstrates to be a suitable framework to enable the test of ALS strategies. As future work, it is foreseen the exploration of machine learning-based techniques to estimate the impact of specific gate pruning in the output error to avoid repetitive simulations.

Acknowledgment

This work was partially supported by the Costa Rica Institute of Technology.

References

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate Computing: A Survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb. 2016.

[2] Y.-K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nhuyen, and M. Smelyanskiy, "Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790–807, May 2008.

[3] J. Castro-Godínez, D. Hernández-Araya, M. Shafique, and J. Henkel, "Approximate Acceleration for CNN-based Applications on IoT Edge Devices," in *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, 2020, pp. 1–4.

[4] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate Logic Synthesis: A Survey," *Proceedings of the IEEE*, pp. 1–19, 2020.

[5] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and Applications of Approximate Circuits by Gate-Level Pruning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1694–1702, 2017.

[6] I. Scarabottolo, G. Ansaloni, and L. Pozzi, "Circuit Carving: A Methodology for the Design of Approximate Hardware," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2018, pp. 545–550.

[7] Y. Wu and W. Qian, "An Efficient Method for Multi-Level Approximate Logic Synthesis under Error Rate Constraint," in *53nd Design Automation Conference (DAC)*, 2016, pp. 1–6.

[8] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate Logic Synthesis Using Boolean Matrix Factorization," in *55th Annual Design Automation Conference (DAC)*, Jun. 2018, pp. 55:1–55:6.

[9] S. Lee, L. K. John, and A. Gerstlauer, "High-Level Synthesis of Approximate Hardware under Joint Precision and Voltage Scaling," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2017, pp. 187–192.

[10] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel, "AxHLS: Design Space Exploration and High-Level Synthesis of Approximate Accelerators using Approximate Functional Units and Analytical Models," in *2020 IEEE/ACM 39th International Conference on Computer-Aided Design (ICCAD)*, 2020.

[11] J. Ma, S. Hashemi, and S. Reda, "Approximate Logic Synthesis Using BLASYS," in *Workshop on Open-Source EDA Technology (WOSET)*, no. 5, Nov. 2019.

[12] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet, "Energy Parsimonious Circuit Design through Probabilistic Pruning," in *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2011, pp. 1–6.

[13] Z. Mrazek, Z. Vasicek, and L. Sekanina, "EvoApproxLib: Extended Library of Approximate Arithmetic Circuits," in *Workshop on Open-Source EDA Technology (WOSET)*, no. 10, Nov. 2019.

[14] J. Castro-Godínez, S. Esser, M. Shafique, S. Pagani, and J. Henkel, "Compiler-Driven Error Analysis for Designing Approximate Accelerators," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2018, pp. 1027–1032.

[15] C. Wolf, "Yosys Open SYnthesis Suite," http://www.clifford.at/yosys/.

[16] S. Williams, "Icarus Verilog," http://iverilog.icarus.com/.