Design Automation for Embedded Systems (2021) 25:161–176
https://doi.org/10.1007/s10617-021-09247-9

CAD synthesis tools for floating-gate SoC FPAAs

Sihwan Kim[1] · Sahil Shah[1] · Richard Wunderlich[1] · Jennifer Hasler[1]

Received: 28 November 2017 / Accepted: 8 March 2021 / Published online: 22 March 2021

Abstract
We present a tool framework to compile and program mixed-signal circuits and systems on Floating-Gate (FG) based mixed-signal System-on-Chips (SoC) consisting of a digital processor and Field Programmable Analog Array (FPAA) fabric. We have modified the configuration of Verilog-to-Routing (VTR) to cover analog circuits and developed a tool called vpr2swcs to create the list of FG switches, that is going from a high level block description of the system to the addresses and bias values on the SoC. This tool enables users to generate macro blocks and customize block location while designing mixed-signal systems on the FPAA and also enables using routing fabric, composed of FGs, for Vector Matrix Multiplication (VMM), a computing element for an analog neural network. The paper demonstrates system level examples using this tool flow, where the experimental results have been proved in other publications.

Keywords Floating-Gate · FPAA · CAD synthesis · vpr2swc

1 Analog-digital mixed system design

Automated or semi-automated Computer-Aided Design (CAD) tools have played a decisive role in the development and design of digital and mixed-signal systems. In digital systems such as Field-Programmable Gate Array (FPGA) [1], automated synthesis tools convert a Hardware Description Language (HDL) design written by a system designer into a routing and logic information [2,3]. This helps system designers to focus on various applications such as image processing [4] or deep neural networks [5,6]. In analog systems, semi-automated design tools (e.g. Cadence Virtuoso) provide numerous functions to help users to design, simulate, and verify the circuits. Although CAD tools in digital and analog systems have achieved remarkable progress for decades in each area, the application designer still does not have a unified system for digital-analog design flow; it is required to define digital and analog parts at the first stage, test each custom IC/FPGA, and integrate two parts at the system level in Fig. 1.

✉ Jennifer Hasler
jennifer.hasler@ece.gatech.edu

Sihwan Kim
k.sihwan@gmail.com

[1] School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, USA

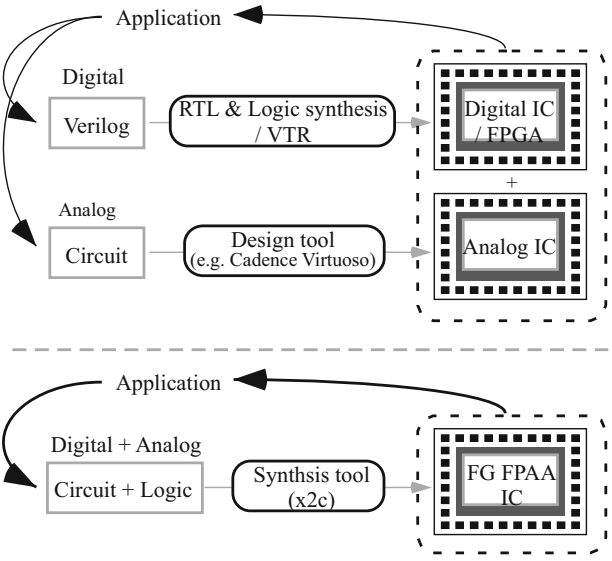

Fig. 1 Comparison of two different hardware implementation flows. To implement an application (e.g. speech recognition), system designers have been taking a traditional approach to separate the algorithm into digital and analog parts, which results in using different tools and requiring efforts on digital/analog interface after testing each custom IC. Floating-Gate (FG) FPAA system provides a mixed-signal design and test flow on FPAA ICs, which is enabled by a synthesis tool, "x2c" meaning Xcos to Chip, compiling the design into necessary hardware files (e.g. switch list)

As a large-scale Field Programmable Analog Array (FPAA) [7] to provide a unified mixed-signal system, Floating-Gate (FG) System-on-Chip (SoC) FPAAs, including analog-digital mixed arrays and FG programming infrastructure with micro-processor and SRAM [8], have been developed. Tools for graphical high-level design environment [9] and FG programming algorithms to precisely target FGs [10] have been integrated for seamless connection to the FPAA.

The focus of this work is on CAD tools to bridge the gap between the high-level user design and the hardware, which are essential for FPAA productive use and development since individuals could only go as far as the tools are capable. The first motivation for the tools is to use existing, working FPAA devices. Devices have been characterized [11], used in classes by multiple students [12] and numerous collaborators [13]. The compilation tools generating a switch list for the FG programming are based on Versatile Place and Route (VPR) due to its capability as well as a possibility of improvements and contribution from a wider CAD community. A second motivation is to explore different FPAA architectures. This paper describes the CAD tool developed for compiling mixed signal systems on an FPAA. Though the tools are generic to be used with different architecture of an FPAA or an FPGA in this work we have used a large-scale FG FPAA [8] architecture.

This works illustrates the hardware configuration of FG SoC FPAA in Sect. 2 and compare compilation tools for FPGAs, FPAAs, and FG FPAAs in Sect. 3. Section 4 introduces a compilation flow with new CAD tools and solutions for the challenges on applying a digital design tool to analog design. We will introduce advanced tools and Vector Matrix Multiplication (VMM) blocks for supporting high-level system designs in Sect. 5 and the system compilation examples in Sect. 6. Section 7 includes the conclusion and discussion.

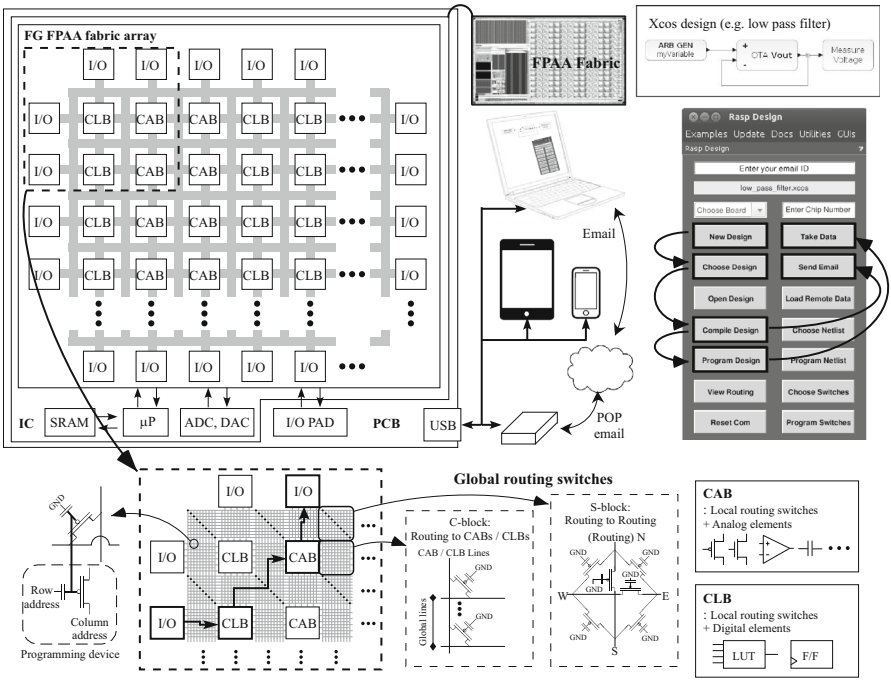

Fig. 2 FG SoC FPAA system. FPAA fabric array consisting of Computational Analog Blocks (CAB), Computational Logic Blocks (CLB), Input/Output blocks (I/O), and Connection (C)/Switch (S) blocks for routing, interfaces with μP, DACs, and ADCs on IC and I/O Pads on PCB. Non-volatile and reconfigurable FG devices are employed for connections in routing fabric, as well as configuration of analog/digital elements (e.g. bias current, logic table). An open-source design environment for mixed-signal system in Xcos/scilab is provided with GUI

2 System design with reconfigurable hardware

The large-scale FG FPAA enables reconfigurable digital-analog system design, as well as analog computation with power consumption at μW level (e.g., Analog VMM calculation using low current in nA range). The 1000× energy efficiency improvement [14,15] compared to digital solutions empowers a whole range of low-power embedded applications such as always-ON context-aware processors [8], acoustics [16], vision, and robotics [17]. This section introduces the configuration of the FG SoC FPAA and the compilation tool flow.

Figure 2 shows the architecture of an FPAA array, a chip photo [8] including FPAA array, an Xcos design example with GUI, and the external interfacing devices available for programming them. The FPAA fabric array consists of Computational Analog Blocks (CAB), Computational Logic Blocks (CLB), Input and Output (I/O) blocks, and routing switches for connecting them. Certain application requires analog or digital processing whereas some require seamless integration of both analog and digital components (e.g., classifiers, Analog to Digital Converters (ADC), Digital to Analog Converters (DAC), etc.) and the FPAA architecture enables this by having both CABs and CLBs.

CAB and CLB consist of FG-based analog and digital elements with local intra-block routing fabric. I/O blocks provide access to μP, DACs, and ADCs on the IC for built-in self test as well as I/O Pads on Printed Circuit Board (PCB) for applying or measuring the voltage

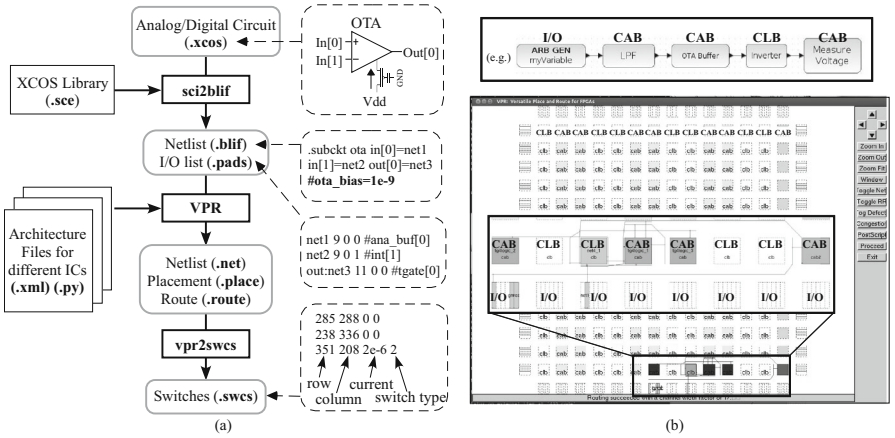

Fig. 3 *x2c* compilation flow. **a** Compilation tools and the resulting files. *sci2blif* converts graphical design file (.xcos) into blif netlist file (.blif) and I/O list (.pads). *VPR* places the blocks and calculates necessary tracks for global routing. *vpr2swc* maps the FG addresses based on the place and route information. **b** Snapshot of *VPR* usage in an example. *VPR* provides a graphical interface, where a user can check the place and route results visually

or current externally. Non-volatile and reconfigurable FG devices form the routing fabric, where the FG node is shared with the programming pFET device indirectly. Global routing structure for inter-block connections consists of Connection (C) blocks connecting global lines to local lines in CAB, CLB and Switch (S) blocks connecting between bidirectional global lines.

The FG SoC FPAA interfaces with user systems such as laptop, tablet, or smartphone through USB, which also supplies the system power for the IC and its infrastructure on the PCB. PCB includes power management components such as charge pumps, and voltage regulators and interfacing chip between USB and μP.

We provide a mixed-signal system design environment using Xcos in Scilab [18] and a Graphical User Interface (GUI) for compiling and testing the system, which are based on open-source codes. On the GUI, "New Design" starts a new application design, "Compile Design" creates necessary files for programming FG devices and built-in self test of the system. "Program Design" sends programming files to the USB-connected FG FPAA IC and programs FG devices. "Take Data" tests the system and shows measured output data. For the users who do not have access to FG FPAA ICs, "Send Email" enables testing of the design by compiling it to a remote system via a simple PoP protocol [13].

3 Tools for FPGAs, FPAAs, FG FPAAs

This section introduces different CAD tools used for FPGAs and FPAAs and tool requirements for FG FPAAs.

FPGAs have look-up tables as a basic unit which has enabled reconfigurable and reprogrammable digital system design post-fabrication of ICs, for over three decades based on CAD tools' support. Verilog-to-Routing (VTR) [2,19] is one of the open-source tools for FPGAs, where ODIN II transforms a given digital circuit described in a Verilog code to a Berkeley Logic Interchange Format (BLIF) [20] netlist, ABC optimizes BLIF netlist by syn-

thesizing logic and performing technology mapping, and Versatile Place and Route (*VPR*) maps the BLIF to the placement of CLBs and routing track configuration on the FPGA architecture.

Similarly, reconfigurable analog CAD tools for CAB-based FPAAs have been proposed [21,22]. A tool called Generic Reconfigurable Array Specification and Programming Environment (GRASPER) [22] is a solution for an automated place and route in FPAA systems. This tool takes a SPICE netlist as an input, places analog circuits based on Modified Hyperedge Coarsening (MHEC) order of cells [23], and generates optimized switches.

In case of FG SoC FPAA one has to handle both CAB and CLB with an integrated processor; it has been essential to develop a new automated design flow enabling mixed-signal applications, as well as for it to be an open-source software for wider adoption and development of such tools. Although one might consider a modified version of GRASPER as an option, the tool has limitations in scalability on different hardware architectures and capability to cover digital circuits. Also, the high-level graphical interface of GRASPER is based on MATLAB Simulink, which constrains the outreach of the tool set. One might consider utilizing Verilog and extending ODIN II but the compilation tool needs to fit with the high level design tool, *sci2blif* [9], provide a graphical design environment, and convert the design to a BLIF file directly.

4 CAD tools for mixed mode design

We propose a tool set to provide a high-level graphical design environment in Xcos/Scilab. Our tool compiles a system to a switch list, which is transferred to the FG SoC FPAA and programmed for the application, while handling heterogeneous elements for mixed-signal circuits in different hardware architectures, as well as relying on a completely open-source code. Figure 3a shows the proposed compilation flow integrating three different open-source tools, *sci2blif* [9], *VPR* [19], and *vpr2swc*. In the next subsections, we summarize *sci2blif* and *VPR* and the following subsection focuses on *vpr2swc* developed in this work and solutions to challenges caused by using VPR for analog design.

4.1 sci2blif: Xcos → blif

Xcos is a graphical system design environment in Scilab, which is an open-source software with similar functions to Matlab. A user designs a system in Xcos by dragging blocks from a palette browser, connecting blocks' inputs and outputs, and setting parameters such as bias current or DC voltage conditions. Analog and digital elements and basic functional blocks (e.g. I/O, ADC, DAC, LPF, etc.) are predefined in the Xcos library, also user-defined functional blocks can be added to the library through a macro block generation tool.

sci2blif converts block level information in Xcos into a blif netlist and a pad file, which are inputs to *VPR*. The blif netlist has a description of the block, connections, and parameters. Each FG parameter in a block is described with "♯" and "&." A pad file includes necessary input/output information and connected I/O blocks.

sci2blif also builds a file set to be transferred to SRAM and generates assembly codes to be executed by μP. For example, the tool converts the user-defined input vector to a HEX file when it compiles "ARB GEN" block, which is a DAC applying voltage values stored in SRAM at a given frequency. "Measure Voltage" block, an ADC using FG device and programming infrastructure in the IC, requires the tool creating a specific assembly code interfacing with

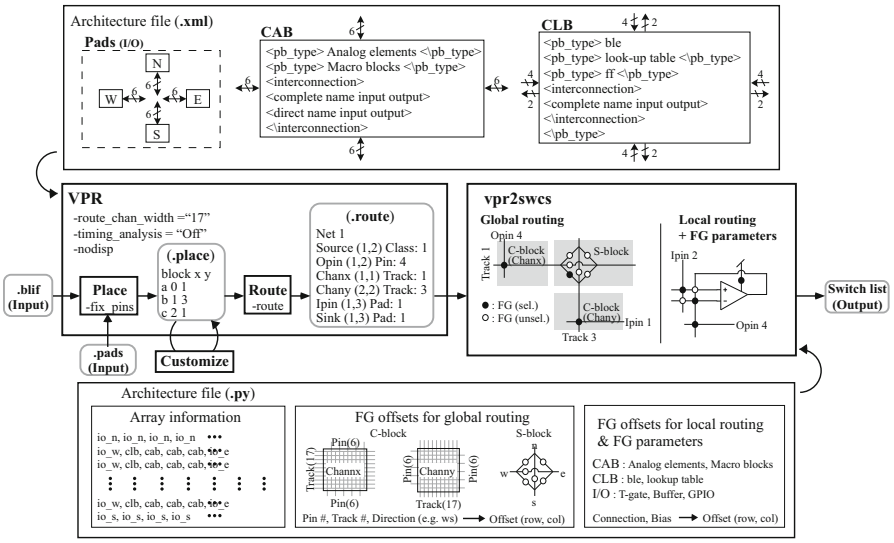

Fig. 4 Detailed tool flow and configuration of *VPR* and *vpr2swc*. Architecture file (.xml) for *VPR* requires new definitions for analog elements and macro blocks in CAB and modification of CLB and I/O blocks corresponding to the hardware. *vpr2swc* creates a switch list of global routing, local routing, and FG parameters for analog and digital devices based on the architecture file (.py) including array information, offsets, and FG parameters

μP for data acquisition. The structure and compilation of assembly language modules are beyond the scope of this discussion and will be presented elsewhere.

4.2 VPR: blif → route

We utilize *VPR* tool to calculate optimized place of blocks. Figure 3b shows a graphical result of *VPR* for a mixed-signal system. The *VPR* architecture file is customized to include CABs, as well as CLBs, in the fabric array. The global routing structure is directly applied to *VPR*, where S-blocks switch the direction of routing (North, South, East, West).

Figure 4 shows how to configure the *VPR* architecture file and specify the tool options in our compilation flow. Architecture file with ".xml" extension requires definition of array blocks, which are I/O pad blocks, CABs, and CLBs. North, South, East, and West I/O pad blocks have 6 pins connecting the array. CAB uses <pb_type> tag, to specify the properties of a complex block, for defining analog elements and macro-blocks consisting of analog elements. CAB includes the definition of 24 pins (6 pins in each direction), which can be assigned to input or output. <Interconnection> tag maps inputs and outputs of each block to CAB's pins. <complete> tag connecting input/output to any pin at the output/input is used for general blocks, while <direct> tag is used for specified blocks such as Vector-Matrix Multiply (VMM). Similarly in CLB, Look-Up-Tables (LUT) and Flip-Flops (FF) in BLE are defined as a complex block with the <pb_type> tag, the <complete> tag connects inputs and outputs of blocks to CLB pins. CLB has 16 input and 8 output pins (4 input and 2 output pins to each direction).

Based on the architecture file, *VPR* places and routes the elements from blif netlist and pads information. The use of "-fix_pins" option locks each I/O pad to a desired location

listed in the pads file and results in the output file (*.place*) which includes details such as block name and locations. In certain applications one can customize the *.place* file where the system designer can assign a specific location for a block in the Xcos design. The option "-route" invokes this functionality to create a route file (*.route*) including switch/source block locations and track numbers.

VPR runs with a default options of "-nodisp" which hides the graphical interface, "-route_chan_width = 17" indicating the number of tracks, "timing_analysis = Off" turning off the timing analysis while performing global routing. Although the global routing optimization in this paper is based on congestion information, which minimizes the parasitic capacitance, a timing driven optimization can be integrated into this system by measuring and modeling line resistance and capacitance [24].

4.3 vpr2swc: route to switch list

As the final step of the compilation process, a *vpr2swc* code developed in python calculates FG addresses and creates a switch list. Figure 4 shows the configuration of *vpr2swc* architecture file and how to map global and local routing elements and FG devices to their physical addresses.

The architecture file (*.py*) includes information on physical arrangement of the array and offsets for FG devices, which varies according to different FG FPAA IC architecture. Row and column addresses of CAB, CLB, I/O blocks are described in the array information. The *.py* file defines FG offsets for global routing, where Chanx in C-block connects horizontal tracts to vertical pins, Chany in C-block connects vertical tracks to horizontal pins, and the tracks intersect in S-block, as well as local routing and analog/digital/I/O devices in blocks.

vpr2swc reads and parses routing file to get necessary information for global routing. The routing of each net begins on a Opin (a certain output pin), goes through Chanx and Chany, and ends on a Ipin (a certain input pin). (x,y) location and pin/track numbers of each channel are described at each line. Based on pin, track, and FG offsets defined in the architecture file, the tool creates C-block FG switch addresses from pin number in Opin/Ipin and track number in Chanx/Chany and S-block FG switch addresses from track numbers in two adjacent Chanx or Chany.

A list of local switches and FG parameters in CAB/CLB/I/O blocks, handled by *VPR* as black boxes, is generated based on the FG offsets defined in the architecture file and block location information assigned by the placement. *vpr2swc* parses the placement file and integrates circuit parameter information described in blif netlist.

Figure 5 shows detailed configuration of blocks, including local routing and elements in the recent FG FPAA IC [8]. Local switches in *routing map* enables connection of local lines, which are inputs/outputs of block elements, to global lines from C-block, to the power rails gnd/V_{dd}, as well as interconnection between local lines. Each FG switch in the *routing map* corresponds to an *FG address* which shares the same FG node and has a row and column address.

Analog elements in CAB include two Operational Transconductance Amplifiers (OTA) using a FG device for bias current, two OTAs using FG devices for the input transistors and a bias current, four capacitors using FG devices for selecting capacitor sizes, two nFETs, two pFETs, four Transmission gates (T-gate), and two N-mirrors. Digital elements in CLB are comprised of eight Basic Logic Elements (BLEs) made of 4-input Look-Up Tables (LUT), a Flip-Flop (FF), and FG switches for the logic configuration. A FG switch is set to program the user-defined logic and is embedded in the LUT. The FG switch also enables a sequential

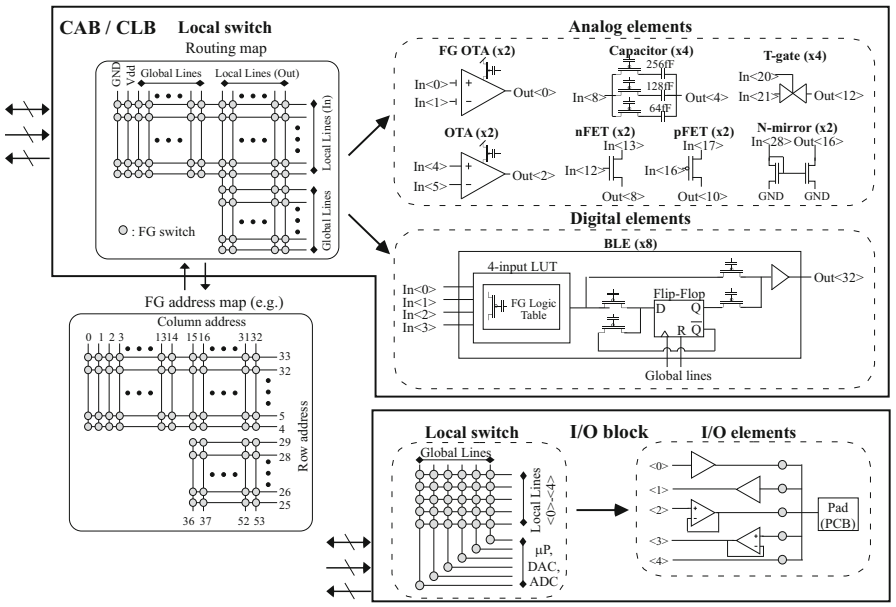

Fig. 5 FG mapping for local routing and analog/digital/I/O elements. *vpr2swc* calculates row and column address from *FG address map* sharing FG node with each corresponding FG device in the *routing map*. Local switch matrix in the *routing map* connects global routing lines, gnd, and V_{dd} to input/output of analog/digital/I/O elements. Local switches in I/O block connect global routing lines to I/O elements as well as DACs, ADCs, μP. *vpr2swc* adds FG device addresses in the elements (e.g. OTA, FG LUT, I/O buffers) into the switch list

or combinational logic based on whether the output is either routed through a FF or not respectively. Local switches of I/O block enable connection to pad on PCB as well as DAC, ADC, General-Purpose Input/Output (GPIO) that interface with the μP. A user could also route the output via a digital or analog buffer, or choose an unbuffered pad.

4.4 Challenges on applying VPR to Heterogeneous systems

Since *VPR* and blif netlist have been developed for the description of logic gates in FPGAs, extending it to heterogeneous systems is accompanied by two big challenges, listed in Fig. 6. One challenge is to handle the input/output directionality of analog circuits. Logic circuits, e.g. AND gate, function with explicitly defined input and output ports. On the other hand, the ports of analog circuits are close to a concept of bus, which requires bidirectional definition depending on the application. As an example a Shift Register (SR) block could be configured in multiple ways. A user could compile a shift register having either 16 inputs/1 output or 1 inputs/16 inputs for a variety of usages. We modified CAB definition in *VPR* architecture file to map each physical port to have either input or output, which enables bus concept of input/output for analog blocks.

Another challenge is conflict arising from connection of multiple outputs. *VPR* does not allow multiple drivers for a single net since it is regarded as a logical error in digital circuits. In analog circuits, however, multiple outputs on a single node is important for its functionality. For example, a voltage divider using two OTAs requires combining two outputs of OTAs to a single node, a Winner-Take-All (WTA) circuit has an architecture which requires a common

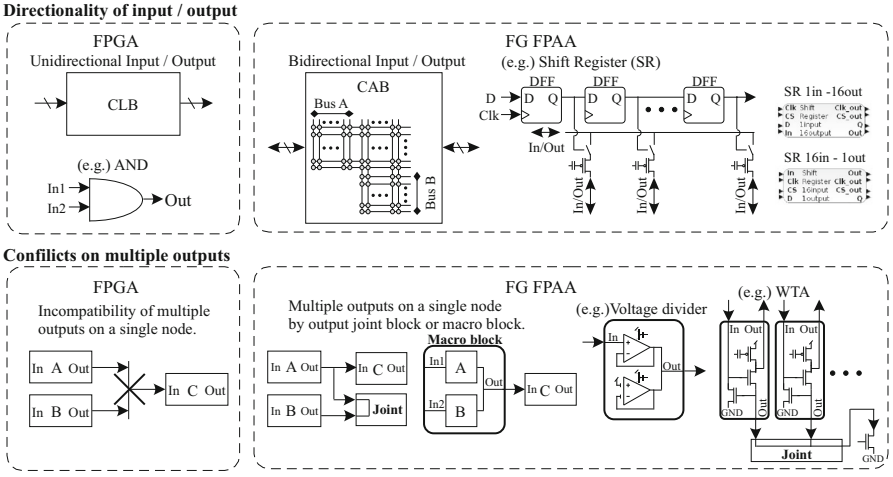

Fig. 6 Challenges on applying *VPR* to analog system. **Directionality of input/output:** *VPR* originally designed for FPGA handles element blocks with explicitly defined inputs and outputs, which functions on logic circuits (e.g. AND gate). On the other hand, input or output of analog circuits are required to be defined bidirectionally. The architecture file of FG FPAA defines global lines in CAB as bus A/B, which allows analog blocks to have different definition on the same global line depending on the application. The example of Shift Register (SR) shows two blocks defined differently with same bus. **Conflicts on multiple outputs:** Although *VPR* does not provide multiple drivers for a single net, it is a required function for analog circuits, shown in the examples of voltage divider or WTA. We enable this functionality by using output joint blocks in global routing or generating macro block with local routing

current bias and hence multiple outputs of the WTA have to be connected to an input. A macro block approach provides a solution by routing locally inside the CAB, encapsulating complex circuits with interconnections on a node for multiple outputs. As a global routing level solution, we provide a joint block allowing multiple inputs to be driven by an output, which is compatible with *VPR*.

5 Advanced design tools and VMM

The tools introduced so far enabled compilation of the user's design by creating a switch list. In this section, we introduce advanced tools for generating macro block and customizing block's location to support complicated and specific system design, as well as designing Vector-Matrix Multiply (VMM) blocks to enable power and area efficient computation by using routing FG devices. Figure 7 shows the whole compilation flow integrating two advanced tools.

5.1 Macro block: encapsulating complex circuits

The tool abstracts complex mixed signal circuits to a high-level block by two different methods of encapsulation as illustrated in Fig. 8. The first is macro-blif block integrating circuits into a blif netlist. The tool extracts the inputs, outputs, interconnections, and FG parameters from the original circuit design and adds a compilation description of the macro block to the Xcos library. For internal nets, unique net names using the user-defined macro

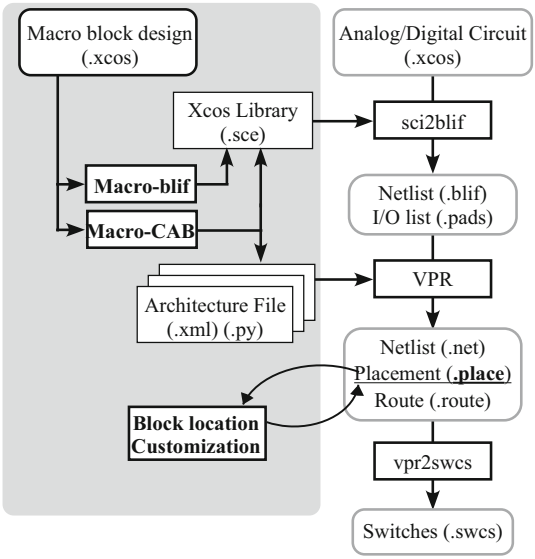

Fig. 7 Compilation flow with advanced tools. A tool generating a user-defined macro block encapsulates a complex analog circuits into a high-level block in Xcos library and architecture files. The tool also customizes the location of the block which could be specified by a user in the Xcos design and used as a parameter by the placer

block name are assigned to avoid errors which may arise from overlapped net names in a blif file. The second is macro-CAB block integrating circuits into FG switches in a CAB. The macro-CAB block generation tool provides a Xcos file where analog elements and interconnection FG switches in CAB are predefined, which includes mapping information of each FG address corresponding to each FG device. A new user-defined macro-CAB block is designed by modifying the provided Xcos file. FG devices are used for interconnecting analog element by connecting their inputs to outputs or gnd/V_{dd}, as well as for setting the bias value of a circuit, for example a bias of an OTA, by specifying a targeted current. After the inputs, outputs, and names of parameters with default values are set by the user in the Xcos design, the tool generates a macro-CAB block, which encapsulates the users design, and its necessary files to add Xcos library and architecture files for *VPR* and *vpr2swc*.

Macro-CAB block enables compact design, which means efficient area and lower parasitic capacitors, using the limited number of analog elements in a CAB. On the other hand, macro-blif block is not limited by the number of elements in the CAB and hence is suitable for system-level design.

5.2 Customization of block location

Customizing the placement of blocks in a specific CAB in the FPAA fabric is required for certain applications. For example, a ramp ADC calibrated with a CAB needs to be compiled at a fixed location, since the slope of the ramp changes depending on the capacitor mismatch and biasing current. A Gm-C filter which is sensitive to parasitic node capacitance is also an example, where we can calculate routing capacitance based on [8] when the block location is fixed, and set the filter parameters.

For Customizing the placement of blocks, a user sets the location of the block in the Xcos design by changing a block parameter, “Fix_location”. The tool searches the block name in the placement (having a suffix .place) file and swaps the location for the defined value.

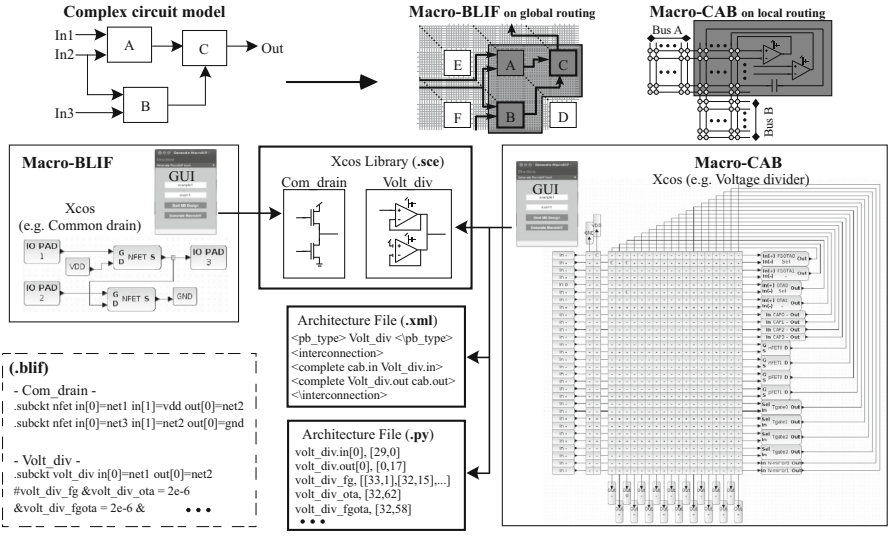

Fig. 8 Automated generation tool for macro blocks integrating a complex circuit model into a single block. A tool for macro-blif block encapsulating the circuit on global routing adds the compilation description to Xcos library. Another tool for macro-CAB block encapsulating the circuit on local routing adds the information to Xcos library and architecture files for VPR and *vpr2swc*

5.3 Analog VMM: computation with routing

Vector-Matrix Multiply (VMM) block is a core component for variety of signal processing and machine learning algorithms, performing a multiply operation between a vector inputs and a matrix of weights trained. FG devices on routing nodes, storing the weight and converting a voltage input linearly into a current, enable a power-efficient and compact implementation of VMMs. Figure 9a shows examples of the application, an analog classifier combining VMM with a WTA [25] and an image convolution combining VMM with shift register and integrator [26].

Figure 9b illustrates an example of 8×8 VMM implementation with a macro-CAB block in local routing. In local routing fabric, FG devices connected to inputs (V_{in}) are programmed to a target current level corresponding to each weight value. The converted currents in a row are summed to each output (I_{out}) through a programmed switch. To pair the weight matrix with dedicated FG switches in the local routing, the inputs/outputs of VMM block are assigned to fixed global lines by using "direct" option in the architecture file.

Based on the VMM blocks, which are special type of macro-CAB blocks, it is easy to extend a VMM block with large number of input/output by using macro-blif block. An example of 16×16 VMM with shift register is shown in Fig. 9c. The description in the architecture file for blif file includes four of 8×8 VMMs and a shift register.

6 System examples

Table 1 shows several system design examples based on the compilation using the proposed tools in this work and implemented in our FPAA SoC [8]. The table includes number of blocks in the placement file and number of FG devices created in the switch list. In this section,

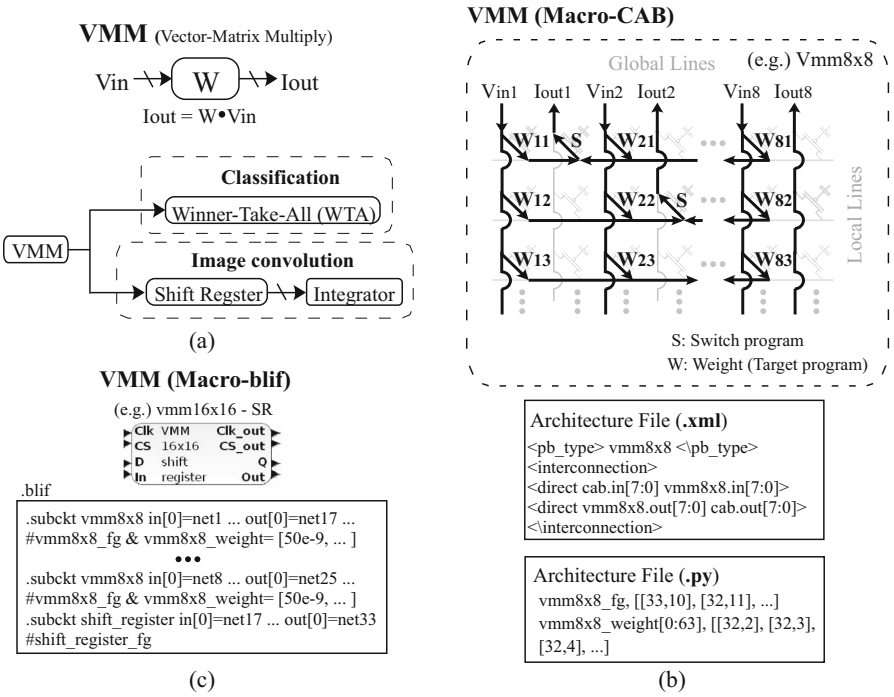

Fig. 9 Vector-Matrix Multiply (VMM). **a** Applications using VMM. **b** VMM macro-CAB block conducting a multiply operation between a input voltage vector and a weight matrix stored in the FG by using local routing FG devices. **c** A large number of input/output VMM using a macro-blif block

Table 1 System compilation examples

	No. of blocks			No. of FGs	Refs.
	CAB	CLB	I/O		
DAC+Com_drain+ADC	2	0	2	40	
DAC+Volt_div+ADC	2	0	2	44	
$\overline{A}\overline{B}C + AB\overline{C}$	1	1	4	63	[8]
DAC+LPF+ADC	2	0	1	39	[13]
Universal approximator	7	0	4	222	[9]
Speech processing	5	0	3	131	[27]
Speech classifier	19	0	5	995	[8]

we illustrate three different systems built using a low pass filter, universal approximator, and a speech classifier as a complex system design example using macro blocks, where each functionality of the system has been proved with experimental data [8,9,13,27].

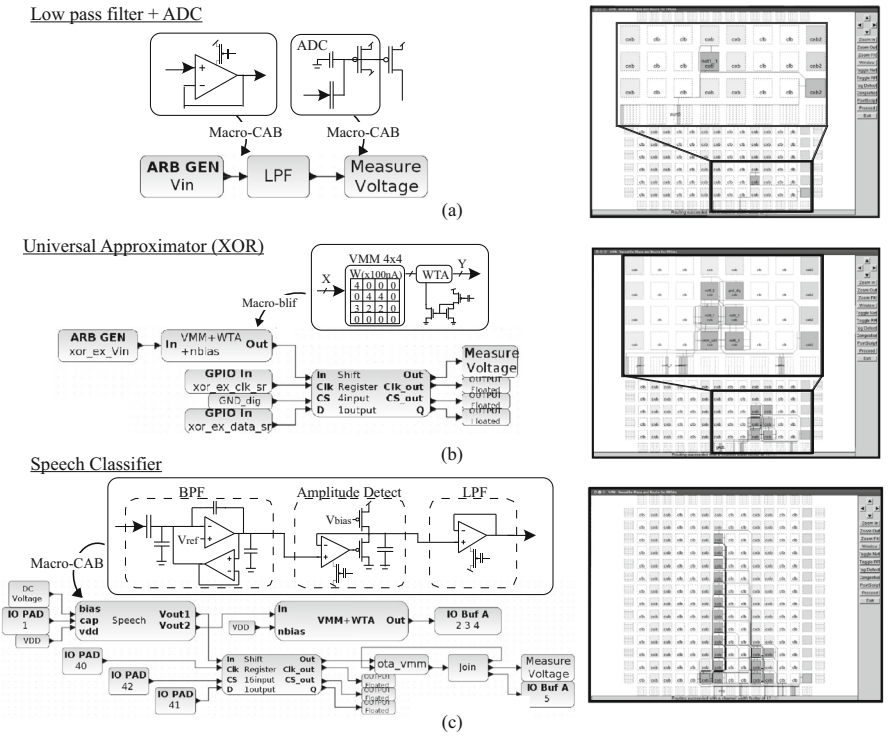

Fig. 10 System examples using FG SoC FPAA. **a** DAC + Low-Pass Filter (LPF) + ADC. **b** Universal approximator. XOR is implemented by using VMM + WTA. **c** Speech classifier

6.1 Low pass filter and ADC

Figure 10a shows a Xcos design and the *VPR* result of a first-order low-pass filter (LPF) system with a DAC and an ADC. An OTA connecting the output to (-) input and a FG device ADC using FG infrastructure are integrated into macro-CAB blocks, respectively. A dedicated circuit converting user-defined input vector into a voltage on FG SoC FPAA is utilized. The tool creates switch list based on the *VPR* placement and routing, and the experimental results have been proved in [13].

6.2 Universal approximator

A boolean function XOR using a VMM and Winner-Take-All (WTA) is an example of universal approximator [25]. Figure 10b shows the Xcos design, weight information, the expected input/output logic, and the resulted *VPR* placement and routing. The second and third input of the circuit and the weights forms the XOR output on the third output of the WTA. A macro-blif block including VMM, WTA, and a bias current FG nFET mirror circuit is implemented. A shift register block and GPIO logic signals are employed to measure the third output of WTA. The experimental results based on the compiled switch list have been introduced in [9].

6.3 Speech classifier

Figure 10c shows an example of speech classification detecting a word in a sentence. A macro-CAB block, "Speech," in Xcos design integrates band-pass filter (BPF), amplitude detection, and low-pass filter (LPF). Twelve speech blocks perform continuous-time decomposition with different Q values on BPF. A VMM + WTA block classify each of the resulting spectrum into simple symbols. A shift register block and ADC are employed to measure intermediate nodes. Location of speech blocks are customized for the performance, the experimental results have been shown in [8].

7 Summary and discussion

This paper presented a mixed-signal co-design environment using FG SoC FPAAs. The tools developed in this work take an essential role in the compilation flow, converting a user's Xcos design into a switch list and enabling experimental measurements in the same integrated design tool framework. The tool set is an open source setup provided in a Virtualbox package with Linux Ubuntu OS[1].

We expect our tools to empower a wider community for analog and digital system designers, as well as share the opportunities with VTR community. This paper opens up interesting questions in the optimization capabilities of the *VPR*. For example, we employed a "Joint" block to solve the incompatibility problem of multiple outputs on a single node, which results in using an extra block. Also, a constraint on the VPR array structure, in which CAB/CLB should be arranged in a column direction, limits the flexibility of the array structures. We believe that an extended version of VTR/*VPR* covering both FPGA and FPAA can cope with the problems in easier and more efficient way.

This paper starts the discussion on formulating the benchmarks for analog and mixed computation. Benchmarks imply understanding computation, which required the effort of this paper and parallel efforts to reach a point where developing a reasonable benchmark is possible. A benchmark likely would be composed of components seen in Fig. 10, although they are not at the necessary complexity for a well formulated benchmark. The initial benchmarks include small number of CLBs and CABs, however, we believe more complicated system-level benchmarks (e.g., Image convolution, image classification or speech recognition) will fully utilize most of digital and analog blocks.

One can infer more about the computation by choosing the right benchmark. Digital computation benchmarks are about matrix equation solutions (e.g.LINPACK [28]), including LU decomposition. Analog computation benchmarks would look at different optimization metrics such as Ordinary Differential Equations (ODE) and Partial Differential Equations (PDE) [29]. Or Digital/Analog mixed-signal computation [30] could be a good benchmark. The system examples illustrated in this paper show, what the benchmarks can be, on the path and are beginning to be clear for such systems. These would be the critical next steps as we move forward from our efforts.

Acknowledgements The authors would like to thank Suma George for her help while debugging the tools at the early stage of development and Sung Kyu Lim for valuable comments and advice on the tools.

[1] http://users.ece.gatech.edu/phasler/FPAAtool/index.html.

References

1. Betz V, Rose J, Marquardt A (2012) Architecture and CAD for deep-submicron FPGAs, vol 497. Springer, Berlin
2. Rose J, Luu J, Yu CW, Densmore O, Goeders J, Somerville A, Kent KB, Jamieson P, Anderson J (2012) "The VTR project: Architecture and CAD for FPGAs from verilog to routing," In: Proceedings of the ACM/SIGDA international symposium on field programmable gate arrays, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp 77–86
3. Kim J, Kim KT, Chung E-Y (2018) "Cad tool flow for variation-tolerant non-volatile stt-mram lut based fpga," In: Proceedings of the 2018 7th international conference on software and computer applications, 2018, pp 312–316
4. Saegusa T, Maruyama T, Yamaguchi Y (2008) "How fast is an fpga in image processing?" In: 2008 international conference on field programmable logic and applications. IEEE, 2008, pp 77–82
5. Nurvitadhi E, Venkatesh G, Sim J, Marr D, Huang R, Ong Gee Hock J, Liew YT, Srivatsan K, Moss D, Subhaschandra S et al. (2017) "Can fpgas beat gpus in accelerating next-generation deep neural networks?" In: Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays, pp 5–14
6. Ma Y, Cao Y, Vrudhula S, Seo J-s (2018) Optimizing the convolution operation to accelerate deep neural networks on fpga. IEEE Trans Very Large Scale Integr (VLSI) Syst 26(7):1354–1367
7. Kutz HM, Williams TJ, Sullam BS, Snyder WS, Shutt JH, Byrkett BE, Mar M, Thiagarajan E, Kohagen NW, Wright DG et al. (2019) "Combined analog architecture and functionality in a mixed-signal array," Jul. 2019, uS Patent 10,345,377
8. George S, Kim S, Shah S, Hasler J, Collins M, Adil F, Wunderlich R, Nease S, Ramakrishnan S (2016) A programmable and configurable mixed-mode FPAA SoC. IEEE Trans Very Large Scale Integr (VLSI) Syst 24(6):2253–2261
9. Collins M, Hasler J, George S (2016) An open-source tool set enabling analog-digital-software co-design. J Low Power Electron Appl 6(1):3
10. Kim S, Hasler J, George S (2016) Integrated floating-gate programming environment for system-level ICs. IEEE Trans Very Large Scale Integr (VLSI) Syst 24(6):2244–2252
11. Kim S, Shah S, Hasler J (2017) Calibration of floating-gate soc fpaa system. IEEE Trans Very Large Scale Integration (VLSI) Syst 25(9):2649–2657
12. Hasler J, Natarajan A, Shah S, Kim S (2017) "Soc fpaa immersed junior level circuits course," In: 2017 IEEE international conference on microelectronic systems education (MSE), May 2017, pp 7–10
13. Hasler J, Shah S, Kim S, Lal IK, Collins M (2016) Remote system setup using large-scale field programmable analog arrays (fpaa) to enabling wide accessibility of configurable devices. J Low Power Electron Appl 6(3):14
14. Mead C (1990) Neuromorphic electronic systems. Proc IEEE 78(10):1629–1636
15. Hasler J, Marr H (2013) Finding a roadmap to achieve large neuromorphic hardware systems. Front Neurosci 7:118
16. Shah S, Teague CN, Inan OT, Hasler J (2016) "A proof-of-concept classifier for acoustic signals from the knee joint on a fpaa," In: 2016 IEEE SENSORS, Oct 2016, pp 1–3
17. Koziol S, Hasler P, Stilman M (2012) "Robot path planning using field programmable analog arrays," In: 2012 IEEE international conference on robotics and automation, May 2012, pp 1747–1752
18. Enterprises S et al (2012) Scilab: Free and open source software for numerical computation. Scilab Enterprises, Orsay, France, p 3
19. Luu J, Goeders J, Wainberg M, Somerville A, Yu T, Nasartschuk K, Nasr M, Wang S, Liu T, Ahmed N, Kent KB, Anderson J, Rose J, Betz V (2014) VTR 7.0: Next generation architecture and CAD system for FPGAs. ACM Trans Reconfigurable Technol Syst 7(2):6:1-6:30
20. Berkeley U (1992) Berkeley logic interchange format (blif). Oct Tools Distrib 2:197–247
21. Doboli A, Vemuri R (2003) Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies. IEEE Trans Comput-Aided Des Integr Circuits Syst 22(11):1556–1568
22. Baskaya F, Anderson DV, Hasler P, Lim SK (2009) "A generic reconfigurable array specification and programming environment (grasper)," In: 2009 European conference on circuit theory and design, Aug 2009, pp 619–622
23. Baskaya F, Reddy S, Lim SK, Anderson DV (2006) Placement for large-scale floating-gate field-programable analog arrays. IEEE Trans Very Large Scale Integr (VLSI) Syst 14(8):906–910
24. Hasler J, Kim S, Adil F (2016) Scaling floating-gate devices predicting behavior for programmable and configurable circuits and systems. J Low Power Electron Appl 6(3):13
25. Ramakrishnan S, Hasler J (2014) Vector-matrix multiply and winner-take-all as an analog classifier. IEEE Trans Very Large Scale Integr (VLSI) Syst 22(2):353–361

26. Schlottmann CR, Shapero S, Nease S, Hasler P (2012) A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing. IEEE J Solid-State Circuits 47(9):2174–2184
27. Natarajan A, Hasler J (2017) Modeling, simulation and implementation of circuit elements in an open-source tool set on the fpaa. Analog Integr Circuits Signal Process 91(1):119–130
28. Dongarra JJ, Luszczek P, Petitet A (2003) The linpack benchmark: past, present and future. Concurr Comput: Practice Exp 15(9):803–820
29. Hasler J (2016) "Opportunities in physical computing driven by analog realization," In: 2016 IEEE international conference on rebooting computing (ICRC), Oct 2016, pp 1–8
30. Ovtcharov K, Ruwase O, Kim J-Y, Fowers J, Strauss K, Chung ES (2015) Accelerating deep convolutional neural networks using specialized hardware. Microsoft Res Whitepaper 2(11):1–4