# ASHES 1.5: Analog Computing Synthesis for FPAAs and ASICs

Afolabi Ige and Jennifer Hasler
School of Electrical and Computer Engineering,
Georgia Institute of Technology, Atlanta, GA
aige3@gatech.edu, jennifer.hasler@ece.gatech.edu

***Abstract*—Synthesis tools can unlock the potential of analog architectures to achieve real-time computation, signal processing, inference and learning for low SWaP systems in commercial timescales. We present a methodology and results towards system analog and mixed-signal synthesis both for FPAAs and Custom Analog IC design. Building on previously efforts on large-scale Field Programmable Analog Arrays (FPAA) targeting tools enables tools capable of synthesizing new ICs. The IC synthesis is built upon our recent work on analog & mixed-signal programmable CMOS standard cell library that can be used across a range of CMOS process nodes (e.g. 180nm, 130nm, 65nm, 28nm, and 16nm CMOS). The entire tool-flow is being developed as an open-source tool that can be widely available. These approaches enable moving analog and mixed-signal design towards structured Design Space Exploration (DSE).**

***Index Terms*—Analog Synthesis, FPAA, Floating-Gate, Analog Standard Cells**

## I. Analog Computing: The what and why

Analog computing has ancient roots, from primitive counting with fingers and toes to Lord Kelvin's mechanical tide predictor in the 1880s [1]. Analog computing uses physical phenomena for computation. This definition of analog computing expands the often-perceived narrower view of analog computing being simply a Vector-Matrix Multiplication (VMM) [2] or Computing in Memory (CiM) [3] to a wider space of analog and mixed-signal architectures enabling a renaissance driven by new applications and technological capabilities.

The resurgence of interest in analog computing stems from its potential for significant improvements in area and power efficiency as well as near zero latency compared to digital implementations in specific applications [4]. Demonstrations in speech recognition [5], sound [6] recognition, Hopfield networks [7], multi-layer perceptrons [8], analog sorting [9], and large-scale Field Programmable Analog Arrays (FPAA) [10] show existing data points in the argument for analog computing as a viable path.

The opportunity to see improvement by exploring more applications motivates the need for tools to enable such investigation. A historical perspective of digital tools show that the field of VLSI did not take off until after tools became reliable. As such our contribution is an integrated open-source tool, Ashes 1.5, that is capable of both programming a reconfigurable device like an FPAA but also generating Application Specific Integrated Circuit (ASIC) layout from a high level description with a focus on primarily optimizing design area in its placement (and routing) approach. Experimental results

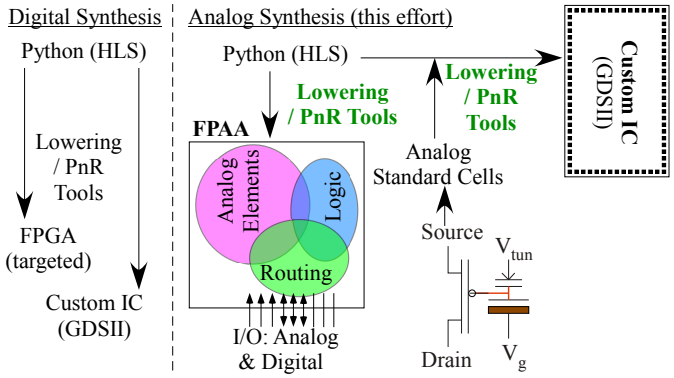



Fig. 1. Paralleling digital design that can start from a high-level formulation (HLS) that can be lowered into an FPGA and/or a Custom IC (GDSII), this effort discusses our analog synthesis tools (Analog HLS) that can start from a high-level python formulation that can be lowered into large-scale Field Programmable Analog Arrays (FPAA) and/or a Custom IC. A critical part of the Analog synthesis to a Custom IC is the recent development of analog & mixed-signal standard cells that are enabled through Floating-Gate (FG) devices available on standard CMOS processes.

| | Parse Std Cells ? | FG-aware placement ? | Reconfigurable fabric porgram ? |
|---|---|---|---|
| OpenLane/OpenRoad [11] | Yes | No | No |
| Align [12] | No | No | No |
| Magical [13] | No | No | No |
| Yosys-ABC9 [14] | No | No | Yes |
| **This Work** | Yes | Yes | Yes |

Fig. 2. Comparison of contemporary open source tools with this work.

show that the tool is capable of both programming to FPAAs for circuit protoyping with FPAA flow while also able to synthesize real world analog computing circuits with layout showing 25% area improvement, 64% wirelength improvement 98x and 2.52x speedup in placement and routing over Cadence tools.

Starting from discussing the recent innovation of analog standard cells (Sec. II) and the analog tool philosophy (Sec. III), we describe the Ashes tool architecture (Sec. IV), FPAA synthesis & measurements (Sec. V), and analog ASIC synthesis (Sec. VI). We show multiple compiled ASIC synthesis designs (Sec. VII) before summarizing the work (Sec. VIII).

## II. Analog Standard Cells

The recent development of analog standard cells (Fig. 3) makes the synthesis to an analog IC possible. The analog standard cell philosophy builds on FG elements to enable a tractable number of analog standard cells, a common library that spans a number of IC processes. Two standard cell libraries

978-3-9826741-0-0/DATE25/© 2025 EDAA

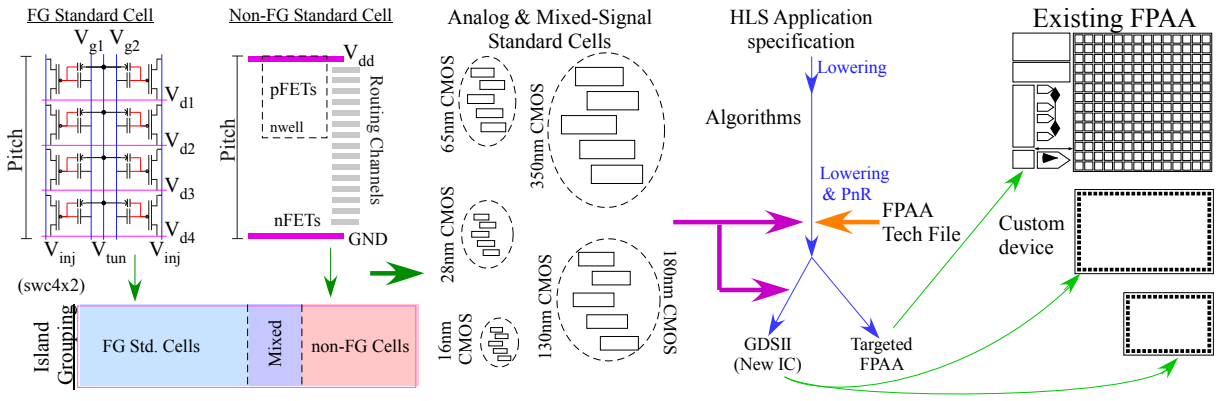



Fig. 3. Programmable Analog Standard Cells are a combination of FG-enabled and non-FG enabled standard cells that would be gathered into a single Island. FG-enabled cells tend to be tight packing of FG cells with a tight packing of required routing; non-FG cells follow a typical Vdd and GND lines with designated routing channels. Some cells are a mixture of FG and non-FG components, cells often on interface between FG & non-FG cells in an Island. These cells have been demonstrated, or are being demonstrated across multipleIC processing nodes, where each of these cells could be used as part of the synthesis (lowering and PnR) tools, enabling the synthesis of Custom IC devices in any of these IC process nodes.

(130nm [15] and 65nm [16] ) have been reported, and four additional standard cell libraries (16nm, 28nm, 180nm, and 350nm) will be reported shortly. This effort shows examples using the 350nm library. The set of cells for a given process node will have device parameters and cell sizes for the designer to develop their high-level systems. An analog synthesis tool must produce a correct Custom IC result for any of the IC process nodes, given the tools have sufficient information for the PnR for that IC process.

The pitch is set by a swc4x2 FG cells, a tight packing of 4-high by 2-wide array of FG devices. The devices are designed to abut with multiple useful cases, such as larger FG arrays and/or classifier terminations (e.g. Winner-Take-All cells). The cell width is allowed to change, although the sizes are often bounded to improve packing and routing of cells. The cells use the lowest three layers of metal for routing, leaving the higher metal layers for higher routing. The analog cells utilize a number of fine-grain components (e.g. transistors), medium level components (e.g. transconductance amplifiers), and infrastructure cells (e.g. drain-line decoding for programming).

The wide FG programming range from pA to nA to $\mu$A currents, enabling at least 8-9 orders of magnitude of critical parameters like operating frequency ( 1Hz $\rightarrow$ 1GHz ), enables an analog standard-cell library with only a few types & sizes of transistors and other components (Transconductance Amplifiers (TA) ). The medium level components (e.g. WTA block) enable dense compilation of a range of mixed-signal classification / inference / on-chip embedded machine learning algorithms, Recent efforts in analog abstraction [17] provided the framing for defining these standard cells.

## III. Analog Computing Design Philosophy

At one level, analog computing platforms are similar to digital computing platforms. Both approaches enable both reconfigurable platforms (FPGA, FPAA) and IC designs. In both cases, the reconfigurable platform can be a targeted prototype for the IC design, as well as may be a sufficient commercial solution not requiring the cost of additional IC fabrication. The reconfigurable platform enables rapid simulation for an eventual custom design.

At another level, analog computing platforms significantly lack the tool framework that currently exists for digital systems. Digital systems have C-level and RTL-level simulation tools developed over decades of IC designs. Only recently have analog standard cells been developed. For FPAAs, some abstraction and cell modeling & simulation are possible [18], although these systems struggle with large designs, although less than full SPICE simulations. This effort opens up the opportunity for a range of abstracted and faster analog simulations for system development and for system verification.

The focus for this tool is optimizing the implementation area. Because we can program the FG bias currents driving signal lines, minimizing the implementation area tends to reduce the overall routing capacitance, thereby reducing the required power and likely optimizing the resulting performance. Further, the FG standard cell lines have fixed potential lines used for programming between the signal lines, partially shielding the lines from line-to-line coupling. Therefore, Ashes optimizes for area as its primary metric, and minimizing overall line capacitance is the desired PnR optimization metric where required.

## IV. Tool Architecture

Extending from our design philosophy, the tool (Fig. 4) has two major flows unified by a common python library interface. Starting with the cell store, the python library stores cell information in json for both standard cells and FPAA primitives. This library can act as a single source of truth to be synchronized when new additions are made to either cell or FPAA primitives. From the store a set of base python classes is generated for both types of items. In the case that a designer wants to add more functionality to a class, this can be done in the extensions file which will not be overwritten by the class generator. Additionally, a system library file and examples are provided to demonstrate how larger designs can be formed from the provided classes.

For the FPAA flow, there exists a parallel path into it in the form of a GUI. This was made using the open source tool

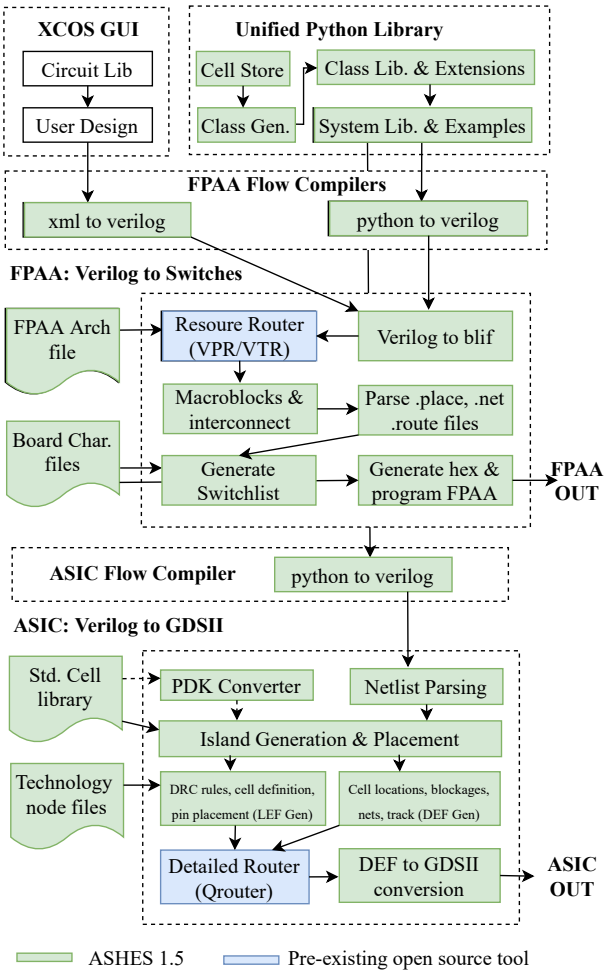


Fig. 4. An overview of the ASHES tool chain.

Scilab and its extension XCOS. A detailed history of the legacy FPAA programming tools is available in [19]. In addition to the historical perspective for XCOS, it provides a user friendly interface for circuit design in the form of a data flow graph. The circuit design can originate in either the GUI or python interface and the resulting output will be lowered through one of a pair of compilers in the FPAA flow, XML-to-verilog or python-to-verilog. Both compilers preserve the connectivity between cells as well as the programmable properties of cells and generate a slightly modified version of verilog. Verilog was chosen for its syntax in handling hardware description and not with the intention of running simulations.

Similarly for the ASIC flow, it is the job of the compiler to understand the relationship between the cells, islands and frame if present. Gathering the cells into islands, demarcating the boundaries for decoders and switch insertion as well as cell instance selection for these elements are all the job of this compiler. Finally, it also generates its own slightly modified version of verilog syntax preserving information such as island identification, matrix versus cell clarification and details of net connections.

## V. Analog Synthesis for FPAA

Once the input has been lowered to the verilog representation on the FPAA side, it enters the backend flow where majority of the changes to it were made for ASHES 1.5. Specifically the overhaul of the verilog-to-blif module and others below it into python from scilab. Additionally, the interconnect generation, VPR output file parsing and a host of supporting class files were converted from python 2 [20].

The verilog-to-blif module is responsible for preparing the inputs for the resource router VPR/VTR [21] as it accepts netlist in blif files. One of the first major features it handles is the 'fix location' feature which allows the user to specify which computational block location they would like to place a particular primitive. Fixing the location has the benefit of ensuring consistent parasitics for repeatable experimental results. In addition to the netlist, other relevant information like the architecture file is passed to VPR. This file specifies items like the size, location and definition of elements available on a particular FPAA, requiring it to be kept updated and is dynamically regenerated when new circuits are created for the board.

The outputs of VPR are a series of files like .place, .net, .route files which need to be parsed for their connectivity and converted into a series of floating gate switch locations to be uploaded to the FPAA. Additionally, things like certain interconnects at the interface of the computational analog blocks need to be specially handled. Once the switch list is ready, there is a separate process by which the proper assembly files and correct programming routines are fetched for the on chip microprocessor which handles the floating gate programming.

To demonstrate the functionality of the FPAA flow of ASHES, we exponentially space and measure a band pass filter on the FPAA. The first building block if a transconductance amplifier (TA) with a floating gate transistor as the bias transistor. The non-volatile charge at the gate allows for the bias current to be programmed. The next primitive is another TA but with additional floating gates for the input differential pair (FG TA). This circuit offers a few advantages, first of which is offset cancellation given that mismatch between the FETs can be tuned out. It also allows for a fixed offset between the pair for a fixed output voltage from the TA. Furthermore it expands the linear range of the amplifier as the extra capacitance at the input forms a dividers which lowers all incoming signals. When these primtives are arranged in the shape (Fig. 5, left) the result if a bandpass filter whose corners can be moved independently when far enough apart and for which the center is set by the ratio of the bias currents of the amplifiers. For the experiment we exponentially space the centers of the bandpass filters to match the filter spacing seen in the human ear (Fig. 5, right). A designer might transition the FPAA flow to an ASIC flow with a focus on creating a tapeout ready circuit (Sec. V), or the designer might have a sufficient component for their application.

## VI. Analog Synthesis for ASIC

ASHES 1.5 also saw significant improvement in its ASIC flowl. The detailed router was replaced, the checkered pattern support and variable decoder tree generation were all added. The netlist parsing reads through the top level cells and reconstructs them from the polygons defined in the gds

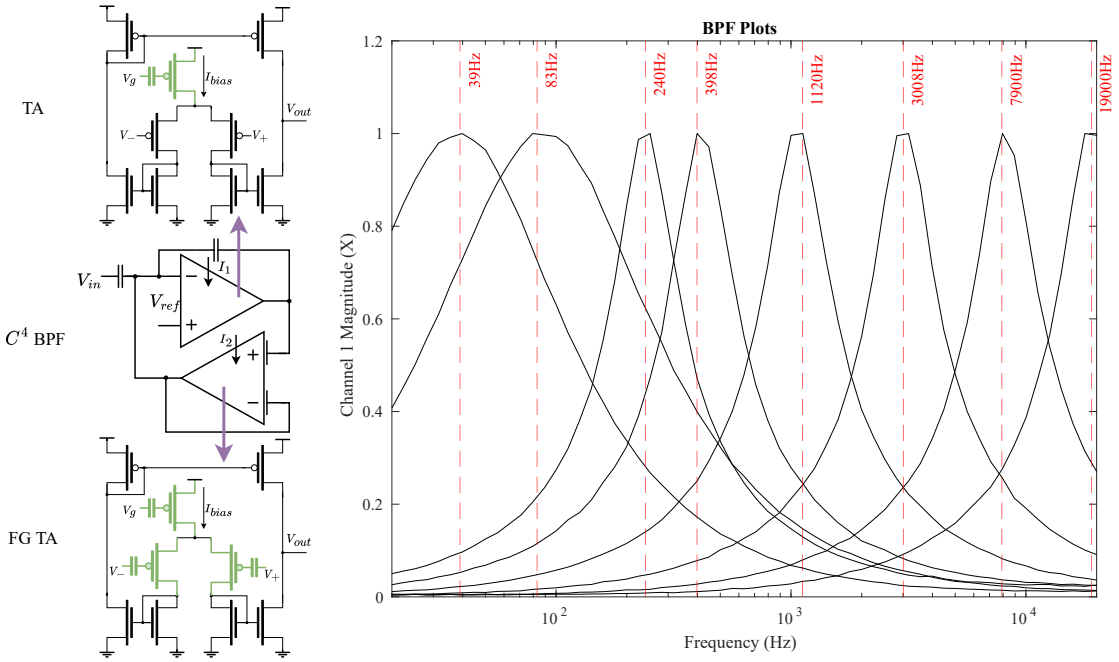



Fig. 5. Left: Schematic of C4 and FGs. Right: Experimentally measured Bandpass filters with exponentially spaced center frequencies programmed on an FPAA.

file. This stage handles pin extraction, cell size calculation (which requires checking for rotation and mirrors) and sub cell tracking to ensure cells are only defined once or top level cells account for the sub cell size. After which, the island generation occurs (Figure 6). For abutting islands, cells are arranged according to connectivity specified in the netlist. For non abutting islands, cells are placed in a checkered pattern to avoid causing DRC errors while connectivity information is updated in the nets table. If required decoder synthesis is then performed to generate the correct size decoder tree to append to a vector of switch which sit vertically and horizontally adjacent to the core. These are used for floating gate programming. Once done the islands can be finalized and placed with equal spacing in the area provided.

After the netlist and placement step, the required files are then generated for the router. These include process specific information like DRC rules and design specific information like pin definition are all placed in the LEF file. The DEF file is even more expansive as it covers individual cell placement, automated blockage insertion which is required for the router to perform DRC clean pin access, net tracking between varying matrix and single cell pins. Furthermore, emperical tuning of parameters like track size, routing costs, jog costs, stage effort can be required to find an clean routing solution. Once Qrouter is done, a conversion module reads the routed DEF output and merges it with the previously placed locations to generate a final output. This section also handles things like hole detection, via cleanup and any other post processing tasks.

### A. More on the Island Approach

It is important to note that the primary metric our flow focuses on is area efficiency within the constraints of not violating DRC or LVS rules. Our island approach stems from the fact that we control our standard cell design meaning we can design the cells to abut to maximize cell density. A strong motivating factor for the development of the island approach is due to the high number of control and power cells a typical FG cell requires, The number of extraneous routes must be minimized. The island architecture allows for sharing of edge infrastructure in decoders and switches. To further improve our efficiency we design our decoder tree to be able to construct arbitrarily larger decoders from abutting cells. This turns a dense routing problem into a fairly tractable placement problem. The area efficiency and synthesis time effects of this are seen in the experimental section.

However, not all applications allow for cell abutting. For a truly universal standard library, it would be impractical for every cell to be able to connect to every other cell. This is what brings about essentially three types of islands. The first is an island within which the cells abut which is straightforward. An example of this is seen in the speed system in the experiment. The second type is an island where no cells abut. An example, seen in Fig. 7, shows how we create an island for characterizing standard cells in a new process node which are then routed to a frame. The analog cells are arranged in a checkered pattern at the core while still sharing multiplexers and floating gate power lines at the edges. The third type is a hybrid where you might have a dense abutting FG enabled section needing to interface with a sparse interchangeable set of cells. An example of this would be found when creating custom computational blocks in a brand new FPAA. This last island type would still fit within our framework and would still take advantage of the efficiency gains described above. Thus we use the island approach for creating modular large scale analog computing blocks when synthesizing custom circuits.

## VII. Experiments and Results

The ASHES toolchain is written in python with calls to libraries (like numpy) written in C. We adopt Qrouter as our detailed router which is written in a mix of C and Tcl. We

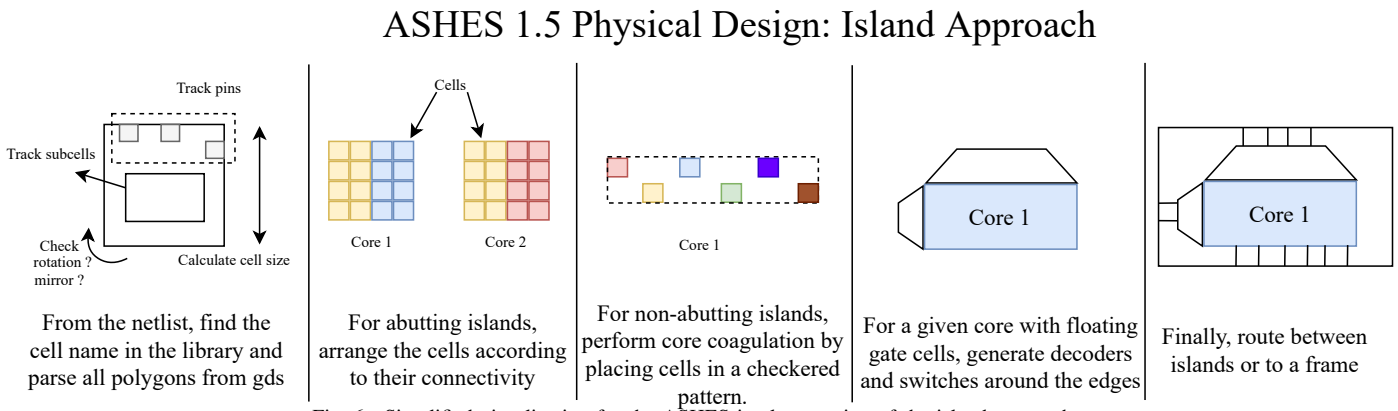



Fig. 6. Simplified visualization for the ASHES implementation of the island approach.

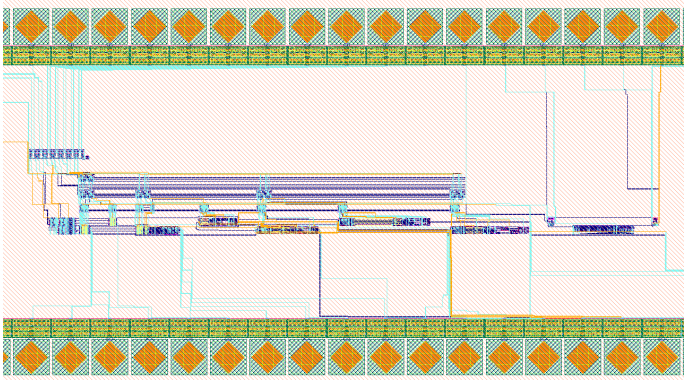


Fig. 7. A synthesized, non-abutting floating gate island showing a compact approach for grouping cells which would characterize a new process node.

compare the synthesis performance against state of the art tools in Cadence for a real world analog computing application that we taped out. The goal was to synthesize the same speed system given the same analog standard cells that were taped out in 350nm. The Cadence tools were run on an 88 core 2.2Ghz Intel Xeon CPU linux server. The ASHES tools were run on a 2 core 1.3 Ghz Intel i7 windows subsystem for linux laptop.

The speed system (Fig. 8a) is designed to be an on chip fully analog universal classifier that can output and classify arbitrary analog patterns and store the digital classification result in a register bank without the use of analog to digital converter (ADCs) or digital to analog converters (DACs). It does using 2 islands, the first is an arbitrary waveform generator (AWG) that will retrieve a vector of 64 analog voltages stored in the FG array depending on which column of the array is activated by a scan chain which sits above it. Next is the VMMWTA [22] island which classifies the output and returns a one-hot vector using the WTA network + an inverter to clean up the signal. These digital outputs are loaded into a 64x32 register bank. Its called a speed system as it can be used to characterize the maximum classification speed of a process node given all computation happens on chip and can later be read out serially for confirmation.

The results of the experiments are summarized in Fig. **??**b. The Cadence tool flow used 1.33x more area for the full speed system while using 1.04x and 1.21x more area for the individual islands. The Cadence tools also used 2.79x more wirelength to route the same speed system while using 1.96x more and 5.45x more routing wires for decoder synthesis. Finally the placement and routing operations both saw the Cadence tool flow use more time with a 98x longer placement and 2.52x longer routing time. Overall our tools demonstrates the advantages of our island approach where by combining the cell design with shared infrastructure can lead to smaller area, less wirelength while running much quicker on a consumer device.

## VIII. Summary and Discussion

This effort showed Ashes 1.5, an analog & mixed-signal synthesis tool that can target FPAAs as well as generate GDSII for fabricating a Custom IC. Experimental results show that the tool is capable of both programming to FPAAs for circuit prototyping with FPAA flow while also able to synthesize real world analog computing circuits with layout showing 25% area improvement, 64% wirelength improvement 98x and 2.52x speedup in placement and routing respectively.

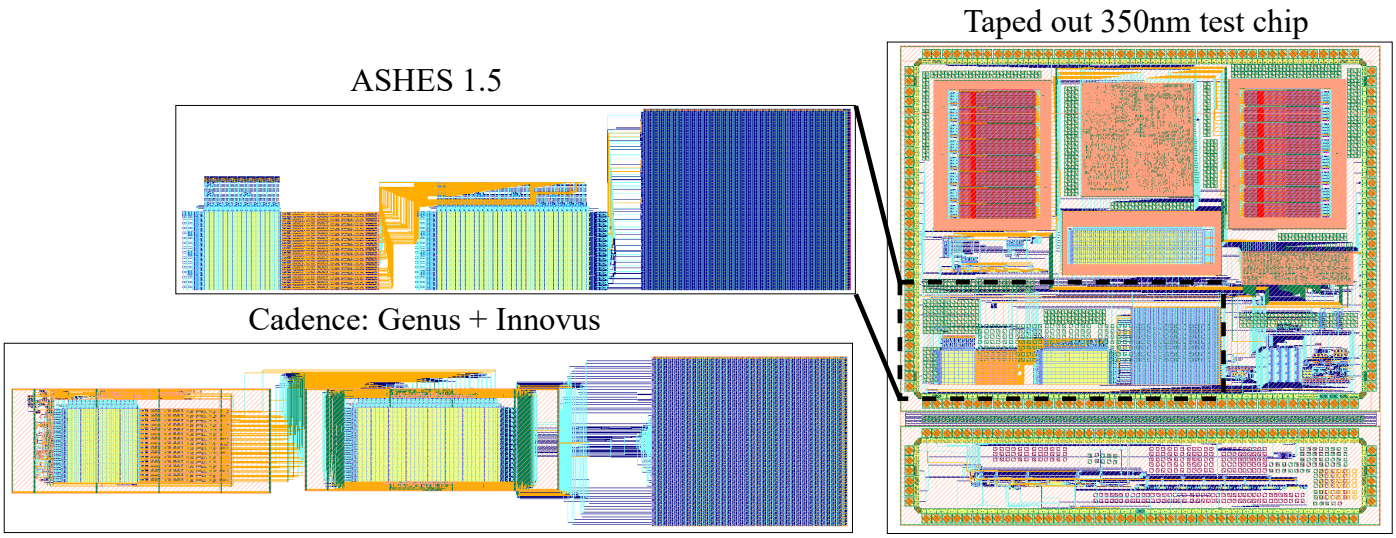
Analog standard cells over a range of CMOS process nodes enables these opportunities. This open-source tool is demonstrated to show multiple synthesized outputs using a 350nm CMOS standard cell library that is similar to the reported 65nm and 130nm CMOS libraries. The tool demonstrated synthesis and routing a high-level formulation through developing an arrangement of standard analog cells that was place and routed to a padframe.

Ashes 1.5 shows an impressive analog synthesis tool, and its development shows the opportunity for further innovations to further improve the capability of this toolset:

**FPAA Fabric Synthesis**: Defining a high-level definition for an FPAA fabric and extending the Ashes tool to synthesize an FPAA would complete the tool capabilities to both create new FPAAs and then target designs on that FPAA.

**Fast and High-level Analog Simulation**: Digital simulation for system development and/or verification happens at many levels; Analog simulation must reach these similar levels for system development and verification. The computational cost of SPICE simulating analog benchmark circuits for a particular implementation could require days or longer on a single machine for one of thousands of instances; analog design space exploration requires nearly intractable simulation resources.

**Demonstration over available analog standard cells**: A demonstration of the scaling and correct operation of Ashes




(a)

| | Area $\mu m^2$ | | | Wirelength $\mu m$ | | | Placement (Routing) time s | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cadence | ASHES | ratio | Cadence | ASHES | ratio | Cadence | ASHES | ratio |
| Arb. waveform gen | 469,560 | 450,065 | 1.04 | 40,149 | 20,490 | 1.96 | - | - | - |
| Universal Classifier | 506,212 | 418,576 | 1.21 | 173,172 | 31,764 | 5.45 | - | - | - |
| Full Speed System | $3.2*10^6$ | $2.4*10^6$ | 1.33 | 375,321 | 134,472 | 2.79 | 52 (1050) | 0.53 (416) | 98(2.52) |

(b)

Fig. 8. (a) Tool synthesis of the speed system measurement that uses a arbitrary waveform generator and VMM+WTA blocks. (b) Experimental result summary.

over the entire range of analog standard cells, including with experimental measurements, showing the scaling capability over a wide range of IC process nodes. When the measurements from at least the four other process nodes are available, this study becomes possible.

These approaches enable moving analog and mixed-signal design towards structured Design Space Exploration (DSE). The Ashes project will remain an open-source tool to enable the wider community to use and contribute to the opportunity of mixed-signal IC system synthesis.

## REFERENCES


[1] B. Parker, "The tide predictions for d-day," *Physics Today*, vol. 64, no. 9, pp. 35–40, 09 2011. [Online]. Available: https://doi.org/10.1063/PT.3.1257

[2] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and Hasler, "A 531 nw/mhz, 128/spl times/32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity," in *Proceedings of the IEEE 2004 Custom Integrated Circuits Conference (IEEE Cat. No.04CH37571)*, 2004, pp. 651–654.

[3] M. Kucic, Hasler, J. Dugger, and D. Anderson, "Programmable and adaptive analog filters using arrays of floating-gate circuits," in *ARVLSI 2001*, 2001, pp. 148–162.

[4] J. Hasler, "Energy-efficient programable analog computing: Analog computing in a standard cmos process," *IEEE Solid-State Circuits Magazine*, vol. 16, no. 4, pp. 32–40, 2024.

[5] S. Shah and J. Hasler, "Low power speech detector on a fpaa," in *ISCAS*, 2017, pp. 1–4.

[6] J. Hasler and S. Shah, "Vmm + wta embedded classifiers learning algorithm implementable on soc fpaa devices," *IEEE JETCAS*, vol. 8, no. 1, pp. 65–76, 2018.

[7] P. O. Mathews and J. O. Hasler, "Physical computing for hopfield networks on a reconfigurable analog ic," in *ISCAS*, 2023, pp. 1–5.

[8] A. Ige and J. Hasler, "Efficient implementation of a fully analog neural network on a reconfigurable platform," in *2023 Midwest Symposium on Circuits and Systems*, 2023.

[9] S. Bhattacharyya, L. Yang, and J. Hasler, "Buzzsort: A linear-time, event-driven data conversion and sorting framework for approximate computing architectures,," in *International Conference on Rebooting Computing (ICRC)*, 2023.

[10] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, W. R, S. Nease, and S. Ramakrishnan, "A programmable and configurable mixed-mode FPAA SoC," *IEEE Transactions on VLSI*, vol. 24, no. 6, pp. 2253–2261, 2016.

[11] M. Shalan and T. Edwards, "Building openlane: A 130nm openroad-based tapeout," in *2020 IEEE/ACM ICCAD*, 2020, pp. 1–6.

[12] T. Dhar, K. Kunal, Y. Li *et al.*, "Align: A system for automating analog layout," *IEEE Design and Test*, vol. 38, no. 2, pp. 8–18, 2021.

[13] H. Chen *et al.*, "Magical: An open- source fully automated analog ic layout system from netlist to gdsii," *IEEE Design and Test*, vol. 38, no. 2, pp. 19–26, 2021.

[14] B. L. Barzen, A. Reais-Parsi, E. Hung, M. Kang, A. Mishchenko, J. W. Greene, and J. Wawrzynek, "Narrowing the synthesis gap: Academic fpga synthesis is catching up with the industry," pp. 1–6, 2023.

[15] J. Hasler, P. R. Ayyappan, A. Ige, and P. O. Mathews, "A 130nm cmos programmable analog standard cell library," *IEEE Circuits and Systems I*, 2024.

[16] P. O. Mathews, P. R. Ayyappan, A. Ige, S. Bhattacharyya, L. .Yang, and J. Hasler, "A 65nm cmos analog programmable standard cell library for mixed-signal computing," *IEEE Transactions on VLSI*, 2024.

[17] J. Hasler, S. Kim, and A. Natarajan, "Enabling energy-efficient physical computing through analog abstraction and IP reuse," *Journal of Low Power Electronics Applications*, vol. 8, no. 4, pp. 1–23, 2018.

[18] M. Collins, J. . Hasler, and S. George, "An open-source toolset enabling analog–digital software codesign," *Journal of Low Power Electronics Applications*, vol. 6, no. 1, pp. 1–15, 2016.

[19] S. Kim, S. Shah, R. Wunderlich, and J. Hasler, "Cad synthesis tools for floating-gate soc fpaas," *Design Automation for Embedded Systems*, 2021.

[20] L. Hanks, C. Lonergan, K. Richardson, J. Hasler, P. Mathews, and A. Ige, "Analog high-level synthesis for field programmable analog arrays," in *2024 IEEE Opportunity Research Scholars Symposium (ORSS)*, 2024, pp. 28–31.

[21] J. Rose, J. Luu *et al.*, "The vtr project: Architecture and cad for fpgas from verilog to routing," in *Proceedings of the ACM/SIGDA FPGA*, ser. FPGA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 7786.

[22] S. Ramakrishnan and J. Hasler, "A compact programmable analog classifier using a vmm + wta network," in *2013 IEEE CASSPI*, 2013, pp. 2538–2542.