Analog High-Level Synthesis for Field Programmable Analog Arrays

Luke Hanks, Cullen Lonergan, Karsten Richardson, Jennifer Hasler, Pranav Mathews, Afolabi Ige
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia, USA

***Abstract*—In this paper, we describe our effort to extend the development of a standard framework for analog computing through further developing and integrating an existing high level synthesis (HLS) tool for analog system design. These Python and Scilab based tools allow designers to design and implement reconfigurable systems on field-programmable analog arrays (FPAA). In doing this, we can provide a way to have the same ease of development that digital integrated circuits (ICs) have with the field-programmable gate-array (FPGA). We describe the importance of analog computing, the state of the old tool flow, our contributions to upgrading the tool flow, and our demonstration of the working tools.**

***Index Terms*—Field-Programmable Analog Array (FPAA), Analog Tools, Analog Synthesis,**

I. Analog Computing and High-Level Synthesis

In the traditional computing space, there are two broad categories of computing devices: digital and analog. Digital computing operates with discrete information and logic. Analog computers use continuous and real-valued mechanical and electrical phenomena (i.e. voltages, currents, etc.) to perform real-time computations. Although analog computing arose first, the development of the transistor led to digital largely dominating due to scalability, cost-effectiveness, and programmability. Widespread adoption of digital systems led to the development of a robust ecosystem that supports digital computing: nearly all programming languages, CAD tools, EDA tools, and other supporting hardware and software efforts are targeted specifically for digital systems. Contrastingly, because analog computing was largely ignored in favor of digital systems, it lacks this standardized framework. For most of computing history, this has not been a significant concern.

However, as the demand for computing power continues to rapidly increase, there has been a resurgence in research efforts and application development in analog computing. Analog computing inherently uses lower power than digital, and certain operations are much more efficient than digital implementations [1]. In particular, hardware implementations of machine learning and AI systems have been demonstrated to have superior power and computational efficiency [2]. These benefits have large implications on the direction computing is taking in general.

To be able to take advantage of these benefits and increase accessibility to analog systems, analog computing necessarily needs a supporting framework of tools and platforms comparable to that of digital. Addressing this need is the purpose

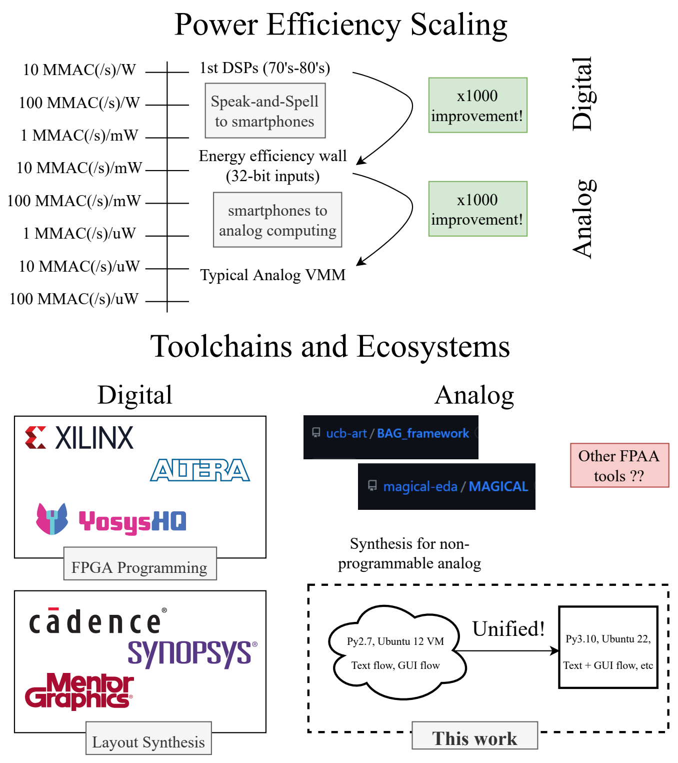


Fig. 1. The power and area efficiency gains of the analog domain motivate the need for analog computing applications. This figure highlights the disparity in ecosystems between the two computing paradigms, and shows how this work contributes to closing the gap.

of this research: to further the effort to create a standardized framework for analog computing. In particular, we focus our efforts on extending a high-level synthesis (HLS) hardware design tool for Field-Programmable Analog Array (FPAA) [4] and application specific integrated circuit (ASIC) applications [3].

II. Field Programmable Analog Arrays

Traditional analog circuit development is a slow process, requiring a multitude of steps from design to testing and verification for each new design or specification. The behavior of analog circuits is much more sensitive to process and design parameters than their digital counterparts, which contributes to these long design times.

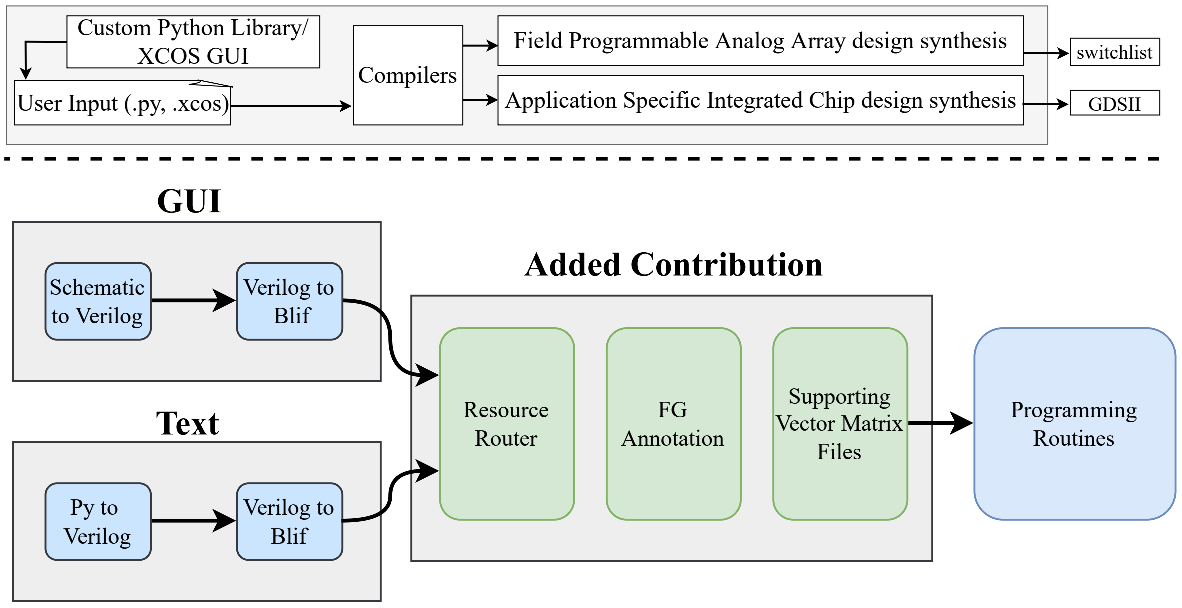
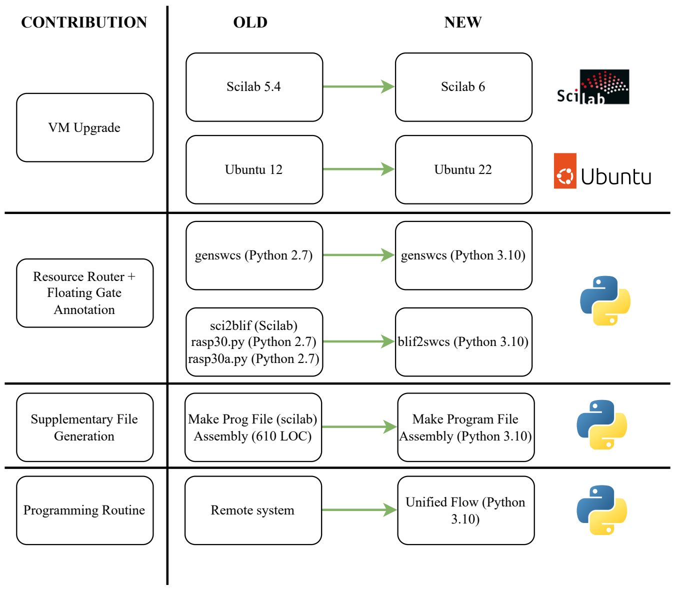


Fig. 2. Top figure: Overview of the compilation flow from user inputs to FPAA and ASIC syntehesis Bottom Figure: Detailed compilation flow illustrating added contribution



CONTRIBUTION	OLD	NEW
VM Upgrade	Scilab 5.4	Scilab 6
	Ubuntu 12	Ubuntu 22
Resource Router + Floating Gate Annotation	genswcs (Python 2.7)	genswcs (Python 3.10)
	sci2blif (Scilab) rasp30.py (Python 2.7) rasp30a.py (Python 2.7)	blif2swcs (Python 3.10)
Supplementary File Generation	Make Prog File (scilab) Assembly (610 LOC)	Make Program File Assembly (Python 3.10)
Programming Routine	Remote system	Unified Flow (Python 3.10)

Fig. 3. Outline of new contributions to the toolset

FPAAs aim to address this problem by allowing rapid prototyping of analog circuits similarly to digital field programmable gate arrays (FPGAs) [5]. The FPAA consists of a fabric of computational analog blocks (CABs) and computational logic blocks (CLBs). The CLBs provide similar programmability to that of blocks on an FPGA while CABs enable analog programmability. Central to the programmability of the CABs on the FPAA is the floating-gate transistor [6]. These devices eliminate a large weakness of analog design: device mismatch. With digital computing, the operations are abstracted to binary logic, so variations in individual device parameters after fabrication does not translate to a large impact on performance. However, because analog computing is not abstracted, mismatch in device threshold voltage, size, and other parameters significantly impacts performance. With floating-gate transistors, through tunnelling and hot-electron injection, a static charge can be programmed on the gate of the transistor that can effectively eliminate the effects of device mismatch [6]. However, not only can we eliminate mismatch, we can program these devices to change the static behavior of the transistors, effectively meaning we can control the behavior of circuits implemented on the FPAA.

This charge programming, along with routing specific devices together, is the method through which analog programmability and reconfigurability is achieved on the FPAA [7]. This process is broadly what the HLS tool performs when programming circuits, and the scripts and programs that achieve this are the focus of the content of the subsequent sections.

III. Upgrades and Additions to the Tool-flow

To begin discussing our work with the tools, we give an overview of the state of the tools before our work. There are two tool flows present on the old system: the Recongfigurable Analog Signal Processing (RASP) Tools, and the Analog Synthesis for High Level Systems (ASHES).

A. Upgrades to the Existing RASP Tools

The RASP tools are a GUI based system that allow the user to design circuits graphically with a drag and drop environment called XCOS, compile them, and program them to an FPAA all within a Scilab environment. At a high level, this flow is described in Figure 2. A Scilab script

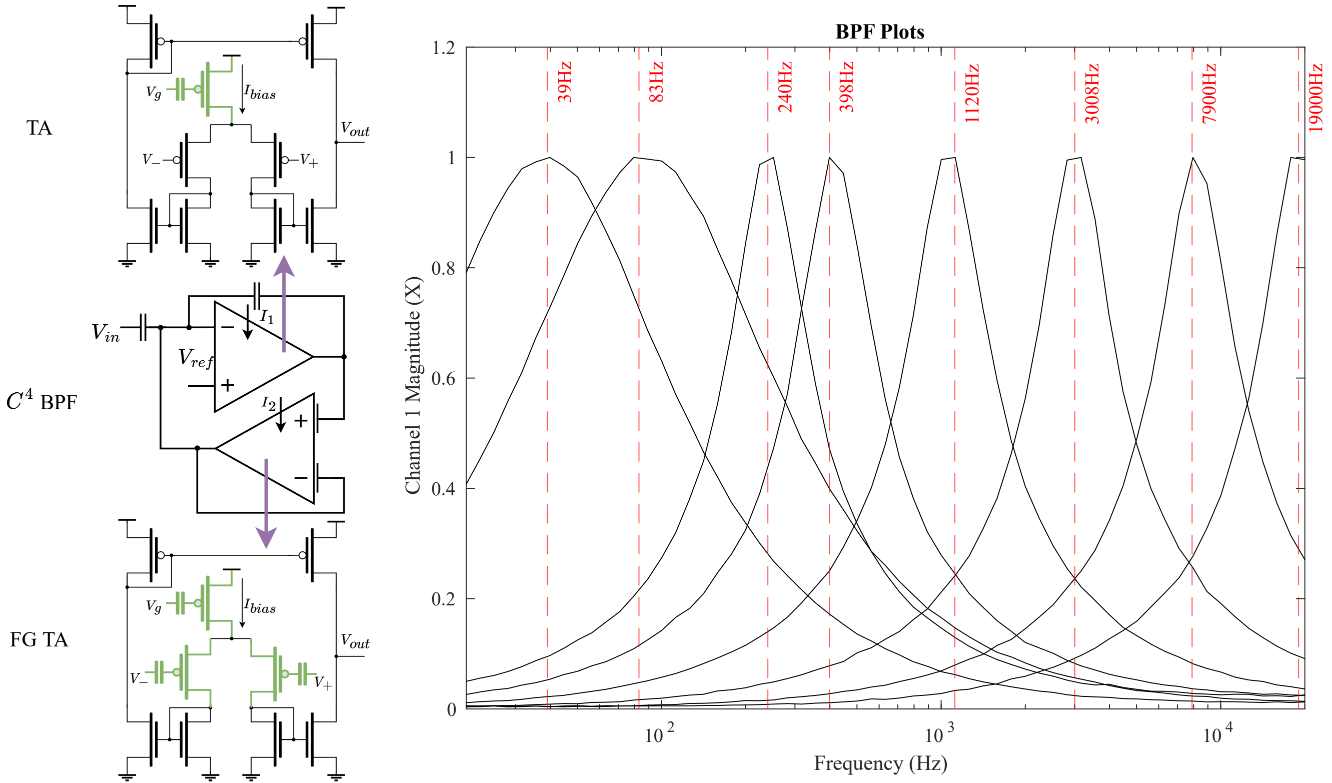


Fig. 4. Left: Schematic of C4 and FGs. Right: Plot of Bandpass filters with exponentially spaced center frequencies.

TABLE I
FILTER BANK BIAS CURRENTS

f_{center} (Hz)	OTA I_{bias}	FG OTA I_{bias}
30	.4nA	4nA
75	.85nA	10nA
189	1nA	5nA
476	1.5nA	8nA
1197	2.5nA	25nA
3008	11nA	60nA
7560	30nA	150nA
19000	94nA	474nA

called sci2blif.sce converts the XCOS representation of the circuit to a .blif netlist file and generates a .pads file, both describing the circuit structure. These are compatible with a CAD tool called Versatile Place and Route (VPR) that is typically for FPGA applications but has been repurposed here for the FPAA. Given an architecture file describing the FPAA, the netlist, and the pads file, VPR determines how to route the FPAA resources to create the circuit. It outputs .place, .route, and .net files. sci2blif.sce then calls a python script genswcs.py that uses the output of VPR to create a switch list that determines what specific switches are used on the FPAA and how they are programmed. Finally, a Scilab script MakeProgramlist_CompileAssembly.sce uses the switch list to create files that interact directly with the microprocessor on the FPAA in assembly to program the circuit. This compilation flow is "Python-wrapped" using Python 2.7.

To begin, the base virtual machine (VM) that hosts all the tools was updated. The VM was initially built on an older deprecated version of Ubuntu (12.04) and was updated to a secure modern LTS version (Ubuntu 22) in an entirely new virtual machine. This update allows for a more secure system due to modern support by Linux, and documentation of the steps taken to update versions allows for easier integration of Ubuntu updates in the future. Scilab, an open-source Matlab alternative that the tools are built around, also needed to be updated when moving VMs. The Scilab version was updated from Scilab 5 to Scilab 6, and syntax changes that differed between the two versions were updated in the code.

After upgrading the VM and Scilab versions and Scilab scripts, the Python 2.7 components of the compilation flow were updated to Python 3.10. This involved updating the genswcs.py, rasp30.py, rasp30a.py files. The rasp30.py and rasp30a.py files are scripts that contain architectural information about the type of FPAA board being used, and this information is used when genswcs.py creates the switch list of the circuit. With these changes, the RASP tools were successfully updated to modern software distributions and programming practices.

B. Extending the Functionality of ASHES

The ASHES tools are a text-based tool flow rather than a GUI based flow, and they are implemented entirely in python. This tool works in conjunction with the RASP tools, but it allows designers a simpler method to design large scale systems hierarchically, increasing versatility. Further, these tools also have the functionality to compile designed circuits down to GDSII files for application specific integrated circuits (ASIC) fabrication. However, our focus is on the FPAA flow. For the FPAA flow, ASHES functionality was implemented up to the generation of the .blif netlist describing the circuit. Thus, the goal of our work was to implement the functionality described in the sci2blif.sce file including creating scripts to call VPR, generate the switchlist, create the compilation files to interact with the microprocessor, and program the circuit onto the FPAA.

The text-based tools were built on a Python 3 framework, eliminating the need for Scilab scripts entirely. The existing text-based toolchain compiled the text description down to the blif file format. Given a python input describing the circuit, the file new_converter.py converts the input to verilog description of the circuit. The file verilog2blif.py then converts the verilog file to a .blif file.

To extend this functionality to be able to program circuits onto the FPAA, scripts that received the .blif file and performed the processing through programming were written in Python 3. The file blif2swcs.py was created that encapsulated a majority of the functionality contained in sci2blif.sce from the RASP tools. Given a .blif input, this file performed the necessary resource routing through VPR and floating gate annotation steps to generate a switch list. To facilitate integrating the two tool flows, we hierarchically call the same genswcs.py and incorporate the architectural function rasp30.py and rasp30a.py.

The completion of the text-based toolchain required that the tool be capable of programming the FPAA directly without the use of the graphical Scilab tools. The assembly compilation of the switchlist requires a multitude of helper files to be generated inside a hidden directory upon each compilation. The implementation of this generation was started with promising results. MakeProgramlilst_CompileAssembly.py is a compilation script for the text-based tools that generates these necessary helper files for the final compilation. This file has been shown to correctly generate most of the required files, but these files have not yet been verified. The programming step using a script labeled program_fpaa.py has been demonstrated to have full functionality using the files generated from the GUI version of the programming interface.

IV. Toolchain Verification

With the tool upgrades and additions successfully complete, we need to demonstrate and verify the functionality of the new tool set. We chose to design, implement, and test a frequency decomposition system on the FPAA. This system can be viewed as an analog version of the fast fourier transform (FFT) algorithm that digital systems use. Unlike the FFT, this system performs decomposition on the input signal directly and in real time. To implement this, we designed an exponentially spaced filter bank of bandpass filters using the capacitively-coupled-current-conveyor (C^4) topology pictured in Figure 4 labelled C^4 BPF.

The C^4 consists of one regular operational transconductance amplifier (OTA), one FG OTA, and an input and feedback capacitor. The OTA and FG OTA transistor level schematics are pictured in Figure 4, labelled TA and FG TA respectively. Each of these OTAs has a bias current that is programmable through the tail FG PMOS highlighted in green on the schematics, and through tuning each of these bias currents we are able to alter both the center frequency and the Q factor of the bandpass response. Further, the FG OTA has FG inputs on its differential pair highlighted in green on the schematic. These allow for the elimination of effects of mismatch on the circuit.

To create the bank, we tuned eight filters with exponentially spaced center frequencies. We chose frequencies that are within the human hearing range of 20Hz to 20kHz. By tuning these filters with a high Q value, and thus a large peak in the frequency response, the filters effectively select the frequency components of the input signal that are present. The current values and associated center frequencies are listed in Table I. These results were obtained entirely through the new tools, particularly the RASP tools, and demonstrate the full functionality of the toolchain.

V. Conclusion

This research consists of a two phase project with the goal of upgrading, integrating, and demonstrating an existing high-level synthesis tool flow for analog design. This tool, being the first of its kind, was in the nascent stages of development, and through our work we were able to make significant progress towards making a complete, robust design tool.

References

[1] J. Hasler, ”The Rise of SoC FPAA Devices,” 2022 IEEE Custom Integrated Circuits Conference (CICC), Newport Beach, CA, USA, 2022, pp. 1-8, doi: 10.1109/CICC53496.2022.9772732.

[2] J. Hasler, ”Opportunities in physical computing driven by analog realization,” 2016 IEEE International Conference on Rebooting Computing (ICRC), San Diego, CA, USA, 2016, pp. 1-8, doi: 10.1109/ICRC.2016.7738680.

[3] Ige, Afolabi, Linhao Yang, Hang Yang, Jennifer Hasler, and Cong Hao. 2023. ”Analog System High-Level Synthesis for Energy-Efficient Reconfigurable Computing” Journal of Low Power Electronics and Applications 13, no. 4: 58. https://doi.org/10.3390/jlpea13040058

[4] Kim, S., Shah, S., Wunderlich, R. et al. CAD synthesis tools for floating-gate SoC FPAAs. Des Autom Embed Syst 25, 161–176 (2021). https://doi.org/10.1007/s10617-021-09247-9

[5] J. Hasler, "Large-Scale Field-Programmable Analog Arrays," Proceed ings of the IEEE, Aug. 2020.

[6] P. Hasler, C. Diorio, B. Minch, and C. Mead, "Single Transistor Learning Synapses," in Advances in Neural Information Processing Systems, 1994.

[7] M. Collins, J. Hasler, and S. George, "An Open-Source Tool Set Enabling Analog-Digital-Software Co-Design," Journal of Low Power Electronics and Applications, Feb. 2016