CREATION OF PROGRAMMABLE ANALOG STANDARD CELL LIBRARIES ENABLING RECONFIGURABLE LOW POWER SYSTEMS-ON-CHIP

A Thesis
Presented to
The Academic Faculty

By

Afolabi Ige

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2021

CREATION OF PROGRAMMABLE ANALOG STANDARD CELL LIBRARIES ENABLING RECONFIGURABLE LOW POWER SYSTEMS-ON-CHIP

Thesis committee:

Dr. Jennifer Hasler
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sung-Kyu Lim
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Callie Hao
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Date approved: December 10, 2021

Have you been tried in the fire?

Dave

Dedicated to my immediate family: Mum, Dad, Biola, Bolaji, Olabode and Arin.

ACKNOWLEDGMENTS

I would like to start by thanking my Baton Rouge family who helped shape me into the person I am today. Tj, Dani, Adamu, Richard, Neville, Fumnanya, Chidinma and others too numerous to mention. Thank you. I discovered my passion for research in Dr Daniels-Race's lab and for that I am forever grateful.

I would like to thank my Atlanta family and the awesome people I met during my tenure so far. Kehinde, Ore, Vanessa, Uty, Kayode, Seun, Aadit and so many more. You got me through tough times and I will always cherish our friendships.

I would like to thank the members of the Integrated Computational Electronics (ICE) lab, both past and present, without whom this project would not have been possible. Starting with Dr Hasler, whose brilliant work and encouraging personality allowed me to develop a passion for the topic. Aishwarya, Joyita, Linhao, Eric, Pranav, John, Praveen and Adrija. It has been a pleasure working with you all and I know we will continue to do interesting things together in the future.

Special thanks to the committee members, Dr Lim and Dr Hao for all their efforts.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A standard cell is a level of abstraction that creates logical circuit building blocks that can be assembled to build complex architectures. The concept of abstraction using standard cells is a well-established notion in digital architecture. In fact, productivity of digital designers has been greatly supported by these cells, yet there isn't any widespread equivalent in the analog domain. Generally, due to the large number of design parameters that tend to change across process nodes, it has not been viewed as a worthwhile endeavor to create analog standard cells without reconfigurability of those parameters. This work aims to show how leveraging floating gates can create abstractable analog circuits which build into standard cells that enable large-scale, low power, mixed signal System-On-Chip (SoC)s.

CHAPTER 1
WHAT ARE STANDARD CELLS?

1.1 Introduction to Standard Cells

Standard cells in the context of Very Large Scale Integrated Circuits (VLSI) generally refer to a combination of transistors – for the purpose of this discussion Metal Oxide Semiconductor Field Effect Transistor (MOSFET) – which build up to a basic computing element. Those basic computing elements then combine to form higher order logic. These higher order functional blocks can also be standard cells which are combined to build more impressive blocks.

1.1.1 Building blocks in Analog and Digital Domains

Digital Building blocks

In digital systems, the basic elements are boolean logic gates (nand, not, nor etc) which can be combined to form functional blocks like adders, latches, muxes etc.

Analog building blocks

In analog systems, the transistors build up to transconductance/operational amplifiers which when combined with primitives like capacitors and diode connected transistors can produce functional blocks like Low Noise Amplifier (LNA)s, band pass filters, delay elements[1] etc.

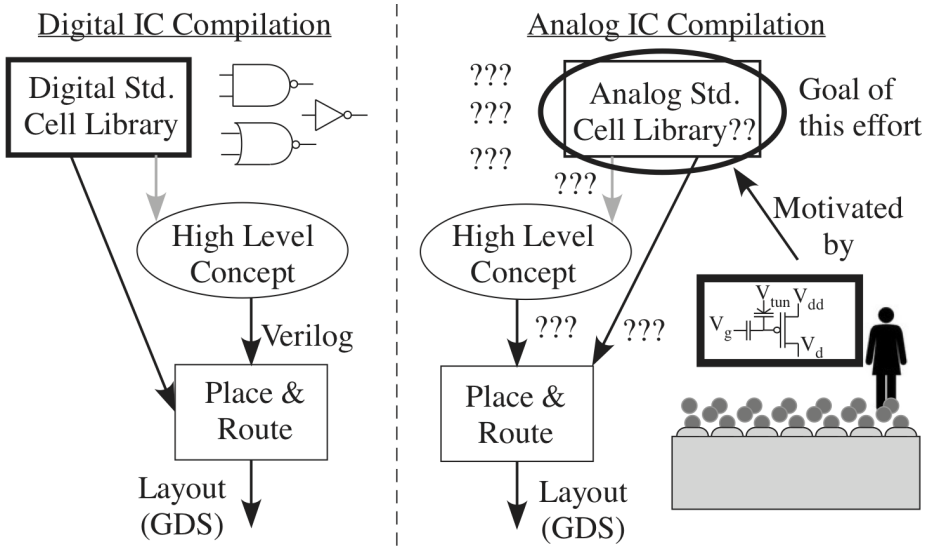

Figure 1.1: Digital standard cells enable design automation in the circuit design process while analog domain circuit design lacks an established equivalent

1.1.2 Addressing the Gap in Standard cells

In the digital domain, standard cell libraries allow circuit designers to focus on higher order descriptions of the design. The majority of the logic is represented by hardware description languages (while automation tools convert that logic to standard cells and even place and route the cells to generate an Integrated Circuit (IC) layout). This does not exist or is not very popular for analog/mixed signal systems and definitely does not exist at anywhere near the same maturity level as illustrated in Figure 1.1 The traditional approach to designing amplifiers (analog building blocks) is to first choose a topology and then to size the bias network and strength ratios of the transistors. However, it is important to note that the bias network and strength ratio primarily change the channel currents flowing through each branch. A designer can effectively create a new amplifier by tuning the gate voltages to adjust current flow of certain transistors in an opamp. This is where floating gates enter the conversation. By storing and adjusting charge on a MOSFET gate of the differential pair or biasing current sources, a reprogrammable amplifier can be created to allow for higher level analog abstraction. This can help with specifications like gain, input range, linearity etc.

of the amplifier. It also allows for post-fabrication mismatch adjustments leading to higher yield and better matching without some traditional analog layout techniques. A separate discussion would be required to properly address the myriad of specifications that can be accounted for with floating gates as well as how to design temperature insensitive circuits using floating gates [2], but the takeaway for this conversation is that programmability gives us the ability to create useful abstractions in analog circuits[3].

1.2 Reconfigurable Computing

1.2.1 FPGAs as a subset of FPAAs

The idea behind reconfigurable computing is to be able to change the circuit being used based on the needs of the designer. The ability to upload a custom circuit to a platform is an incredibly powerful tool for rapid prototyping especially when Application Specific Integrated Circuit (ASIC) fabrication can be very costly. Field Programmable Gate Arrays (FPGA) are able to accomplish this for digital systems by using Look Up Table (LUT)s in combination with muxes and registers for storage. These three main items can form a Computational Logic Block (CLB). A LUT can implement any combinational logic circuit (digital building blocks) given the truth table. Putting enough CLBs in a massive 2-D array and routing between them is what enables custom digital architectures to be uploaded to an FPGA.

Floating Gates (FG) based Field Programmable Analog Arrays (FPAA)s – an invention from Dr Hasler's Lab [4, 5] – expand on this idea by creating Computational Analog Block (CAB)s which are routed in the same array as CLBs. These CABs contain those analog building blocks discussed earlier but are enabled by floating gates to give them programmability. That means amplifiers whose parameters are tuned to the user's requirement can then be combined with other primitives available in the CAB such as nfets, pfets, current mirrors, capacitor banks etc. to create and test custom analog circuits without the cost of ASIC fabrication.

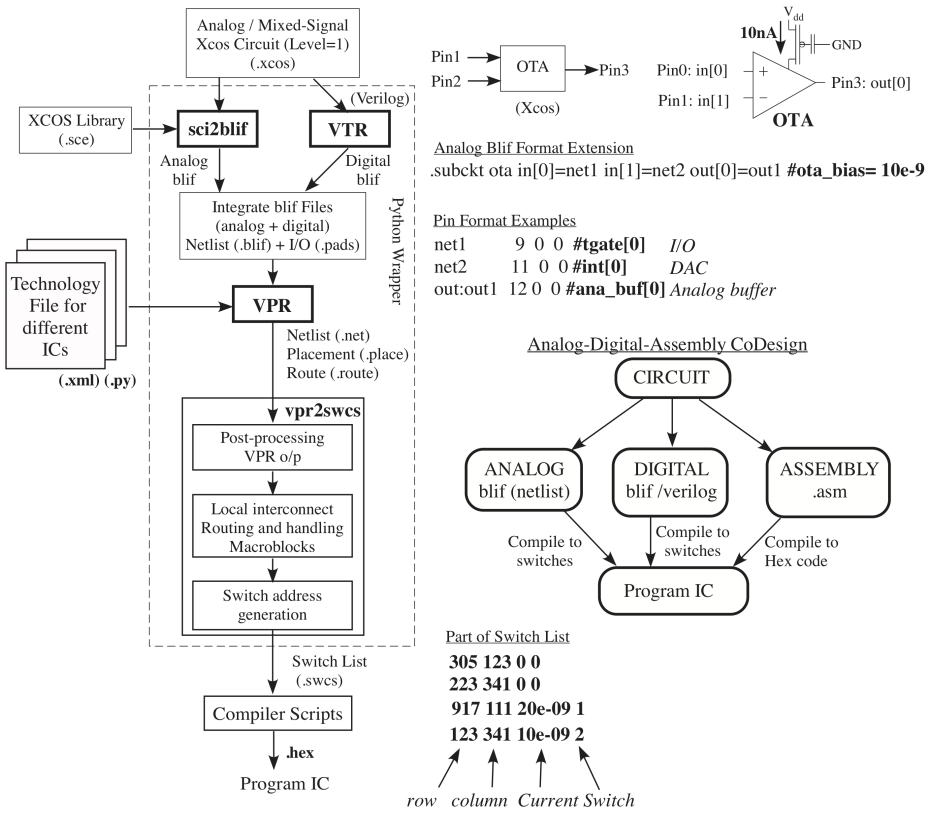

Figure 1.2: Current Tool flow detailing how circuits are programmed to the FPAA

1.2.2 Current FPAA Tool Chain

Collins et al. tool paper [6] shows an in-depth discussion on the design philosophy and implementation of the current tool chain. As a quick summary, the current set of tools that enable FPAA programming include Scilab and Xcos programs as well as Python and Assembly Scripts for translating schematics into machine-readable byte code as seen in Figure 1.2. Analog/Mixed Signal circuits are designed in the graphical Xcos program while any digital circuits used in the same design are created in Verilog. A custom ”sci2blif” script converts the Xcos circuit to Berkeley Logic Interchange Format (BLIF) while Verilog-To-Routing (VTR) converts digital circuits to BLIF as well. These then feed as input to Virtual Place and Route (VPR) which in conjunction with the technology file generates a netlist that is placed and routed in the FPAA fabric. The next step is for ”vpr2swcs” to do some post-processing and generate a switch list which represents transistor locations and their floating

gate charge. The compiler then uploads these to the on-board microprocessor for FG programming. This tool flow is currently being expanded to include a text based high level circuit description for analog/mixed signal circuits.

CHAPTER 2

WHAT ARE FLOATING GATES?

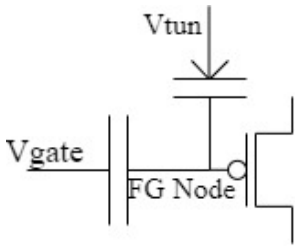

Figure 2.1: A symbol for a pFET floating gate

A floating gate transistor is a MOSFET with an external capacitor in series with the gate capacitor to hold charge between them Figure 2.1. A tunneling capacitor is used to remove charges from the floating gate (capacitive coupling to the floating gate node to extract charge carriers). It is important to note that there are no contacts to the floating gate node to prevent leakage. Due to the floating gate node being surrounded by high quality insulators, it guarantees the long term storage of charge. Once charge is placed on the FG node, from the perspective of Vgate, the threshold voltage of the MOSFET can be changed to suit the needs of the circuit [7]. This is the programmability as discussed in Ch.1 that allows for analog abstraction. Knowledgeable readers might already be familiar with floating gates as memory elements which forms the basis for Electronically Erasable Programmable Read Only Memory (EEPROM)s, Solid-State Drive (SSD)s, Flash memory etc. However, for the purpose of this discussion, floating gates will be examined as computing elements.

2.1 Programming Floating Gates

2.1.1 Fowler-Nordheim Tunneling

To pull electrons off the FG node, a capacitively coupled MOS Capacitor (MOSCAP) is used. A MOSCAP is preferred because of the substantially better oxide quality and as such improved reliability[7]. By raising Vtun, electrons tunnel through the SiO_2 insulator, effectively raising the voltage at the FG node.

2.1.2 Hot Electron Injection

Hot electron injection is used to put electrons on the floating gate node. The idea behind it is having electrons from the channel shoot through the gate insulator to reach the floating gate node. This is achieved by raising the drain line to an injection voltage, and keeping the gate voltage at fixed ground potential. Electrons starting at the drain edge of the channel flowing to the source with higher kinetic energy than the Si-SiO_2 barrier are injected into the oxide and transported to the floating gate.

2.2 Using Floating Gate Transistors in Circuits

Once the charge has been programmed it is important to measure it to ensure accuracy. The amount of charge on the floating gate is estimated by measuring the current the transistor produces when the source is connected to a supply voltage. In fact, when programming the floating gate, charge is added until the transistor produces the target current. This however raises an interesting question: how does one measure the drain terminal current of a transistor when programming, while also connecting said drain terminal to say another transistor to act as a bias with the minimum number of total transistors used.

2.2.1 Direct Programming

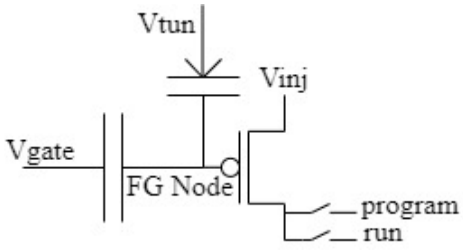

Figure 2.2: Directly programmed Floating Gate

In direct programming, the transistor being programmed is the same transistor that is being used. A couple of switches are required at the drain terminal to switch between the program line and the run line as seen in Figure 2.2. The injection supply can be moved from injection voltage to supply voltage to avoid adding extra switches. The disadvantage of direct programming is the requirement of multiple switches in the FG cell. The advantage it offers is the certainty of the target current programmed.

2.2.2 Indirect Programming

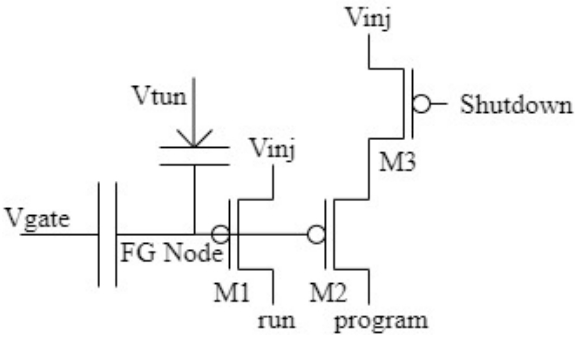

Figure 2.3: Indirectly programmed Floating Gate

In indirect programming, the programmed transistor and the in use (aka run mode) transistor are separate but share the same floating gate node (Figure 2.3). There is also an added shutdown transistor M3 which minimizes leakage current from the measure (aka program mode) transistor. This is important because the shared floating gate charge could turn on

the program mode transistor but having a shutdown FET separating it from a supply voltage prevents unwanted current draw. The advantage of this setup is the dedicated run and program lines allow for fewer transistors per FG cell, but the disadvantage of this setup is the mismatch between the program mode transistor and run mode transistor could cause slightly different currents in the branches. Yet, once this mismatch is characterized, it can be accounted for when programming future charge to the node.

2.3 FG and Non FG Cell Island Placement

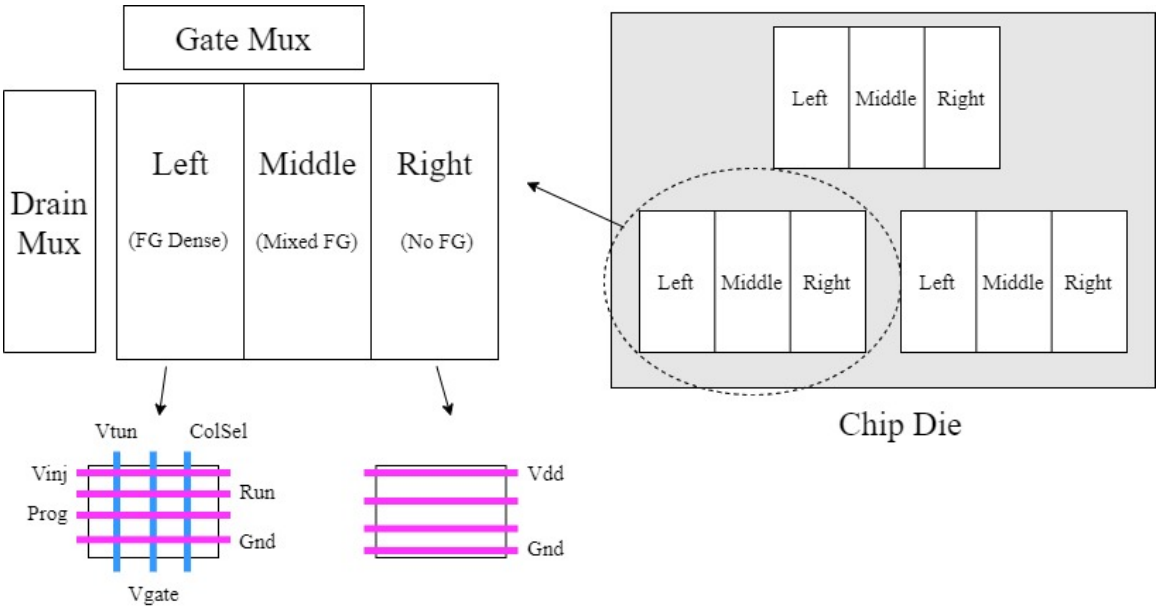

Figure 2.4: Overview showing the use of island structures to mix FG dense cells which come with many required signal and supply lines with non-FG cells that only require supply and ground

A significant motivator behind designing these floating gate circuits as standard cells is to bring the current state of high level analog design closer to the realm of digital as seen in Figure 1.1. This would enable much larger system architectures and faster design cycles in analog circuit design. However, floating gates bring many signal and supply lines to each standard cell that must be accounted for when laying out the circuit. The density of these lines is what inspired the use of an island architecture where many FG elements can

be grouped and routed together. Even within islands, some cells are completely FG based, others only partially make use of floating gates, while some do not use any floating gates at all. To account for this, it makes sense to group the FG cells together (left) as well as grouping the non-FG cells together (right) and to transition between them using cells that are designed to be half of each (middle) as seen in Figure 2.4. To make efficient use of space, the cells should be arranged in a gradient starting from the most FG dense to the least dense as seen in Figure 2.5.

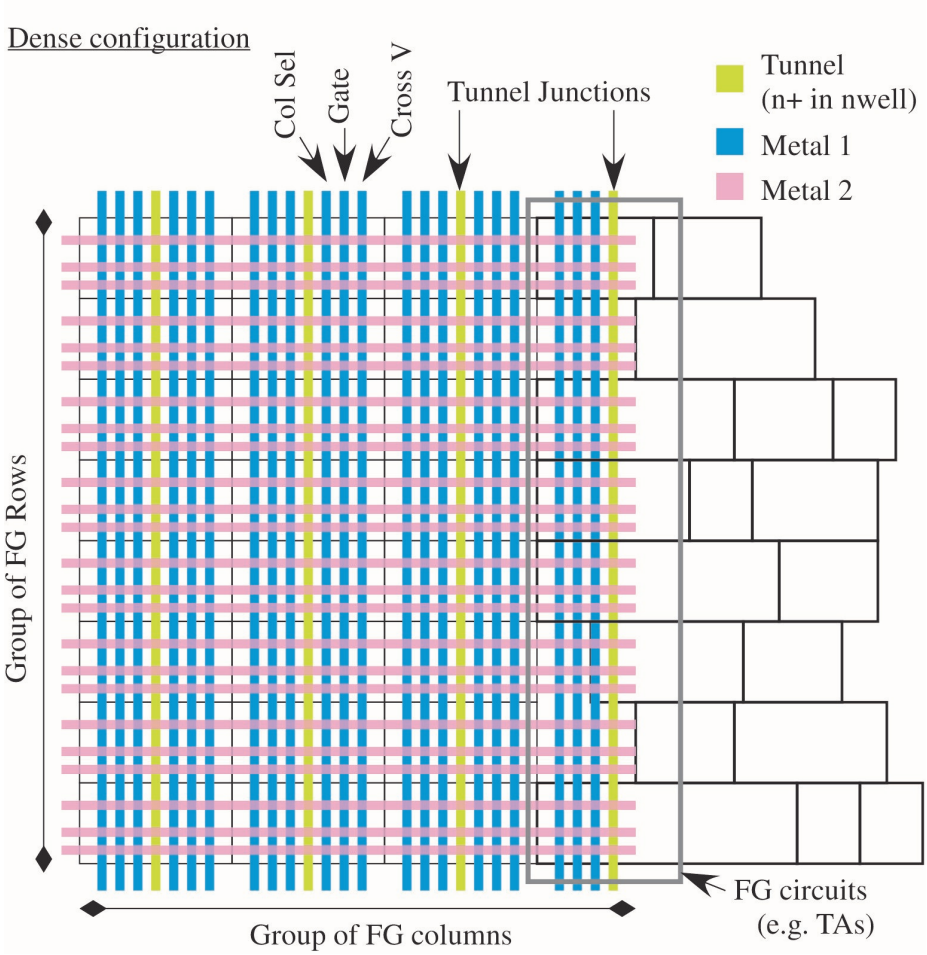

Figure 2.5: Detailed illustration showing how the gradient of FG and non-FG cells can be placed in an island

CHAPTER 3

CREATING STANDARD CELL LIBRARIES ACROSS PROCESS NODES

3.1 350 nm Process

The discussion starts in a 350 nm Complimentary Metal-Oxide Semiconductor (CMOS) process. It is convenient because this process node makes readily available two layers of polysilicon which is ideal for creating highly linear capacitors.

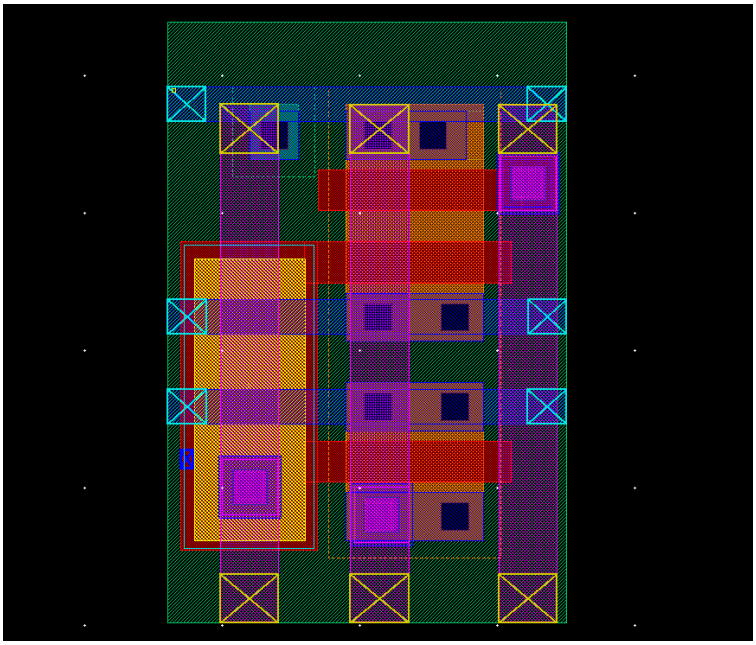

Figure 3.1: An indirect floating gate transistor circuit

Figure 3.1 shows an indirect floating gate transistor layout. This core cell is the basic building block for other FG based cells. The signal and power lines of the cell are designed to abut both horizontally and vertically which allows for efficient tiling of cells.

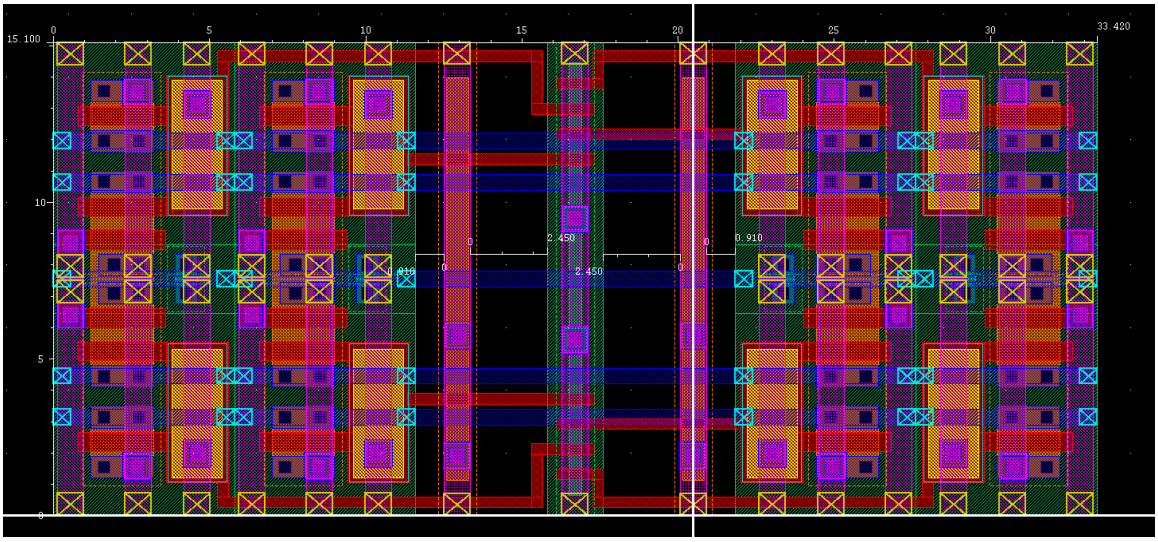

Figure 3.2: 2x4 bias cell used for VMM

Figure 3.2 is an example of tiling of the core FG cell. There are 8 FG fets: 2 rows and 4 columns. Each row shares a programming drain line and each column shares a gate line. This forms the basis for a crossbar array which can be used for a Vector Matrix Multiply (VMM) block or as routing fabric in an FPAA. It also sets the pitch for the entire 350 nm standard cell library of 15.1 μm.

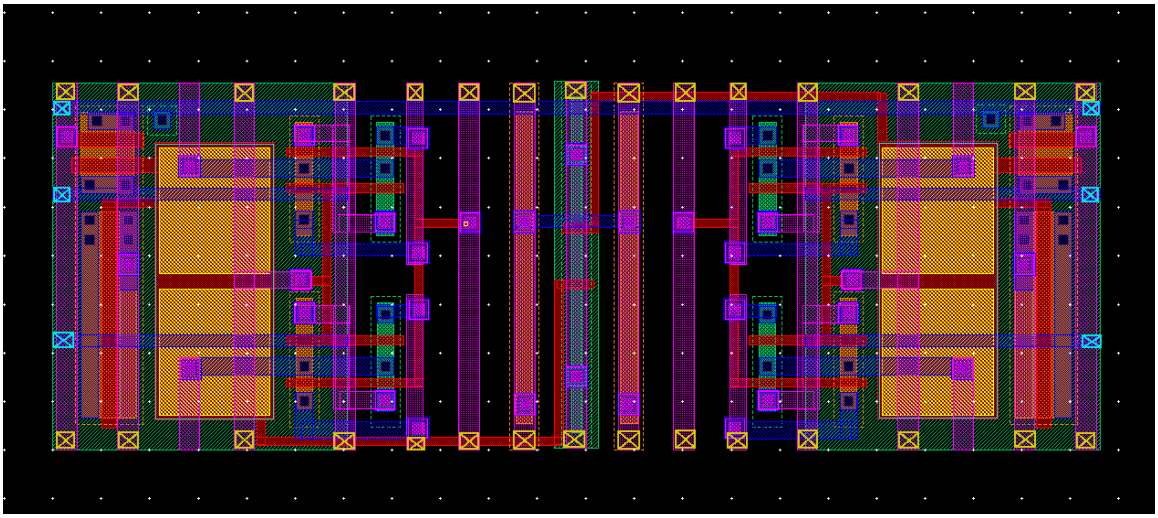

Figure 3.3: 1x2 bias cell with larger cap

In addition to the core FG bias cell, other variations of an indirect FG cell with multiple

capacitive inputs were designed[1]. Figure 3.3 shows two indirect FG cells with two capacitive inputs. The two inputs form a capacitive divider resulting in a lower sub-threshold slope when sweeping the gate voltage in comparison to non-floating gate pFETs.

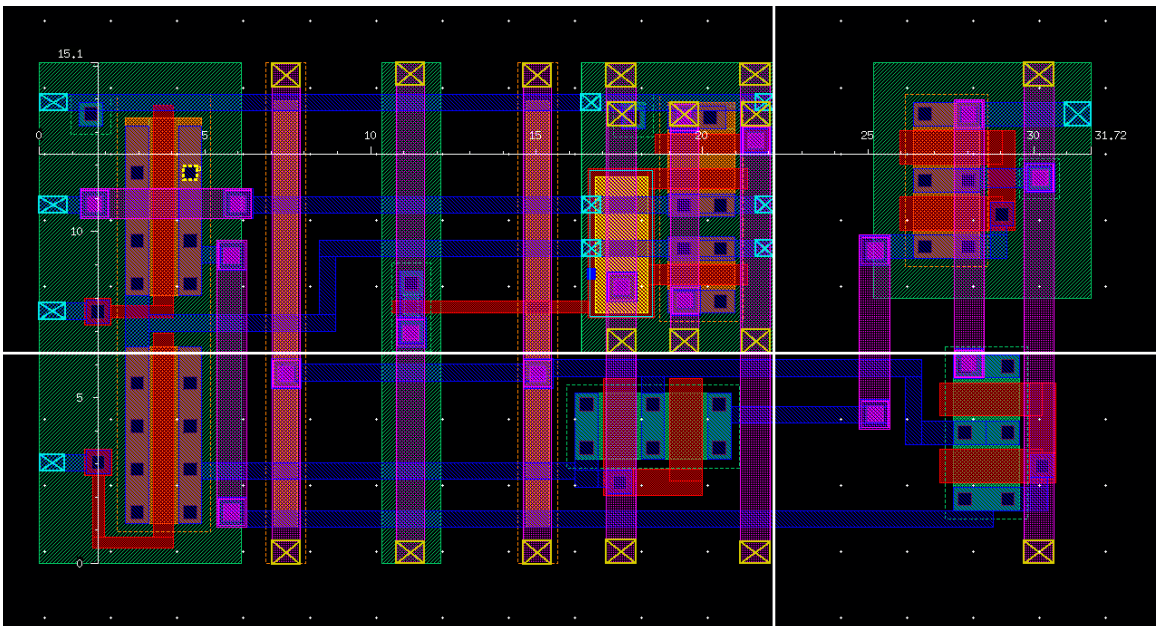

Figure 3.4: A 9T operational transconductance amplifier

Figure 3.4 is a classic transconductance amplifier structure with a FG cell as the current bias. This allows for tuning of parameters like gain post fabrication. With a FG cell on the differential pair, more parameters like input range and linearity can be also be tuned. Having OTA's in the library opens up the possibility for many circuit applications like capacitvely coupled current conveyor bandpass fiters, LNA's, delay circuits, amplitude detectors etc.

[1]This FG variation cell was a collaboration with another student Joyita Roy

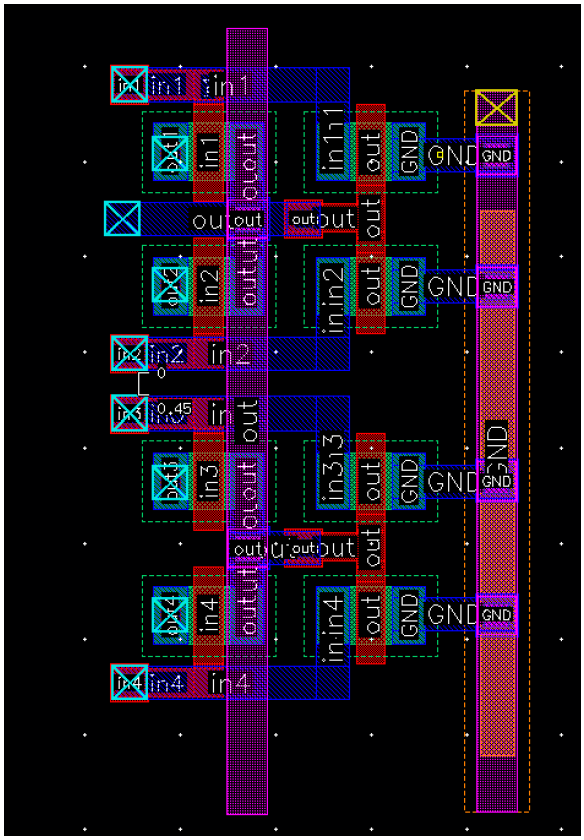

Figure 3.5: 4 Winner Take all circuits in pitch

A winner-take-all block is a circuit that selects the highest input current from an array of input currents. It is possible to have multiple winners selected, but the scope of this chapter limits the discussion to a simpler one winner implementation. It can be thought of as the output layer of an analog classifier network. Figure 3.5[2] shows 4 winner-take-all blocks connected to a single output line. When combined with a couple of the 2x4 Bias structures in Figure 3.2, one creates a fully analog classifier whose weights are represented by the charge on the floating gate node. This then allows for the VMM based ”Compute-in-Memory” architecture first posited in [8].

[2]The WTA cell was drawn by another lab student Linhao Yang

3.2 130 nm Process

A significant factor to consider when scaling FG-cells to lower process nodes is the availability of device primitives in each process, specifically the capacitors. The advantage of a smaller minimum channel length comes at the cost of the second layer of polysilicon that was previously used to create FG capacitors. While the 130 process does include Metal-Insulator-Metal (MIM) capacitors, this would require a contact to the FG node which would prevent long term storage of charge due to leakage.

The solution is to use a Varactor. This forms a capacitor between an n-diffusion in an n-well and the polysilicon gate, similar to the structure used for the tunneling junction (MOSCAP) thus preventing any contacts from being made to the polysilicon gate.

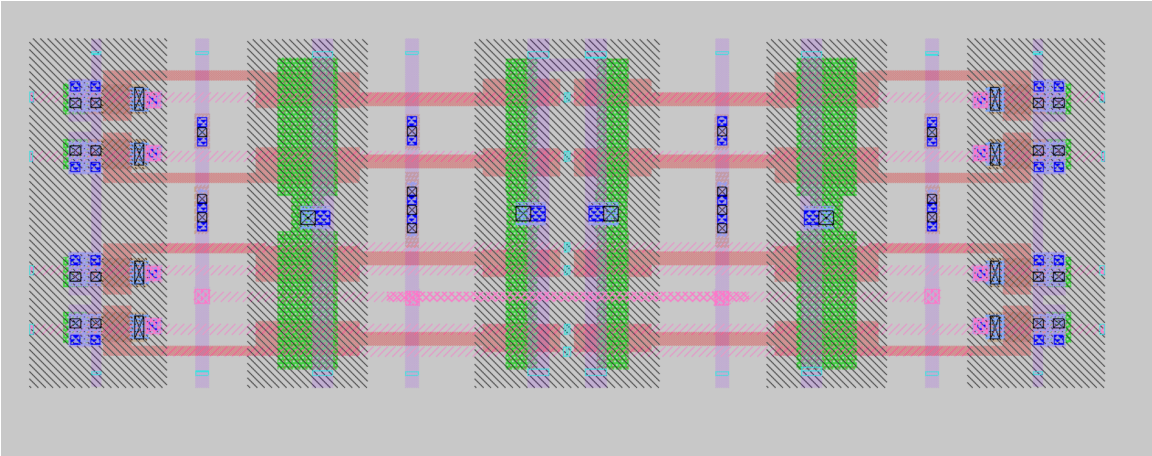

Figure 3.6: 4x2 bias cell used for VMM in 130 nm

The reduced minimum length of transistors in the 130 nm process allows the bias cell in this library to fit four rows and two columns of directly programmed FG cells in pitch as seen in Figure 3.6. Each transistor has its gate first connect to a gate-input varactor and then the tunneling junction varactor in the center n-well block. This cell also sets the pitch of the library at 6.5 μm. These are directly programmed cells as they were designed to for a large VMM array for which the extra switches could be placed on the edges[3].

[3]The 4x1 and 4x2 bias cells were contributed by Dr Hasler for the 130 nm process [9]

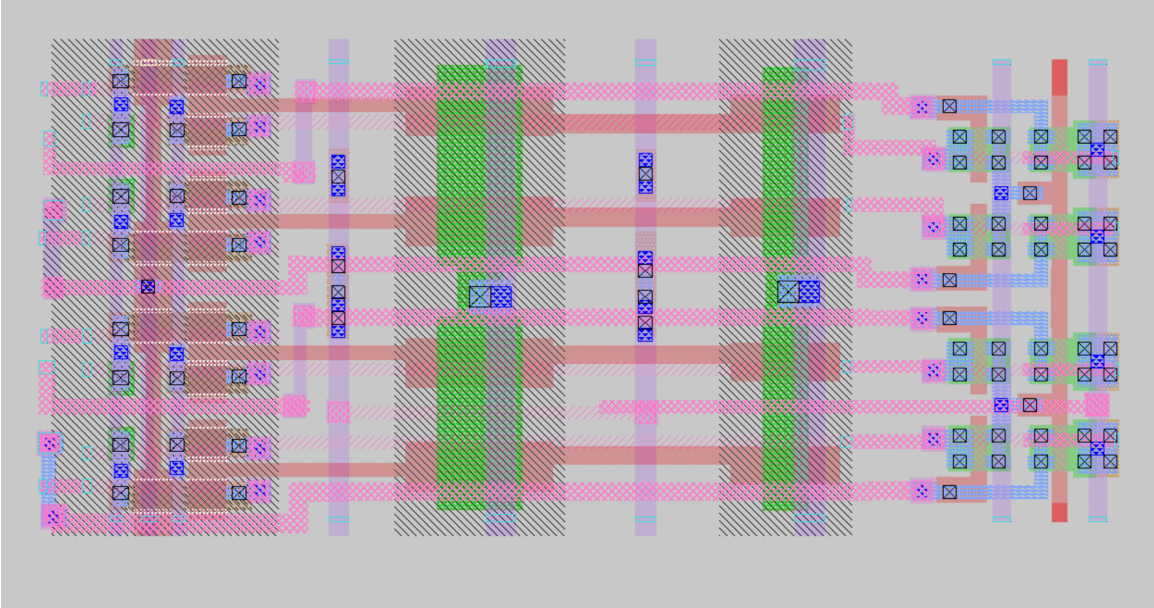

Figure 3.7: 4 winner take all cells biased by a 4x1 FG bias cell

The bias network for the WTA cells in this library was created with indirect FG cells as shown in Figure 3.7. Four rows of FG cells supply the current bias while the current input for the WTA cell is also made available on the left of the cell.

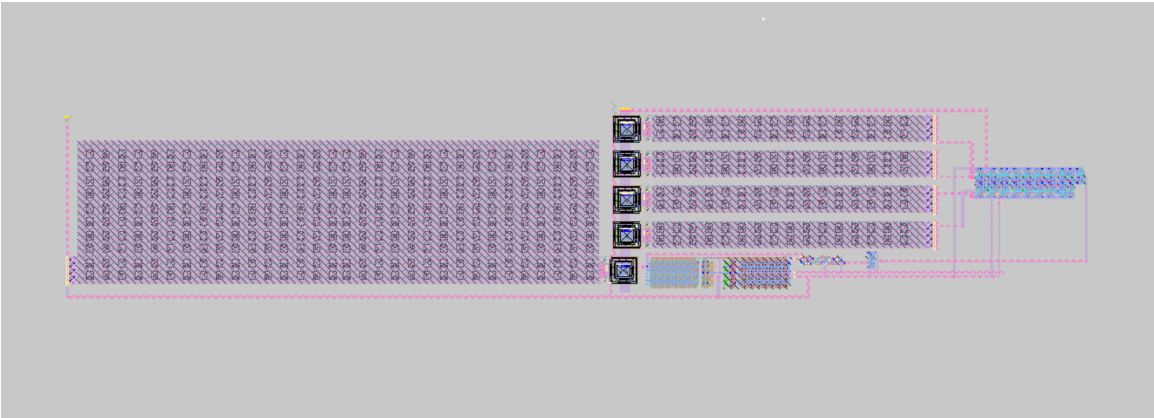

Figure 3.8: A 5 Stage On-chip Dickson charge pump for generating a 6V injection voltage from a 1.8V supply implemented in a 130 nm CMOS process

To flesh out the 130 nm library to a point where synthesis of a full chip is possible, additional cells were required. An example being charge pumps to generate injection (6V) and tunneling (11-12V) voltages on chip. Due to the large size of the capacitors, the standard cell (Figure 3.10) was designed as a diode and capacitor stage within pitch and chaining

those cells together formed the full pump. The size of the final storage capacitor is much larger than the previous stages and was allowed to occupy 4 pitch heights. Other circuitry involved to complete the charge pump is a NAND gate based non Overlapping clock generator (Figure 3.11), Large W/L Transmission gate (T-gate) to switch the injection supply to regular (1.8V) supply voltage and a level shifter + inverter for the T-gate selection logic.

The capacitors were implemented using two parallel MIM capacitors stacked vertically on each other to double the capacitance per unit area and provide extra shielding for the charge stored in each stage. This can be seen in Figure 3.9

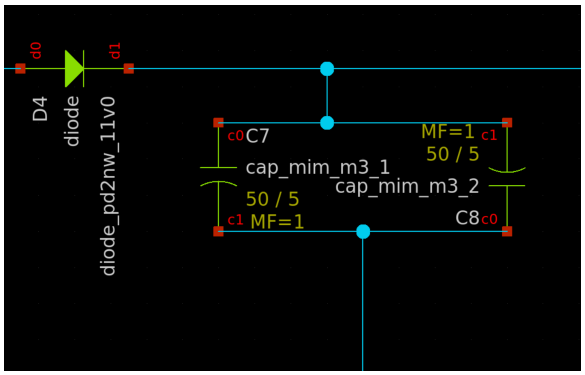

Figure 3.9: Schematic of a single stage of a Dickson charge pump consisting of a rectifying element connected to a capacitor

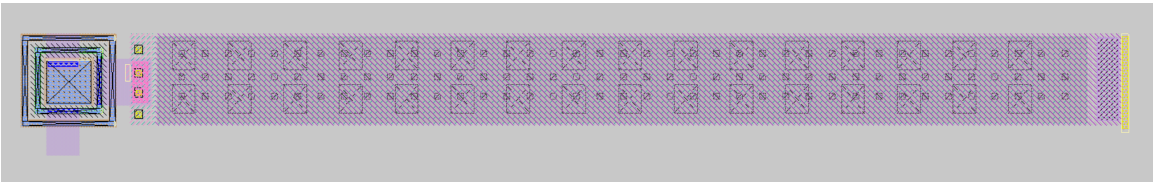

Figure 3.10: Layout of a single stage of a Dickson charge pump - sitting in pitch - showing a rectifying element connected to a capacitor

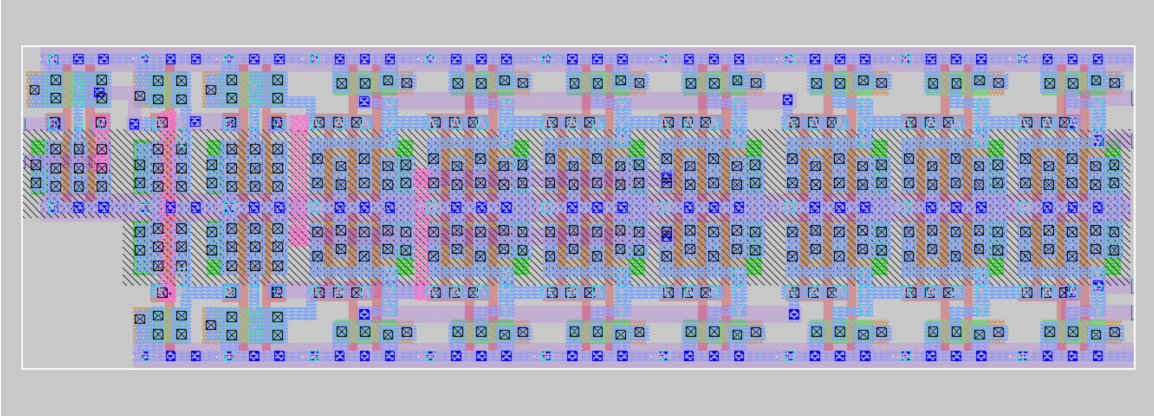

Figure 3.11: Non-Overlapping clock generator layout for the charge pump

Table 3.1: Comparison of key cell sizes between the 350 nm and 130 nm CMOS processes

Name	Process	Size	Width
Crossbar Cell	350nm	2×4 cell	33.42μm
	130nm	4×2 cell	20.12μm
Transconductance	350nm	1TA	31.72μm
Amplifiers (TA)	130nm	2TA	17.92μm
TA FG bias	130nm	1 FG, 1 non-FG, TA	28.09μm
Winner-Take-All	350nm	4 stages	26μm
with threshold biases	130nm	4 stages	14.07μm

3.3 90 nm, 65 nm Process

The 90 and 65 nm processes are fairly similar to the 130 nm process as they are all planar, single poly, MOS processes. As a result, there should be a relatively linear scaling [10] between cells in these libraries. Early estimations put the cell pitch at 4.5 μm and 3.25 μm for the 90 and 65 nm processes respectively. However, there is uncertainty as to how the scaling of the well spacing might change due the voltage levels being very similar across these processes. Additionally, the substrate doping might change which would also affect the scaling. Once access is granted to these process nodes, uncertainties can be clarified.

CHAPTER 4

APPLICATIONS ENABLED BY PROGRAMMABLE ANALOG STANDARD CELLS

4.1 ALICE: A Listening Caravan

The "A Listening Caravan (ALICE)" chip is a custom designed, hand synthesized[1], standard cell based, analog command word recognizer IC. As part of the Institute of Electronics and Electrical Engineers (IEEE) Solid-State Circuits Society (SSCS) Platform for IC Design Outreach (PICO) competition, the ALICE chip was developed to be fabricated by program partners Google, Skywater and Efabless in their efforts to support the open source IC design community.

4.1.1 Dataflow

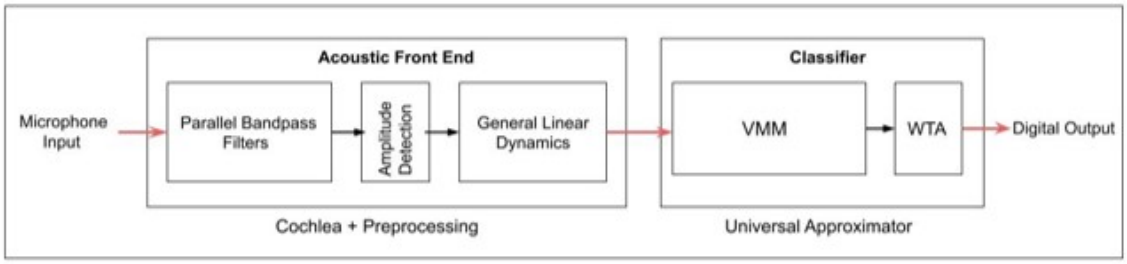

Figure 4.1: Block diagram showing the Alice chip dataflow

For the theory of operation, there are five relevant blocks as highlighted in Figure 4.1. The raw microphone input flows into the 16 parallel band pass filters. These divide the input into 16 frequency bands covering the range of human voice. Next, the 16 parallel outputs flow into the amplitude detection stage to determine if there is a strong enough

[1] ALICE was hand synthesized due to existing open source tools being unable to perform place and route with the 130 nm library. See Ch.5 for more details

signal within the isolated frequency band. The output of the amplitude detect stage then flows into a ”general linear dynamics” stage which is composed of seven delay stages, specifically ladder filters[1]. These delay lines expand the region of time over which input is collected as certain words and even phrases are multi syllabic and require a slightly longer time frame to characterize. The seven outputs of the delay stages and the one output of the amplitude detect stage forms eight individual time shifted versions of the input. There are also 16 frequency ranges for which this operation is performed generating a 128 output lines going to the VMM structure. This is the Acoustic Front-End (AFE) of ALICE.

A fact that might be taken for granted is the corners of the bandpass filters, the strength of the amplitude detects and delay of the ladder filters can all be tuned after fabrication thanks to floating gates. Traditional analog designs would have to make final decisions about these items before fabrication and any slight errors would lead to a lot of wasted effort and possibly a whole new chip run. Not to mention, if the user wanted to change the range of frequencies detected or shorten the delay window for classification purposes that would not have been possible.

The VMM and Winner-Take-All (WTA) form the on-chip neural network/classifier structure[11, 12]. There are two islands of 1600 x 128 directly programmed FG cells which will hold the weights and biases used to identify the bank of command words. Each row of cells flows into a WTA circuit at the last column. These are actually K-Winner Take All Circuits that allow for multiple winners at once. The digital logic value of the WTA’s win/loss is then scanned out via a shifter register attached to the VMM array. See Figure 4.2 for individual circuit schematics of each stage.

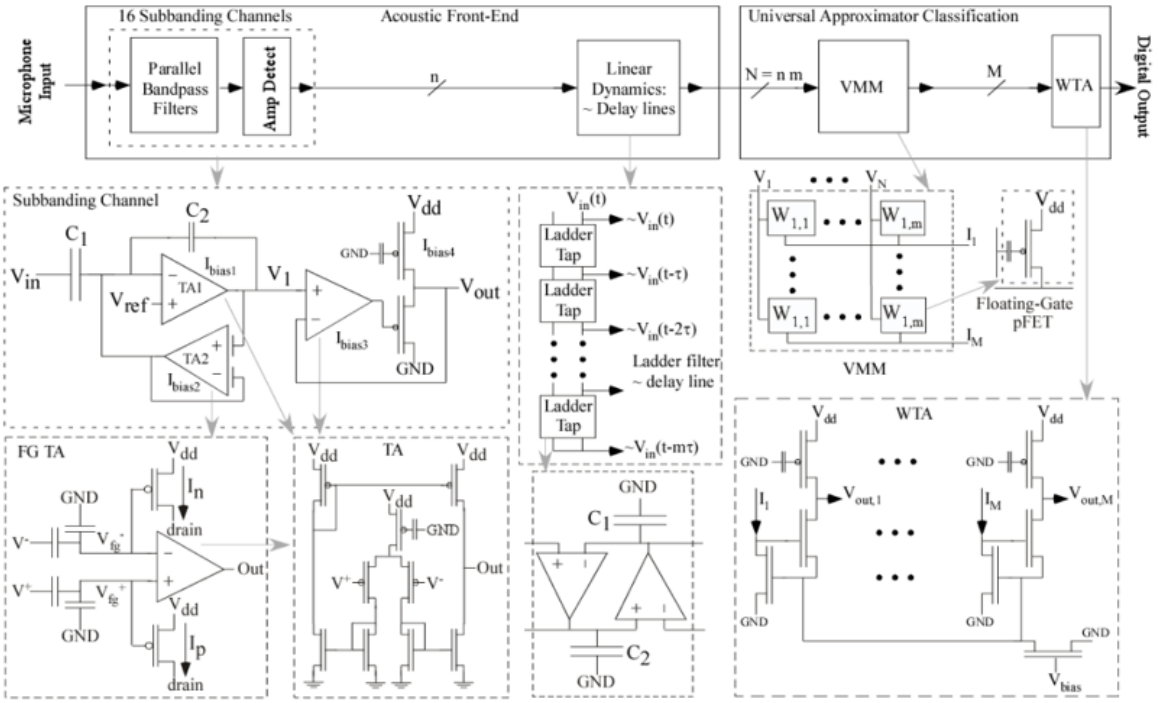

Figure 4.2: Schematics showing the circuit implementations of major functional blocks

The scanner is controlled by a shift register which is driven by a digital signal from an on-board processor. Each register element controls a t-gate that connects a WTA output to the shared scan out line as seen in Figure 4.3. One should note that Figure 4.3 is primarily to help convey the idea and is not an exact depiction of the implemented circuit.

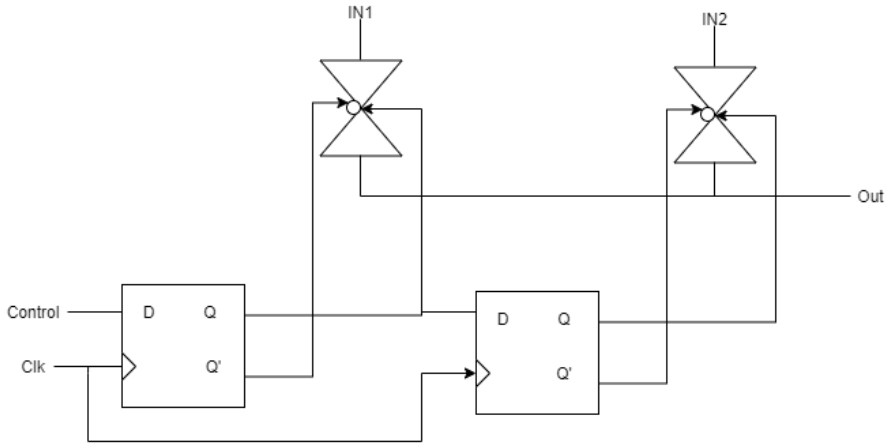

Figure 4.3: WTA Shifter Register controlled Scanner Cell Schematic

These scanner cells were designed to be connected to the VMM array and as such

need to fit four t-gates and four D-flip flops in pitch while abutting with the WTA output. The D-flip flops are digital standard cells from the Skywater provided library. Figure 4.4 shows two of these standard cell tiling vertically while in the full layout, 1600 of these tile vertically.

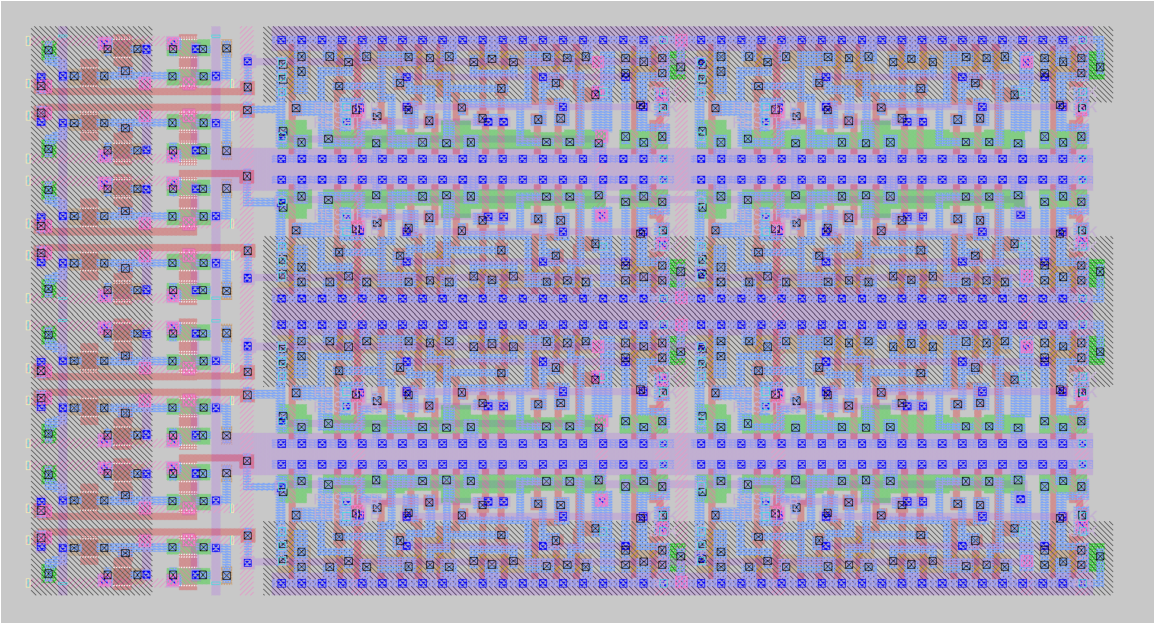

Figure 4.4: Layout showing two vertically tiling scanner cells

4.1.2 Programming Infrastructure

The total number of FG cells on the ALICE chip is around 410,000+ transistors which all need to be programmed. To accomplish such a task, programming infrastructure [13, 14] needs to be designed around the 3 core islands: 1 for the AFE and 2 for each VMM island. As the reader will remember from Ch.2 page 8, the two lines of interest when programming a FG cell are the gate line and the program drain line. The gate lines are the columns and the drain lines are the rows. Floating gate circuits will generally have two modes, run mode and program mode. Run mode is when the circuits are being used for their intended purpose while program mode is when the FG cells are being programmed. Another reason for placing cells on an island in a crossbar array is to allow addressing of individual FG cells during program mode.

Gate Line Programming

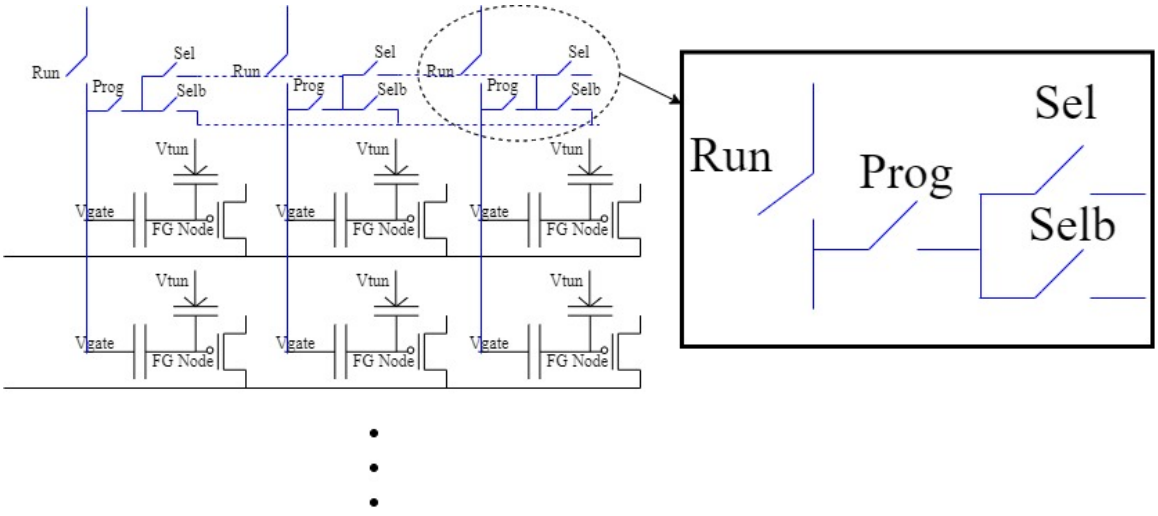

Figure 4.5: Switches used to connect the gate line to either an input or the programming select and unselect gate lines

In run mode the gate lines are connected to some input but in program mode, only one column should be connected to the gate select line while the other columns connect to the gate selectb (unselect) line. The select and unselect signals are generated by decoders which receive input from an on-board processor.

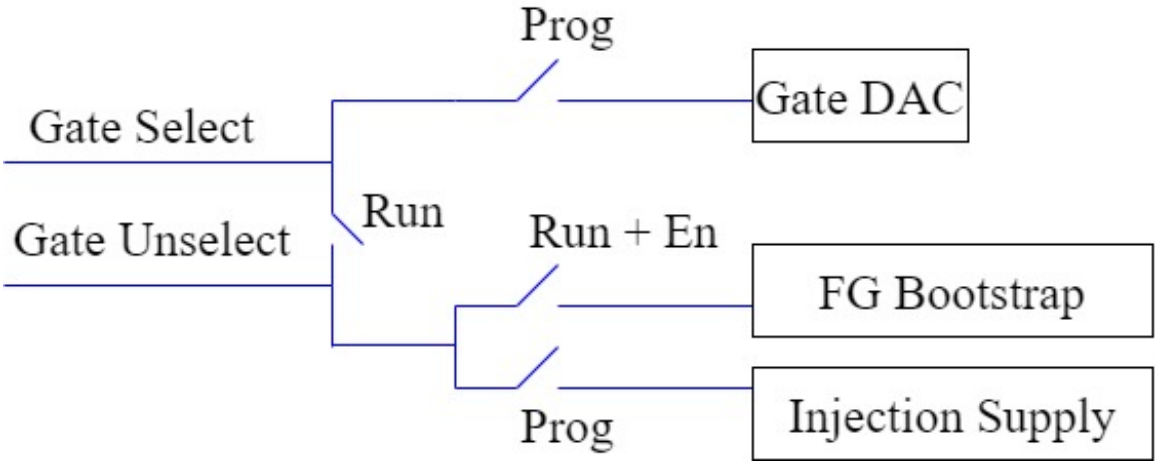

Figure 4.6: Switches used to connect the select and unselected gate lines to one of a gate DAC, Floating Gate Bootstrap Supply or Injection Supply

At the end of the gate select and unselect lines is the set of switches in Figure 4.6. In

run mode, both the gate select and unselect lines are shorted to the FG bootstrap source. In program mode, the selected gate line is controlled by the gate DAC and the unselected gate lines are tied up to the injection power supply to shut them off.

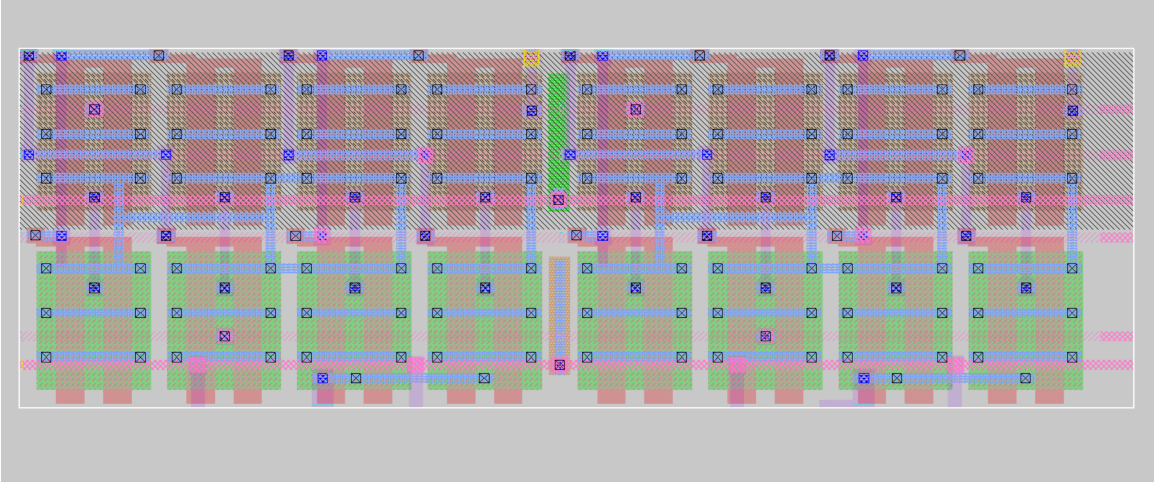

Figure 4.7: Layout of 2 horizontally tiled switches used to connect the gate line to either an input or the programming select and unselect gate lines

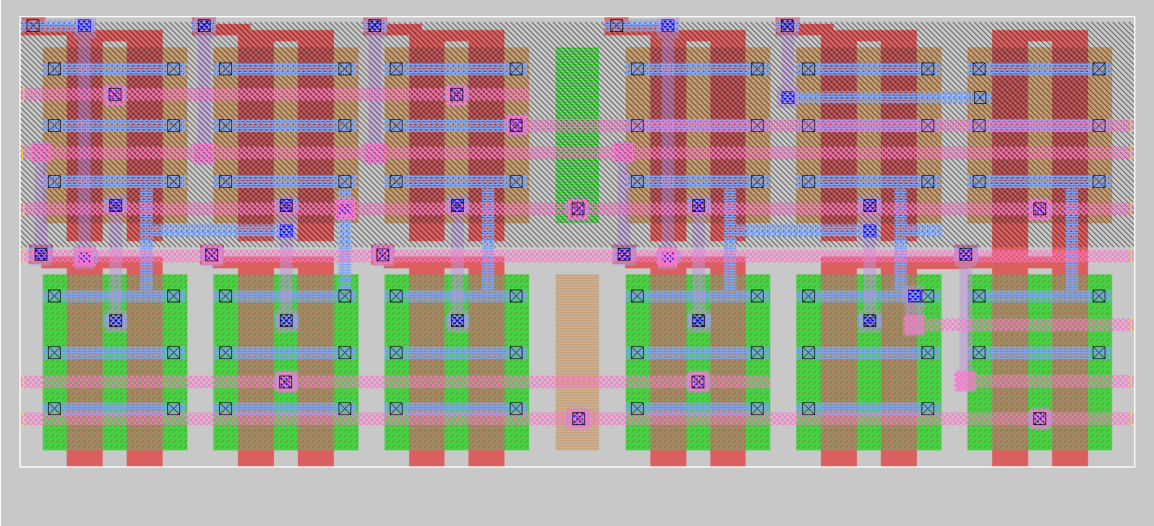

Figure 4.8: Layout of switches used in the programming architecture to connect gate lines to one of the DAC, Floating Gate Bootstrap or Injection Supply

Drain Line Programming

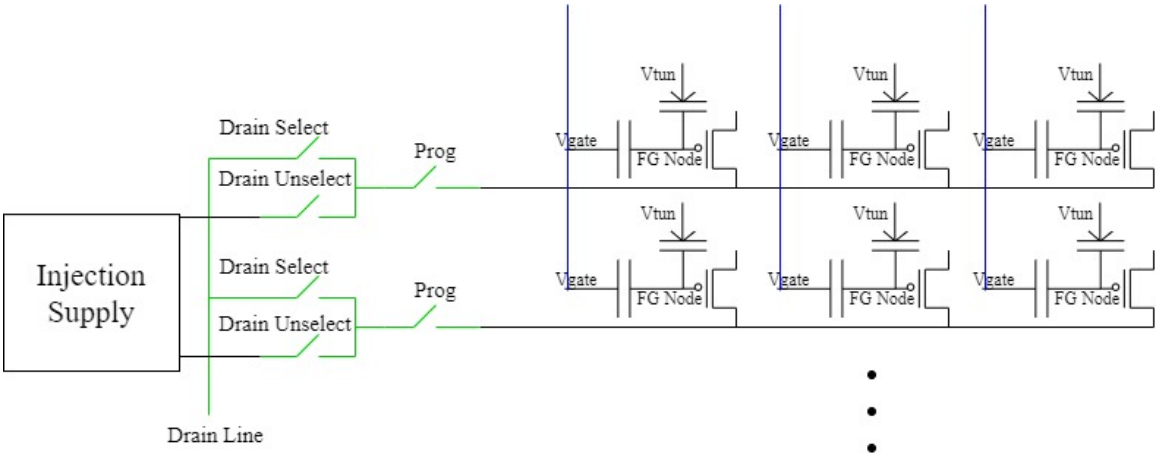

Figure 4.9: Switches used to select a drain line from a crossbar array

In program mode, each drain line is connected to the drain selection switches. Selection signals come from a drain decoder which receives input from the processor. The selected drain is connected the common ”Drain Line” as seen in Figure 4.9. Transistor drains that are not selected are tied to the injection supply. To finalize programming, the drain current of the selected drain needs to be measured as well as injected. The switches in Figure 4.10 serve this purpose.

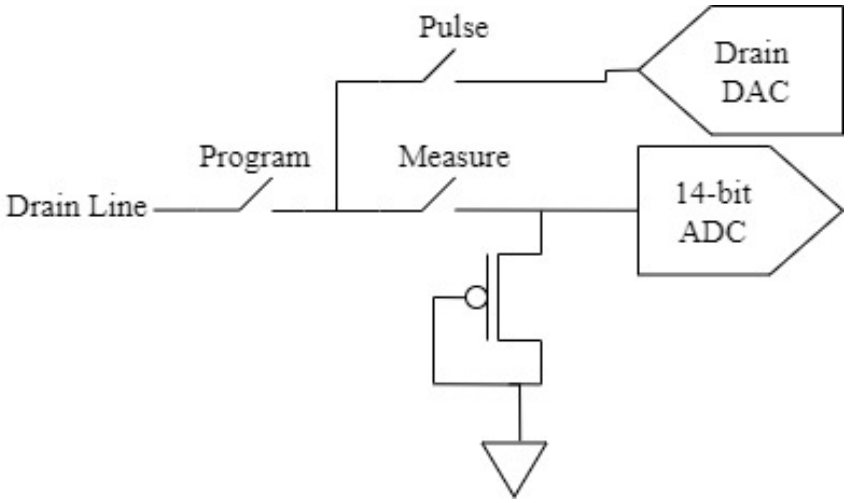

Figure 4.10: Switches to either pulse the drain line for injection or measure the drain current

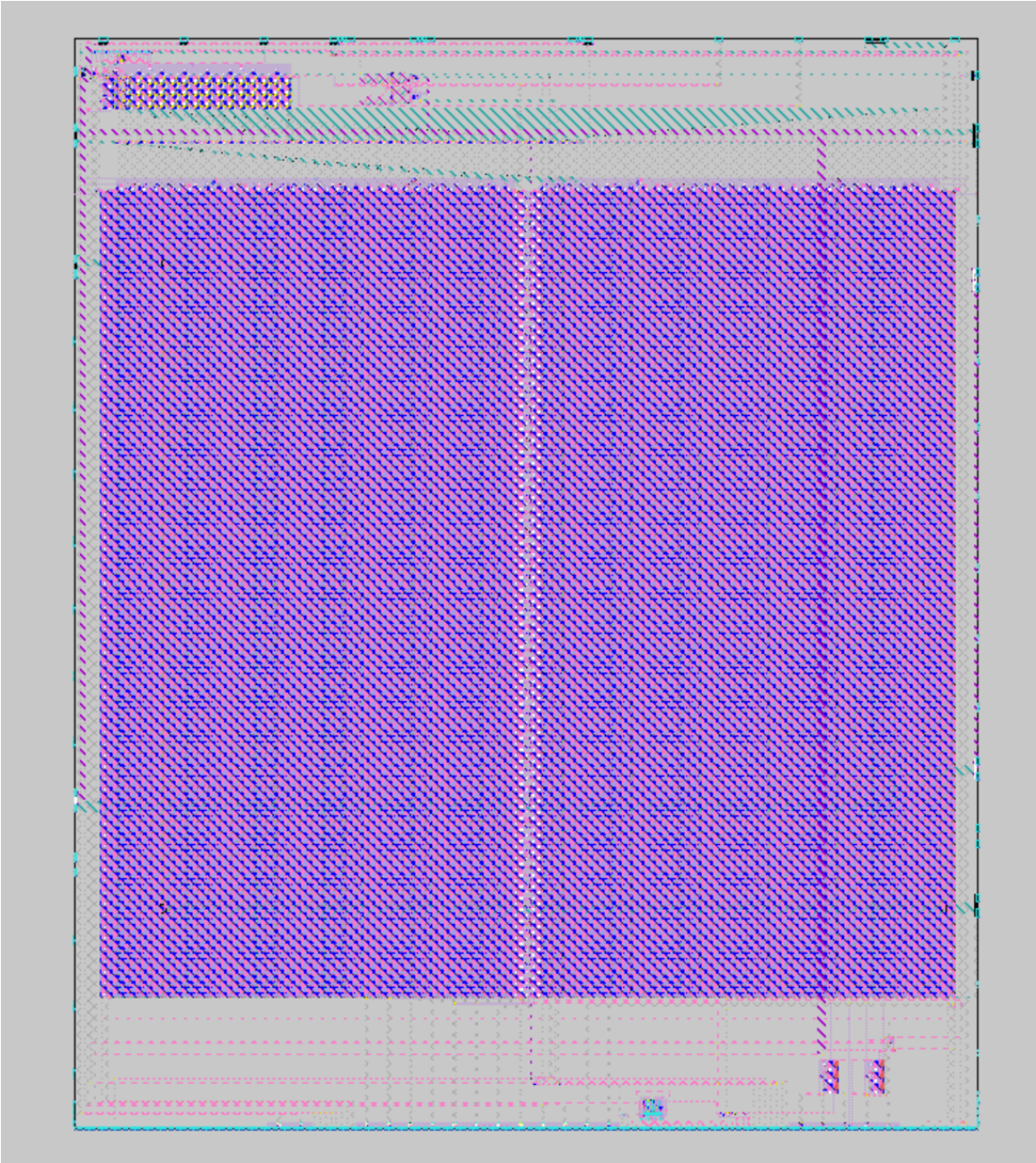

Figure 4.11: Hand synthesized[1] layout of the ALICE chip using the 130 nm standard cell library

[1]Synthesis of the full ALICE chip was done in collaboration with Dr Jennifer Hasler, Pranav Mathews, Praveen Raj and Dr Barry Muldrey

Table 4.1: Functional grouping of ALICE chip circuits showing their on-chip area and estimated power consumption

Component	Size	Area	Power
Bandpass filters + Amp detect	16 Channels	0.006 mm^2	2 - 6 μW
Linear Dynamics Computation (Delay lines)	16 input taps (7 delay blocks) 16 x 7 = 112	0.053 mm^2	0.5 - 2 μW
2x VMM + WTA Classifier	2x (128 x 1600)	7.03 mm^2	4 - 16 μW
FG Programming Infrastructure (During inference)			$\sim$0 μW
AFE Gate Mux	31 lines, 5 bit address	0.024 mm^2	
AFE Drain Mux	16 lines, 4 bit address	0.003 mm^2	
2x Classifier Gate Mux	128 lines, 7 bit address	0.110 mm^2	
2x Classifier Drain Mux	1600 lines, 11 bit address	0.184 mm^2	
Total[1]		7.41 mm^2	7 - 24 μW

The reader might notice that the VMM array alone contains around 400K FG transistors while consuming about 16 μW of power. Running at a 1.8V supply, this works out to nanoamp current levels for each row. (This deep sub-threshold current level is what gives analog computing significant scalability over its digital counterpart.)

[1]The area above does not include the charge pumps, DACs + ADC, and global routing area.

CHAPTER 5
SUMMARY AND LOOKING TO THE FUTURE

5.1 Summary

Chapter 1 explained the concept of standard cells and addressed how floating gates can be used to bridge the gap between digital and analog design flow to create abstractable analog circuits. Chapter 2 explained what floating gates are and how they can be programmed and used in a circuit. Chapter 3 examined the FG building blocks for both a 350 nm and 130 nm analog standard cell library. Chapter 4 demonstrated an application enabled by the 130 nm standard cell library and shows the usefulness of programmability in circuit design. My work was centered around implementation of the core FG cell, 4x2 Bias cell, 9T OTA cell and assisting other lab students like Joyita and Linhao in designs of the 350 nm library. Furthermore, my work also includes design and implementation of the charge pump cells, – both injection and tunneling (not shown) – vertical scanner elements, gate and drain line switches used in the programming infrastructure as well as the portions of the global routing done to synthesize the ALICE chip. All other aspects of the ALICE chip were in collaboration with Dr Jennifer Hasler, Pranav Mathews, Praveen Raj and Dr Barry Muldrey.

5.2 Looking to the Future

A running theme throughout the discussion has been the concept of laying the groundwork to elevate the analog circuit design process to approach that of digital. The only way to achieve that is through the power of automation. While the ALICE chip is a decently sized structure, access to place and route tools would allow for far more complex and ambitious projects. Even within the digital space access to synthesis tools have historically

been locked behind massive licensing contracts and only recently have the OpenRoad and OpenLane projects surfaced to help democratize hardware design.

5.2.1 Existing Open Source Synthesis Tools

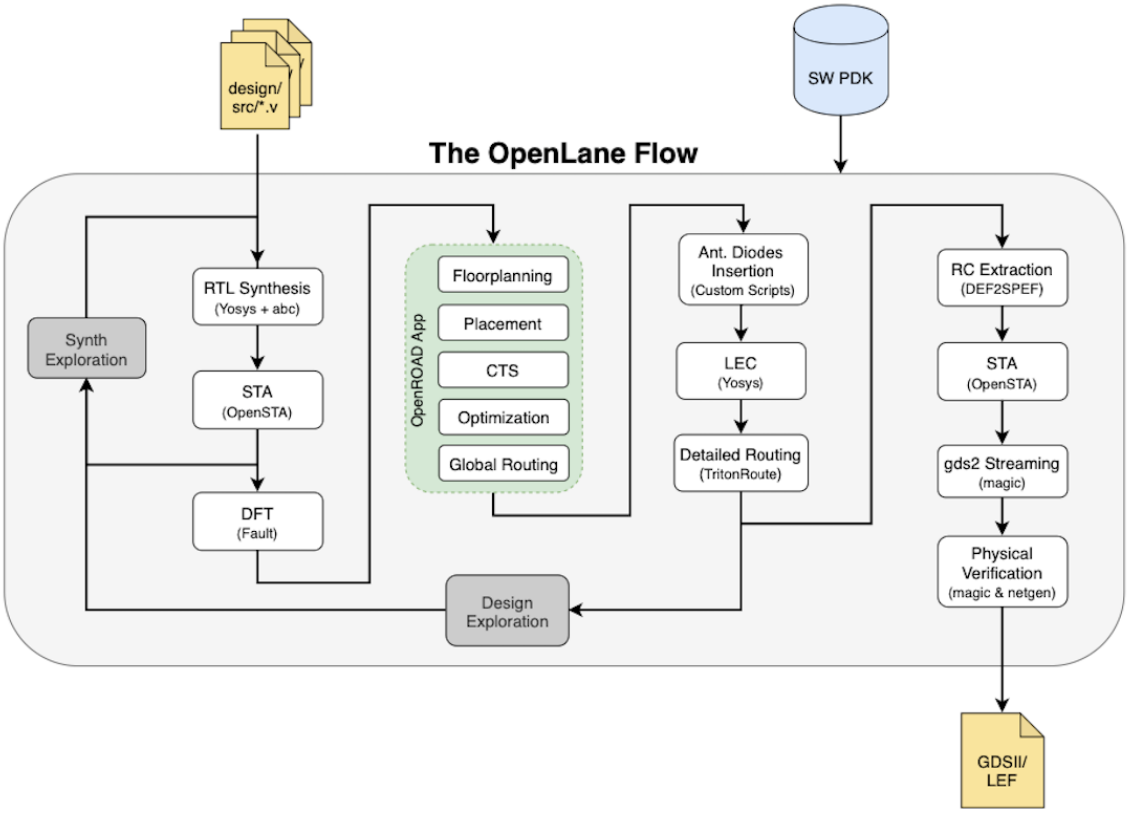

Figure 5.1: OpenLane connects many open source IC design tools sitting at various points in the stack to achieve one of the first fully open source HDL to GDSII Synthesis tool chains for digital designs.

The OpenLane project is an impressive tool that shows the power of an open source community and has made significant strides in accomplishing its goals. For the ALICE chip, the counter used by the 14-bit ramp ADC was synthesized by OpenLane. Attempts at synthesizing smaller analog projects using the 130 nm library were made, but unfortunately the way the tool tried to use the Library Exchange Format (LEF) files ultimately did not work. While the LEF and .lib files were generated, the tool had issue using them during place-and-route.

5.2.2 Novel Approach to Synthesis

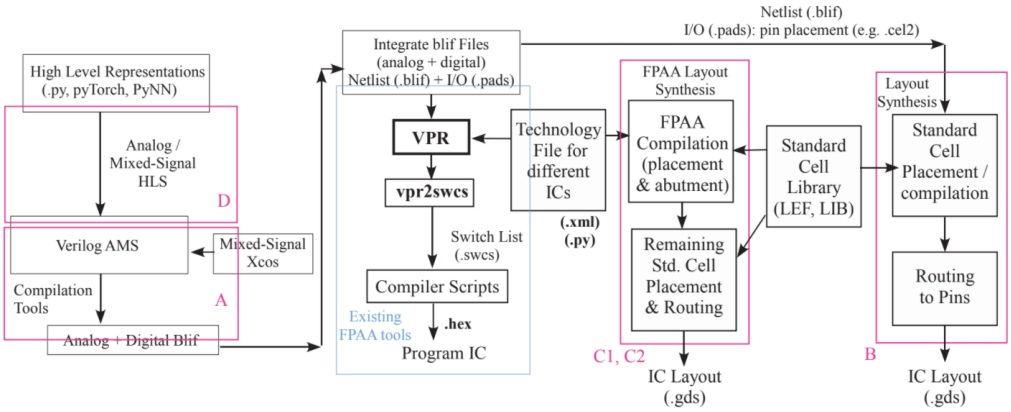

Figure 5.2: A novel synthesis flow that is able to program circuit implementations to an FPAA for rapid prototyping, ASIC layout generation and even synthesize the layout of new FPAA architectures

To achieve the future of synthesizable analog VLSI, especially with FG enabled standard cell libraries, a novel synthesis flow is described in Figure 5.2. As seen for the digital counterparts, circuit designs implemented on the FPAAs could only have matched or significantly better performance when moved to an ASIC. Power consumption would drop significantly considering unused elements are removed from the fabric while routing and programming infrastructure would be tailored for the application leading to a boost in computing efficiency as well. These additional reasons highlight the importance of establishing a novel synthesis flow that is able to program circuit implementations to an FPAA for rapid prototyping, ASIC layout generation and even synthesize the layout of new FPAA architectures with the same high level description language.

5.2.3 ALICE as a Benchmark

When discussing Electronic Design Automation (EDA) tools, it is important to have a benchmark against which to characterize the performance of said tools. With the 130 nm library being sufficiently fleshed out to hand synthesize the ALICE chip, it serves as an

excellent comparison point for the proposed synthesis tools in Figure 5.2. Parallel work is already underway tackling both task A: converting the existing Xcos tool chain to a text based representation and subsequently converting that to BLIF and task B: This work which contributes to creating standard cell libraries across various process nodes to enable synthesis. Contributing to the creation of the proposed flow is part of the work to be undertaken in my pursuit of a PhD here at Georgia Tech.

REFERENCES

[1] J. Hasler and S. Shah, "An soc fpaa based programmable, ladder-filter based, linear-phase analog filter," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 2, pp. 592–602, 2021.

[2] S. Shah, H. Toreyin, J. Hasler, and A. Natarajan, "Models and techniques for temperature robust systems on a reconfigurable platform," *Journal of Low Power Electronics and Applications*, vol. 7, no. 3, 2017.

[3] J. Hasler, "Defining analog standard cell libraries for mixed-signal computing enabled through educational directions," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[4] ——, "Large-scale field-programmable analog arrays," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1283–1302, 2020.

[5] S. George *et al.*, "A programmable and configurable mixed-mode fpaa soc," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2253–2261, 2016.

[6] 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*, vol. 6, no. 1, 2016.

[7] P. Hasler, B. Minch, and C. Diorio, "Adaptive circuits using pfet floating-gate devices," in *Proceedings 20th Anniversary Conference on Advanced Research in VLSI*, Mar. 1999, pp. 215–229.

[8] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. 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.

[9] J. Hasler, B. Muldrey, and P. Hardy, "A cmos programmable analog standard cell library in skywater 130nm open-source process," in *Workshop on Open-Source EDA Technology*, 2021.

[10] J. Hasler, S. Kim, and F. Adil, "Scaling floating-gate devices predicting behavior for programmable and configurable circuits and systems," *Journal of Low Power Electronics and Applications*, vol. 6, no. 3, 2016.

[11] J. Hasler and S. Shah, "Vmm + wta embedded classifiers learning algorithm implementable on soc fpaa devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 65–76, 2018.

[12] S. Shah and J. Hasler, "Soc fpaa hardware implementation of a vmm+wta embedded learning classifier," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 28–37, 2018.

[13] S. Kim, S. Shah, and J. Hasler, "Calibration of floating-gate soc fpaa system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2649–2657, 2017.

[14] S. Kim, J. Hasler, and S. George, "Integrated floating-gate programming environment for system-level ics," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2244–2252, 2016.