Circuit Design and Configuration for Low Power FPGAs

A Dissertation by

Seyi Ayorinde

6 July 2016

Submitted to the graduate faculty of the

Charles L. Brown Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the Dissertation and

subsequent Ph.D. in Electrical Engineering

Approved By:

John Lach, Committee Chair

Scott Acton

Steve Bowers

Kevin Skadron

Ben Calhoun

Outline

Abstract

Today, society is moving towards a ubiquitous computing (UbiComp) environment, where sensors and other integrated circuits (ICs) are ever present in daily life, creating a smart environment for people to constantly monitor and react to their environments. For UbiComp to be fully realized, a host of requirements are necessary for the ICs deployed in this vast network. These requirements include low cost, low power consumption, flexibility, and adequate computation power. Current IC design methodologies do not quite meet all UbiComp requirements. Application Specific Integrated Circuits (ASICs) are low power and have the necessary computing power for UbiComp applications, but they are prohibitively inflexible. Ultra-Low Power (ULP) General Purpose Processors (GPPs) have the necessary flexibility, but are prohibitively high power (by multiple orders of magnitude). Field Programmable Gate Arrays (FPGAs) are another IC implementation that could potentially be used for UbiComp applications, because they bridge the gap between the high efficiency of the ASIC and the high flexibility of the GPP. However, FPGAs have historically been targeted for high performance applications, where performance (or speed) is the main metric. As a result, commercial FPGAs are too power hungry for ULP applications (like UbiComp). Relatively little research has been done to bring reconfigurable logic into the ULP application space. However, if FPGAs can be retargeted for ULP applications, then they will intrinsically bridge that same gap, this time between ULP ASICs and ULP GPPs, by providing energy efficiency than ULP GPPs, but also have the flexibility to update or re-target their applications (a necessary requirement for UbiComp) without the expensive, time-consuming respins that ULP ASICs require.

This dissertation explores the steps necessary to both build and configure Ultra-Low Power FPGAs. These steps include:

1. developing a toolflow that will not only allow researchers to quickly co-optimize FPGA fabrics for different circuit and architectural parameters, but also users in the future to quickly be able to generate FPGA fabrics with the necessary configurations for a given benchmark circuit

2. re-assessing FPGA architectural parameters to determine architectures most suitable for ULP operation for FPGAs as opposed to high-performance operation

3. revisiting circuit design of FPGA-building blocks and re-designing them for low power operation

4. exploring the integration of the ULP FPGA fabric into a SoC designed for wireless sensing applications

Acknowledgements

I would be remiss if I didn't first thank God, through my Lord and Savior Jesus Christ, for providing me with the countless opportunities that led to this accomplishment, a PhD in Electrical Engineering (WHAT!?). I've been quoting the words of Paul and Timothy from the bible since I could talk: "I can do all things through Christ who strengthens me (Phil. 4:13)." I'm happy to attest to that, and proclaim that PhD degrees definitely qualify!

I'm so incredibly blessed to have a strong support system in my life, which was integral in my completion of this degree. I can't thank all of the people in my life enough. I first want to thank my father. He has been an inspiration my whole life, especially academically. He has more degrees than you can shake a stick at (BS, MS, CE, MBA, PhD). But more importantly, he has high expectations for me and my only brother. In fact, right before I set of for college, he told me: "I don't care what you do, son. Just make sure you get a doctorate." No pressure, huh? But I'm happy to say that I think I made my father proud, and that's all a son can ask for. I love you, Dad, and thanks.

My mother is also incredibly supportive, but in a very different way. She's a constant source of support, never wavering in her love, and is incredibly selfless with her time and her energy. She's still working in my father's retirement, and she frequently volunteers to travel here from DC to visit, despite the fact that she had to wake up at 3 AM for work that same day. I love you, mommy, thank you so much for being there the way you have my ENTIRE life, but particularly as I completed this degree.

I would also like to thank my little brother, Tope, who has been my best friend for years, and has helped to push me in ways he probably doesn't even realize. I've come to learn that he loves me very much, and looks up to me as something to aspire to. I don't mean to say that in a boastful way. In fact, I think that would be true whether or not I managed to achieve some level of success or not; it's just the way our relationship works. But because he looks up to me, it provided an additional impetus for pursuing my doctorate degree. Since I also have him looking up to me, I knew I needed to be the best student, brother, son, and man that I could be, to make sure that he followed suit. He's currently working on his master's degree at Virginia Commonwealth University, and I couldn't be more proud of him. I love you, Top'.

About two years into my degree program, I met a girl by the name Elizabeth Baker. Long story short, she's now my girlfriend of almost 2 years, and I couldn't be happier about it! Straight up, my friends get so annoyed by us. But thanks for giving me some sanity while things were at their toughest. A perfect example came when I was testing the first FPGA test chip that I personally

worked on. Nothing seemed to be working, and I complained to Lissie, saying "No matter what I do, these chips won't work!" She responded, without batting an eye, "Did you try putting Old Bay on it?" (Get it? Like potato chips?) In the moment, I was NOT amused, but Lissie has brought a joyous sense of levity to my life, and I love her very much for it. Baby I did it! Thanks for everything! I LOVE YOU!

My academic advisor, Prof. Ben Calhoun, is kind of a rock star at UVa. He brings in a ton of grant money, started his own company that is doing well, has an amazing family. 5 years ago I stepped into his office to make my pitch for working for him. In retrospect, I was probably a little too honest...I told him that I have absolutely no prior experience in electrical engineering or chip design. I told him that his works looks very interesting, and that I'm smart enough to pick it all up, and I promised I would do good work. For some reason, Ben thought that sounded like a good idea, which honestly still BLOWS MY MIND. But 5 years later, I think I made good on my promise. Ben, I appreciate you taking a chance on me, and I hope I did you proud. Thanks for your guidance, your high expectations, and your support in my pursuance of this degree. I couldn't have done it without you!

I definitely want to thank all of the people that have worked on the FPGA project in Ben's research group over the years. Dr. Joe Ryan, thanks for getting the project rolling and in a good direction. I especially want to thank He Qi, who has been a joy to work with. Not only did he treat me with the utmost respect (almost like I'm his boss, which I obviously am not), but he's also low-key HILARIOUS, and always has kind words to say. Keep up the hard work, He, and I'm excited to see where the FPGA project goes!

I'd like to also thank other members of the research group, both past and present. I'd especially like to thank Chris Lukas and Farah Yahya, who have given immense support over the last couple years in particular. From figuring out what a particular Cadence tool is trying to do to helping debug Verilog code, you guys have been patient and helpful, and I appreciate it! I'd like to thank past BenGroup members, particularly Dr. James Boley, Dr. Kyle Craig, Dr. Yousef Shakhsheer, and Dr. Alicia Klinefelter. Jim, way to be an athlete, thanks for being a good roommate and a good research colleague. Kyle, thanks for teaching me how to do top-level integration for a chip, and how to bowl over 100. Yousef, thanks for the Hulk Smash! button and the automated nerf rocket launcher. Oh, and the technical stuff too...Alicia, thanks for humanizing this research group from when I first got here. You're probably the biggest reason why I got comfortable in this research group, and I'll forever be indebted to you for that. Also thanks to all of the other BenGroup members (Divya, Harsh, Abhishek, Jacob, Ningxi, and Arijit) for the various ways that you've helped me along this

journey. Best of luck!

Lastly, I want to thank the staff here at UVa. Often times, their work is overlooked because it isn't quite as technical in nature, but the fact of the matter is we as researchers can't do our technical work without having the space in which to conduct the research, and the materials to do the research, from physical research equipment to IT infrastructure and CAD tools. So with that I want to thank my work mother, Terry Tigner, for all that she's done for me and for this research group. Thanks, and I love you! I also want to thank Gary Li for keeping our servers up and running, and for putting up with our constant requests and dealing with them gracefully. I also want to thank Dan Fetko, Yadi Weaver, and David Durocher. Thanks for making this research work possible!

Actually, one more shout out. A big part of why I've pushed myself to this point is becaue the people closest to me are also doing big things. Two of my best friends from high school are finishing up their PhDs this summer: Gregg Tabot at University of Chicago in Computational Neuroscience, and Chris Gilmore at MIT in Aerospace Engineering. My best friend from college, Clint Smith, is currently pursuing his PhD in Education at Harvard. I'm in good company, and I couldn't be more greatful about. In the words of the wise Wiz Khalifa, "Hol' up hol' up, we dem boyz!!" Or as GOOD Music put it, "Ain't no body [messing] wit' my clique."

Thanks again to everybody in my life. I'm humbled, and excited for the future.

1 Introduction

1.1 Motivation for Low-Power FPGAs - Ubiquitous Computing

Today, in a world where people have immediate access to massive amounts of information, increased control and awareness of the surrounding environment is becoming more and more viable, and perhaps necessary. For this to occur, integrated circuits (ICs) need to be redesigned such that a large number of sensors can be deployed simultaeneously in different environments (wearable sensors, smart home sensors, infrastructure monitoring, etc.) and users can access the information provided by these sensors and respond accordingly. This landscape, known popularly as ubiquitous computing (UbiComp), is not far off. There are, however, certain restrictions in this application space that prevent UbiComp adoption on a large, commercial scale. These challenges are as follows:

- **Ultra-Low Power and Energy Operation** – For UbiComp to occur, there will need to be a large number of ICs deployed in various environments (homes, wearable sensors, etc.). Constantly changing batteries for all of these devices, or connecting all of them to wires, becomes invasive and infeasible. Therefore, these sensors need to be very-low power, so as to maintain functionality with a small power budget, and also be low energy, so as to not drain power sources too quickly. This will minimize battery changes, allowing the sensors to spend extended periods of time functioning.

- **Computational Flexibility** – With many sensors deployed in the field, constantly gathering information, removing them from their deployments can be problematic. Moreover, algorithms used for sensing applications are subject to change, either from discoveries of new methods, or from updated versions of the current approach. Additionally, these sensors are subject to different environments, and thus may need to perform slightly (or completely) different functions based on the environments. For all of these reasons, these sensors need to be designed with a level of flexibility, in order to meet all of the requirements of the environments they are sensing.

- **Computational Power** – It's not enough for these sensors just to sense information. If the devices simply sent all information collected to a user, it would be highly inefficient, as the sensor would send more information than is necessary, and the information would likely not be discernible by the user. Ideally, all of the sensors in this UbiComp platform will be able to process some of the information it collects on the node itself, in order to give the information to whoever needs it in a way that is easy to understand and work with.

- **Low Cost** – As mentioned before, UbiComp would require thousands, millions, and potentially billions of sensing nodes to be deployed all at the same time. The cost of developing and fabricating these chips would be very high, and that cost goes up even further if these chips need to be re-designed and re-fabricated with each new or updated algorithm.

Unfortunately, current IC options cannot meet all of these requirements. Ultra-low power (ULP) Application Specific Integrated Circuits (ASICs) provide the most efficient computing platform, and can function at low power consumptions with high computing power. Unfortunately, they have little or no flexibility, so updating or changing the functionality of the ASIC would require a full re-design and re-fabrication. This inflexibility also contributes to higher costs for implementation. General purpose processors (GPPs) that are targeted for low power, like ULP microcontrollers (MCUs), like the MSP430 from Texas Instruments, boast low power operation high flexibility and computing power[15]. However, the power consumptions of these devices is still too high for certain applications within the UbiComp space.

A more viable option is a ULP system-on-chip (SoC), like the Body-area Sensor Node (BSN) described in [27]. This device leverages both MCU and ASIC blocks to get extremely low power consumptions (<20 μW), allowing the node to run on harvested energy. Battery-less operation is *perfect* for UbiComp applications. Unfortunately, devices like this one still suffer from lack of flexibility. Although the node can perform many functions (ECG, EEG, aFib detection, etc.), it is limited to those functions, and needs to be re-fabricated and re-designed for any changes or updates.

Field Programmable Gate Arrays (FPGAs) are reconfigurable ICs that, at nominal voltages, split the difference between ASICs and GPPs in terms of flexibility and efficiency. The reconfigurability of the devices makes them inherently more flexible than ASICs, as they can be reconfigured to any functionality that fits on the logical resources available. Because FPGAs are still hardware implementations of algorithms, they still consume less power and energy than GPPs for many applications. The flexibility of FPGAs is limited to the amount of logic resources on the FPGA. Both ASICs and GPPs have been retargeted for ULP operation, as discussed earlier. However, there are fewer options for ULP FPGAs. This is primarily due to the current market for FPGAs. Applications today that primarily use FPGAs include aerospace, defense, automotive, high-power consumer electronics (like digital cameras), and medical devices (ultrasound, endoscopes, etc.), among others[13]. For these applications, power and energy consumption are not as important as performance. Thus, the FPGA industry has been driven by a different set of metrics, and designs for FPGAs have progressed accordingly. That being said, there are FPGAs from companies like Microsemi and Lattice Semicon-

ductor that boast extremely low power consumptions[6][29]. But again, those devices still consume 10s of mW in active mode, which is high for the UbiComp requirements. If sub-mW FPGAs were to be realized, they would provide an ideal solution for UbiComp applications; a power-efficient computing platform that is also flexible and powerful. This proposal argues that if the circuits and architectures of FPGAs are retargeted for ULP operation, FPGAs can fill the space needed to realize UbiComp pervasively.

1.2 Motivation for Configuration Scheme for Custom-FPGAs

In order to adequately test whether circuit and architecture optimizations are valid for minimizing power consumption in FPGAs, it will be important to test how incremental changes effect the FPGA circuit as a whole. To do this, we will need to build schematics of FPGAs with the different circuit-level and architectural parameters, and configure them to perform different functions. This poses two major problems.

- **Design Time** FPGAs are large structures, which require the use of a large number of transistors. One single configurable logic block (CLBs) consisting of 9 4-input basic logic elements (BLEs) can have as many as 20,000+ transistors (including test structures like registers). Thus, building entire FPGAs, which include multiple CLBs and a global interconnect, is very time consuming. Multiply that by the different FPGAs required for a thorough design space exploration, and comparing FPGA parameters through SPICE-level simulations becomes virtually impossible by hand.

- **Configuration for Custom-built FPGAs** As it stands now, there is no commercially available way to configure a custom-built FPGA fabric. Commercial FPGA companies, like Xilinx and Altera, have their own compilers that work for mapping a configuration bit stream that can be loaded to their hardware to implement verilog code provided by the user. Unfortunately, this method will not work for custom-FPGAs, because the custom hardware (circuit designs and architectures) doesn't match the commercial tools. Configuration bit locations are different, so the configuration bit stream wouldn't configure the same configuration bits. Configuration mechanisms vary from FPGA to FPGA, so it's entirely possible that the configuration bit stream generated by the commercial tool would be completely invalid for the custom FPGA built. Moreover, the underlying architectures will vary for commercial FPGAs, meaning that algorithms will be partitioned to different amounts of logic and interconnect resources. For all of these reasons, using commercial configurations is impossible for custom

fabrics.

What is needed is a method of taking a set of circuit level and architectural parameters (which define your physical FPGA structure) and a function for the FPGA to implement, and generating from those a netlist ready for simulation.

1.3 Thesis Statement

By re-targeting circuit elements and architecture parameters of reconfigurable fabrics for ultra-low power operation, FPGAs can provide the adequate combination of efficiency, flexibility, and computing power to enable ubiquitous computing to become a reality. These fabrics could be deployed as stand-alone ICs, or be included in ultra-low power SoCs to increase their flexibility. In order to study FPGA circuits and architectures fully, the FGC toolflow is created to generate FPGA fabrics, and configure them with algorithms used in ULP applications, in order to see how they will function. The tool extends open source FPGA mapping tools (i.e. the Verilog-to-Routing (VTR) flow [26]) to generate configurations for FPGAs. These generated FPGAs are used to explore circuit-level and architectural parameters to find optimal solutions for ULP FPGAs to target ULP applications like UbiComp.

1.4 Goals

The overarching goal of this work is to explore and determine circuit designs and architectures to allow FPGA operation with extremely low power consumption (<1mW). The individual goals of the work are as follows:

- Develop a tool-flow enabling circuit-level and architectural co-optimization for FPGA fabrics
- Explore viable circuit designs for FPGA sub-circuits (logic blocks, interconnect structures, and configuration bits)
- Determine best practices for FPGA sub-circuit design
- Re-visit FPGA architectures to determine if there are better architectural parameters for ULP FPGAs
- Determine optimal FPGA architectures for different classes of ULP applications
- Leverage toolflow to create FPGA fabrics that perform tasks with ultra-low power and energy consumptions

- Use toolflow to help create embedded reconfigurable fabric to be used in ULP SoCs, illustrating the benefits that ULP FPGA fabrics can provide

2 Background

2.1 General FPGA Architecture

FPGAs are reconfigurable ICs that consist of a collection of distributed logic blocks whose inputs and outputs are connected to each other through a reconfigurable interconnect, filled with switches and buffers. Configuration bits (usually SRAM) are distributed through the FPGA and control the reconfigurable interconnect and the logic, and these configuration bits, if altered, can change the functionality of the logic blocks, the connections between different logic blocks, or both. The I/Os to the FPGA chip also include configuration bits that determine whether it will be an input or an output, allowing flexibility in the number of input and output pins the device has. A high-level illustration of the general FPGA architecture is shown in Figure 2.1.

Each configurable logic block (CLB) consists of one or more Basic Logic Elements (BLEs). These

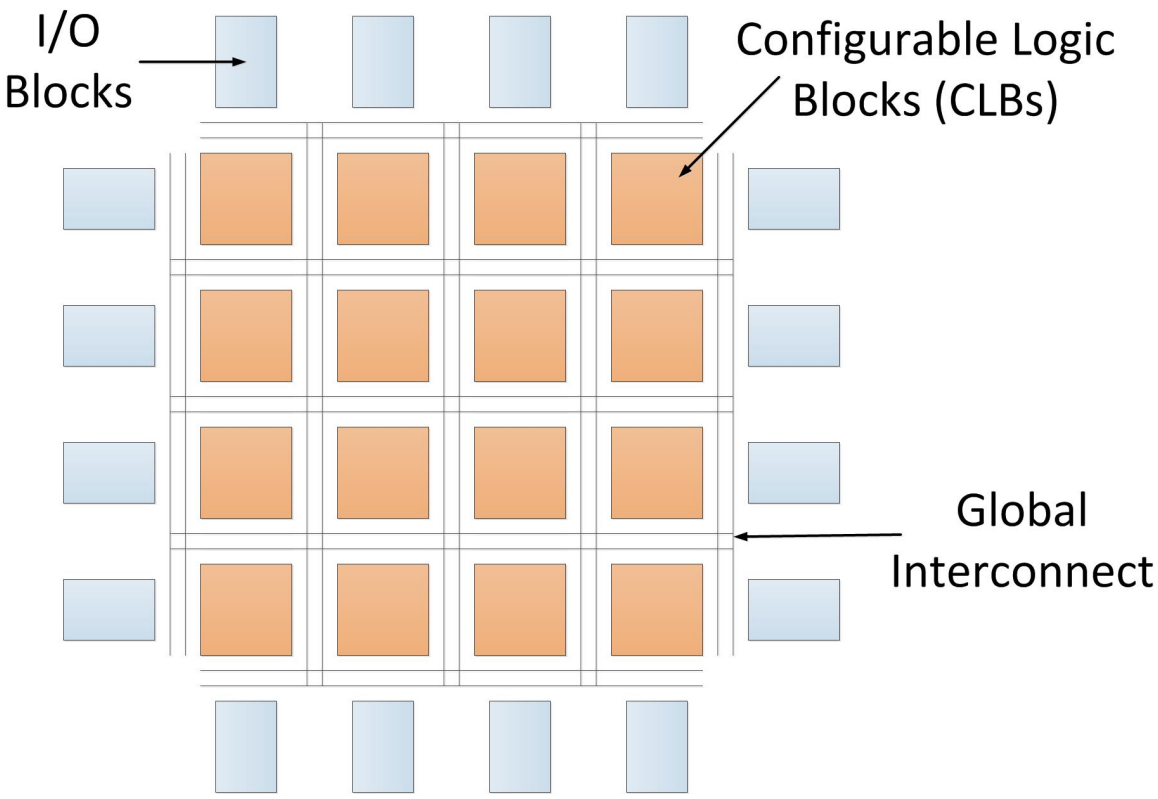

Figure 2.1: A 4x4 2D-mesh FPGA. Configurable Logic Blocks (CLBs) are distributed and connected to each other and input/output (I/O) blocks by a reprogrammable global interconnect.

BLEs include a look-up table (LUT), a flip-flop (FF) and a multiplexer (mux). A k-input LUT is implemented by 2^k SRAM configuration bits that hold the truth table values of any k-input Boolean function. Those bits are then fed into a 2^k-to-1 mux, whose select signals are controlled

by the inputs of the BLE. The output of teh LUT is then connected to both an additional output mux and the input of a FF, whose output is connected to the other input of the output mux. The output mux is controlled by a configuration bit, which determines if the BLE is purely combinational, or if it will utilize the FF to perform sequential logic. A detailed image of the BLE is shown in Figure 2.2. The global FPGA interconnect is generally organized as a 2-dimensional mesh structure

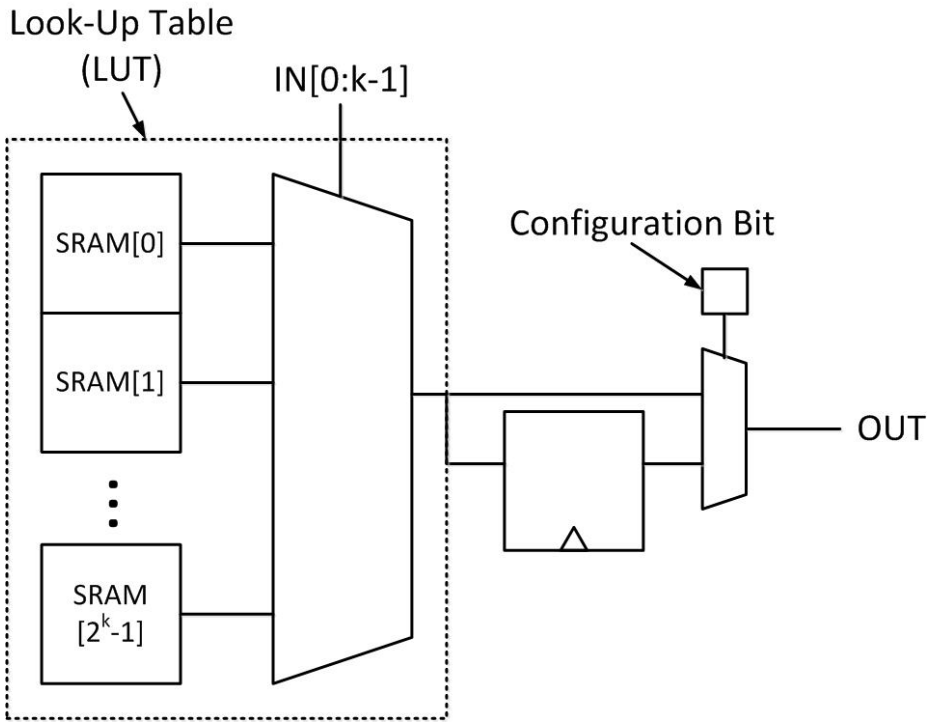

Figure 2.2: A basic logic element (BLE). One k-input look-up table (LUT) has 2^k SRAM configuration bits that are connected to a 2^k-to-1 multiplexer. The LUT is either configured to be combinational or sequential by a multiplexer and a configuration bit at the output.

with horizontal and vertical wires. In 2D-mesh FPGAs, each individual wire is considered a "track," and the collection of adjacent tracks in a given direction are called "channels." Horizontal and vertical channels are connect to each other through switch boxes. Connection boxes connect the CLBs to the FPGA interconnect wires. Configuration bits control which tracks are connected to each other and to the CLB inputs and outputs. Figure 2.3 gives a detailed picture of interconnect resources and how they are connected.

2.2 Commercial Ultra-Low Power FPGAs

Most of the market share for FPGAs is shared by Xilinx (47%) and Altera (41%) as of 2012[31]. However, these companies do not make FPGAs with low power consumptions. These companies'

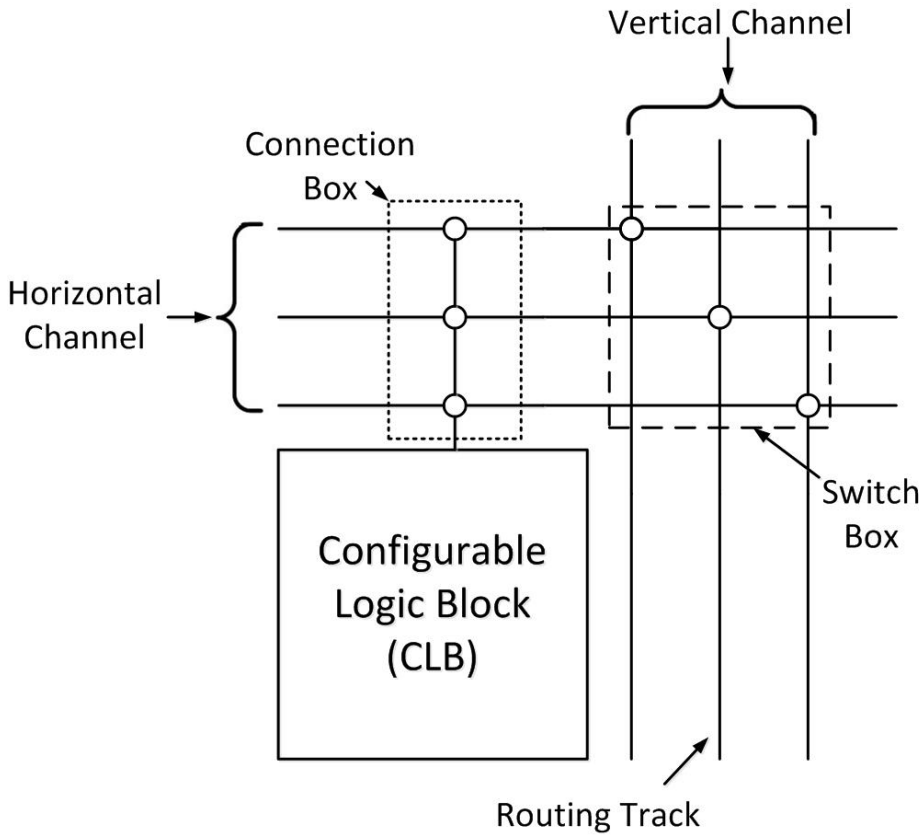

Figure 2.3: Programmable FPGA interconnect. Connection boxes connect signals from CLBs to individual routing tracks of horizontal and vertical routing channels. Switch boxes connect the channels to each other.

current consumption are consistantly in the mW-W range, which is too high for ULP applications. There are companies, however, who do make FPGAs with power consumptions in certain modes that are sub-mW, and their circuit designs and architectures are described in this section.

2.2.1 Microsemi IGLOO nano

One of the competitors to Xilinx and Altera is Microsemi, a company that makes some of the lower power FPGAs on the market. Their devices use non-volatile FLASH configuration bits instead of the SRAM bits that most industry FPGAs use. By using these, they can get leakage currents for their devices as low as 2 μW. The smallest logical element in the IGLOO nano, which is analogous to a BLE, is called a VersaTile, which is a 4-input block that can have one of three functions:

1. 3-input LUT

2. Latch with clear or set

3. D flip-flop with clear or set

The IGLOO has no explicit clustering of their VersaTiles, but clusters are implicitly created by the routing structures in the FPGA. There are multiple levels of hierarchy in the interconnect of the IGLOO. First, there are Local-Line resources that connect every VersaTile with its eight nearest neighbors. Next, Long-Line resources span 1, 2, or 4 VersaTiles, in either the x or y direction. Lastly, there are Very-Long-Line resources that act as the global interconnect for the FPGA, which span 12 versatiles in the x-direction and 16 VersaTiles in the y-direction. Because of the Local-Line interconnects, we will consider the cluster in these FPGAs to be 9 VersaTiles [6].

2.2.2 Lattice iCE40 Ultra

Lattice Semiconductor is a company who designs SRAM-based FPGAs for low-power operation. Their iCE40 device boasts typical standby currents of 71 μA, with IP cores including embedded block RAMs, communication interfaces, and DSP blocks. The device clusters 8 4-input LUTs into each programmable logic block (PLB). Like other commercial FPGAs, their logic blocks have dedicated carry logic for more efficient arithmetic functionality. Their routing structure consists of three different segment lengths spanning 1, 4, and 12 PLBs [29].

2.3 Academic Ultra-Low Power FPGAs

While some commercial FPGAs (like the Microsemi IGLOO and the Lattice iCE40) have sub-mW operation, they only achieve this low power consumption in sleep mode. There have been academic ventures to design ultra-low power FPGA designs that consume low power actively. One example is introduced in [9], which features a low power FPGA that consumes as little as 40 μW in the active mode. This FPGA uses 6T latches instead of SRAM bits that are used in commercial FPGAs. For interconnect resources (such as connection boxes and switch boxes), this FPGA uses buffered, uni-directional wires with buffered multiplexers at the switch points. Limitations to this FPGA design come mostly from the size of the device. This FPGA achieves record-low power consumption, but only has 4 configurable logic blocks. That severely limits the computation power of the device, and almost assuredly is not enough resources for many of the DSP algorithms needed for UbiComp.

Another low-power FPGA, proposed in [33], uses the Xilinx Stratix III architecture as its baseline. Thus, it also uses standard SRAM configuration bitcells and buffered unidirectional multiplexers for the interconnect routing as well. This FPGA included IP blocks like multipliers and block RAMs. The low-power optimizations employed in this FPGA include mid-oxide, high-VT transistors in the

Table 1: FPGA architectures of commercial and academic low-power FPGAs

FPGA	Size (# of LUTs)	Power (μW)	Config. Bit Topology	Frequency (MHz)
Latice iCE40[1]	384-7680	Static: 21-250 Active: just $\downarrow$ 1k[7,8]	SRAM	275
Microsemi IGLOO nano[1]	100-3000	Static: 2 Active: 400[6]	FLASH	160-250
Ryan et. al. [28][2]	1134	Static: $\sim$ 35[3,4] Active: $\sim$ 12.5[3,4]	5T-SRAM	$\sim$ 33[3]
Grossmann et. al. [9][2]	128	Static: 8.9 Active: 34.6	6T Latch	16.7
Tuan et. al. [33][2]	1500-15000	Static: 46-460 Active: 13k-130k	SRAM	244[5]

[1] – Commercial ULP FPGAs
[2] – Academic ULP FPGAs
[3] – Estimated from plots in the paper
[4] – Simulation result for 780 LUTs
[5] – Reported approx. 27% reduction from Xilinx Spartan-3
[6] – Obtained from Microsemi Power Calculator worksheet
[7] – Mid-range iCE40 model
[8] – From news article in EE times: Ultra-low power FPGAs enable always-on sensor solutions for context-aware mobile apps

configuration bits. They also power-gate unused resources, and have a stand-by sleep mode. With these techniques, this FPGA consumes 13-130mW of active power and 46-460 W of sleep power for 1500-15000 logic cells (BLEs).

In [28], researchers also used SRAM bitcells for configuration, but used 5T cells as opposed to the standard 6T. This made the cells harder to write, but because the cells are generally in the hold state, using 5T configuration bits only requires extra effort during the configuration stage, in the form of boosted voltages for the writing circuitry. For the interconnect, this FPGA doesnt use buffers at all, but instead uses a purely pass-gate interconnect. To make up for the degraded swing across the interconnect, a modified-Schmitt Trigger sense amp is used at the inputs to the logic blocks in order to restore the signal to full swing. This interconnect leverages the low-swing that occurs throughout the interconnect to reduce power. Using these circuit design techniques, researchers were able to reduce area per LUT by almost 3x, delay at a constant energy by 14x, and energy at a constant delay by almost 5x.

Table 1 shows different low-power FPGAs and their architectures. Both the commercial and academic FPGAs cover a wide range of capacities. In terms of power consumption, only the academic have power consumptios that fall within the UbiComp specs (<1 mW). The different FPGAs use a variety of bitcell topologies, which inspires the bitcell exploration in chapter 5. It is also important to note that all of the frequencies in this table are much higher than necessary for UbiComp

applications, which range from 100s of kHz to MHz. That suggests that there is room for further reducing power for FPGAs at the expense of performance.

3 FPGA Generation and Configuration (FGC)

3.1 Motivation

There are many circuit and architectural knobs that need to be turned in order to design an FPGA that is optimized for ULP functionality. While testing each individual knob (such as switch topology) can provide interesting information, its also very important to see how these knob changes affect the functionality of the FPGA as a whole. FPGA modelling (like what is done in VTR) can provide some notion of how the full FPGA fabric behaves with different circuit-level and architectural parameters, but a more accurate representation of the circuit performance would be to conduct SPICE-level simulations of FPGA fabrics. To do this, it is necessary to build full FPGA schematics, employing the different circuit-level and architectural parameters to be tested, configuring them to perform some function, and simulating them. However, there are many challenges to conducting these FPGA-level simulations. First, we are exploring multiple knobs, creating potentially 1000s of different combinations for FPGA fabrics. Each one of those FPGAs is also a large IC, using 1000s of transistors. As a result, building the necessary FPGA schematics by hand becomes impossible, given time and monetary constraints. Secondly, in order to simulate these FPGAs, they need to be configured to perform certain tasks. Unfortunately, no tool is readily available for configuring custom-built FPGAs. Commercial FPGAs have their own compilers for configuration, but these software packages are specific to their own FPGA products, and will not work for our custom-built FPGA fabrics. In this chapter the FPGA Generation and Configuration (FGC) tool is introduced. This tool not only generates FPGA fabrics based on circuit-level and architectural parameters, but that can also create configurations for those FPGA schematics in order to conduct SPICE level simulations.

The design space of FPGA hardware design can be abstracted to 3 axes of flexibility: circuit parameters, architecture parameters, and benchmark circuit designs. Commercial FPGA companies, like Xilinx or Altera, allow for large flexibility in terms of the types of circuits that can be mapped to them, but are extremely limited otherwise. They provide only one alternative in terms of circuit parameters, and a small set of different architectures, ranging from smaller, low-end devices to large high performance devices. The VTR tool [26] is slightly limited in the types of benchmarks it can map (due to the inflexibility of the included CAD tools to Verilog syntax), but has a large range of architectural flexibility, meaning it can handle a large design space in architectural parameters. Our proposed tool-flow will cover a much larger portion of the possible design space, and be able to

cover both architectural and circuit-level knobs while also leveraging a variety of benchmark circuit algorithms. Figure 3.1 gives a graphical representation of the design space and how other tool-flows fit into it.

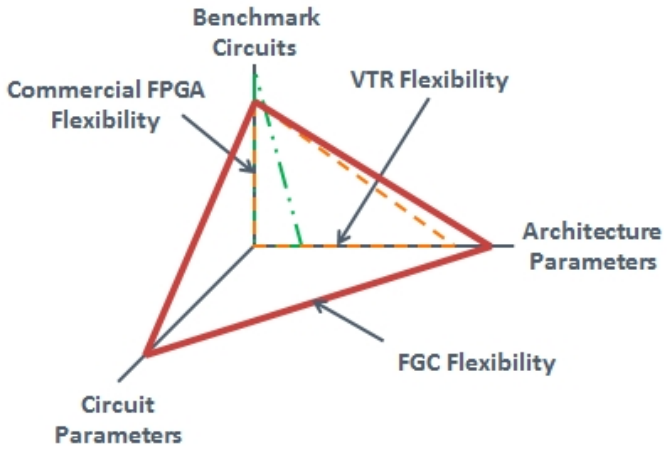

Figure 3.1: Graphical representation of FPGA hardware design space. Proposed tool-flow will address all three portions of the design space to move beyond other tools capabilities.

3.2 FGC Overview

The FPGA Generation and Configuration (FGC) tool-flow, which I created, allows users to quickly generate full-FPGA schematics, configure them to perform any Verilog-based function, and prepare simulations to be run by the user to observe metrics of interest. It does so by extending the VTR toolflow [26], as illustrated in Figure 3.2. This tool-flow leverages the VTR toolset to configure an FPGA with the architecture of our choosing, Cadence SKILL scripts to build schematics, and perl code to edit and generate scripts and text files, and control the different tools used. The FGC flow primarily targets researchers looking to build custom-FPGA fabrics, and allows them to explore new circuit- and architecture-level design choices, and see the effects of these changes at an FPGA-system level. This tool is also useful for designers making systems-on-chip (SoCs) looking to potentially include FPGA fabrics. Additionally, the FGC tool requires minimal user input, allowing a user who is not an expert in FPGAs to still use the tool-flow. Figure 3.3 is a flow-chart describing the FGC toolflow.

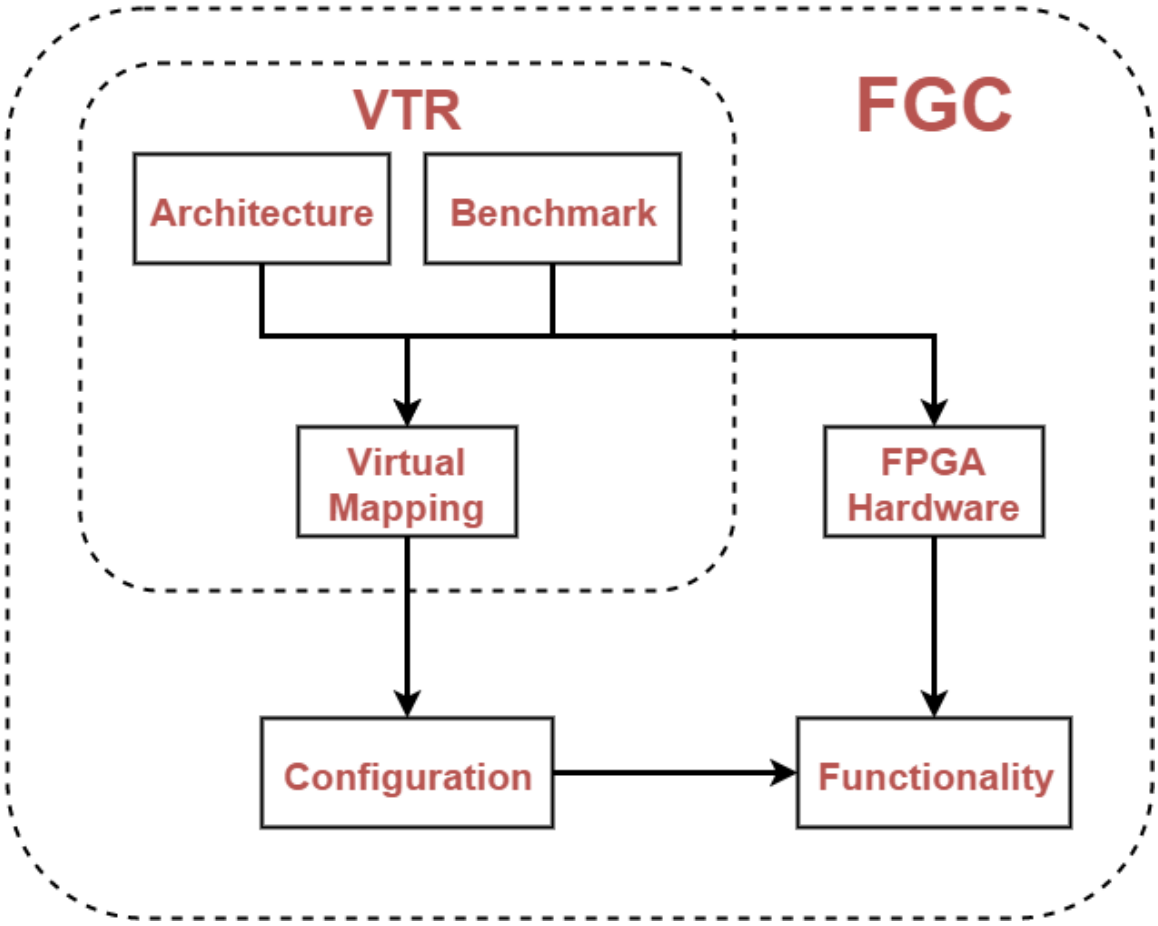

Figure 3.2: High-level picture of the FGC flow. The capabilities of the VTR flow [26] are extended to include generated FPGA schematics and simulation results for given benchmark circuits.

3.3 Prior Art

Various methods for generating configurations for custom-built FPGAs exist, but with varying limitations. In [30], researchers develop a bitstream generator, called DAGGER, as part of their larger research effort to create a custom FPGA. DAGGER focuses on creating bitstreams for their specific FPGA architecture and includes additional functionality, such as partial reconfigurability, bitstream compression and encryption. [30] also highlights the many other bitstream generators/editors ([10],[23],[11],[25]), but all of these tools target Xilinx FPGAs specifically. The FGC tool described in this dissertation has different goals, and is instead designed for allowing design space exploration across many different architectures and circuit-level designs.

In [17], researchers at University of Toronto extend the VTR toolflow to include physical synthesis and place and route using commercial synthesis tools (i.e. Cadence, Synopsys, etc). The flow also generates bitstream configurations for the synthesized FPGA. This provides a path toward SPICE-

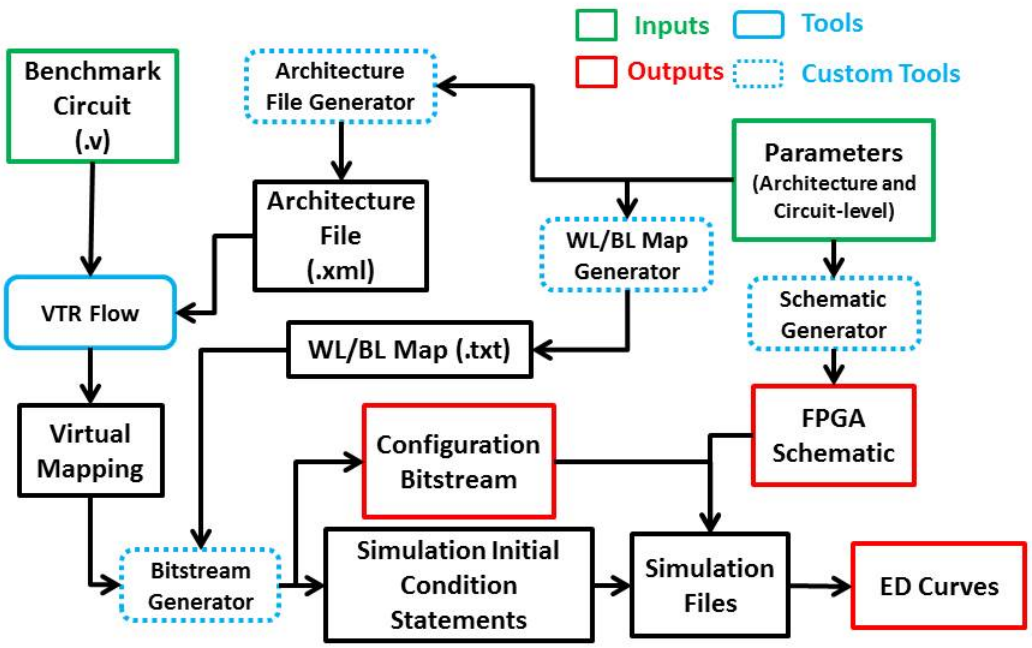

Figure 3.3: Flow-chart of the FGC flow. The inputs are a parameter file explaining the target FPGA circuitry, and a benchmark Verilog circuit describing the target FPGA functionality. The flow creates a full FPGA schematic and configuration files representing the functionality, and can be used to generate simulation results.

level simulations for FPGA fabrics that at least span the flexibility of the VTR tool. The FGC tool developed for this dissertation takes this level of flexibility a step further, and includes circuit-level parameters (like transistor sizing, additional circuit topologies, etc.) that are not included in the VTR tool. Additionally, the tool provides initial condition statements for simulation as well as bitstream, giving the user direct control over individual configuration bits, to quickly assess functionality without needing to simulate configuration, which can take prohibitively long. These configuration generation schemes above have the following drawbacks, which are addressed by this new FGC flow that I have developed.

- **Bitstream configuation only** – Both DAGGER and the VTR extension simply create configuration bitstreams. Thus, to simulate FPGA functionality, the user must also simulate the configuration each time. This can take prohibitively long for larger FPGAs, and will greatly increase design space exploration time regardless of size.

- **Lack of flexibility** – Both of these approaches have limits in the amount of the design space that can be explored. The DAGGER flow is made specifically for the AMDREL FPGA, and only works with FPGA fabrics with similar architectural and circuit-level parametes. The VTR extension has increased flexibility, but is limited to the level of flexibility of the VTR tool, which is primarily flexible with respect to architectural parameters (LUT size, CLB clustering, channel width, etc.)

- **Incomplete flow** – In order to efficiently explore the design space, one needs to vary multiple parameters, including circuit-level parameters, architectures, circuit functionalities, and operating conditions, and simulate across this large design space. In order to do this, FPGA fabrics need to be generated, configured, and simulated. The DAGGER tool and the VTR extension include one piece (configuration) of the puzzle.

The proposed FGC tool achieves all three goals. It completes the loop, going from design parameters and ending with simulations of FPGA fabrics. The configuration includes initial condition statements, allowing users to simulate FPGAs without having to also simulate configuration. This tool is the most flexible FPGA-generation or configuration tool to date, allowing for different circuit-level and architectural parameters, operating conditions, target Verilog circuits, and simulation options.

3.4 Running the FGC flow

Figure 3.4 outlines the steps to perform in order to run the flow. The flow is divided into 3 sections: mapping, schematic generation, and simulation. First, the user sets up the toolflow by creating a parameter file, which is the main input for the toolflow. This file provides descriptions of both architectural and circuit-level parameters for the target FPGA to be generated. Next, the user places verilog code for the benchmark(s) to be mapped to the target FPGA within the FGC file structure in the correct directory. After placing the verilog files, the first portion of the toolflow can commence, which generates the necessary files for FPGA configuration, which can be either initial condition statements for the configuration bits, or a configuration bitstream which can be used either in a chip-level simulation, or for hardware configuration. Once these files are generated, the user then creates the schematics by setting up a library in the proper schematic environment, running a SKILL script to generate all of the FPGA schematics, and extracting the netlist of the full FPGA circuit. Following the schematic generation steps, the user can then run the second portion of the toolflow, which uses the mapping of the FPGA as well as the netlist to generate a set of simulation files, and kick off a simulation to observe metrics such as delay and energy. In the

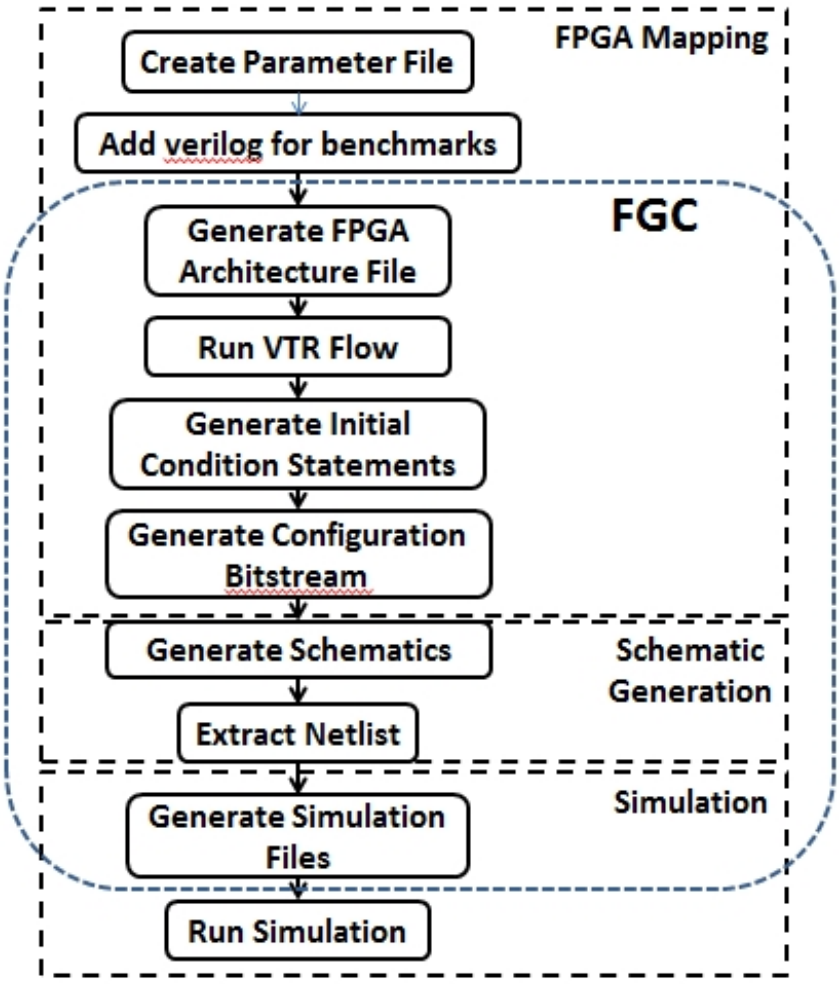

Figure 3.4: Steps for running the toolflow. The toolflow is divided into 3 sections: FPGA mapping, schematic generation, and simulation. File names and directory locations are also provided.

following sub-section, I give a more detailed, step-by-step tutorial for running the FGC Flow.

3.4.1 Step-by-Step FGC Flow

Step 1: Create Parameter File The first step for the toolflow is generating the input parameter file, titled **<task name>.txt**, where **<task_name>** will be the name that will organize all of the outputs of the flow. A template parameter file is included with the flow, and copying the template, renaming it, and changing the values of the parameters ensures that all of the necessary parameters are set. Figure 3.5 illustrates the parameter file opened in a text editor.

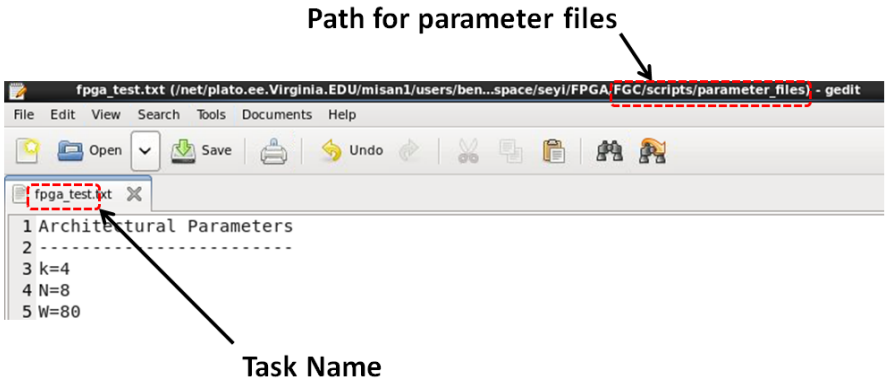

Figure 3.5: Excerpt from an example parameter file. The **task name** (used to organize the outputs) and the file location are highlighted.

Step 2: Add Verilog Circuits to FGC Flow Before running the flow, the user needs to add Verilog descriptions of the circuits that are to be mapped to the generated FPGA for testing. These Verilog files are to be placed into the **FGC/vpr_files/verilog/<task name>** directory, which needs to be created as well. Multiple Verilog files can be added, and the FGC tool supports multiple configurations in one run. The tool will generate multiple outputs, one for each circuit to implement.

Step 3: Edit and Run FGC Flow Now that the parameter file and Verilog circuits are in place, the user can kick off the toolflow. First, the driver script for the flow (**FGC.pl**) should be opened and edited. Only the top 3 lines of the script need to be changed. The **$parameter_filename** variable should match the task name in steps 1 and 2. **$fgc** is the location of the FGC flow, and **$fgc_output_dir** is the location for all of the outputs to be generated (called **FGC_OUT_DIR** from now on). Figure 3.6a shows the script and the lines of code to edit. Once the script is properly

edited, the script can be executed with the following command in the terminal:

perl FGC.pl

The script performs the following functions:

1. Runs the VTR flow to generate a virtual mapping of the FPGA
2. Generates initial condition (IC) commands for configuring FPGAs to perform target Verilog functions
3. Generates configuration bitstreams for true FPGA configuration of target Verilog functions
4. Generates full FPGA schematics
5. Generates simulation files (including netlists, stimuli, and driver scripts) for analysis of FPGA behavior

Figure 3.6 shows the terminal output after running the FGC flow. In this example, a 10x10 FPGA is generated, and configured with 10 4-bit counters. The total runtime for the flow is approximately 8 minutes, not including simulation time.

Step 4: Edit Stimulus Upon completion of the FGC flow, simulation files will be available for simulation, located in the **FGC_OUT_DIR/simulation_files** directory. A sub-directory exists for each target circuit that is mapped. For the simulations to be functional, the user must change the stimulus in order to see the functionality. The default stimuli are piece-wise linear (PWL) inputs for each input signal specified in the Verilog for the target functionality. The onus is on the user to set up the stimulus such that the critical path of the FPGA is exercised in the simulation. The simulation driver script (**sim.sp** for HSPICE and HSIM, **sim.scs** for Spectre or UltraSim) in the simulation file directory shows the input and output signals of the critical path, and uses those signals to calculate delay. It is up to the user to use that information to set up the stimulus properly.

Step 5: Run Simulation Once the stimulus is edited, simulations can be run using the generated files. To run the simulation, navigate to the **FGC_OUT_DIR/simulation_files/<simulator>/ <circuit>** directory and run the following command:

bash run_sim.sh

Simulation data is saved to the **FGC_OUT_DIR/sim_results/<circuit>** directory, and delay and energy calculations are saved into **<circuit>.mt** file in the **FGC_OUT_DIR/simulation_files/ <simulator>/<circuit>/<voltage>** directory.

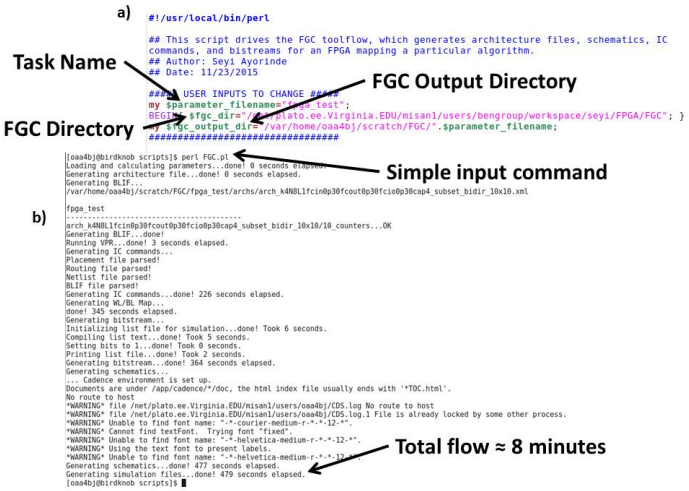

Figure 3.6: a) FGC.pl driver script, with task name, FGC directory, and FGC output directory highlighted. b) Single FGC run for 10x10 FPGA. Schematic, initial condition, bitstream, and simulation file generation all together takes about 8 minutes.

3.5 Architectural Assumptions in FGC

There are a set of assumptions that I have made in designing this FPGA-synthesis toolflow.

3.5.1 Inputs to CLBs

The number of inputs to the CLB is a variable that has been well studied. In this dissertation, I use the equation derived in [2], namely that

$$I = k\left(\frac{N+1}{2}\right)$$

where I is the number of inputs to the CLB, k is the number of inputs to an LUT, and N is the number of BLEs in the CLB.

3.5.2 Word-line (WL) and Bit-line (BL) configuration

In this toolflow, the FPGA is assumed have its configuration bits (CBits) arranged through a network of horizontal word-lines (WLs) and vertical bit-lines (BLs), similar to an SRAM. WLs control access

to CBits, and BLs control the value written into the CBits. Figure 3.7 illustrates the configuration mechanism. The tool does not yet support single bitstream configuration topologies, where each configuration bit is part of a large scan chain that can be scanned in through one port. Future work will include expanding the FGC flow to support additional FPGA configuration topologies.

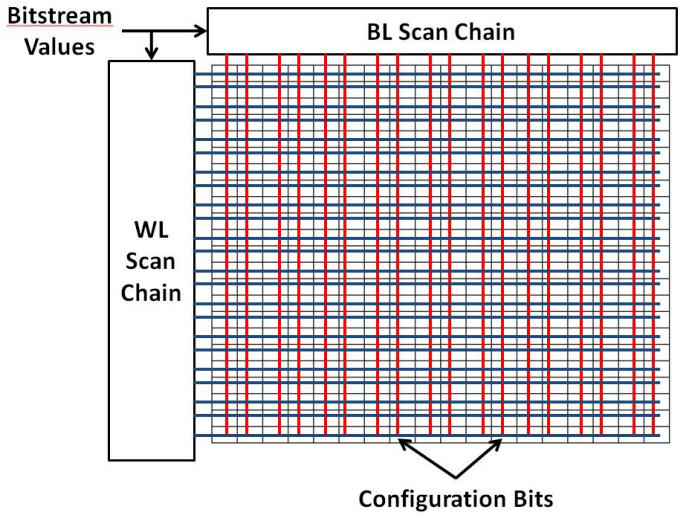

Figure 3.7: Dual scan-chain configuration for FPGAs. Bitstreams are generated for both the WL and the BL scan chains, such that each configuration bit (CBit) is written with the proper values. WLs control access, and BLs control values to be written into the CBits.

3.5.3 Depopulation Implementation

Multiplexer Depopulation (as discussed in [20]), refers to accessing less than 100% of the possible inputs for basic logic element (BLE) inputs. Without depopulation, multiplexers for BLE inputs can select from all of the inputs and outputs of the larger configurable logic block (CLB), which includes one or more (N) BLEs. With depopulation, the muxes have a percentage of the total signals to choose from. The FGC flow implements depopulation by accessing the proper proportion of inputs and outputs. An example is shown in figure 3.8. When the depopulation is 50%, the BLE input connects to half of the inputs, and half of the outputs. Complete connectivity to the BLE is achieved over multiple inputs, unless the depopulation is less than $\frac{1}{K}$, where K is the number of inputs to the BLE. The first $N \times [depopulation]$ BLEs connect to the first subset of the signals, the second set of BLEs to the second subset of signals, etc. For example, if the depopulation is $\frac{1}{3}$ and $N = 9$, then BLEs 1-3 connect to the first third of the inputs and the first third of the outputs, BLEs 4-6 connect to the second third of inputs and outputs, and BLEs 7-9 connect to the rest.

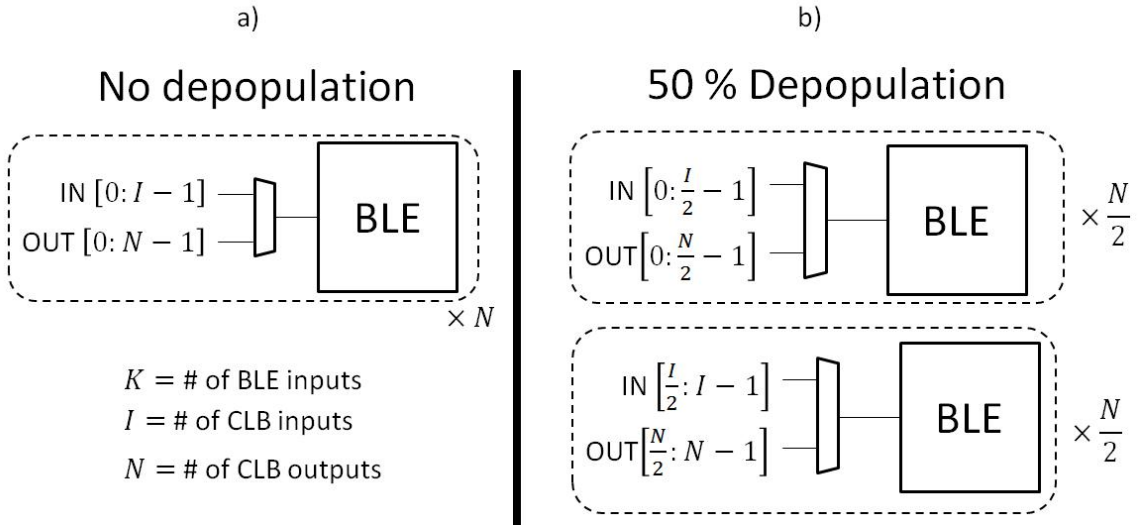

Figure 3.8: Example of BLE input multiplexers with a) 0% and b) 50% depopulation.

3.6 FPGA Tile

In this tool-flow, FPGA resources are organized into "tiles," which include a single CLB, all of the connection boxes attributed to that CLB, and the switch box in the top-right hand corner. Tile coordinates go from (0,0) to ([fpga width+1],[fpga height+1]). Tiles on the edges lack CLBs and include fewer connection boxes. Tiles on the right and top edges also have switch boxes removed, as the neighboring tile has the necessary switch box. The different blocks are labeled in Figure 3.9.

3.7 FGC Directory Structure

In this section, I explain the different directories within the FGC toolset, illustrated in Figure 3.10. The directory structure is very important, and needs to be maintained in order to guarantee tool functionality. There are a set of directories and files that are also generated by the tool, and the directory structure of these output files is also explained in this section. More information about the individual files in each directory can be found in the next section, entitled File Descriptions.
The main FGC directory (**FGC**) has four directories inside:

- The **cadence** directory is the assumed launch location for Cadence Virtuoso, which will be used to generate the schematics for the FPGA. Because the schematic generation files are written in SKILL, Cadence Virtuoso is required for running them.
- The **scripts** directory houses the driver scripts for the FGC flow, as well as the modules which hold the sub-routines for generating all of the necessary files, and parsing all of the intermediate

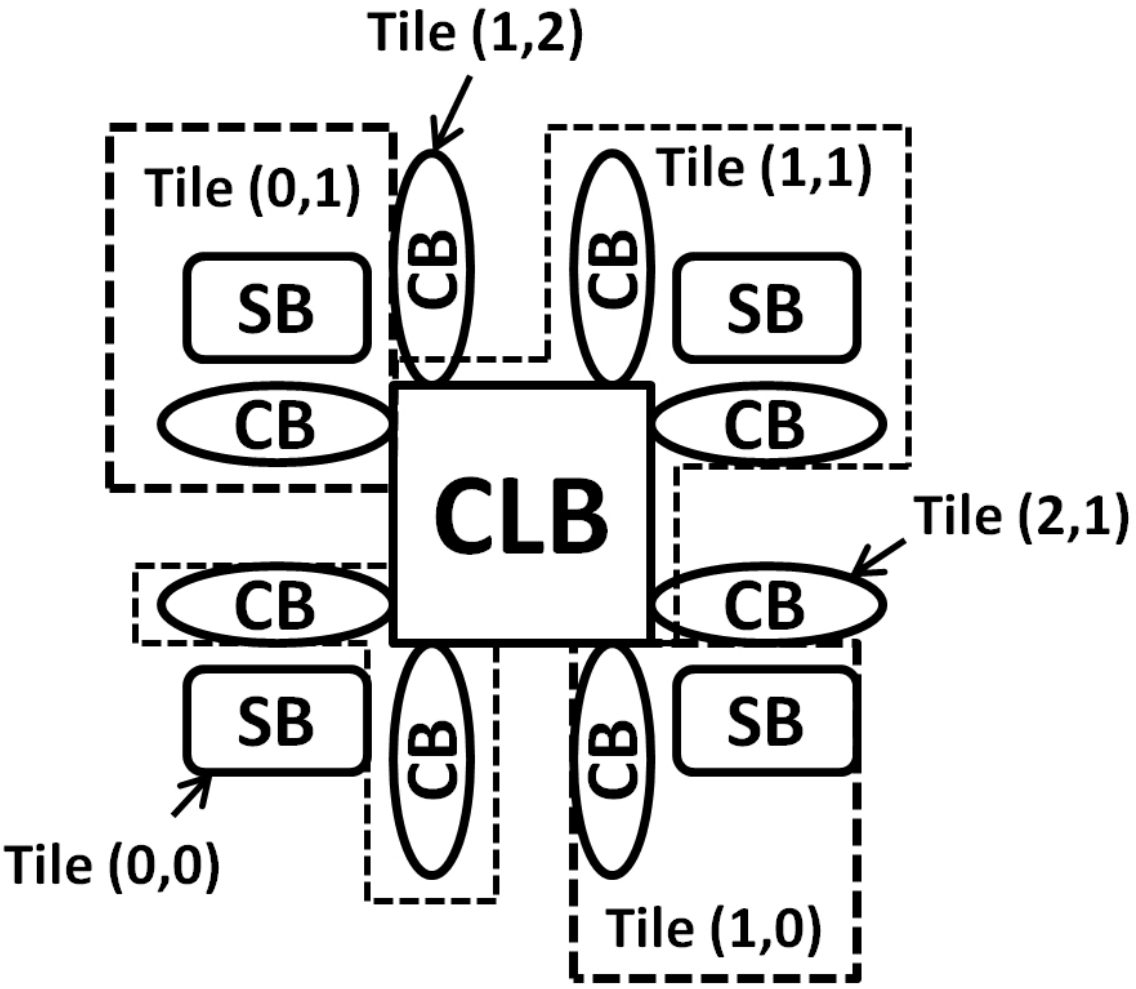

Figure 3.9: CLB tile organization, illustrated in a 1x1 FPGA. CLB Tiles include CLBs, connection boxes, and a single switch box. Left and bottom edge tiles include connection boxes and switch boxes. Right and Top edge tiles include only connection boxes. The tile in the bottom left corner (tile 0,0) has only a switch box.

files. The user executes the FGC flow in this directory.

- The **vpr_files** directory includes template architecture files for potential FPGAs, BLIF (Berkeley Logic Interchange Format) files, and Verilog code for potential benchmark circuits.

- The **vtr** directory is the open-source Verilog-To-Routing (VTR) mapping tool developed by University of Toronto. The FGC flow extends this tool to allow for schematic-level FPGA generation and configuration. The VTR flow can also be run stand-alone, as it was originally intended.

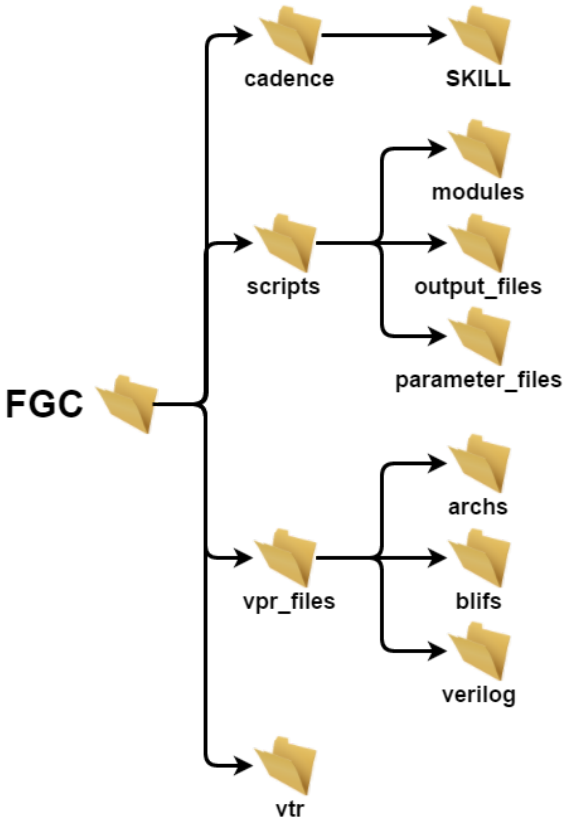

Figure 3.10: Directory structure for the FGC toolset. 'vtr' houses the Verilog-To-Routing (VTR) open-source tool. Most of the custom code resides in 'scripts'. Code to generate FPGA schematics is in the 'SKILL' directory

3.7.1 FGC Outputs Directory Structure

Once the FGC flow is completed, a set of output directories and files are generated. All of these outputs are generated in the FGC output directory (**$FGC_OUT_DIR**). The location of that directory is specified by the user in the FGC driver script. The output directory structure is illustrated in Figure 3.11. The contents of the individual files will be discussed in greater detail in the File Descriptions section.

The following list describes each directory in the **$FGC_OUT_DIR** directory:

- The **archs** directory is the location of the generated VPR architecture file, which provides descriptions of the target FPGA hardware used to map algorithms to a virtual FPGA through the VTR flow.

- The **activity factors** directory houses the activity factor file generated by the ACE tool, which is part of the power estimation flow (VersaPower) in VTR.

- The **blifs** directory is the location of the generated Berkeley Logic Interface Format (BLIF) file, which describes the logic of the target benchmark circuits to be mapped using the VTR flow.

- The **extracted_netlist** directory is where the schematic for the full FPGA should be extracted after it is generated, allowing for successful simulation.

- The **csv_files** directory holds a comma-separated value file that represents the bitstream for configuring the target FPGA. This file is generally used for the physical testing, as many pattern generators support CSV inputs.

- The **ff_output_mappings** holds the mapping between flip-flop names and verilog output signal names. The VTR outputs describe the FPGA output signals by the name given to the flip-flop that drives that signal. This directory is only generated for the mapping of sequential circuits.

- The **ff_signal_mappings** directory holds the mapping between the flip-flop name and the schematic net that refers to the Q-node of that flip flop. This allows for the delay calculations to choose the proper nodes in the schematics. This directory is only generated for the mapping of sequential circuits.

- The **ic_commands** directory holds the initial condition statements generated by the toolflow for the FPGA schematics, in order to properly perform the target benchmark circuit functionality.

- The **list_files** directory holds a representation of the stimulus file for streaming in the configuration bits. This is list file is used to generate other versions, including a comma-separated value (CSV) file (used by many pattern generators to generate stimulus for a chip) and a piece-wise linear (PWL) file (used for simulation).

- The **parameter_lists** directory includes an updated version of the initial parameter list input to the toolflow, which includes calculated parameters as well. A full description of the additional parameters will be shown later in the document.

- The **schematic_generation_script** directory includes a copy of the SKILL code that is generated to create the FPGA schematic.

- The **schematic_libraries** directory is where the Cadence library to hold the FPGA schematics should be.

- The **simulation_files** directory is the location of the generated files (netlist, stimulus, etc) used for running simulations of the full-FPGA fabric.
- The **sim_results** directory is the waveform data for the simulation is generated as the simulation of the generated FPGA schematic progresses.
- The **track_map** directory holds the list of tracks that each FPGA I/O (both chip-level and logic-block-level) connect to. This is important for ensuring matching connectivity between the mapping that comes from the VTR flow and the schematics for the FPGA that are being generated by the SKILL code.
- The **vpr_outputs** directory is the final location of the files that are generated by the VTR flow. More information on each of those files will be given in the File Descriptions section of the document.
- The **wlbl_map** directory holds the generated WL/BL map file, which tells the WL and BL location of each configuration bit in the generated FPGA schematic.

3.8 File Descriptions

3.8.1 Parameter file (<task name>.txt)

The main input for the toolflow is a parameter text file, which is located in the *$fgc/scripts/parameter_files* directory. This file includes the following information:

1. architectural and circuit-level parameters for the target FPGA
2. options for running virtual place and route (VPR)
3. simulation options

Architectural Parameters The architectural parameters are as follows:

- **k** - number of inputs to each look-up table (LUT) in the target FPGA.
- **N** - number of basic logic elements (BLEs) clustered inside each configurable logic block (CLB) in the target FPGA.
- **W** - channel width, or the number of routing tracks in each routing channel. If left as *VPR*, VPR will determine the number of routing channels.

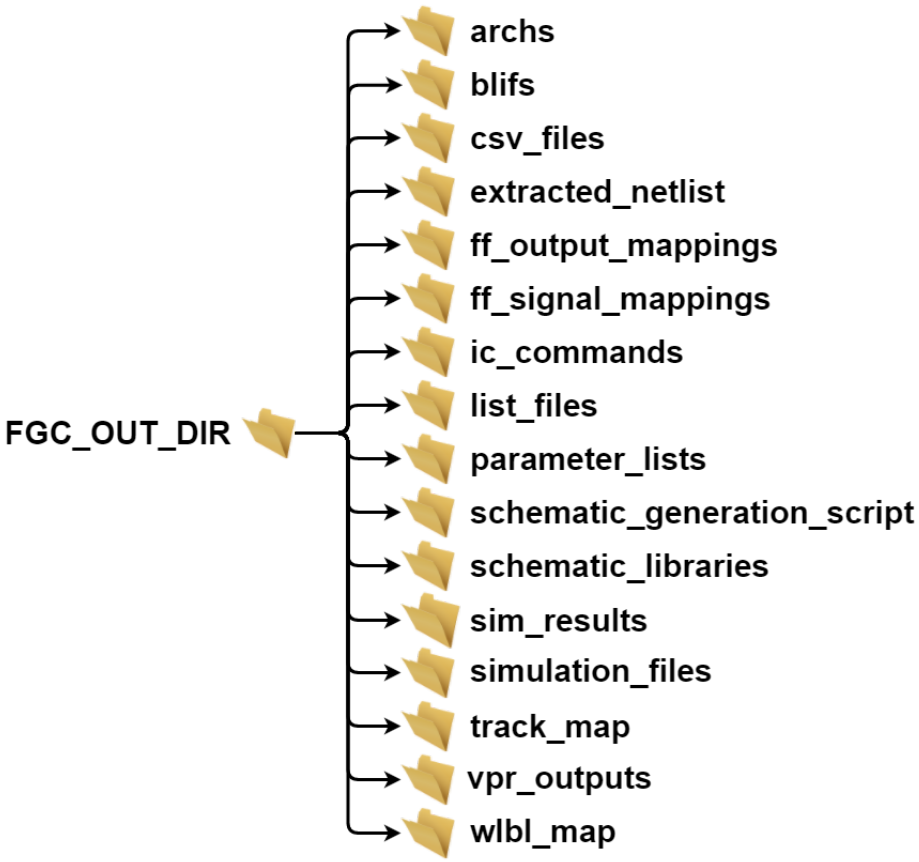

Figure 3.11: Directory structure for the generated FGC outputs. These outputs are the intermediate files for the toolflow, and provide visibility for debugging purposes. All of the directories are generated by the toolflow.

- **L** - segment length, or the number of CLBs each routing segment spans.
- **fc** - channel fanout, or the percentage of tracks in each routing channel that each pin connects to. The parameter file calls for 3 different channel fanout values: one for the input pins of logic blocks, one for the output pins, and one for the I/Os of the FPGA.
- **I/O capacity** - number of physical ports at each X-Y location for an I/O block. If I/O capacity is equal to n each I/O block in the FPGA will have n inputs and n outputs.
- **FPGA width and height** - overall dimensions of the target FPGA, in terms of number of CLBs. If left as *VPR*, VPR will determine the dimensions for the FPGA.
- **CLB width and height** - dimension of the CLB, in terms of number of BLEs.
- **Track directionality** - sets the tracks to be either uni-directional (signals can only pass in one direction) or bi-directional (signals can pass in both directions).

- **Logical Equivalence** - sets the logical equivalence of the inputs to logic blocks in the FPGA. A value of *true* means that the inputs are interchangeable, and are therefore logically equivalent. A value of *false* means the inputs to the logic blocks are not interchangeable.

- **Depopulation (multiplexer-based CLBs only)** - sets the amount of depopulation in the input multiplexers of each BLE. Values range from 0 to 1.0. Depopulation of 0.33 means that each BLE input multiplexer connects to 0.66 of the possible signals (inputs and outputs) of a CLB.

- **W_clb (mini-FPGA CLBs only)** - sets the channel width of the intra-CLB routing in a mini-FPGA CLB.

Circuit-level Parameters The circuit-level parameters are as follows:

- **clb_type** – sets the type of intra-CLB connectivity. At this point, the supported options are *mux* and *minifpga*.

- **clb_powergate** – chooses whether or not CLBs in the FPGA are power-gated. *0* omits power gates, and *1* includes them.

- **clb_powergate_size** – sets the size of the power gates, measured in minimum transistor widths (e.g. clb_powergate_size=300 means the power gate is 300× the minimum width for the technology).

- **sbox_type** – sets the switch box topology of the switch box. At this point, the supported switch box topologies are *subset*, *wilton*, and the custom *folded_subset*, designed here at UVa.

- **sbox_powergate** – chooses whether or not switch boxes in the FPGA are power-gated. *0* omits power gates, and *1* includes them.

- **sbox_powergate_size** – sets the size of the power gates, measured in minimum transistor widths.

- **spoint_wls** – sets the number of word-lines (WLs) for the switch points that make up the switch boxes in the global interconnect of the FPGA.

- **spoint_bls** – sets the number of bit-lines (BLs) for the switch points that make up the switch boxes in the global interconnect of the FPGA.

- **lut_switch** – sets the switch topology for the switches inside the look-up table (LUT). This includes the transistors in the *k-1* multiplexer, as well as the output multiplexer that chooses between combinational and sequential functionality. Switch options supported by FGC are *pg* for pass gates, *tx* for transmission gates, and *tristate* for tri-state buffers.

- **ble_switch** – sets the switch topology for the switches inside the basic logic element (BLE). In multiplexer-based CLBs, this option controls the switches in the input multiplexers, which route the inputs and outputs of the larger configurable logic block (CLB) to the individual BLEs. In mini-FPGA style CLBs, this option also determines the switch topology in the intra-CLB interconnet, namely switch boxes and connection boxes. Switch options supported by FGC are *pg* for pass gates, *tx* for transmission gates, and *tristate* for tri-state buffers.

- **fpga_switch** – sets the switch topology for the switches in the global interconnect of the FPGA. More specifically, this option controls the switches in the switch boxes and connection boxes. Switch options supported by FGC are *pg* for pass gates, *tx* for transmission gates, and *tristate* for tri-state buffers.

- **ble_switch_size and fpga_switch_size** – sets the width of the transistors in the switches in the interconnect of the BLE and FPGA, respectively. Width is measured in minimum-widths of the technology.

- **cbox_wls_vert** – sets the number of word lines (WLs) for connection boxes connecting to the vertical channels in the FPGA. The number of bit lines (BLs) needed for these connection boxes is determined by the channel width (W) and the channel fanout for that pin (fc).

- **cbox_bls_horz** – sets the number of bit lines (BLs) for connection boxes connecting to the horizontal channels in the FPGA. The number of WLs needed for these connection boxes is determined by the channel width (W) and the channel fanout for that pin (fc).

- **cbox_vert_spacing and cbox_horz_spacing** – sets the number of WLs or BLs between adjacent connection boxes. In building FPGA layouts, it became apparent that additional space was needed for the connection box, such that another connection box could not fit on the adjacent WL or BL.

- **mux_type** – sets the multiplexer topology. The options thus far are *full*, *two_level*, and *flat*.

- **mux_buffering** – determines how much buffering for the multiplexers. The options supported by FGC are *none*, *single_buffer*, and *full*.

VPR Options The VPR options are as follows:

- **graphics_display** – controls whether the VPR graphics come up when VPR is running. The options are either *on* or *off*
- **exit_temp** – controls the exit temperature (i.e. the number of iterations) of the temperature annealing algorithm used for the placement of logic blocks through the VPR. A lower number results in more iterations, and the default value in VPR is 0.01. I found that certain, smaller benchmark circuits would not map to large FPGAs with limited routing resources, because all of the I/Os and logic blocks would be packed together in a corner, and most of the routing resources wouldn't be used. By raising this exit temperature, you can ensure that the blocks are more spread out through the FPGA, resulting in sub-optimal placement, but successful routing.
- **power_analysis** – determines whether or not the power estimation flow (VersaPower) is used in the VTR mapping. This flow provides an estimation of power consumption, but only works for sequential circuits.

Simulation Options The simulation options are as follows:

- **benchmarks** – list of the verilog circuit implementations to map onto the target FPGA, separated by spaces
- **benchmark_types** – list of the types of circuits listed above, either **combinational** or **sequential**. This list must be the same size as the list of benchmark circuits.
- **simulator** – chooses the simulator to create the schematics for. Supported simulators include *spectre*, *ultrasim*, *hspice*, and *hsim*.
- **scan_chains** – chooses whether or not scan_chains for configuration are to be included in the FPGA schematic
- **models_path** – points to the transistor-level models file that will be used in the simulations
- **design_path** – points to the location of the design file that will be used in the simulations
- **corner** – design corner to simulate in. Supported options are *tt*, *ff*, *ss*, *fs*, and *sf*.
- **save_select_signals** – determines whether or not to save specific individual nets. *0* saves all of the voltages and currents, and *1* saves only the inputs, outputs, and supply voltage currents.

- **vdd** – supply voltage of the FPGA fabric. Multiple VDDs can be listed, and the simulation will run for each voltage
- **vddc** – configuration bit supply voltage. A separate voltage supply is used for the interconnect circuitry in the FPGA to observe the effects of changing that voltage on overall system performance and efficiency. Multiple VDDc's can be used, but the number of VDDcs must match the number of VDDs. If the goal is to simulate a single VDD with multiple VDDc's, then the VDD list is the single VDD three times
- **simulation_time** – the length of time for the transient FPGA simulation
- **frequency** – Frequency of clock signal (used for sequential FPGA circuit implementations
- **step_size** – sets the transient simulation step size
- **buffer_size** – Option for size of the bitstream stimuli. Because FPGA bitstream stimuli tend to be large, this option specifies the maximum size (# of lines) of any bitstream file, and will break the bitstream stimulus into multiple files, each with a maximum size of the specified buffer size.
- **stimulus_type** – Chooses the format of the stimulus. **standard** creates a simple stimulus that must be edited by the user in order to have proper functionality. **binary_count** exercises all of the possible input combinations, each for equal fractions of the total simulation time.
- **simulation_type (UltraSim only)** – sets the type of simulation for UltraSim. The supported options are digital fast (*df*), mixed-signal (*ms*), and Spectre (*s*).
- **simulation_speed (UltraSim only)** – sets the simulation speed for UltraSim. Increased speed results in decreased accuracy in the simulation. Speed ranges in integer values from 1-8.
- **probe_depth (UltraSim only)** – determines how many levels of hierarchy deep to save voltage data when running the simulation. The default value is just the top-level.
- **probe_current (UltraSim only)** – determines whether or not current data is saved. The number of levels of hierarchy that are saved match the *probe_depth* command. *0* does not save the current data, and *1* does.
- **hsim_speed** – sets the simulation speed for HSIM. Increased speed results in decreased accuracy in the simulation. Speed ranges in integer values from 0-8.

3.8.2 Driver Script (FGC.pl)

This script is the main driver for the toolflow, and is located in the *$FGC/scripts* directory. This script runs the proper sub-routines, which are dispersed among a set of perl modules that are located in the *$FGC/scripts/modules* directory.
At the beginning of the file, there are (only) three variables that need to be changed:

1. **$parameter_filename** – the name of the parameter file. It will be used by the script to find your parameter file and also create the output directories in the proper place.

2. **$fgc_dir** – The full path to the main FGC directory. All of the subroutines use this directory as a reference.

3. **$fgc_output_dir** – The full path to the output FGC directory. All of the subroutines use this directory as a reference.

Sub-routines in the FGC.pl script *FGC.pl* runs the following sub-routines (which will be describes in more detail in later portions of this section):

- **do_params** – converts the parameter file into a data structure for future use in the toolflow

- **do_arch** – generates an architecture file (.xml) using the architectural and circuit-level parameters given in parameter file. This architecture file is one of the inputs to the VTR open-source toolflow that is leveraged by this FGC flow.

- **map_fpga** – runs the VTR toolflow in order to generate a virtual mapping of the FPGA, which will then be used to generate initial condition commands and configuration bitstreams. Instead of running the full flow end-to-end, we run from up until the final place and route step (VPR), but then run VPR separately, allowing access to additional VPR options.

- **do_ics** – parses the outputs of the VTR flow to generate IC commands for the coming FPGA schematics. These IC commands ensure proper circuit functionality in the FPGA schematic during simulation.

- **do_wlbl_map** – generates mapping of configuration bit locations in the generated FPGA, used for generating configuration bitstreams.

- **do_bitstreams** – creates the configuration bitstreams for scan chains used for simulation and silicon verification of the FPGA.

- **do_schematics** – generates the necessary SKILL code for creating the FPGA schematics. The actual schematic generation takes place outside of the script, and must be done by the user in the circuit design environment (i.e. Cadence).

- **do_sims** – which generates the necessary simulation files for the generated FPGA schematic performing the given benchmark(s). The simulation also calculates delay and energy for each benchmark as well.

3.8.3 Modules

The Perl modules described below are the meat of my personal contribution to this FGC toolflow. All of these scripts were written by me, to allow for the generation of IC commands, bitstreams, and schematics to save on the extreme effort required to do all of these things by hand for each differing benchmark and parameter.

Setting Parameters (param_control.pm) This Perl module takes the parameters from the parameter file the user creates, and puts them into a data structure (hash) that can then be used by the other Perl modules. After creating the hash, the module calculates other important parameters than can be derived from the given parameters, and includes these new parameters in the hash also. The list of parameters that need to be set in the parameters file is the smallest set of parameters than can be used to fully describe the FPGA. The parameters that are calculated are:

- **I** – number of inputs for each CLB.

- **cbox_w** – number of channels each connection box connects to. There are three seperate connection box widths, for inputs, outputs, and I/O connections.

- **mux_depth (mux-based CLBs only)** – number of levels for the input multiplexers in a multiplexer-based CLB.

- **ble_wls and ble_bls** – number of word lines (WLs) and bit lines (BLs) that span each BLE in the target FPGA.

- **clb_wls and clb_bls** – number of WLs and BLs that span each CLB in the target FPGA.

- **sbox_wls and sbox_bls** – number of WLs and BLs that span each switch box in the target FPGA.

- **fpga_tile_wls and fpga_tile_bls** – number of WLs and BLs that span each FPGA tile. The description of the FPGA tile is in the Architectural Assumptions section.

- **fpga_wls and fpga_bls** – number of WLs and BLs that span the entire FPGA.

The entire list of parameters, both given and calculated, can be observed by the user in the *$FGC_OUT_DIR/parameter_lists* directory.

Architecture Generation (gen_arch.pm) This module takes architectural parameters from the parameter hash that has been generated, and creates an architecture file to be used in the VPR flow. The general architecture file format can be found in the VTR documentation. The output architecture file is printed to the *$FGC_OUT_DIR/archs* directory.

Running the VTR Flow (virtual_mapping.pm) This module runs the VTR flow, which virtually maps the target benchmark circuit(s) to the target FPGA. Both the architectural parameters for the FPGA and the benchmark circuits are described in the parameter file that was the main input to the FGC flow. The VTR tool takes Verilog code and converts it to Berkeley Logic Interchange Format (BLIF), then maps that BLIF onto an FPGA using the Virtual Place and Route (VPR) tool. In the FGC flow, VTR is stopped after the BLIF file is generated. Then, VPR is run on its own, outside of the larger VTR flow. Running VPR stand-alone gives the user access to more options, and gives the FGC tool more control in the FPGA mapping. The outputs from VPR are a set of files that describe a theoretical FPGA that is configured to perform the given function(s). While there are many outputs generated by the VTR flow, the files that are important for the FGC flow are described briefly below:

- **Routing graph (rr_graph.echo)** - This file enumerates every node in the FPGA. The FGC flow uses this file to determine how the connection boxes in the FPGA connect to the tracks. When the channel fanout (fc) is less than 1, the connection boxes skip over some of the tracks. Moreover, each different connection box connects to different tracks. Thus, it is important to map which tracks are connected to by each connection box in the FPGA.

- **Netlist file (<benchmark>.net)** - This file describes the connectivity of all of the blocks in the FPGA. The FGC tool-flow uses this file to determine intra-CLB connectivity.

- **BLIF file (<benchmark>.blif)** - This file represents the logic in each LUT of the FPGA. This tool-flow takes the logic, which is represented in the Berkeley Logic Interchange Format

(BLIF), and converts it into a binary representation, to be configured in the LUTs of the FPGA.

- **Placement File (<benchmark>.place)** - This file gives the relative X-Y locations of all of the logic, IP, and IO blocks mapped by the VTR tool. This tool-flow uses this file to keep track of all of the different configuration bits, as they are organized by the location of their FPGA-tile location.

- **Routing File (<benchmark>.route)** - This file describes which pins of different blocks are connected to each other through the global interconnect of the FPGA. The tool-flow uses this file to determine what values to set each configuration bit in the global interconnect.

The VPR outputs are written to the *$FGC_OUT_DIR/vpr_outputs* directory. The routing graph is the same for each benchmark, and the other VPR outputs are separated into sub-directories based on benchmark.

Generating Initial Conditions (gen_ics.pm) Once the VTR generates the virtual mapping of the FPGA, a set of scripts parse these files and create initial conditions for a simulation of the FPGA. These initial conditions set the values of each configuration bit throughout the entire FPGA fabric. The syntax of each initial condition needs to match the naming conventions of the schematics that will be generated in the future. Additionally, every configuration bit must be set in order to guarantee proper functionality, as bits that are not explicitly set could cause short circuits between used and unused resources. The script first creates all of the possible initial conditions for the FPGA and sets them all to a known default state. For configuration bits controlling access to routing resources or storage elements, such as connection box switches or LUT data, the default value is a logical *0*. For configuration bits that tie floating nodes down to V_{SS}, such as those that tie down the inputs to LUTs, the default value is a logical *1*. Power gates, if included, also use a default value of *1*, as we decided to use PMOS headers for our power gates. After the initial conditions statements are initialized, the tool parses the VPR output files and creates IC commands based on the contents of the files. If the created IC command matches one of the commands in the full list, the value of the bit is changed from 'off' to 'on.' The completed list of initial commands shows up in the *$FGC_OUT_DIR/ic_commands/<benchmark>/<simulator>/<clb type>/ic_commands_fpga.scs* file. <benchmark>, <simulator>, and <clb type> are options set in the parameter file. For help with debugging purposes, there is another file called *ic_post_parsing.txt* that is also written in order to highlight only the configuration bits that are set to a logical *1*. The modules for parsing the VPR

outputs are all in the *$FGC/scripts/modules/vpr7* directory.

Parsing the routing graph (inside gen_schematics.pm) In order to reduce routing overheads in an FPGA, the number of tracks in the routing channels that each connection box connects to is often times far less than 1. But, in order to maintain routeability through the FPGA, each connection box has to connect to different tracks, in order to allow for multiple signals to pass through the channel. As a result, the channels that a connection box will connect change with each mapping through VPR. This creates a problem from generating schematics, because it is necessary to know to which track each of the different connection points in a connection box are connected. A sub-routine called *build_track_map* opens the routing graph generated by VPR and determines which tracks are connected to each type of connection box. Luckily, each tile in the FPGA has the same connection box connectivity. The track map is saved to the *$FGC_OUT_DIR/track_map/track_map_<task name>_generated.txt* file. The task name matches the parameter file name.

Parsing the VPR placement file (parse_placement.pm) Parsing the placement file is the first step in parsing the VPR outputs. All of the parsed information is organized by x-y location on the FPGA, based on the organization of the VPR outputs, and more importantly the organization of the generated FPGA schematics. The placement file provides that x-y information for each block. The purpose of this script is to parse the placement file and create an hash of coordinates based on the name of each block. Power gates for used CLBs are also turned on (bit value is set to 0), enabling the CLBs.

Parsing the VPR routing file (parse_routing.pm) The routing file describes the connectivity between logic blocks and I/Os across the global interconnect. This script parses the routing file and creates initial condition commands for the following configuration bits:

- Global switch boxes
- Power gates for the switch boxes
- Global connection boxes
- Intra-CLB switch boxes and connection boxes (mini-FPGA CLB only)

In order to properly set connection box initial condition statements, a mapping is generated from the track map that maps the different tracks of each channel to the specific bit in the connection box.

If the flow is mapping mini-FPGA CLBs, VPR runs additional times to map each CLB. Each time, all of the VPR outputs are saved, including the routing file. These routing files are also parsed, and generate slightly different initial condition statements from those generated for the global routing file.

Parsing the VPR netlist file (parse_netlist.pm) The netlist file describes the connectivity of the blocks internal to the CLB. It describes which of the inputs to the larger CLB are used, and which inputs or outputs of the CLB are connected to the inputs of the BLE. The FGC flow parses the netlist file and extracts information to generate initial condition commands dealing with whether or not inputs to BLEs and CLBs are being used or not. These initial condition commands include:

- LUT inputs
- BLE output multiplexer select signal
- Sense amp control in the CLBs.
- BLE input multiplexer select bits (multiplexer-based CLBs only)

In order to minimize static current in the FPGA fabric, certain nodes have pull-down NMOS transistors are used at nodes where floating nodes could be a problem (due to the power gating of logic blocks and interconnect circuitry) and force those nodes to logical *0*. These nodes of interest include the LUT inputs and the CLB inputs. The multiplexer at the output of each BLE determines whether or not the register in the BLE is used (i.e. whether this BLE is a sequential element or not). The netlist file gives the connectivity between the CLB inputs/outputs with the BLE inputs, so that the proper select signals for the input multiplexers can be set in order to match this connectivity. Lastly, this sub-routine creates data structures that provide important information necessary to create initial condition statements in the BLIF file.

Parsing the VPR BLIF file (parse_blif.pm) The Berkeley Logic Interchange Format (BLIF) file describes the logic in each LUT. Unfortunately, this information is strictly organized by LUT name, and each LUT is named for its output. Thus, data structures that were created in the **parse_netlist.pm** module help create the initial condition commands from the BLIF, which require information from the netlist. This sub-routine converts the BLIF format into a 2^k-bit truth table, which maps to the configuration bits in each LUT. Initial condition statements for each bit are then generated, implementing the logic described in each LUT in this BLIF file.

Generating WL/BL Map (gen_wlbl_map.pm) This module generates the WL/BL map, which describes the configuration bits at each WL/BL intersection. An FPGA can be thought of as a sparse memory array, where relatively few WL/BL intersections have configuration bits. The WL/BL map is a file with 3 columns: WL, BL, and bit description. Each bit description matches an initial condition statement, or is described as "N/A" if there is no configuration bit with the WL/BL pair. The bitstream generator first matches the activated configuration bits (given through the initial conditions) to the WL/BL pairs required to set each bit. The WL/BL map is saved to *$FGC_OUT_DIR/wlbl_map/wlbl_map.txt*.

Generating Bitstreams (gen_bitstreams.pm) This Perl module generates bitstreams for configuring the generated FPGA fabric through scan chains as opposed to the initial condition statements generated by the flow. These bitstreams can be used to simulate the configuration mechanisms in the FPGA, and also can be used to configure fabricated FPGA chips that have configuration schemes that match the FGC flow. The bitstream generator has been verified in prior FPGA chip tapeouts, and successfully creates timing between word line (WL) and bit line (BL) scan chains to configure all of the configuration bits in the FPGA fabric. This bitstream generator takes two inputs:

1. Initial condition statements (already generated by the tool)
2. WL/BL Map

The bitstream generator creates the proper stimulus for the inputs to the scan chains for the WLs and BLs, such that the proper configuration bits are set. The generated bitstreams are written to the *$FGC_OUT_DIR/simulation_files/<benchmark>/stimuli_scan.scs* file.

Generating Schematics (gen_schematics.pm) This perl module creates a SKILL script that is used to generate the schematics for the FPGA sub-circuits, and eventually the full FPGA fabric. The generated script is simply a driver script; the necessary SKILL scripts for generating the FPGA sub-circuits are already completed, and are housed in the *$FGC/cadence/SKILL* directory. The sub-circuit SKILL scripts are as follows:

- **genMux_<mux type>_<mux buffering>.il** – generates schematics for multiplexers based on the multiplexer type and amount of buffering
- **genMux_ble.il** – generates the schematic for BLE input multiplexers (meaning the select signals are controlled by configuration bits).

- **genMux_lut.il** – generates the schematic for LUT multiplexers (meaning the select signals are pins that pass through inverters to generate the select-bar signal)
- **genLUT.il** – generates the LUT schematic
- **genBLE.il** – generates the BLE schematic
- **genCLB_<clb type>_BLEtile.il** – generates the schematic for a CLB tile that includes the BLE and the intra-CLB interconnect circuitry attributed to that BLE.
- **genCLB_<clb type>.il** – generates the CLB schematic
- **genCBox_in.il** – generates two different connection box schematics, one for connections between block inputs and vertical channels and one for input-to-horizontal-channel connections.
- **genCBox_out.il** – generates two different connection box schematics, one for connections between block outputs and vertical channels and one for output-to-horizontal-channel connections. Output connection boxes include sense amps to drive the signal into the interconnect.
- **genSPoint.il** – generates the schematics for the switch points in the interconnect
- **genSBox_<sbox type>.il** – generates the schematic for the switch box
- **genFPGA_CLBtile.il** – generates the schematic for the FPGA CLB tile, which includes 1 CLB, all of the connection boxes connected to that CLB, and the switch box located to the upper-right of the CLB. Figure 3.9 illustrates an FPGA CLB tile.
- **genFPGA_IOtile.il** – generates the schematics for each of the four FPGA IO tiles (top, left, right, and bottom side) which includes connection boxes to connect the interconnect to the FPGA I/Os. IO tiles on the bottom and left sides of the FPGA have switch boxes included.
- **genFPGA.il** – generates the schematic for the full FPGA circuit
- **genFPGA_topLevel.il** – generates the top-level schematic for the FPGA. This schematic would include the scan chains if the user of the flow chooses the scan chain option. Scan chains need to be synthesized or created outside of the flow.

With each differing FPGA architecture, the connection boxes connect to different routing tracks, depending on the width of the routing channel (W) and the channel fanout (Fc). Thus, the schematic generation needs to use the track map (generated from the routing graph) to update the CLB tile and IO tile SKILL scripts such that they connect to the proper channels. There is a *templates* directory

in the SKILL directory that doesn't include any channel information. The **gen_schematics** Perl module takes those templates and augments them with code for connecting channels. This module also creates a driver script, located at *$FGC/cadence/SKILL/<task name>.il*, that calls each of the necessary SKILL scripts with the parameters provided in the *<task name>.txt* parameter file.

Building and Running Simulations (gen_sims.pm) This Perl module generates the necessary files to run a transient simulation, and then calculate critical path delay, power consumption, and energy. Once the schematic for the FPGA is generated and a netlist is extracted, a directory is generated for each benchmark given in the *<task name>.txt* parameter file. In each directory, several files are generated. The filenames are either **.scs** or **.sp** files, depending on the choice of simulator.

- **sim** file – This file runs the simulation. It sets the simulation parameters, includes the necessary files (stimulus, initial condition commands, etc.) and calculates the delay and energy. The signals used to calculate the delay are extracted from the critical path output file from the VTR flow (*<benchmark>.critical_path.out*). Power is calculated by measuring the average power consumption, and multiplying by the voltage. Energy is calculated by integrating the current drawn from the voltage sources (VDD and VDDC) over the delay, and multiplying by the voltage.

- **ic_commands** file – This file holds the initial condition statements, and is copied from the *$FGC_OUT_DIR/ic_commands* directory.

- **save** file – This file sets the signals to be saved by the simulation. Because FPGAs are such large circuits, simulation times can be extremely long. This file limits the number of nodes saved to the inputs and outputs of the FPGA, and the currents from the main voltage sources. The *save_select_signals* option in the *<task name>.txt* parameter file sets whether this file is used or not. If set to 0, all signals in the FPGA are saved.

- **stimuli** file – This file provides the stimulus for the FPGA simulation. The default stimulus creates piece-wise linear (PWL) voltage sources for each input. The inputs can also be set as binary counters, such that every possible input combination is exercised. If the input stimulus is PWLs, then the user needs to edit the stimulus file in order to achieve proper functionality before running the simulation.

- **stimuli_scan** file – This file provides the stimulus for the FPGA configuration. This stimulus

	F_PG	F_TX	F_TS
Switch used in interconnect	Pass Gate	Transmission Gate	Tristate Buffer
# of LUTs	200		
K (# of LUT inputs)	4		
N (LUTs per CLB)	8		
W (# of routing tracks)	80		
FC (fraction of tracks connected to)	0.3		
Mapped Circuit	5 4-bit adders		

Figure 3.12: Parameters for the FPGAs that were simulated using the FGC flow.

is only present if bitstream generation is turned on .

- **run_sim.sh** – This file is a Bash script that gets generated by the FGC flow to run the simulation. The format of the script changes depending on the simulation type. Spectre simulations are run through OCEAN.

Once all of the simulation files are created, the script kicks off the simulation by running the *run_sim.sh* Bash script (described above). Delay and Energy information calculated from the simulation are saved into the simulation directory once the simulation is completed. For UltraSim simulations, the data is saved into the *input.meas0* and *input.mt0*files.

3.9 Proof-of-Concept Simulations

To show how the FGC tool can be used for design space explorations, I used the flow to generate both the schematic and configurations for three different 200-LUT FPGAs. Each of the FPGAs was configured to implement 5 4-bit adders. One FPGA has tri-state buffered switches in the interconnect , and the others use the low-swing interconnect introduced in [28], one with pass gates (PGs) and the other with transmission gates (TXs). All three implement standard multiplexer-based CLB structures illustrated in [26]. Figure 3.12 shows the different parameters for each FPGA simulated.

3.9.1 Simulations Across Switch Type

Figure 3.13 shows ED curves taken from simulations generated through the flow. Each data point is taken from a simulation at a different operating voltages, ranging from 0.7 V to 1 V. For the PG and TX FPGAs, each data point in the simulation represents the minimum-energy VDD/VDDc pair,

which are the two voltages in the dual-VDD scheme shown in [28]. These simulations use SPICE netlists generated by the FGC flow. These plots were generated using HSIM, which is a high-speed SPICE simulator. Using a high-speed simulator reduces the accuracy of the simulation, but because the same simulator is used for each FPGA, comparisons can still be made. With that said, more accurate SPICE-level simulations will be required to verify delays and energy consumptions. From this simulation, we can observe the pareto-optimal curve, and see that the optimal solution for implementing a 4-bit adder would be either using the PG or TX FPGA, depending on the delay/energy constraints.

3.9.2 Simulations Across Depopulation

Figure 3.14 shows another example of the use of the flow. This time, the knob of interest is depopulation of the multiplexers used in the CLBs of the FPGA, another parameter that is not addressed within the VTR flow. The FPGA used in this exploration has the same design parameters as the tri-state FPGA described in figure 3.12, but instead implementing 10 4-bit counters. This analysis allows the user to see how the depopulation effects top-level FPGA metrics. Some of the simulation points from the data sets in this plot were removed due to simulation error (simulation timed out, server crash, etc.). For these architectural and circuit-level parameters, using 66% depopulation minimizes energy consumption, and using 50% depopulation minimizes delay.

The same approach used for these example design space explorations can be taken with more complicated benchmark circuits, and with the changing of many more knobs, allowing for circuit designers to rigourously determine the best set of design choices for attacking specific problems. The FGC toolflow allows circuit designers to run SPICE-level simulations for full FPGA fabrics, and see how specific design choices at lower levels in the FPGA design effect its overall performance.

3.10 Limitations and Future Work

3.10.1 Stimulus Generation

In the FGC flow, users have two options for generating stimulus: a simple, standard stimulus framework to be edited by the user, and a binary counter that exercises all possible inputs. These options present the following problems.

Exercising the critical path In the course of the FGC flow, the input and output signals of the critical path of the circuit are taken from the output of the VTR tool, and are used for calculating

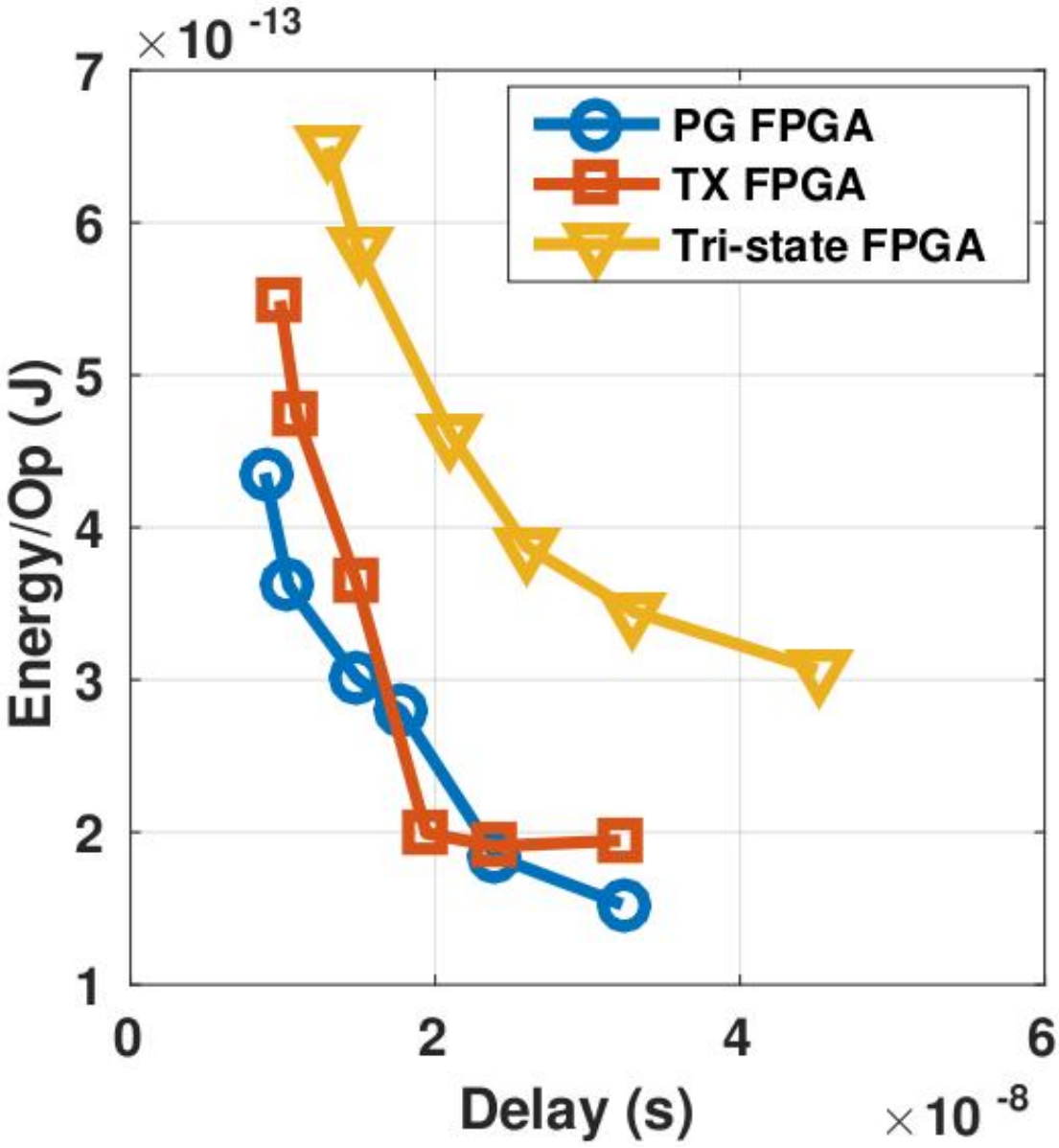

Figure 3.13: Energy-Delay (ED) curves of 3 different FPGA at different voltages. The simulation files and schematics are generated by the FGC tool. The tool allows full FPGA SPICE-level simulations showing the effects of low-level design changes like interconnect switch type, shown in this figure.

delay. At this point, it is left to the user of the flow to determine what input stimulus is necessary to exercise this critical path in simulation. In the case of the adders simulated in section 3.8, the critical path was between a carry-in input and a carry-out output. Thus, the inputs to the combinational circuit needed to be such that a transition in that particular carry-in input caused a change in the carry-out output, so the critical path can be measured. Figuring out the proper stimulus will become more difficult with benchmark circuits of higher complexity.

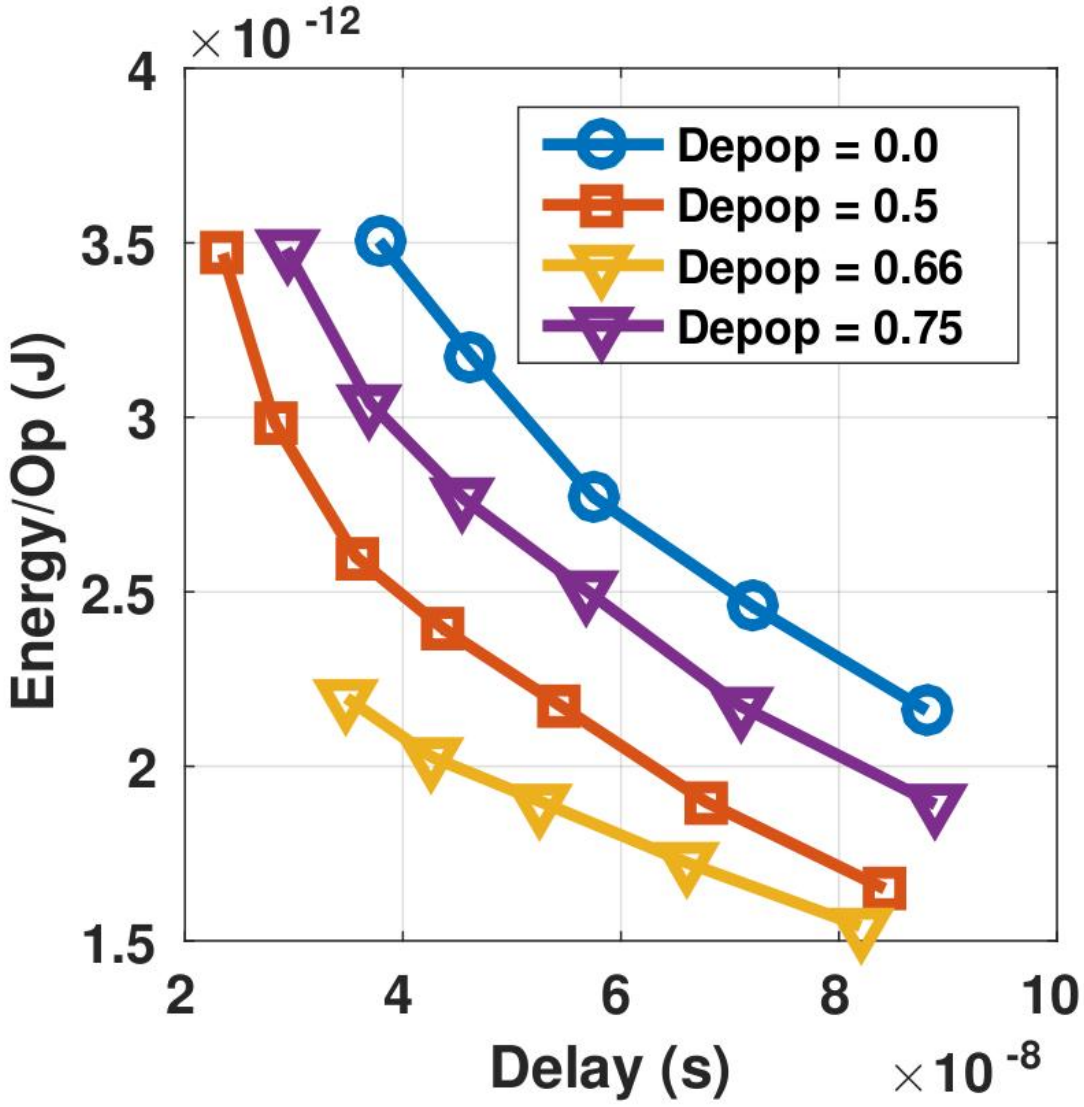

Figure 3.14: Energy-Delay (ED) curves of a 5x5 FPGA implementing 10 4-bit counters across different multiplexer depopulations in the CLB. Using this set of FPGA architecture and circuit-level parameters, 66% depopulation minimizes energy, and 50% depopulation minimizes delay. The FGC flow is the only tool that allows for simulations across depopulation.

Prohibitively large stimulus generation time The amount of time to generate binary stimulus increases greatly with the number of inputs. As an example, if the generated FPGA is to implement five parallel four-bit additions, the terminal crashes before the input stimulus can be generated successfully. Additionally, the simulation time for the FPGA would need to be $T_P(2^I)$, where T_P is the propagation delay, and I is the number of inputs to the FPGA, and that simulation time gets very long for many inputs. To address large stimulus generation and simulation time, generating an I-bit LFSR as part of the flow might be beneficial. The LFSR can be synthesized using the Cadence synthesis flow to provide pseudo-random inputs to the FPGA, providing a manageable, yet

characteristic number of inputs.

3.10.2 Layout Generation

While the FGC flow quickly generates the schematics for full FPGA fabrics (within minutes), the flow currently does not address layout. If designers want to build custom FPGA fabrics, designing the FPGA sub-circuit layouts is extremely time consuming. [17] addresses this issue by leveraging the commercially available synthesis flow, and building FPGA-style standard cells. Another possible solution is to continue to leverage Cadence's SKILL code (like the schematic generation scripts to), and tile smaller FPGA block layouts together. Including layout generation into the flow would allow for more complete design space explorations, because considerations like FPGA area can be taken into account very realistically. It is important to note, however, that even without the layout generation, the FGC flow greatly reduces the design time for building custom FPGAs.

3.10.3 Simulation Time vs. Accuracy

The simulations conducted in the Proof-of-Concept section of this paper used the HSIM, a SPICE simulator that sacrifices accuracy to achieve very high speed. Because FPGA circuits are so large (thousands of FETs), using an accelerated simulation platform is necessary. On one hand, higher HSIM speed refers to lower HSIM accuracy, and the difference in energy consumption across from the high speed to the low speed is very different. On the other, highly accurate SPICE-level simulations take prohibitively long amounts of time. Simulating an FPGA with 100 CLBs can take multiple days. Users have to choose between inaccurate timely simulations or prohibitively long accurate ones.

To make matters worse, both the inaccuracies of the simulation at high speeds and the length of simulation time at low speeds are much worse in the pass-gate version of the FPGA. Research (like [28]) has suggested that using purely pass-gate interconnects can reduce energy consumption for low-voltage operation, which means that testing this becomes extremely important. Even though HSIM has inaccuracies in estimation of power consumption for FPGAs, different design choices can still be compared at the full-FPGA level using this tool. The FGC tool also builds the simulation files in different formats, such that the user could use another simulator, if they felt it would do a better job than HSIM or other supported simulators.

Research has been conducted for reducing simulation times for FPGAs. FPGA-SPICE ([32]) extends VTR to allow for SPICE-level simulations on the individual FPGA sub-circuits, which could drastically reduce run-time for low-voltage simulations as well as increase the accuracy. Leveraging a

flow similar to FPGA-SPICE could mitigate the low-voltage simulation problems, but FPGA-SPICE is currently limited to the same architectural and circuit-level flexibility as VTR.

3.10.4 Low Voltage Simulation

Along with introducing inaccuracies into simulation results, using accelerated simulators reduces the range of voltages that can be simulated. HSIM simulations of FPGAs don't work at VDDs lower than 0.7, which means near- and sub-threshold operation cannot be verified using the HSIM simulator. Moreover, the minimum voltage that can be simulated increases (i.e. gets worse) with increased FPGA size. If researchers are looking to reduce power and energy consumption, the simulation capabilities of the FGC flow cannot reach the low voltages that would need to be tested.

3.10.5 Stimulus for Combinational Circuits

In the course of the FGC flow, the input and output signals of the critical path of the circuit are taken from the output of the VTR tool, and are used for calculating delay. At this point, it is left to the user of the flow to determine what input stimulus is necessary to exercise this critical path in simulation. In the case of the adders simulated in section 3.8, the critical path was between a carry-in input and a carry-out output. Thus, the inputs to the combinational circuit needed to create a transition in that particular carry-in input that changed in the carry-out output, so the critical path can be measured. Figuring out the proper stimulus will become more difficult with benchmark circuits of higher complexity.

3.10.6 Delay Calculation for Sequential Circuits

The simulations shown in this chapter are of combinational circuits, to illustrate the ease of configuration, simulation, and delay and energy calculations. Doing the same for sequential (clocked) circuits is a little more difficult and nuanced. Generally, sequential circuit max frequencies can be determined in two ways:

1. Simulating the circuit repeatedly, and increase the frequency of the circuit until it stops working
2. Measuring the propagation delays between each combinational path

As discussed before, simulation times for these FPGA circuits are long, so option 1 becomes infeasible. Option 2 provides a potential solution, but requires additional analysis of simulation waveforms to determine which combinational signal transitions contribute to the critical path delay. Additional research is required to allow the toolflow to automatically parse through the simulation output and

determine which combinational signals to include in delay calculations. While it is difficult for the FGC tool to automatically calculate delays and energies for sequential circuits, the toolflow makes simulating these circuits on FPGA fabrics possible, and reduces the user input to finding ways to calculate necessary parameters from simulation results.

3.10.7 Graphical User Interface

There are many parameters, each with varying options. It would be much easier for the user if a graphical user interface was used instead of a input parameter list. Numerical values would be fields, parameters with specific options could be drop-down menus, and the options that are turned on and off could be push-buttons. The GUI would then generate the parameter text file, and run the FGC flow.

3.10.8 Introduction of IP blocks

Because of the complexity of real-world computing applications, FPGAs that are built purely of CLBs are not optimal implementations. Many commercial FPGAs include accelerating IP blocks for specific functions, from multipliers to full MCU cores. Doing so dramatically expands the usefullness of FPGAs, as tasks that would be prohibitively inefficient for an FPGA fabric to perform can be done by the IP blocks while still remaining a part of the larger flexible fabric. Currently, the FGC tool does not support custom FPGA architecture that include new IP, but doing so is straightforward. VTR already has the capability of adding new IP blocks, so the FGC flow would need to include additional parameters in the input parameter file that choose from a set (which could be expanded) of IP blocks to add to the FPGA fabric. For generating the FPGA schematic, the IP block schematic would come from elsewhere, either created by or given to the user, and a SKILL script would need to be created to include that IP block into the larger FPGA framework.

3.11 Conclusion

In this chapter, I have described the FGC (FPGA Generation and Configuration) toolflow that I have developed for rapid configuration, generation, and simulation of custom-built FPGA fabrics. The toolflow has the following features:

- Ability to sweep both architectural and circuit-level FPGA parameters
- More than 50 different parameters with 100s of different options

- Generates full FPGA schematics
- Generates initial condition (IC) commands for rapid, parallel FPGA configuration in order to streamline design space exploration through simulation
- Generates configuration bitstreams for configuration of physical custom-FPGAs post-fabrication
- Generates simulation files compatible with multiple supported simulators
- Allows for simulation in any available transistor technology

Running the entire flow for relatively large FPGAs (800 LUTs) takes approximately 8 minutes, not including simulation. As a proof-of-concept, simulations across different circuit-level and architectural knobs for full FPGA schematics are shown. This tool, while used in this dissertation to address low-power FPGA design, can also address custom high-performance custom-FPGA development, as the same problems of incomplete CAD infrastructure and design time persist.

This chapter also highlights places where this flow can be improved in the future. These features include:

- a graphical user interface (GUI) allowing the user to easily set different options
- inclusion of hard-IP blocks
- an extension of VersaPower (VTR's power estimation tool) to estimate power consumption based on given circuit-level and architecture parameters
- integrating layout generation into the FGC flow

The work written in this section was submitted for publication to the $34^{h}boxth$ IEEE International Conference on Computer Design, outlined in the publications section under 'Pending Publications.'

4 Architecture Exploration

4.1 Motivation

Modern FPGA design (as made by Xilinx, Altera, and other companies) use circuit elements that allow for increased performance, as FPGA companies design their devices in order to keep up with commercial processors. This is probably even truer of FPGA *architectures*. The newest generations of Xilinx and Altera chips both have much more complicated CLB structures than standard academic FPGAs, mostly to push performance. In addition, commercial FPGAs include many different IP blocks for specific functionalities, embedded block RAMs, and other accelerators. Additionally, many FPGAs have extremely large interconnects. All of these optimizations to traditional FPGA architectures were added to provide maximum logic capacity, and to complete computations as fast as possible (in the 100s of MHz range), as opposed to doing so energy efficiently. Current commercial FPGAs are not feasible for ULP applications. Many of these applications require active power dissipation at less than 1 mW. High end commercial FPGAs have sleep power consumption that are still over 1 W. FPGAs that champion low energy operation, like the Lattice iCE40 and the Microsemi IGLOO, still have operating currents in the mW range, which is much lower than the rest of the field, but still about an order of magnitude too high for ULP applications. In this chapter of the dissertation, I revisit the architectural sweeps done in the past, but look to minimize power consumption and transistor area, with less of an emphasis on FPGA performance. Doing so creates a set of architecture parameters that will better position FPGAs to meet the stringent requirements of ULP applications, such as the Internet-of-Things. In addition, I highlight where commercial FPGAs reside in this design space.

4.2 Prior Art

Work has been done to optimize individual architectural parameters for FPGAs. In [2], LUT-inputs (K) and cluster size (N) were co-optimized to minimize the area-delay product of a mapped FPGA. In this study, the VPR tool was used, and the conclusion was that K = 4 to 6 and N = 3-10 provide the best results for area-delay product. [21] extends that study by including larger cluster sizes (up to 20), and includes different routing architectures, which take into account both different segment lengths and switch topologies (pass gates vs. buffers). Not only that, but this study also explicitly targets power consumption. In this study, researchers found that K = 4 minimized power dissipation across different clustering values, and that a cluster size of N = 12 minimized

both total power of the FPGA and power-delay product. [16] compares uni-directionality and bi-directionality of interconnects in FPGAs. Uni-directional wiring only allow signals to propagate in one direction through the interconnect, whereas bi-directionality allows signals propagating in both directions. Findings from this study show that bi-directional routing (in the form of tri-state buffers) for low-frequency designs have lower energy consumptions, but for high frequency operation uni-directional routing is lower energy. The work in this dissertation extends this work by combining the architectural knobs listed above, while also adding additional knobs to the design space. To the best of my ability, I atempt to show where commercial low-power FPGAs fit in the architectural design space, in order to show how these designs could be influenced at an architectural level to better meet ULP application requirements.

4.3 Architectural Parameters for FPGAs

The architectural parameters investigated in this dissertation are listed below:

- k - number of inputs to LUTs
- N - the number of BLEs in each logic block
- **Channel Fanout (F_c)** - the number of channels in the global interconnect that a logic block or FPGA I/O can connect to
- **Segment Length (L)** – number of CLBs spanned by individual wire segments.

This is not an exhaustive list of architectural parameters, but provide the deepest architectural analysis for ULP FPGAs to date, at least to my knowledge. Figure 4.2 provides the values of each of the architectural parameters that are swept. This includes the architectures used in commercial and academic low-power FPGAs that already exist, which I discuss in the next section. The original plan was to use larger k values than 8, but cannot be increased because of limitations to the VTR tool.

In order to purely compare architecture, and not capacity, the number of routing tracks is fixed, and the number of CLBs changes k with N, in order to provide the same number of total LUT bits in the FPGA. Because the highest K we will be using is 8, and the highest N is 28, we want to create a reasonably sized FPGA with those architectural parameters. As a result, I elect to use the capacity of 100 8-input LUTs as the logical capacity for this exploration, in order to remove capacity from the equation. That capacity is 25,600 configuration bits specifically for the data in the LUTs, and is equivalent to 400 6-input LUTs, 1600 4-input LUTs, and 3200 3-input LUTs. In terms of

Parameter	Symbol	Explored values
LUT inputs	K	3, 4, 6, 8
CLB Clustering	N	4, 8, 12, 16, 20, 24, 28
Channel Fanout	FC	0.05, 0.1, 0.2, 0.3, 0.4, 0.5
Segment Length	L	1, 4, 6, 7, 8, 9, 10

Figure 4.1: Architectural knobs swept for this experiment, and the values observed in this sweep. This architecture sweep checks 1,176 architectural combinations.

benchmarks for analysis, I elect to use the MCNC20 benchmarks, which have been historically used for FPGA benchmarking, as well as other benchmarks that adequately capture different ratios of inputs to outputs. N-input AND gates have relatively large input-output ratios (N), N-bit adders have a medium input-output ratio (2), and N-bit counters have small input-output ratios ($\frac{1}{N}$). A breakdown of the architectures explored is shown in Figure 4.11.

Parameter	Symbol	Explored values
LUT inputs	K	3, 4, 6, 8
CLB Clustering	N	4, 8, 12, 16, 20, 24, 28
Channel Fanout	FC	0.05, 0.1, 0.2, 0.3, 0.4, 0.5
Segment Length	L	1, 4, 6, 7, 8, 9, 10

Figure 4.2: Benchmarks used for this architectural exploration. Historically established benchmarks are used (), as well as benchmarks with varying input-output ratios.

4.4 Commercial and Academic Low-Power Architectures

For somewhat obvious reasons, much of the specific information concerning the architectures of commercial FPGAs are proprietary. As a result, it is difficult to pin-point where each commercial part would fall in this architectural design space. As a result, commercial parts will show up as groups of points on the design space, where specific known architectural parameters are pinned down, and the other parameters are swept. Figure 4.3 shows the architectural parameters of the FPGAs highlighted in the Chapter 1 of this dissertation. Areas with a '?' will be swept. Directionality for this entire exploration is left as uni-directional, because the VersaPower estimation tool only works for uni-directional routing.

Parameter	Symbol	Lattice iCE40	Microsemi IGLOO nano	Ryan [22]	Grossmann [7]
LUT inputs	K	4	3	4	4
CLB Clustering	N	8	1	9	8
Channel Fanout	FC	?	?	?	?
Segment Length	L	1,4,8,12	1,2,4	1	1

Figure 4.3: Architectures of commercial and academic low-power FPGAs.

The commercial FPGAs tend to have a distribution of segment lengths (L) in their interconnects. This allows for connections from one side of the FPGA to the other without passing through multiple switch boxes and connection boxes, removing unnecessary overhead and making the overall interconnect more efficient. However, these same commercial FPGA companies do not give details as to the proportionate number of routing tracks for each segment length, so an equal proportion is assumed.

4.5 VTR Investigation

As an initial comparison of the areas and power consumptions of the different architectures, I leverage the VTR toolset. Instead of using the VTR flow standalone, I elect to use it within the framework of the FGC flow, which I developed as part of this dissertation. Because the lone input to the FGC flow is a parameter file which describes (among other things) architectural parameters, it is easy to sweep some of those parameters and run the VTR portion of the flow.

Routing area estimates and power consumption are taken from the VTR outputs. In this exploration, keeping the total logical capacity the same, which should keep the total logic area equivalent for each combination of architectural parameters. As a result, only the routing area changes, whether it be local to the logic blocks or the global routing. The VTR tool reports routing area estimates, which illustrate the impact of architectural parameters on FPGA area.

Power estimates come from the VersaPower tool in the VTR flow. This tool provides breakdowns of power consumption based on sub-circuit and functionality, proving useful. It does this by drawing from key-words in the architecture file to determine circuit parameters, and using an additional tool, called ACE [19] to measure the activity factor of each node in the circuit. Power estimates from using this tool are much higher than power consumption measured from simulations earlier in this dissertation. This is due to:

1. **Differences in FPGA size** Because of long simulation times, the FPGAs simulated in previous chapters have relatively small sizes. In this chapter, we've set the logic capacity at 25,600 SRAM bits of LUT memory. When simulating the FPGAs with a single CLB in the FGC chapter, those FPGAs have 128 bits.

2. **Assumption about FPGA architecture** In the simulations of the FPGAs shown in earlier chapters of this dissertation, most of the transistors in the FPGA are close to minimum sized, to reduce capacitances and currents. Further inspection into files used in the VTR flow show assumed transistor sizes as high as 20× minimum sized, greatly increasing power consumption as well as area.

Figure 4.4 plots the power consumption of each of the explored FPGA architectures against the area. As expected, power consumption and area have a strong linear correlation. Figure 4.5 displays the 50 best design points, determined by minimizing the power-area product of the different data points. There is a clear optimal point, which minimizes both area and power consumption. From there, there is a pareto optimal surface in which all of the points along it are equally optimal for different constraints. The pareto surface is marked with a curve.

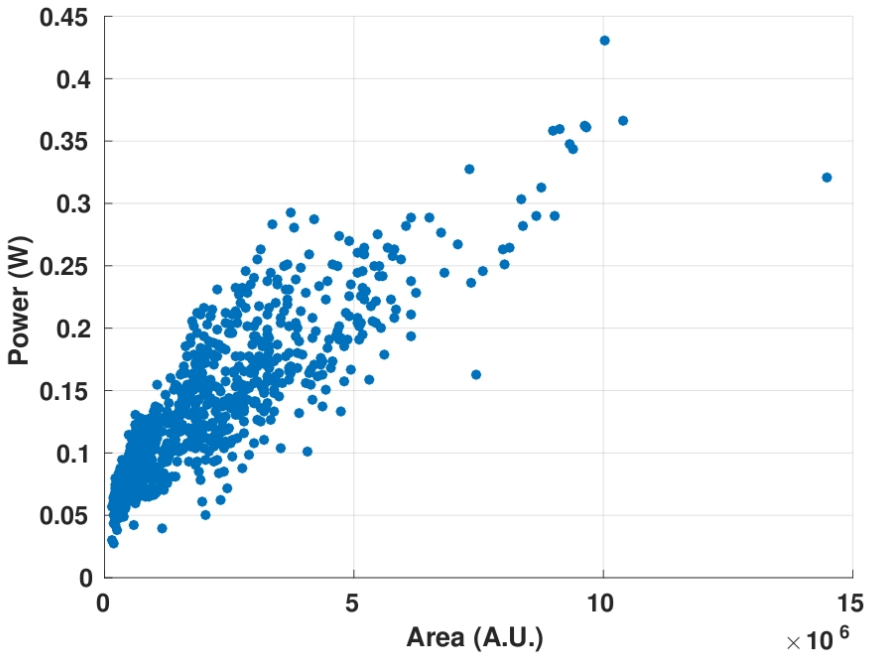

Figure 4.4: Scatter plot of power consumption and area for different FPGA architectures. There is a strong linear relationship between area and power consumption, as is expected.

The table in Figure 4.6 highlights the architectural parameters that are suggested by this study

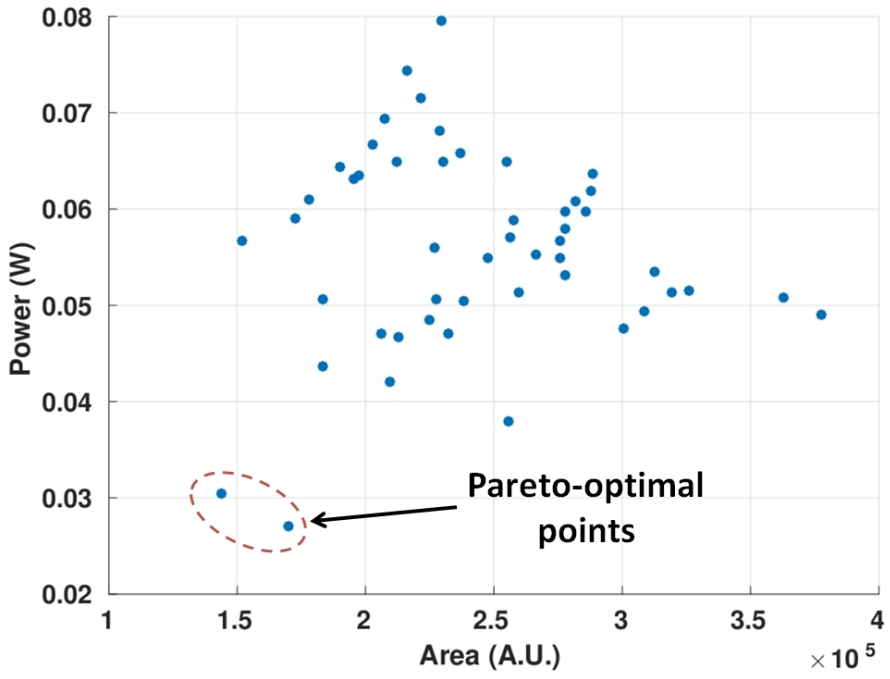

Figure 4.5: The 50 best FPGA architecture combinations, minimizing the power-area product. There is an outright best option, and then a pareto curve for additional options.

to minimize power consumption and area. From this experiment, the best way to minimize area is to maximize k and N, and reduce fc and L. The architectural choices are different for minimizing power consumption, however. The optimal N value is now 12, and L is optimized at 6.

Description	k	N	fc	L	Area (A.U.)	Power (W)
Min. Power	8	12	0.05	6	227858	0.05064
Min. Area	8	28	0.05	1	255247	0.06494

Figure 4.6: Architectural parameters for minimizing power consumption and area in FPGAs. Maximizing k and N and reducing F_c and L minimize area. To minimize power consumption, $N = 12$ and $L = 6$ are optimal..

Depopulated Switch Boxes In doing this initial VTR sweep, it is assumed that there is no depopulation with increased segment length. Extended wire segments still have access points at every intersection, and segment length refers to where the physical wires in the interconnect end. Depopulating switch boxes from long wire segments can reduce power consumption and area effectively, as shown in [5]. It is unclear whether long segments in commercial FPGA architectures are depopulated or not. Some of the literature seems to suggest that the wire segments could be

completely depopulated, aside from the endpoints of the wire segments. To address this, I redo the same experiment with completely depopulated long wires. There is still connectivity for each connection box along the wire segments. An illustration of switch box depopulation is shown in Figure 4.7.

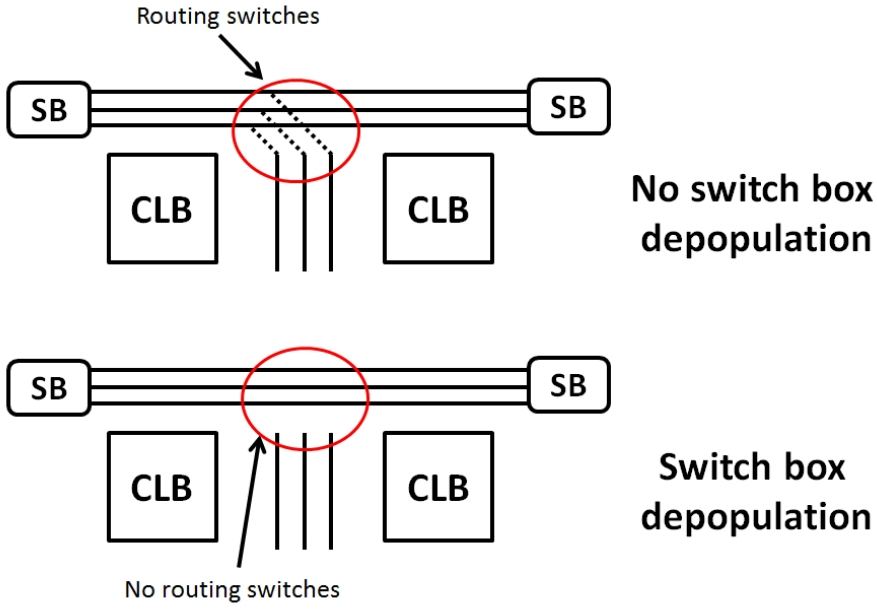

Figure 4.7: Depopulation of switch boxes from wire segments. Instead of providing access to routing channels at every intersection point, many FPGA architectures reduce those. Commercial low-power architectures hint toward complete depopulation.

Figure 4.8 shows the VTR exploration with the depopulated wire segments. The first thing to notice is there are much fewer data points on this exploration. In removing the access between the endpoints of the wire segments, the routeability of the FPGA reduces drastically. As a result, many of the architectural parameter combinations are not routeable for the same algorithm. The exploration that generated Figure 4.4 had a 72.9% success rate (857 successful routings out of 1176 architecture combinations), where as depopulating the switch boxes dropped that success rate down to 19.1% (224 out of 1176). Figure 4.9 states the suggested architectures for minimizing power and area, based on this exploration. Again, maximizing k and N while minimizing F_c and L reduce area. This time, those parameters also minimize power consumption as well. This is due to the reduced routeability of the FPGA with depopulated switch boxes, which makes designs with more necessary connections (i.e. more CLBs) less likely to route.

In order to determine whether or not circuit depopulation is in fact good for reducing power and area, I compare the power and area consumptions of the four best architecture combinations as

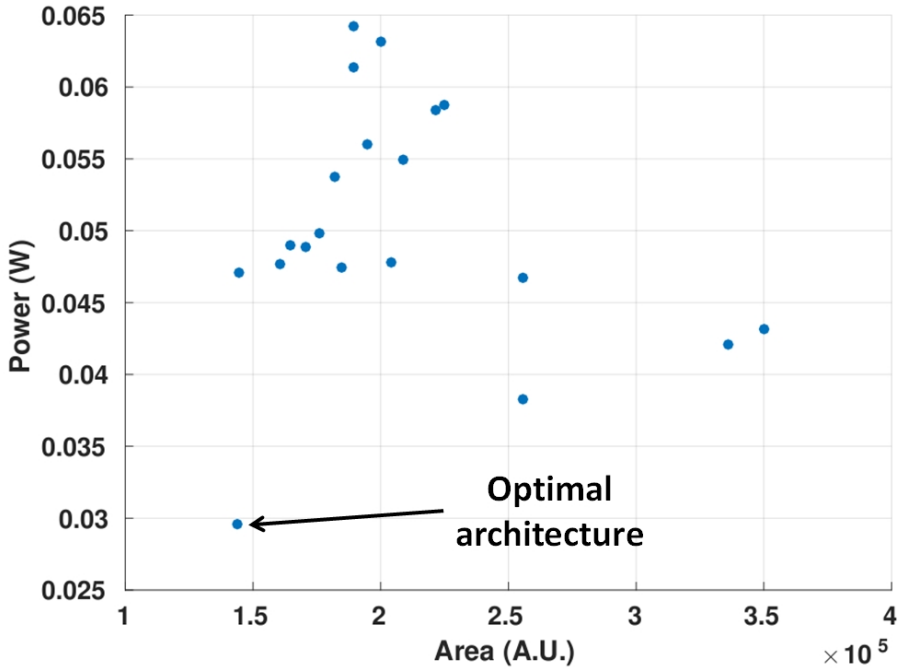

Figure 4.8: The 20 best FPGA architecture combinations for depopulated wire segments (removal of intermediate switch boxes). There is an outright best option, and then a pareto curve for additional options.

Description	k	N	fc	L	Area (A.U.)	Power (W)
Optimal Arch.	8	28	0.05	1	143870	0.02962

Figure 4.9: Architectural parameters for minimizing power consumption and area in FPGAs with fully depopulated long routing segments. The architectural parameters are identical to those that minimize area for the inteconnect without switch box depopulation.

reported. Figure 4.10 illustrates the comparison. The units for both power and area are normalized to the FPGA with no switch box depopulation. Surprisingly, depopulation of the switch box actually increases the power consumption by 9.5%. This is due to the sharp decrease in routeability. The FPGA that minimizes power consumption has multiple CLBs, which cannot be routed in the FPGA with switch box depopulation. For designing ULP FPGAs, reducing power consumption is of more importance, and as a result, no depopulation is recommended for long wire segments in ULP FPGAs and the architecture parameters that minimize power consumption are recommended.

4.5.1 Comparisons across benchmarks

To further explore FPGA architectural choices, I repeat this architectural exploration across benchmark verilog circuits that are mapped to the FPGAs. In order to make a complete comparison, I

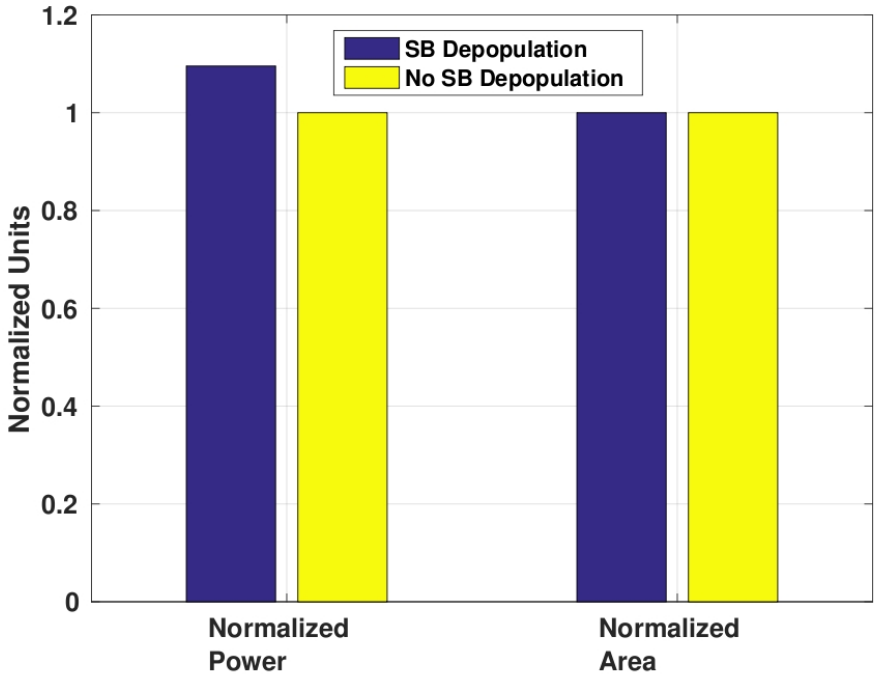

Figure 4.10: Comparison of area and power consumption between global routing interconnects with and without switch box depopulation. Switch box depopulation increases power consumption by 9.5%, and has no effect on area.

choose three different benchmarks. In addition to 4-bit counters (which are used in the previous section), I explore mapping a wide AND gate and a collection of 4-bit adders. These circuits are chosen deliberately because they vary in numbers of inputs/outputs. An N-bit AND gate has N inputs and 1 output. An N-bit adder has $2N + 1$ inputs and $N + 1$ outputs. An N-bit counter has 1 input (the clock), and N outputs. Thus, observing these three types of circuits will give edge cases for the exploration of different types of benchmark circuits. Figure 4.11 displays the different benchmarks, as well as the relative number of inputs and outputs.

Benchmark Circuit	# of Inputs	# of Outputs
N -bit AND	*N*	1
N -bit Adder	*2N+1*	*N+1*
N -bit Counter	1	*N*

Figure 4.11: Benchmark circuits used for this architectural exploration. The circuits have varying numbers of inputs and outputs, which provide limiting cases for benchmark circuits.

Figure 4.12 displays the same power/area design space for the FPGAs, but for a collection of 4-bit adders and a large AND gate as well as the 4-bit counters. Even though the FPGAs are the same size, we see that routing both area and power consumption change with different benchmark circuits. Pareto optimal points in these distributions are highlighted, and the architectural parameters that optimize power and area are given in Figure 4.13. Across benchmark circuit, this exploration suggests that k should be maximized and F_c should be minimized across benchmark circuits. The optimal N value, however, seems to change depending on the metric of interest. For minimizing area (routing area specifically), clustering should be maximized to reduce global routing overhead. It is unclear how VTR estimates intra-CLB routing, and extending this work must include updating the estimation processes used by VTR. For minimizing power consumption, this study suggests using an intermediate value, balances the increasing power consumption of the individual logic blocks with the decrease in power consumption from the interconnect. The suggested value for L is dependent on the benchmark circuit. For counters, that essentially only have outputs, it reduces power and energy to have small segment lengths (1). For benchmark circuits with more inputs, longer segment lengths reduce area and power.

4.5.2 Comparison with Commercial/Academic Architectures

It is important to determine how the current state-of-the-art FPGA architectures fit in to this design space exploration. To do this, architecture files are created to match the commercial and academic FPGAs highlighted in Figure 4.3. As stated before, multiple points on the design space are attributed to the different FPGAs. Each of the reported FPGAs have certain parameters that are not specified, and so the architectural exploration covers all of the possible combinations that fit within the constraints of prescribed architecture.

Figure 4.14 overlays existing commercial and academic FPGA over the design space shown in Figure 4.8. From this plot, it is clear that the architectures used in commercial FPGAs could be changed to better address low power operation, namely by increasing k and N. It is important that this is only true for algorithms that are output-heavy, that is have many more outputs than inputs. Looking at other benchmark circuits with varying numbers of inputs and outputs could give a more complete picture.

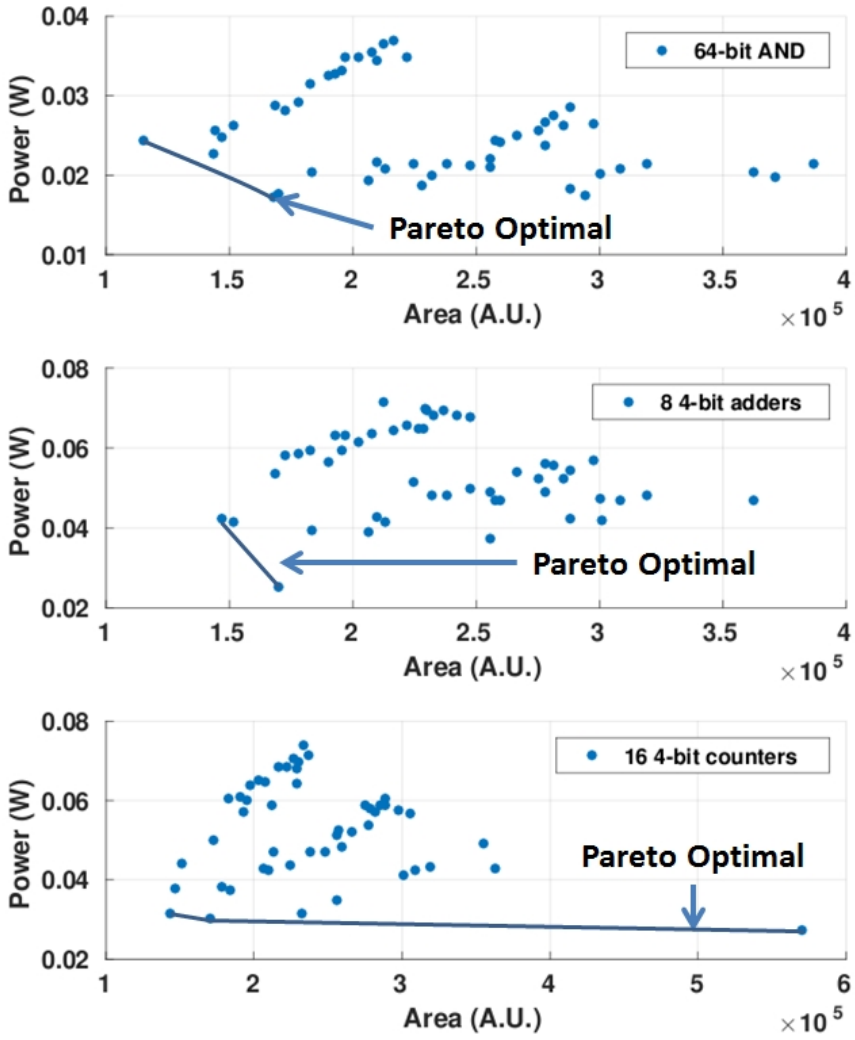

Figure 4.12: Scatter plots of area and power consumption of FPGAs implementing the three different benchmarks. Pareto-optimal architecture choices are highlighted.

Circuit	Description	k	N	fc	L	Area (A.U.)	Power (W)
64-bit AND	Min. Power	8	12	0.05	7	168085	0.01727
	Min. Area	8	28	0.05	4	115307	0.0224
8 4-bit Adders	Min. Power	8	12	0.05	6	170134	0.02541
	Min. Area	8	28	0.1	6	146846	0.0424
16 4-bit Counters	Min. Power	8	4	0.05	1	569837	0.02734
	Min. Area	8	28	0.05	1	143870	0.03152

Figure 4.13: Architectural parameters for minimizing power consumption and area in FPGAs with each of the benchmark circuits. Maximizing k and minimizing F_c reduce both power and area. Maximizing N reduces area, but using a balanced N reduces power consumption. A large L reduces area and power for the counters, suggesting that circuits with many more outputs than inputs should minimize L.

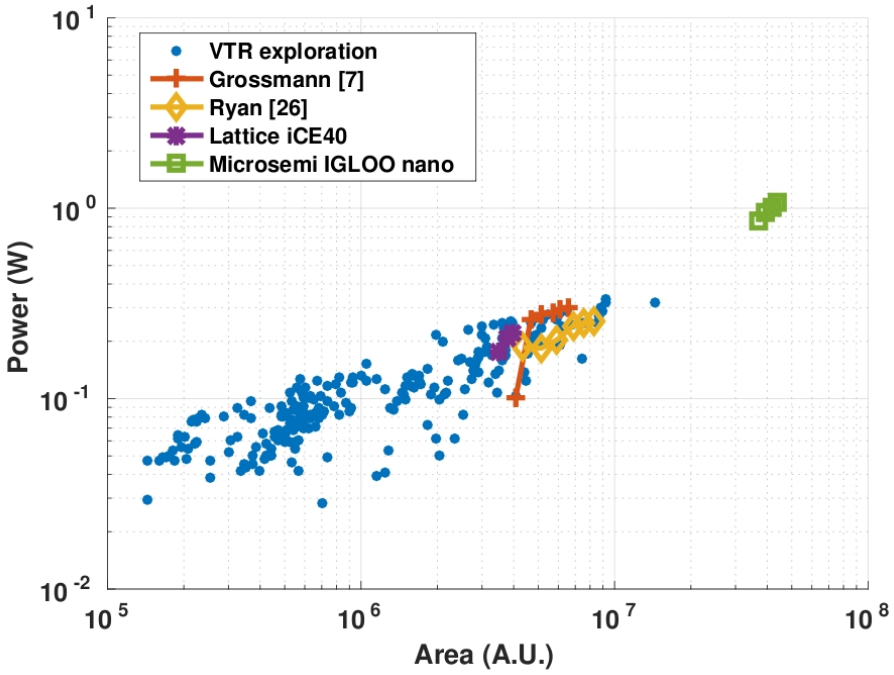

Figure 4.14: Scatter plot of full VTR exploration, with commercial and academic architectures overlayed. The exploration uses architectures that drastically reduce both area and power consumption compared to the accepted low-power solutions.

4.6 Conclusions

In this chapter, I leverage the VTR VersaPower tool to re-investigate architecture choices for ULP FPGAs. Historically, architectural choices focused more on area and performance with little regard for low-power operation. Evidence of this is shown in high-end FPGAs from Xilinx and Altera, boast frequencies of 100s of MHz, but have power consumptions in Ws. Commercial FPGAs designed for low power, like Lattice and Microsemi parts, have very low sleep power, but still have active power consumptions near 1 mW. To further reduce that power consumption, I abstract away the state-of-the-art, low power circuit techniques already employed, and revisit the architectural parameters that have been accepted as optimal. To do this, I look at power consumption as the primary metric of interest, with some interest in area, and no emphasis on performance.

This analysis successfully suggests a different direction for commercial FPGA architectures, whose architectural parameters produce sub-optimal results when observed through the VTR VersaPower tool. This study suggests increasing the LUT size (k) to eight inputs and reducing channel fanout (F_c) for minimizing power and area. This study also suggests greatly increasing clustering value (N) for reduction in routing area, but using slightly higher N values for reduction in power consumption. Depopulation of switch boxes increases overall area and power consumption because

additional FPGA circuitry is required for implementation. This chapter provides a methodology for revisiting FPGA architectures, but provides room for a more thorough investigation of FPGA architectures, as is highlighted in the Future Work section.

4.7 Future Work

Here, I discuss future research directions for pinpointing optimal FPGA architectures for ULP operation.

Update VersaPower models The reported power consumptions of these FPGAs with different architectural parameters is in the 10s of mW, which I have already argued is too high for ULP operation. While conclusions can be drawn from the relative reductions in power consumption based on changes in architectural parameters, it will be important to update the underlying circuits assumed by the VersaPower tool, as results from architectural changes to FPGA fabrics are sure to be different for different physical circuits.

Address VPR bugs Many of the designs that refused to route seemed to be due to bugs in the VPR tool, as opposed to an actual lack of routability. If the FPGA requires a prime number of CLBs (3, 5, 7), and the user attempts to make a 5x1 FPGA, the tool crashes. For that reason, certain k and N combinations failed to work, where the number of required CLBs is prime. Changing the constants in this experiment (especially the logic capacity) such that VTR doesn't could potentially change the results.

Expand design space The result of the design space exploration of architectures is that the borders are the optimal solutions. Thus, it only follows that the borders should be expanded in order for find local (and maybe global) optimum solutions. Clearly, the optimum solution is not simply maximizing both k and N, because the highest possible k would fit the entirety of the logic into a single LUT, which would require $N = 1$. The current limits of the design space were set primarily by time constraints for testing, and the exploration covered 240 architectural combinations. Regardless, expanding the design space is important for finding the true optimal architecture parameters.

It will also be important to expand dimensions of the design space. Many parameters were held constant, such as total logic capacity, interconnect directionality, and switch box architecture, to name a few. Sweeping these parameters as well as those highlighted in this study, with an express focus on minimizing power consumption, can give a more complete picture of the architectural design

space.

Vary depopulation In this discussion, the only option explored for switch box depopulation was full depopulation, because of its apparent use in commercial low-power FPGAs. A more thorough and complete approach would be to vary depopulation, and observe how area and power change as a function of that parameter.

Full FPGA simulations Instead of purely relying on VersaPower estimations, SPICE-level simulation results of FPGAs that employ these differing architecture parameters would give very clear support for an optimal FPGA architecture. The FGC flow that is described in this dissertation is a step towards this, but unforunately cannot yet support all of the necessary architectural parameters. Additionally, the FPGAs generated from this sort of exploration take far too long to simulate. The design space in this chapter of the dissertation tested 1176 different combinations, and simulations for each of those FPGAs at the spice level would take multiple days, possibly weeks. Thus, full FPGA simulation is infeasible. A new simulations strategy with SPICE-level accuracy across architectural parameters will need to be adopted.

More fair benchmarks In this architecture exploration, counters are used for ease of sequential implementation. This poses a few problems. First, these circuits use very few inputs, which may or may not truly address differences in input F_c and k. Secondly, because mapping 20 counters maps mostly outputs, FPGAs with large k reduce the number of logic block outputs, making routing more difficult. Choosing a circuit that better distributes the number of inputs and outputs, but is also sequential (for power estimation), would allow for a more complete architectural sweep.

5 Circuit Exploration

5.1 Motivation

FPGAS are often overlooked for ULP applications, primarily because of the inefficient overhead incurred from reprogrammability. Logic is represented through Look-Up Tables (LUTs), which implemented as memory cells connected to a set of multiplexers. This implementation is inherently less efficient than an ASIC implementation of the same logic. Moreover, FPGAs have reconfigurable interconnect that connect the different logic blocks together, resulting in unused routing resources and configuration memory cells used to select which routing resources are active. ASICs, on the other hand, only have the routing necessary to perform the given function, which greatly reduces the amount of circuitry outside of the logic path. The overall goal of this portion of the research will be to search for better alternatives to the current state-of-the-art FPGA sub-circuits. This chapter focuses primarily on the topologies of switch boxes, CLBs, configuration bits, and LUTs. Other sub-circuits, such as connection boxes, I/Os, and registers (within the CLBs) will not be discussed in this dissertation, and are left for future work. The first step to lowering the overall power and energy consumption of FPGA fabrics is to optimize each FPGA sub-circuit. In this dissertation, I have focused on sense amplifiers (sense amps) for reduced-swing interconnects, configurable logic blocks (CLBs), and the configuration bits used to store configuration information for the FPGA. For FPGAs to be feasible for ULP applications, their energy consumption must be comparable to that of ASICs. To achieve this, each individual circuit element must be re-optimized for new metrics, namely low energy and small area. Once each of the circuit elements are redesigned for more efficient operation, the overall power consumption of the FPGA will be reduced dramatically, closing the gap in efficiency between FPGAs and ASICs. The challenge here is to find new designs for these building blocks that have better power and energy consumptions than the state-of-the-art FPGA sub-circuits, while maintaining a level of flexibility suitable for UbiComp and other ULP applications.

5.2 Configurable Logic Block (CLB) Exploration

Configurable Logic Blocks (CLBs) are the building blocks of FPGAs, and house all of the logic performed by them. Thus, optimizing the CLBs for ULP operation will be paramount for developing ULP FPGAs. In modern FPGAs, CLBs are collections, or clusters, of basic logic elements (BLEs). The focus of this exploration is not to change the topology of the BLEs themselves, but rather redesign the local interconnect between the BLEs inside the CLB. Modern FPGAs (for the most

part) currently use multiplexers to route the inputs and outputs of the CLB with the individual outputs of the BLEs on the inside. However, [28] proposed an alternative solution, building a mini-FPGA style CLB, which uses switch boxes and connection boxes to connect the BLEs. In that design, researchers were able to achieve a smaller CLB area and lower power. In this section, I conduct a deeper exploration of the design space, both in simulation and in hardware, and see under which set of parameters each of the two CLB topologies is optimal.

5.2.1 Prior Art

CLB topologies have not been investigated thoroughly. [28] introduced an alternative idea from the standard multiplexer-based local interconnect inside CLBs which uses FPGA-style routing to connect different BLEs within a CLB together. [20] introduced the notion of depopulating, or removing full connectivity in intra-CLB routing between the inputs/outputs of the CLB and the individual multiplexers inside. However, involved, no study has yet been conducted comparing CLB connectivity topologies against each other across a range of circuit-level knobs (to my knowledge).

5.2.2 CLB Topologies

Figure 5.1 illustrates the two topologies. The mux-based CLB has large muxes at each input of each BLE that allow for connectivity to any of the inputs or outputs of the CLB. In the mini-FPGA case, BLEs are treated like CLBs in an FPGA, connected by a network of switch boxes, connection boxes, and routing wires.

5.2.3 Comparison of CLB Topologies

The knobs we decide to turn for this CLB exploration are:

- **K** – Number of inputs to the individual BLEs of the CLB
- **N** – Amount of clustering, or the number of BLEs total in the CLB
- **Depopulation** – This is a measure of how much the size of the input multiplexers in the mux-based CLBs is reduced, measured in percentage (for example, 50% depopulation would allow connectivity to half of the signals). This is a common technique used in FPGAs to minimize the multiplexer size in the CLBs, reducing area and power consumption.
- **Channel Width** – This parameter refers to the number of routing channels are present in the local interconnect of the mini-FPGA CLB. Reducing the number of channels helps lower

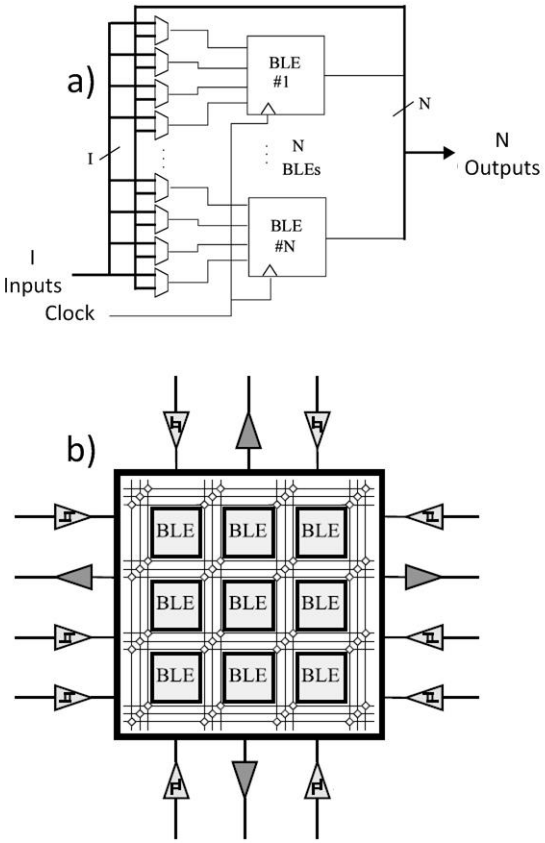

Figure 5.1: a) multiplexer-based[**?**] and b) mini-FPGA[28] Configurable Logic Blocks (CLBs)

power consumption and area, but limits routeability through the CLB.

- **Operating Voltage**

The evaluation metrics for the CLB comparison are:

- **Area** – We will attempt to reduce CLB area as much as possible. Smaller CLB area corresponds to smaller capacitances, corresponding to lower power dissipation and energy consumption inside the CLBs. Moreover, smaller CLBs also reduce the wirelengths of both the intra-CLB and global interconnects, and therefore the power dissipation and energy consumption.

- **Power Consumption**

- **Energy Consumption**

- **Delay Consumption**

Area Figure 5.2 plots the total number of transistors attributed to the local interconnects of both multiplexer-based and mini-FPGA CLBs against clustering. The transistor counts for both fully buffered and unbuffered multiplexers are plotted to show the bounds of possible transistor counts for multiplexer-based CLBs. The number of transistors in the mini-FPGA style CLBs increases lin-

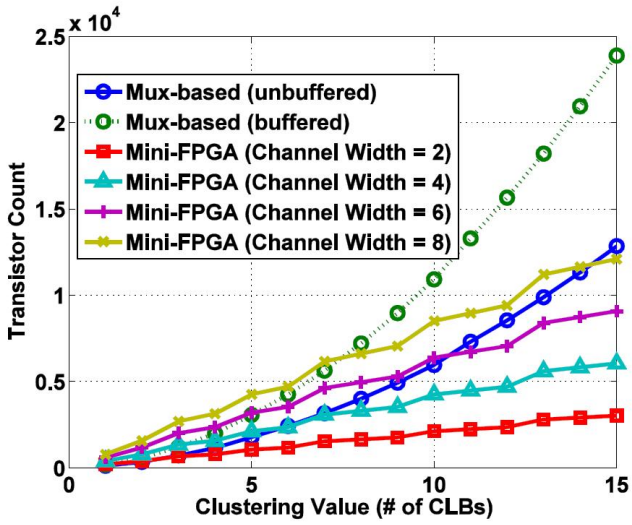

Figure 5.2: Total number of transistors vs. clustering value for different CLB types. Multiplexer-based CLBs become very large with high clustering values. The channel width of Mini-FPGAs contributes heavily to the transistor count.

early with the size of the cluster, whereas the count for the mux-based CLB increases exponentially. The buffered mux-based CLB has more transistors than mini-FPGA CLBs with a channel width of 8 at a clustering factor of 9, which is smaller than the best clustering value for CLBs as determined by [2] and [21]. It has been empirically determined that a channel width of 6 for the mini-FPGA CLB is sufficient for mapping the entirety of the MCNC Golden 20 benchmarks, which are often used to benchmark FPGAs. At this channel width, the transistor count of the mini-FPGA CLB with 15 BLEs (9072 transistors) is approximately 38% of the buffered multiplexer-based CLB (23880 transistors), and 87% of the unbuffered multiplexer-based CLB (12840 transistors).

Figure 5.3 is a table which shows the cluster sizes at which the multiplexer-based design and the mini-FPGA design have approximately the same transistor count, which will be referred to as the break-even point. At cluster sizes higher than this break-even point, there is a smaller area for mini-FPGA style CLBs versus the multiplexer-based designs. As the channel width increases, the break-even point happens at larger cluster sizes, as number of switch boxes and connection boxes increases. For a channel width of 6, the transistor count can break even at around 5 BLEs per CLB, if the multiplexer is fully buffered. By making the dependency of transistor count linear (in

Mini-FPGA vs. Mux (Buffered) - Break Even Points				
	K = 4			
Channel Width	Break Even Points @ Different Depopulation %'s			
	0%	50%	66%	75%
2	Always Less	N = 4	N = 5	N = 6
4	N = 3	N = 8	N = 11	N = 14
6	N = 6	N = 11	N = 16	N = 22
8	N = 9	N = 15	N = 23	N = 29
	K = 6			
2	Always Less	N = 2	N = 3	N = 4
4	N = 2	N = 4	N = 6	N = 8
6	N = 4	N = 6	N = 9	N = 12
8	N = 4	N = 9	N = 14	N = 16

Figure 5.3: Total number of transistors vs. clustering value for different CLB types. Multiplexer-based CLBs become very large with high clustering values. The channel width of Mini-FPGAs contributes heavily to the transistor count.

the mini-FPGA design) instead of exponential (in the multiplexer case), it becomes more feasible to further increase the cluster size of CLBs.

While discussion of transistor area is important, it is also important to characterize the effect of CLB implementation strategies on the overall functionality of the FPGA as a whole. For the purposes of this comparison, we will look at delay for a notion of CLB performance, and energy as a metric for CLB efficiency. We chose architectural parameters for the simulation that were consistent with the mini-FPGA CLB used in [28]. The LUTs were not given true functions to perform, such as AND gates, for example. Instead, we configured with chessboard patterns, such that the odd bit values were set to 1 and the even bit values were set to 0. This allowed us to test each possible delay path through the CLB by driving the inputs with a binary counter (from 0-15). The delay measurements are pessimistic, as all 9 outputs are observed to find the worst-case delay for the CLB. Using all 9 BLEs will also give the worst-case energy estimate, because circuit elements in all nine of the BLEs will be switching.

Delay Figure 5.4 shows the delay for the two different CLB topologies across supply voltage (VDD), which is swept from sub-threshold (0.3 V) to super-threshold (0.8 V). The two topologies

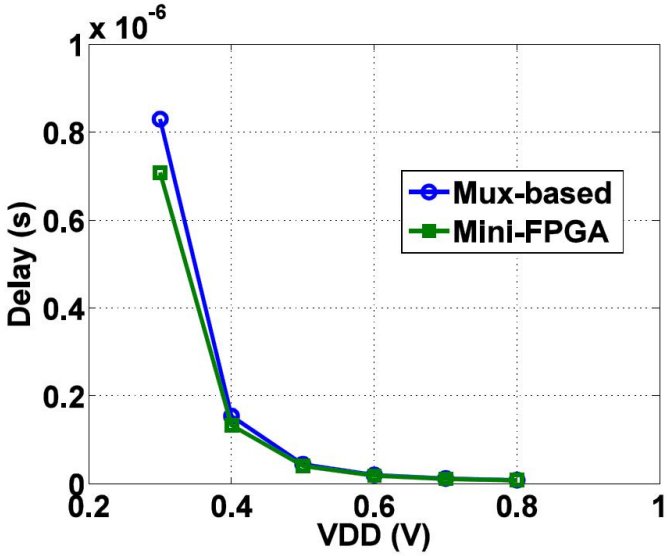

Figure 5.4: Delay vs. Supply voltage for the two CLB types. The two designs have essentially the same delay, w/ the mini-FPGA style having a slight advantage in sub-threshold.

have very similar delays, differing by approximately 7.5% at 0.8 V and by 14.6% at 0.3 V. The mini-FPGA CLB, however, has the better performance, though slightly, across a range of supply voltages, including both sub- and super-threshold. We also observe that delay savings from using the mini-FPGA CLB increase as circuit operation moves deeper into the sub-threshold regime.

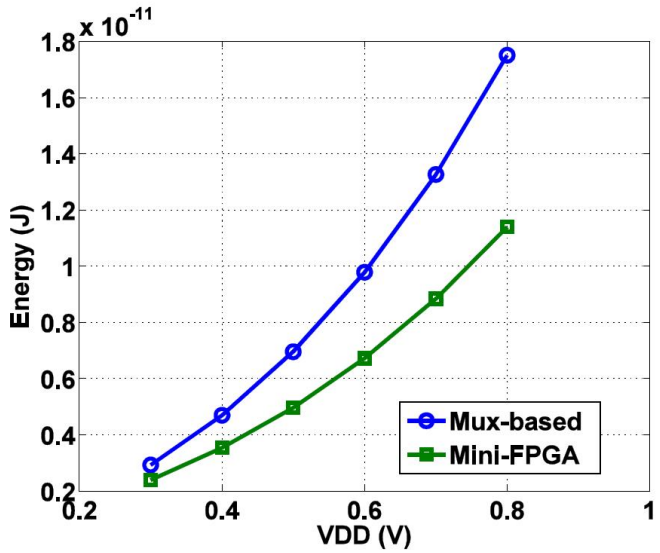

Figure 5.5: Energy vs. Supply voltage for the two CLB types. Mini-FPGA CLB have lower energy across supply voltage than the multiplexer-based CLB due to a lack of switching buffers in the switch boxes and connection boxes.

Energy Figure 5.5 plots energy per operation against supply voltage for the different CLB topologies. For this comparison, we use a frequency of 10 kHz, giving enough time for the operation of the CLBs at 0.3 V. From this plot, we see that the mini-FPGA CLB is a lower energy option across supply voltages, with an 18.1% reduction at 0.3 V and a 34.9% reduction at 0.8 V. This happens for two reasons. First, each of the 29-input multiplexers in the multiplexer-based CLB has 28 buffers, each which actively draw current as the inputs to those multiplexers transition. This is not true for the interconnects of the mini-FPGA style CLBs, which have no buffers at all in the interconnect. Second, the mini-FPGA CLB is slightly faster, which will slightly lower the energy due to leakage per operation as well.

5.3 Configuration Bit Exploration

Configuration bits are large contributors to the overhead of FPGAs over ASIC designs. These bits control the data inside each of the logic blocks, combinational vs. sequential modes for logic blocks, and the large global interconnect. Commercial FPGAs can have on the order of millions of these bits. Because of this, it is important to optimize these circuit elements. To do so, we will compare different memory cell topologies in a way that pertains directly to FPGA functionality. Generally, when looking at different bitcell topologies, researchers tend to focus on reading the data from the memory cell and writing data into it, making sure to balance both of the phases while also minimizing energy consumption. In FPGAs, however, the read and write phase are not important while the device is in use; only the hold phase is. Thus, this configuration bit exploration will be similar to those that have been discussed in the past, but with more of an emphasis on hold margin, data retention, and leakage reduction. In this section, I explore the following bitcell topologies:

- Standard 6T SRAM
- 5T SRAM (proposed in [28])
- 6T sub-V_T SRAM
- 6T sub-V_T latch (proposed in [9])

5.3.1 Prior Art

[9] compared different configuration bit topologies, including standard 6T SRAM, subthreshold 6T SRAM, and a 6T latch. His study found that the 6T latch had the most robust propagation delay, but the slowest write time. The study correctly pointed out that write pulse width is not an important

metric for discussing configuration bit topologies. To extend this study, looking at factors such has hold margin, minimum retention voltage, and leakage current should be highlighted for different configuration bit topologies. [33] compared the differences between using simply high-VT devices vs. using mid-oxide devices, with higher dielectric constants than the other high-VT devices. They found that using transistors that are both mid-oxide and high-VT reduces leakage power by multiple orders of magnitude. In this work, transistor type as well as configuration bit topology will be knobs to turn for optimizing configuration bits for low-power operation.

5.3.2 Configuration Bit Topologies

This section describes the bitcell topologies will be explored in detail. Each has been proposed for use in ultra-low power (ULP) FPGAs.

6T SRAM Cell The standard 6T bitcell includes two access pass transistors, connected to cross-coupled inverters, which are commonly used as storage devices in CMOS circuits. Because inverters are regenerative (inputs that are not full swing result in further swing in the output), the storage node (Q) and it's compliment (QB) are stably held. The bit-line (BL) and it's compliment (BLB) signals are used to provide the values to be written to the bitcell, and the word-line (WL) signal allows for access to the cell. The 6T is illustrated in Figure 5.6a. To write to the cell, the BLs (both BL and BLB) are first loaded with the proper values. Once the BLs are a the proper value, the WL signal goes from 0-V_{DD} to allow the values to write into the cell. The read operation is not discussed in this section, because it is assumed that the bitcells are only held.

5T SRAM Cell (proposed in [28]) Instead of having two access transistors, the 5T cell only has access to the target storage node (Q). This makes writing more difficult, because the bitcell is only written to from one end. The advantage of having only one access point is increased stability. The 5T cell is illustrated in Figure 5.6b.

6T Sub-V_TSRAM (explored in [9] Similar to the 5T cell, the 6T sub-$V_T$cell is also single-ended, but uses a transmission gate instead of a single pass transistor to access the cell. This allows more current to flow through the access, making the write stronger, as well allowing for better passage of high voltages, which makes the writing easier than the 5T cell. The 6T sub-$V_T$cell is illustrated in Figure 5.6c.

6T Latch Cell (proposed in [9]) The 6T latch cell removes the contention in the cell that occurs during a write operation. When writing a value into the cell that is different than the current value, the write operation must overcome the value being forced by the cross-coupled inverters. In the latch, a PMOS transistor is placed between the output of the second inverter and the input of the first inverter, and is turned off whenever the cell is accessed, allowing for the Q node to be changed without contention. When $WL = V_{SS}$, the PMOS is on and the access NMOS is off, allowing for the normal cross-coupled inverter functionality. The 6T latch is illustrated in Figure 5.6d.

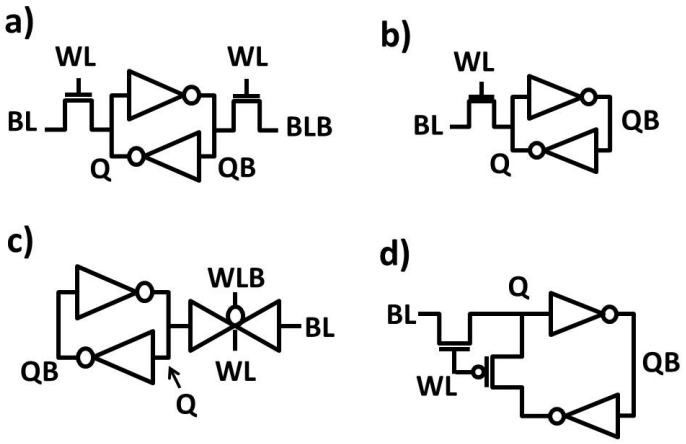

Figure 5.6: Static Random Access Memory (SRAM) topologies. a) Standard six transistor (6T) device. b) Single-ended 5T cell, with lower writeability, but higher stability. c) 6T sub-$V_T$device has increased writeability over the 5T cell. d) 6T latch reduces contention during the write operation.

5.3.3 Bitcell Comparisons

In this section, I compare the different bitcell topologies over multiple design metrics, which are chosen specifically for their importance in FPGA functionality. The evaluation metrics explored in this section of the dissertation for the configuration bitcell topologies are:

- **Leakage Power**
- **Hold Static Noise Margin (HSNM)**
- **Retention Voltage (DRV)**
- **Write Voltage Boost**
- **Bitcell write delay**

Leakage Power Configuration bits are some of the most common circuit elements in an FPGA. These elements are simply holding data and leaking during the functional life of an FPGA. As a result, limiting the leakage power of these bitcells will have a strong impact on the overall leakage power of the device.

Figure 5.7 shows the static current draw across supply voltage for each of the bitcell topologies. Each current measurement shown is the mean current draw across a 1000-pt MC simulation at each voltage. There are two observations to make in looking at these trends. First, all three alternative bitcell topologies reduce leakage heavily compared to the 6T cell. At 0.3 V, for example, the 5T, 6T sub-V_T, and 6T latch cells reduce the leakage of the 6T cell by 41.4×, 36.5×, and 37.3×, respectively. Second, the relationship between the 5T, 6T latch, and 6T sub-V_T cells changes right around 0.5 V.

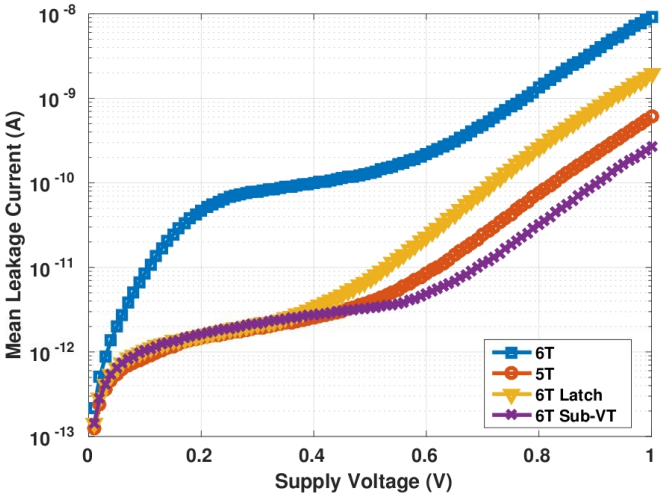

Figure 5.7: Mean bitcell leakage vs. supply voltage for the different bitcell topologies. Each bitcell was simulated using 1000-pt bitcell simulations. 6T cell has much higher leakage than the other three cells. Relationships between bitcell leakages change around 0.5 V.

As a result, I plotted the data without the 6T cell and split at 0.5 V in Figure 5.8. When the supply voltage is less than 0.5 V, the 5T cell minimizes leakage. The 5T cell is at most 66.8% lower leakage than the 6T latch, and at most 16.0% lower than the 6T sub-V_T cell. Above 0.5 V, the 6T sub-V_T cell minimizes leakage, reducing leakage compared to the 5T and 6T latch cells by as much as 57.7% and 87.8%, respectively.

Along with observing the leakage current, it is important to quantify the variability of the leakage current. Figure 5.9 illustrates the standard deviation of the leakage current of the different bitcells across supply voltage. This plot only shows supply voltage values up to 0.5 V, to highlight interesting relationships. Above 0.5 V, the relationship between the variability in the leakage current between bitcells is fairly regular, with the 6T sub-V_T cell reducing variability by as much as 68.7% and 83.6%

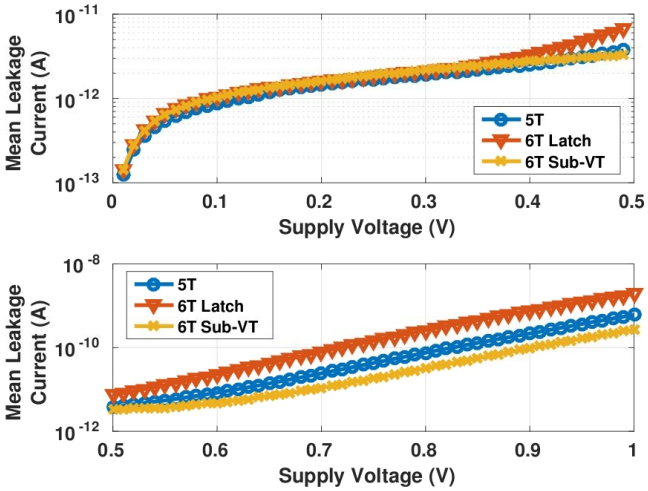

Figure 5.8: Closer look at mean leakage current vs. supply voltage. Below 0.5 V, the 5T cell minimizes leakage. Above 0.5 V, the 6T sub-V_T cell minimizes bitcell leakage.

more than the 5T and the 6T latch, respectively. Below 0.5 V, interesting relationships emerge. The 6T suv-$V_T$cell still minimizes the standard deviation in the leakage, and the 6T still has the highest standard deviation, but the relationship between the 5T and 6T latch cells change across supply voltage. At supply voltages less than 0.1 V, the two are relatively even. The 6T latch is then less variable than the 5T cell, but only up to about 0.38 V, at which point the relative variabilities switch, and the 5T cell becomes more stable (smaller standard deviation) than the 6T latch.

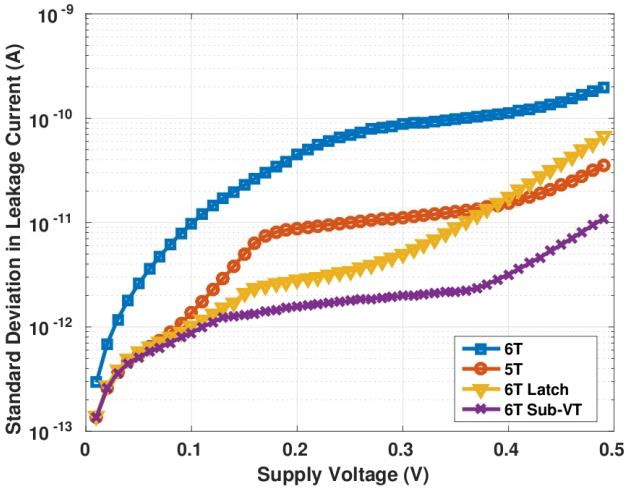

Figure 5.9: Standard deviation of the bitcell leakage across supply voltage Below 0.5 V, interesting relationships emerge. The 6T-sub-V_T cell minimizes variability in the bitcell leakage across supply voltages.

Hold Static Noise Margin (HSNM) While the FPGA is deployed, the configuration bits aren't being read or written to, and are just holding the values. Thus, the most important function of a configuration bitcell is to retain its state, and it's extremely important to measure the stability of each cell while it's holding data, which is quantified by the HSNM.

Figure 5.10 shows the results of a 100-pt MC simulation calculating the HSNM for each bitcell. The 5T, 6T, and 6T sub-V_T cells all have similar distributions, centered between 400 and 500 mV, and skewed to the left. The HSNM in the 6T latch is much less than the others, centered around 200 mV, and is a more normal distribution. A low HSNM shows that the 6T latch is much less stable than the other bitcell choices. This is due to the introduction of the additional PMOS transistor, which is used to eliminate contention when writing to the cell.

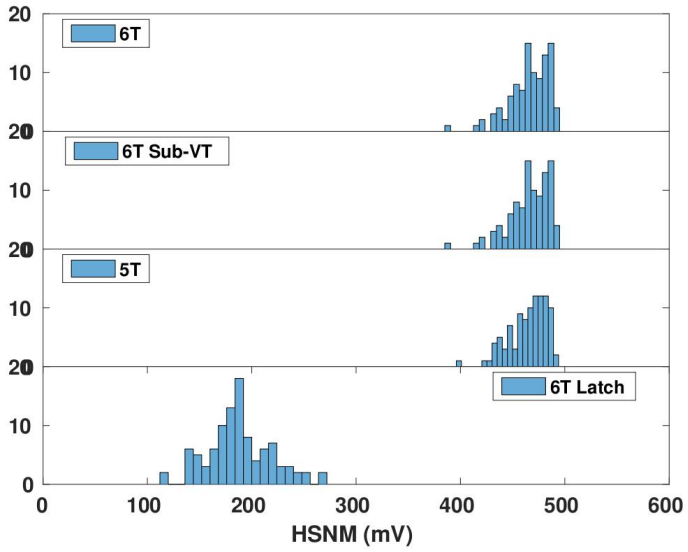

Figure 5.10: Hold Static Noise Margin (HSNM) distributions for the different bitcells. The 5T, 6T, and 6T sub-V_T cells have similar HSNMs, but the introduction of the PMOS between the cross-coupled inverters in the 6T latch cell greatly reduces its stability.

Figure 5.11 displays the mean, worst-case, and standard deviations of the HSNMs for each bitcell, at both 0.5 V (near-V_T) and 0.8 V (super-V_T). Again, the 5T, 6T and 6T sub-V_T cells have similar stabilities. At 0.5 V, the 5T cell has the highest mean HSNM at 267.8 mV, which is only 1.7% and 1.3% higher than the 6T and 6T sub-V_T respectively, but 3x (201%) higher than the 6T latch. The highest worst-case HSNM belongs to the 6T sub-V_T cell at 177.2 mV, which is 14.6× that of the 6T latch, 4% higher than the 6T cell, and 2.9% higher than the 5T cell. The 6T latch is the least robust to variation in HSNM, with the highest standard deviation of the bitcells at 27.6 mV, which is between 46.2 and 48.3% higher than the other three bitcell topologies. The results are slightly different at 0.8 V, with similar conclusions. At 0.8 V, the 5T cell has the highest worst-case HSNM

at 465.9 mV, but the difference between the three stable cells is much lower (less than 1%).

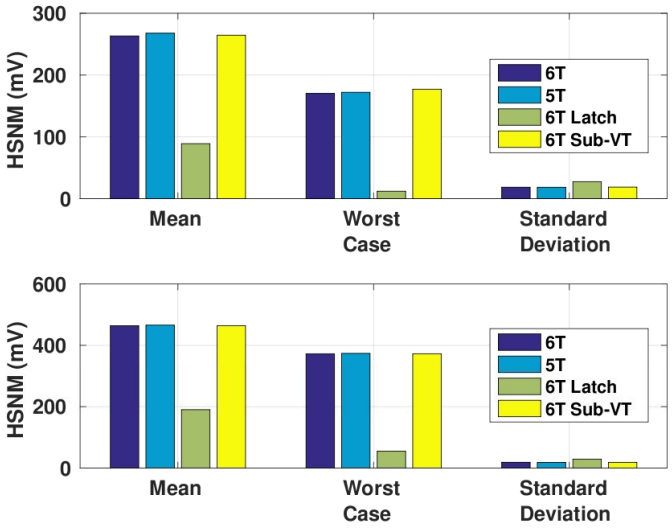

Figure 5.11: Mean, worst-case, and standard deviations of HSNMs for each bitcell. Plots are shown at a) 0.5 V and b)0.8 V. The 5T, 6T, and 6T sub-V_T cells have similar HSNMs. The 6T latch has a much lower HSNM than the others, by about 3× on average, and 14.6× at the worst case.

Because the worst-case HSNM changes with voltage, it is also important to observe the trend of HSNM across supply voltage, and check voltages both near- and sub-V_T . Figure 5.12 shows the mean and worst-case HSNMs across supply voltage. The difference between the 6T latch and the other three bitcell topologies grows with supply voltage, more so for the worst-case HSNM than the mean. The difference between the 5T, 6T, and 6T sub-V_T cells is largest at lower voltages. At 0.2 V, the 5T cell has the highest mean HSNM at 43.1 mV, which is 4.8% higher than the 6T cell (41.1 mV), 95% higher than the 6T sub-V_T cell (22.0 mV), and 123% higher than the 6T latch (19.3 mV). However, 0.2 V is below the data retention voltage (which is explained later in this chapter), and functionality at such low voltages is infeasible. At more realistic voltages (as shown previously), the 5T, 6T, and 6t sub-V_T have very similar HSNMs, and the 6T latch is much more unstable to noise.

Data Retention Voltage (DRV) Reducing voltage is the most effective way of lowering power consumption. As a result, it will be important to characterize different bitcell topologies based on how low the supply voltage of the bitcell can be reduced and still reliably retain the information stored inside. This minimum voltage of storage ability is referred to as the *data retention voltage* (DRV)

In order to test for DRV, a 1000-pt Monte Carlo (MC) simulation is conducted. The supply voltage to each bitcell is reduced until the configuration bit loses state, and that minimum voltage

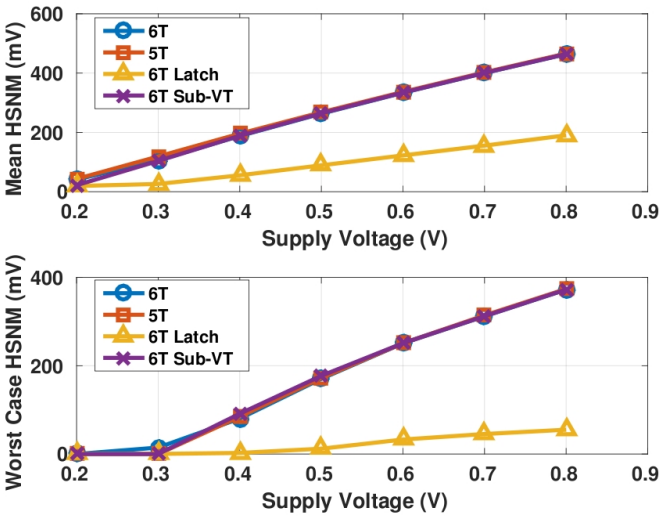

Figure 5.12: Mean and worst-case HSNM as a function of supply voltage (V_{DD}) All three of the cells are very similar across VDD except for the 6T latch, which is markedly lower.

is recorded. It is common practice for DRV measurements for 6T SRAM cells to set both the BL and BLB nodes (illustrated in Figure 5.6) to the supply voltage (V_{DD}) when measuring DRV, which represents the typical case for bitcells. For single-ended bitcell topologies (like the 5T cell, the 6T Latch and the 6t Sub-V_T latch), there is no compromising BL setup. The only two options for the BL value are either V_{SS} or V_{DD}, which are both either the worst-case or the best-case scenario for data retention in the cell. If the BL and Q node in the bitcell are the same value, it is the best case scenario for holding the cell. If the BL and Q node are opposite values, then it is the worst case for retaining the cell value. Figure 5.13 illustrates the problem posed by simply putting the BL and BLB signals high. In order to make a fair comparison between single-ended bitcells and the standard 6T topology, we will simulate across the four different BL/BLB possible values, and average the DRV calculation over those.

Figure 5.14 shows the distributions of DRV measurements for each cell over the 1000 MC iterations. The DRV in each iteration is the higher retention voltage between storing a '1' and a '0', then averaged over the different BL/BLB combinations. The distributions of the different bitcell topologies are comparable, with the DRVs of each topology ranging from about 100-300 mV. The distribution for the 6T cell seems to be centered slightly higher than the others.

Figure 5.15 shows the mean, worst-case, and standard deviations of the DRV for each bitcell topology. The 5T has the lowest DRV in mean and worst case, whereas the 6T sub-V_T cell minimizes standard deviation. At 184.6 mV across 1000 iterations, the 5T cell has a mean mean DRV 15% lower than the 6T cell (219.0 mV), 9.2% lower than the 6T sub-V_T cell (203.3 mV), and only 1.2%

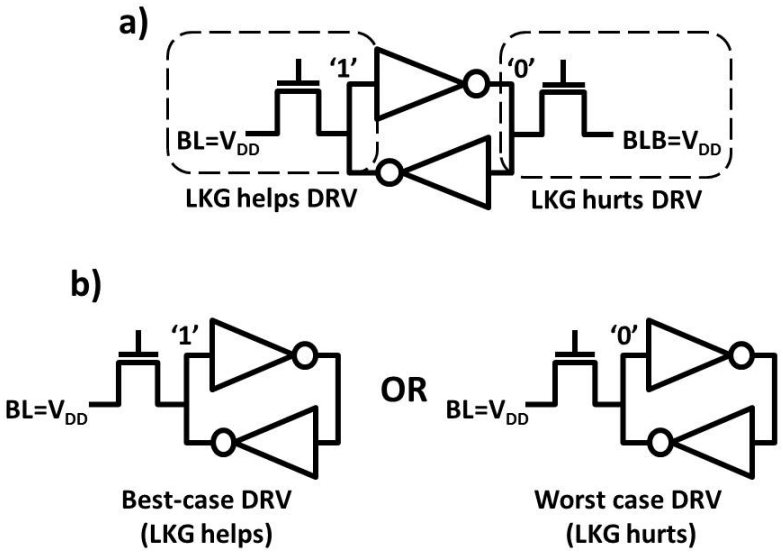

Figure 5.13: Bit-line (BL) and Bit-line Bar (BLB) values and how leakage (LKG) from them effect data retention voltage (DRV) a) 6T bitcell can have BL/BLB combinations that are not worst-case. b) Single-ended bitcells (like the 5T cell) are either best or worst case.

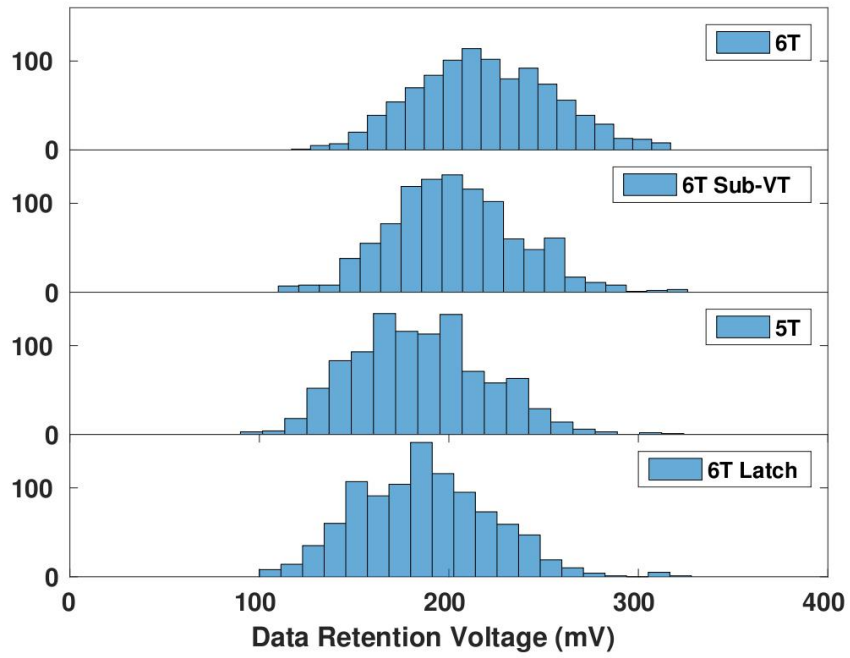

Figure 5.14: Distributions of data retention voltages (DRVs) for different bitcell topologies, taken from a 1000-pt Monte Carlo simulation. Each bitcell has comparable DRV distributions, from $\sim$ 100 - $\sim$ 320 mV.

lower than the 6T latch (186.7 mV). The 5T cell has a worst-case DRV of 305.8 mV, which is 3.1% lower than the 6T cell (315.6 mV), 6.17% lower than the 6T sub-V_T cell (325.9 mV), and 6.23% lower than the 6T latch (326.15 mV). The standard deviations of the cells are even closer together, as evidenced by the similar distributions in Figure 5.14. The 6T sub-V_T cell has a standard deviation of 33.9 mV, which is 6.0% lower than the 6T cell (36.1 mV), 4.8% lower than the 6T latch (35.7 mV), and only 1.3% lower than the 5T cell (34.4 mV).

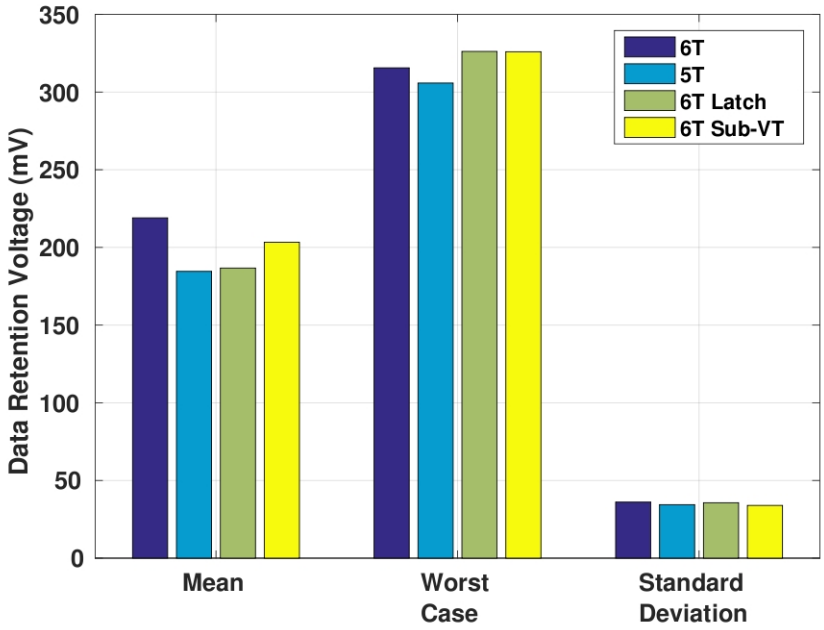

Figure 5.15: Mean, worst-case, and standard deviations of DRV for different bitcells. The 5T cell has the lowest mean DRV at 184.6 mV and worst-case DRV at 305.8 mV. The 6T sub-V_T cell minimizes standard deviation in DRV (33.9 mV).

Overall, the 5T cell minimizes DRV compared to the other cells. The 6T sub-V_T cell is slightly more robust to variations in DRV. The 6T latch has a very similar DRV to the 5T cell overall, but worst-case DRV is closer to that of the 6T sub-V_T cell.

Bitcell Overhead Although some of these bitcells that are designed for stability without consideration given to read and write speed, it should still be understood what kind of overheads will be included in being able to successfully write to the cell. Write operations occur during each configuration, which is usually rare. The overheads incurred by these write operations that will be observed include increased WL/BL voltage, additional control signals, write delay, and bitcell area.

Increased WL/BL Voltage In order to reduce the leakage current in the bitcells, all of the transistors in the cross-coupled inverters are minimum sized and high-V_T. As a result, the WL and

BL voltage need to be boosted in order to successfully write to the cells. While this would be an infeasibly large overhead for standard SRAM operation (because of constant reading and writing operations), this overhead can be ignored for FPGAs. The write operation for FPGA bitcells is not a part of the functional operation of the FPGA, as the write happens on a bench before the FPGA is deployed as a computing solution.

To quantify the overhead of WL/BL increases for writeability, I measure WL boost and BL boost, which is measured as V_{WL}-V_{DD} and V_{BL}-V_{DD}, respectively. Figure 5.16 illustrates the WL and BL boost for each bitcell. The boosted voltage reported is the minimum boost at which all cells in a 1000-pt MC simulation are written to. The 6T latch only requires a boost at a supply voltage of 0.1 V, and only needs a boost at 0.1 V, needing the least amount of increased WL/BL voltage. At more practical operating voltages, no boost is required. The 6T cell also doesn't require much boosting, as the write mechanism helps both sides of the cell. For the 5T and 6T sub-V_T cells, because they are single-ended, it is impossible to write to them without boosting the voltage. Because the 5T cell has a higher resistance access mechanism (pass gate vs. transmission gate), it requires more of a WL/BL boost.

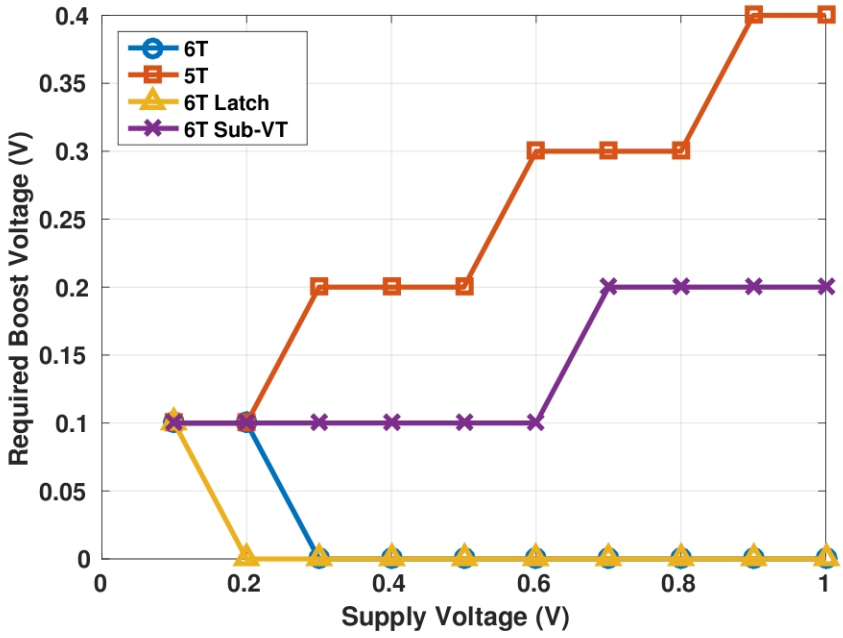

Figure 5.16: Required WL/BL boost in order to successfully write to each bitcell, across a 1000-pt MC simulation. The 6T latch requires the least amount of voltage boost for successful writing, followed closely by the standard 6T cell. The 5T cell and 6T sub-V_T cells require boosted WLs and BLs for successful writes across supply voltage.

Additional control signals Depending on the bitcell topology, additional control circuitry is required. For the 5T cell and the 6T latch, only the WL and BL are required. For the 6T sub-V_T cell, an additional WL signal needs to be generated, and for the standard 6T cell an additional BL is required. These additional control signals needed for the standard and sub-V_T 6T cells require additional inverters (to generate the compliments) that are either local to the cell or included in the top-level of the control signal generation logic, which is a non-negligible overhead.

Write Delay This metric quantifies the time it takes to write into each individual cell. This metric is again less important, because writing to the configuration bitcells happens very rarely in FPGAs compared to regular functionality. However, it is still important to determine the overhead of writing to the cells.

Figure 5.17 shows the mean write delays for the different bitcells. The relative write delays of the bitcells differ at high and low voltages. At higher voltages, the 6T cell has the fastest write. At lower voltages (i.e. below 0.5 V) the 6T sub-V_T cell has the lower average write delay. The 5T has relative low write delay, comapred to the other cells, at low voltage, but becomes the worst at higher voltages.

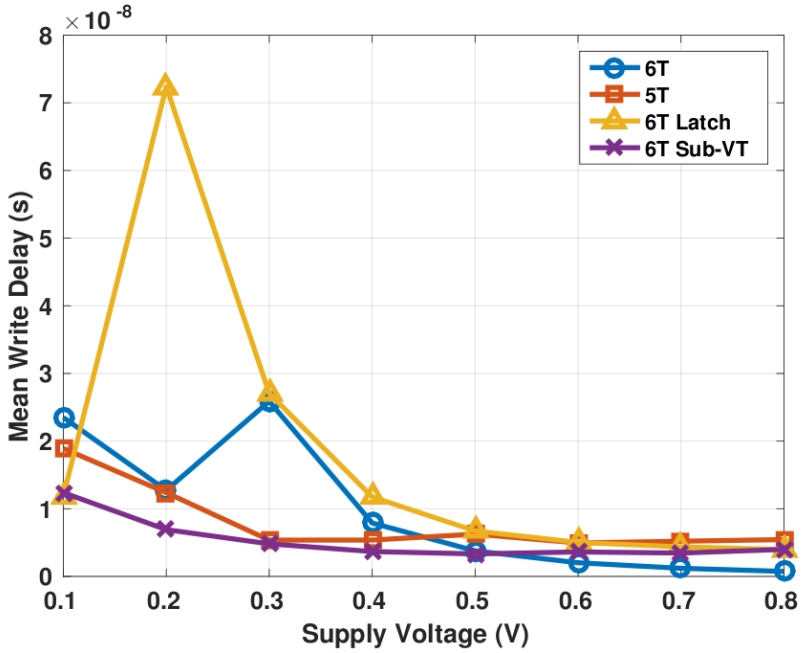

Figure 5.17: Mean write delays for each bitcells across supply voltage, measured from 1000-pt MC simulations. At low voltage, the 6T sub-V_T cell minimizes write delay, where the standard 6T cell does at higher voltages.

Bitcell Area Because most of the FPGA is comprised of configuration bitcells, area consumption of the cell is also of prime importance. All of the transistors have either 5 or 6 transistors, and have similar structures. Currently, as I have not completed the layouts for each of the bitcells, transistor count will work as a proxy for transistor area.

5.3.4 FPGA-level Comparison

In addition to comparing individual bitcell topologies, it is important to see how changes in bitcell topology propagate up to the full-FPGA level. To do this, I leverage the FGC flow (discussed in chapter 2) to generate FPGAs using the different FPGA bit topologies. I then test both bitcell leakage as well as robustness at low voltages at the full FPGA-system level. To do this, I conduct DC simulations to determine bitcell power consumption, and transient simulations to determine proper functionality at low voltage.

To determine the robustness of the bitcells at the FPGA level, I conduct 100-pt MC transient simulations on a small FPGA (1 CLB) configured with a single 4-bit adders at 0.3 V, and observe what percentage of the MC iterations function properly. Because of long simulation times and memory constraints, a much smaller FPGA and much simpler functionality (compared to the 10x10 leakage simulation) are used for MC simulation. 0.3 V is far below the threshold voltage of the transistors, but is where functionality begins to change drastically depending on the voltage. Figure 5.18 compares the percentage of the MC iterations that function properly with each bitcell topology. All of the other physical attributes of the FPGA (circuit design, architectural parameters, etc.) and the algorithms mapped are the same across the 4 FPGAs, so any difference in functionality can be attributed entirely to the bitcell topologies. 0.3 V is far below the threshold voltage of the transistors, but is where functionality begins to change drastically depending on the voltage. The 5T and 6T latch are the most robust at 0.3 V, exhibiting successful functionality 58.5% of the time. The 6T cell is much less robust at 30.9%, and the 6T sub-V_T works surpsingly less frequently than the 6T cell at 18.3%.

In addition to observing the number of successful simulations at 0.3 V, I also checked the power consumption of these small FPGAs. Figure 5.19 shows the distribution of power consumptions for each of the bitcells observed. The 6T cell has a much larger spread than the other cells, and the 5T and 6T latch cells both have relatively tight, low-valued distributions. Figure 5.20 shows the average, worst case, and standard deviations of the power consumptions across the MC iterations of the 1x1 FPGAs. The 5T and 6T latch both have very similar power consumptions. The 5T cell has a slightly lower mean (1.5 nW lower, or) and standard deviation (2 nW lower) than the 6T latch.

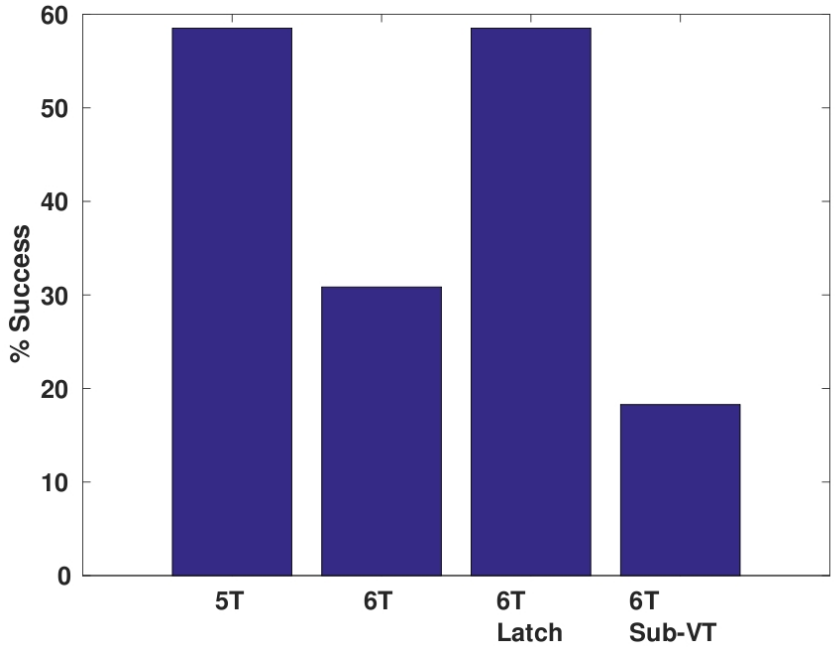

Figure 5.18: Percentage of success across 100 Monte Carlo (MC) iterations of simulations of single-CLB (1x) FPGAs with differing configuration bits. Each FPGA implements a 4-bit adder. Both the 5T and 6T latch succeed 58.

The 6T latch has a lower worst-case power consumption by approximately 15 nW, suggesting that the 6T latch will minimize the necessary power margins that need to be estimated for FPGA use. The differences in these two bitcells are so small, however, that at the FPGA level, looking only at power consumption, the 5T and 6T latch cells are nearly interchangeable. Using either the 5T or 6T latch reduce the mean power consumption from using a standard 6T SRAM cell by 56.9%. They reduce worst case power consumption by 63.6% and reduce variability in power consumption (measured as standard deviation) by 73%.

The table in Figure 5.21 explains the conclusions of the bitcell exploration. Each of the cells are ranked 1-4 based on their performance in each category. The 5T cell, based on the criteria of this dissertation, is the better bitcell topology, having the lowest DRV, lowest low-voltage leakage, and highest bitcell robustness, both at the single bitcell level and at the full FPGA level. Moreover, the categories in which the 5T cell performed poorly (write delay and write assist) are not critical to the functionality of an FPGA. The 6T latch and 6T sub-V_T cells have similar rankings. The 6T latch performed well at high-level FPGA simulations, whereas the 6T sub-V_T cell has strong leakage and HSNM values, and also proved to be most robust to variation, minimizing standard deviation in leakage, DRV, and HSNM.

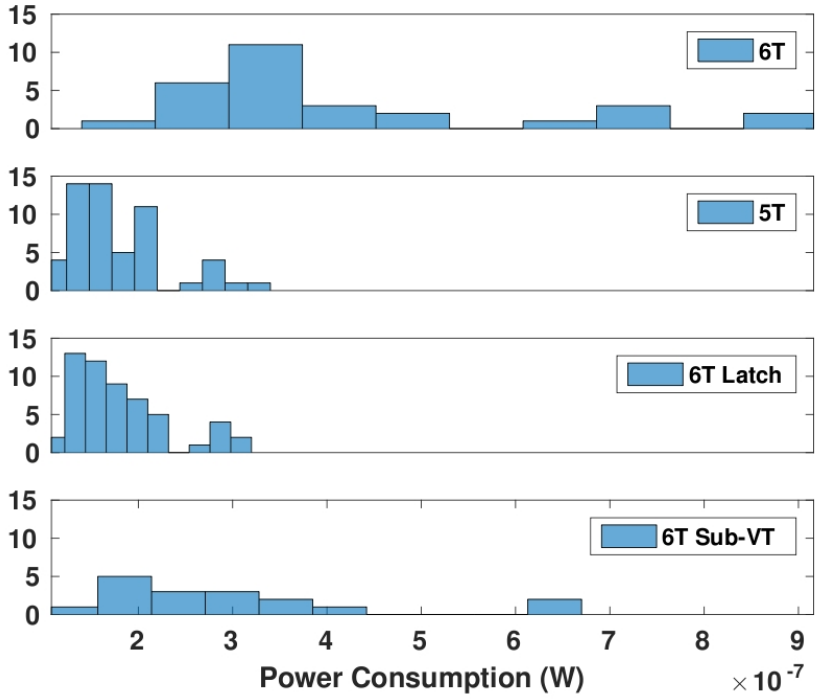

Figure 5.19: Distributions of the power consumption over 100-pt MC simulations of 1-CLB FPGAs with differing bitcells. The 5T and 6T latch cells have much tighter distributions than the 6T and 6T sub-V_T cells.

5.4 Sense Amplifier Exploration

In [28], researchers leverage reduced voltage swing in the configurable interconnect of the FPGA to further reduce power consumption. In order to do this effectively, sense amplifier (sense amp) circuits are required at the output of the interconnect in order to restore the signal to full swing before entering the logic blocks. In this dissertation, I will address the sense amp topology used in [28], and attempt to improve on the design, further reducing the power consumption of the FPGA.

Figure 5.22 shows circuit topologies of different sense amp topologies. The double-buffer is a low swing and a high swing buffer multiplexed together, so that depending on the transition, the proper buffer is used to maximize speed and limit static current. The low-swing buffer is very similar to the modified Schmitt trigger, only with the hysteresis transistor removed. Figure 5.23 shows the voltage transfer characteristics (VTCs) of the different sense amp topologies, as well as the static current consumption across input voltage. In addition to the three custom receivers, we also compare a buffer built from two inverters and a conventional Schmitt trigger. In this plot, the sense amps are compared at an operating voltage of 0.3 V. The double buffer has the best VTC overall, because it has the lowest low-to-high transition, but also the highest high-low transition. This means the

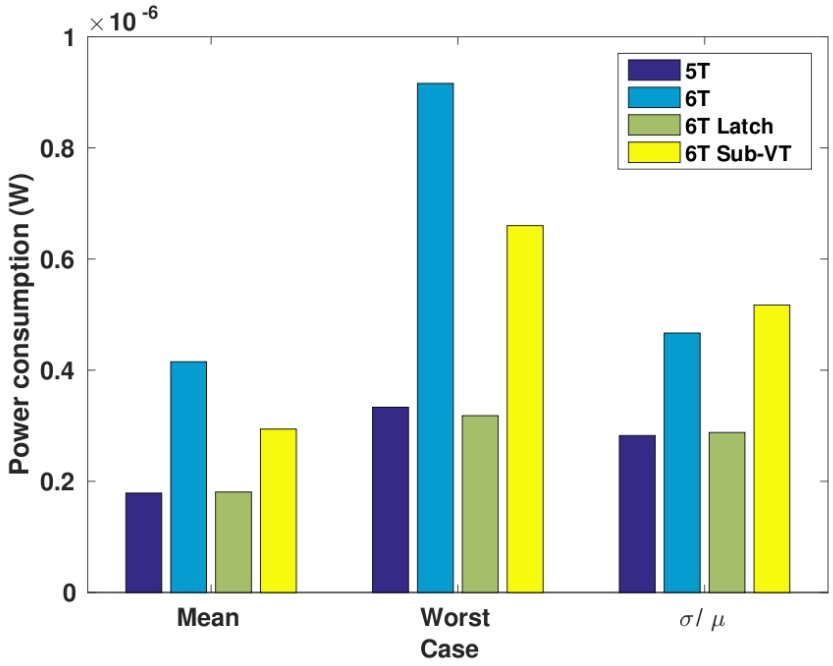

Figure 5.20: Mean, worst case, and standard deviation of the power consumption across 100 MC iterations of 1x1 FPGAs with differing configuration bits. The 5T and 6T latch cells reduce mean power consumption by 56.9%, worst-case power consumption by 63.6%, and standard deviation by 73.9% compared to the 6T cell.

	5T	6T	6T Latch	6T Sub-VT
Leakage - Low Voltage	1	4	2	3
Leakage - High Voltage	2	4	3	1
Hold Static Noise Margin	1	1	4	1
Data Retention Votage	1	4	2	3
Control Signal Overhead	1	4	2	2
Area	1	2	2	2
Write Delay - Low Voltage	2	3	4	1
Write Delay - High Voltage	4	1	3	2
Write Boost	4	2	1	3
Robustness to Variability	2	4	3	1
FPGA-level Power	1	4	1	3
FPGA-level Robustness	1	3	1	4

Figure 5.21: Percentage of success across 100 Monte Carlo (MC) iterations of simulations of single-CLB (1x) FPGAs with differing configuration bits. Each FPGA implements a 4-bit adder. The 5T and 6T latches have tighter distributions, and are centered lower, than the 6T sub-V_T cell, and even more so the 6T.

device will switch quickly in both transitions, and should have decently low static current. It is also important to note that for both Schmitt trigger designs, the high-low transition is at a lower voltage

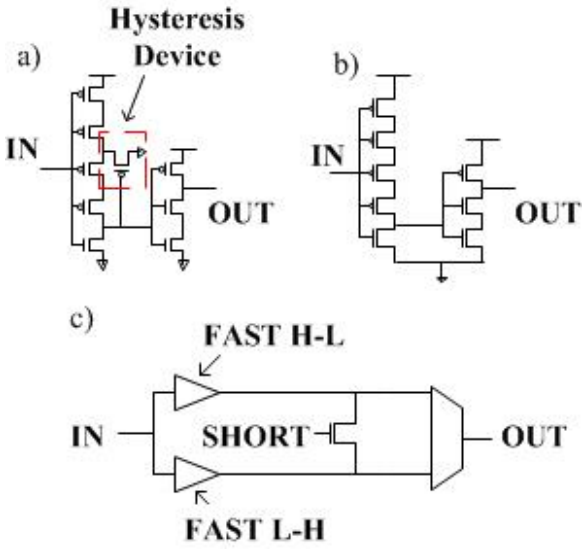

Figure 5.22: a) Modified Schmitt trigger, b) low-swing buffer, and c) double buffer sense amp topologies.

than the low-high transition, which results in very slow high-low transitions. The low-swing buffer has the smallest static current, which also peaks at a lower voltage. As a result, high-low propagation delay is higher for Schmitt triggers, and the static current in these transitions is much higher. In

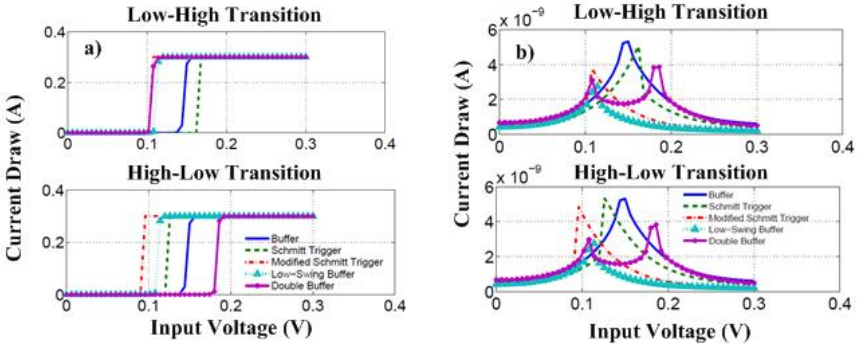

Figure 5.23: a) VTCs and b) static current pofiles for the low-high and high-low transitions of the different sense amp topologies at VDD = 0.3 V. The double buffer has the fastest low-high and high-low transitions, and the low-swing buffer has the smallest static current.

figure 5.24, we compare the energy-delay (ED) curves of the different sense amp topologies. The low swing buffer is generally faster and lower energy than the other alternatives. Each of the sense amps were simulated with 50 passgates leading to them, in order to accurately depict low-swing inputs. In this plot, supply voltage is swept from 0.3 0.8 V. The low swing buffer is the optimal solution, minimizing both energy and delay in almost all cases. Only with the double buffer at very low voltage (0.3 V) can you achieve a slightly faster delay for the same energy consumption.

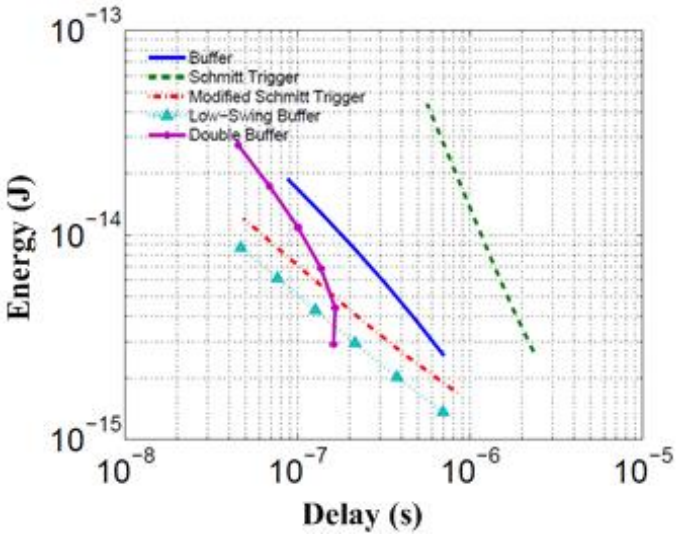

Figure 5.24: ED Curves for sense amp circuits, driven by 50 pass gate switch boxes in series. The low-swing buffer is the overall best option for ULP design

Now that the pass-gate interconnect is optimized, with an improved sense amp design, it is compared to the tri-state buffer alternative in figure 5.25, which shows the ED curves of the tri-state switch box compared to the pass gate switch box. With the low-swing sense amp, the pass gate interconnect design shows lower energy consumption across VDDs. With the same delay (of about 10 s), the pass gate network altenative decreases energy consumption by 87%. In deep subthreshold (VDD = 0.25 V), we find that the pass gate network is 63% faster as well.

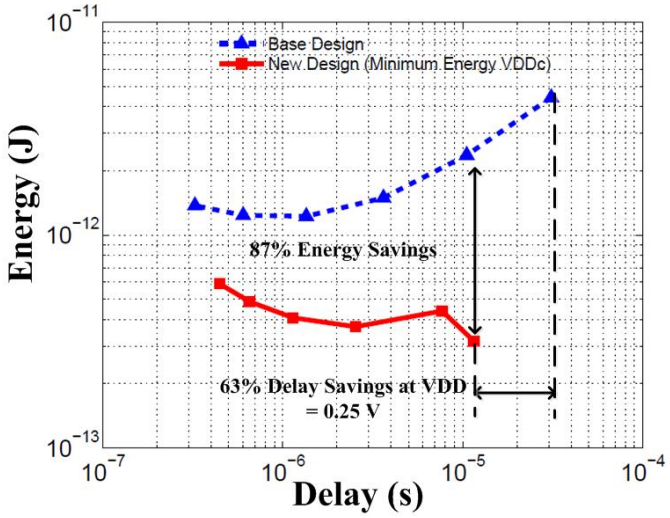

Figure 5.25: ED curves for interconnect networks, sweeping operating voltage. The pass gate interconnect topology is lower energy across operating voltages, and is faster at lower operating voltages.

5.5 Conclusions and Future Work

In this chapter, I've successfully explored different topologies for CLB architectures, configuration bitcell topologies, and interconnect circuitry, in order to further reduce power consumption in FPGA circuits, and further close the gap between the efficiency of FPGAs and more efficient IC implementations like ASICs. Figure 5.26 is a power-delay curve comparing the standard academic FPGA implementation to an FPGA with the same functionality, but using the optimizations shown in this chapter. Both of these FPGAs hold 9 CLBs, amounting to 72 LUTs. The pareto optimal FPGA choice, althrough somewhat marginally, is the optimized FPGA across all voltages.

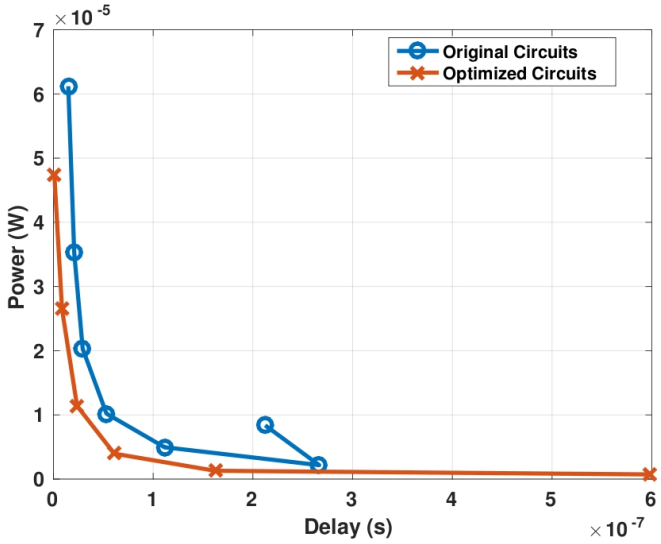

Figure 5.26: Power-Delay (PD) curve for the standard academic FPGA, and an FPGA with this chapters circuit-level optimizations. The optimized FPGA is the optimal choice across supply voltages.

The main goal of these circuit optimizations is to minimize power consumption, which I compare across supply voltages in Figure 5.27. Using the circuit optimizations highlighted in this chapter, the power consumption is reduced by as much as 91.4%. These circuit optimizations work best at low voltage, and the gains from using them drop rapidly as voltage increases. Contributing factors to the high power consumption include leakage into off paths, a poor design choice in isolation of CLB outputs, and potential places for level conversion. These issues will be worked out by the FPGA team here at UVa in the future. Figure 5.28 briefly highlights each of the circuit optimizations, and their outcomes.

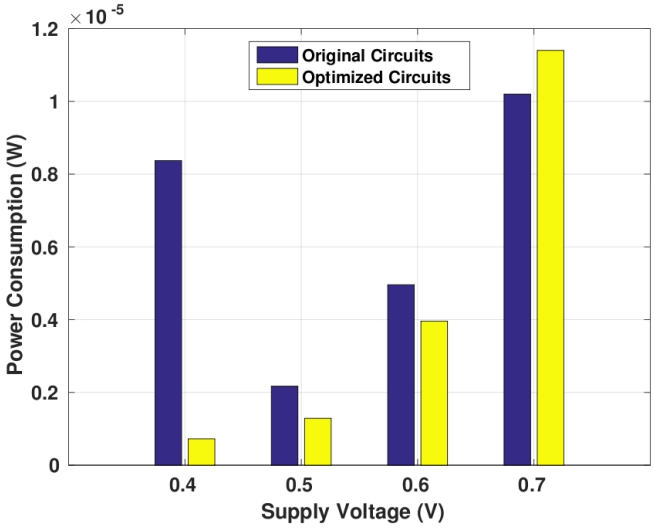

Figure 5.27: Power-Delay (PD) curve for the standard academic FPGA, and an FPGA with this chapters circuit-level optimizations. The optimized FPGA is the optimal choice across supply voltages.

FPGA Sub-Circuit	Choice	Result
Configuration Bitcell	5T Cell	Power ↓ 56.9%
Low-Swing Interconnect	PG + New SA	Energy ↓ 87% Delay ↓ 63%
CLB Architecture	Mux-based or Mini-FPGA	Depends on FPGA arch.

Figure 5.28: Summary of circuit-level explorations, choices, and impact.

6 Embedded FPGAs in SoCs

6.1 Motivation

A popular solution for ULP wireless sensing is developing ASIC-based SoCs with a low-power controller. An example system is introduced in [27]. This SoC is designed to run off of harvested energy, removing the dependency on a battery for power. While the extremely low power consumption is state-of-the-art, there are still potential issues with the design. First, the node is prohibitively inflexible for widespread adoption. Right now, this node can only be used for ECG sensing applications. A node like this would be a viable solution for all IoT applications, but the cost of re-designing the SoC to tailor its design to each application (100s potentially) is extreme, both in time and money. Additionally, making any updates or changes to the algorithms implemented in any of these SoCs also requires a respin of the chip. Aside from flexibility, testing ASIC SoCs can be difficult and costly. Introducing test structures into the different ASIC blocks of the SoC can hinder block performance, and its difficult to attain adequate observability and testability in large, complicated SoCs.

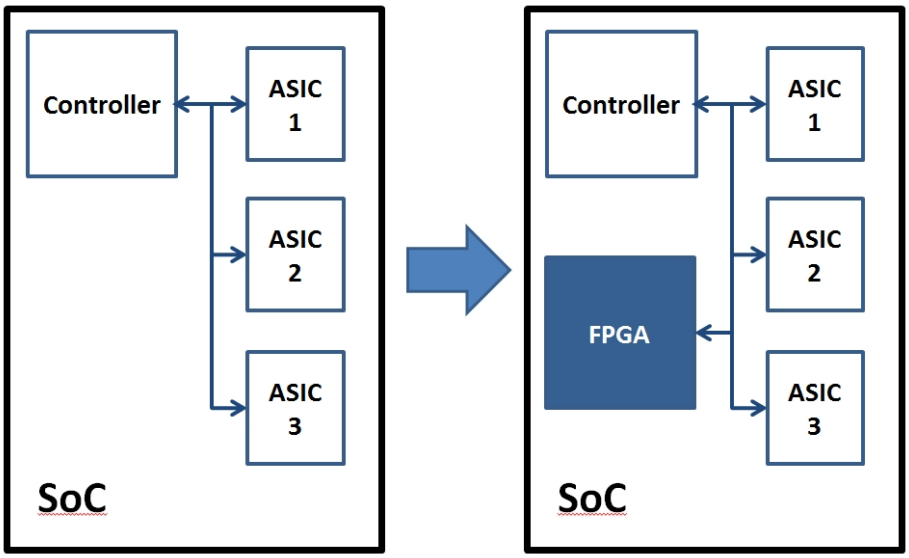

Figure 6.1: Adding an embedded FPGA to increase flexibility and robustness of ULP SoCs.

Including small FPGA fabrics, designed for ULP functionality, could potentially revolutionize ULP sensing. Figure 6.1 illustrates how an FPGA fabric can be included into a larger SoC. Functions that are common to most sensing applications (filtering, arithmetic operations, etc) should remain ASIC-implemented, as ASICs are the most power-efficient implementation of algorithms. Algorithms that are prone to change, or those that are specific to different applications, should be implemented in the low-power reconfigurable fabric in the SoC. That way, a single SoC could be retargeted for a host

of different applications, and the only necessary change would be a re-configuration of the FPGA-portion of the SoC to the specific application. Additionally, the FPGA block can provide redundancy for the SoC. If, for some reason, one of the ASIC blocks in the SoC became disfunctional, the FPGA could be configured to pick up the slack, and implement a less-efficient but functioning version of the ASIC block, allowing the SoC to remain functional. Moreover, algorithms for implementing ASIC functionality is prone to upgrades, especially as more research is conducted in the growing field of the internet-of-things. If algorithms do change for ASIC blocks, the updates can be implemented in the FPGA portion, allowing the SoC to still be useful even after individual ASIC algorithms become obsolete. Figure 6.2 illustrates the many potential uses of an embedded FPGA.

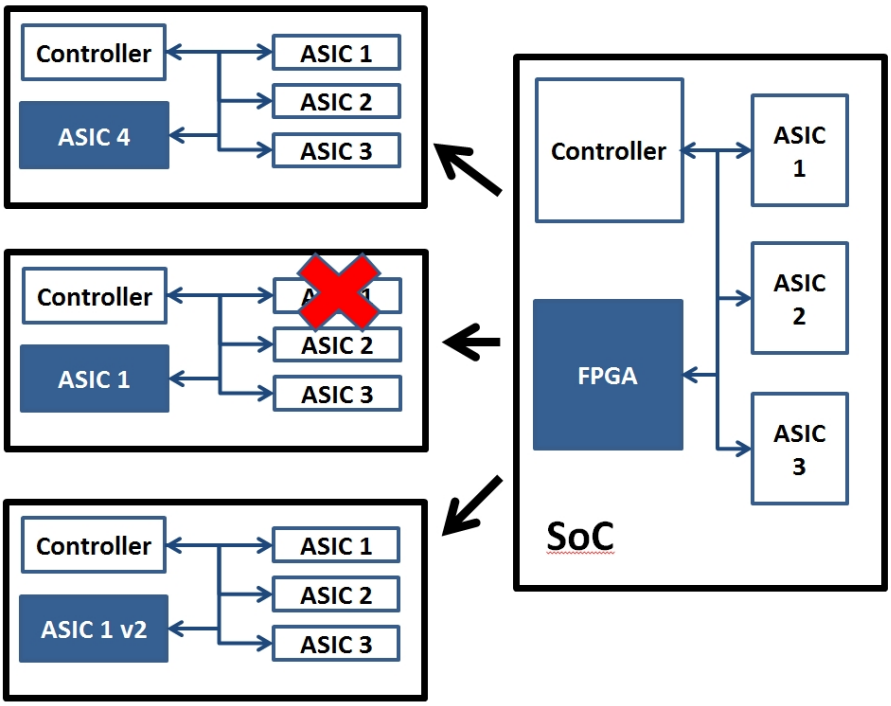

Figure 6.2: Three examples of uses for an embedded FPGA fabric in an SoC. The FPGA can provide additional functionality to the SoC, replace broken ASIC blocks, or update obsolete blocks.

6.2 Prior Art

[1] discusses the benefits of embedding FPGA fabrics into SoCs. This paper gives a methodology for using the FPGA fabric to test itself, and also use it as a means of testing other cores in the SoC. Leveraging the techniques described in this paper would allow a ULP sensor to be tested with as little cost and overhead as possible. Commercial FPGA companies have seen the potential benefits of integrating FPGA fabrics into SoCs, and have begun developing FPGA-based SoCs themselves.

Xilinxs Zynq 7000 All Progammable SoC leverages their state-of-the-art FPGA technology (either Artix-7 or Kintex-7) in combination with an ARM Dual Core processor and a variety of ASIC blocks (DSP slices, floating point processor extension, caches, etc.) [14]. Altera has its own set of FPGA-based SoCs, the newest of which is the Arria 10. This device uses the same ARM processor, but boasts additional security measures, and leverages Altera FPGA technology[7]. These devices, however, are targeted for high-power, high-performance applications, where GHz speeds are necessary. As a result, the power consumption of these devices is on the order of Ws. ULP applications need the power consumption to be in the μW range. Because commercial FPGA companies have illustrated that SoCs with FPGA-fabrics are viable solutions, re-targeting the designs for low-power is promising.

6.3 ASIC Blocks on ULP SoCs

In order to assess the feasibility of using embedded FPGA fabrics on ULP SoCs, we first need to quantify the area and power consumption of the ASIC blocks already present on ULP SoCs. In this chapter, I use the body sensor node reported in [27], a state-of-the-art technology, as an example of an ULP SoC. Figure 6.3 lists ASIC blocks present on that SoC, as well as the area and power consumption for each block. The RR-AFib blocks detects atrial fibrilation by measuring the RR intervals of a heart beat [22].

The information from Figure 6.3 provides comparison points for implementations of the same functionality on FPGA fabrics. ASIC implementations are generally the most area- and power-efficient hardware representations of logic, and thus it is expected that the FPGA implementations of the same functionality will require more area, and consume more power. The goal of this chapter of the dissertation is to determine whether or not the added benefits of flexibility (described in the motivation of this chapter) provide sufficient upside, such that an increase in area and power consumption is worthwhile.

6.4 Mapping SoC functions to FPGA fabrics

In this section, each ASIC block shown in the previous section is mapped to an FPGA, and characterized for area and power consumption. I then compare those power consumptions to that of the ASIC blocks, to see how much overhead is incurred from realizing each of the ASIC functions through FPGA fabrics. Using the VTR tool set, I can take a benchmark circuit, and determine how many LUTs and routing channels are necessary to represent that functionality on an FPGA. That

ASIC Block	Area (mm^2)	Power @ 0.5 V, 200 kHz (µW)
4-Channel FIR	0.307	0.453
CORDIC	0.058	0.830
16-pt FFT	0.207	0.885
Histogram	0.293	0.903
R-R Afib	0.140	0.405
16-bit multiplier	0.019	0.340
Total	**1.024**	**3.816**

Figure 6.3: ASIC blocks on the SoC described in [27], along with the area and power consumption. Each ASIC block is smaller than 1 mm^2, and consumes less than 1 μW of power.

information can be used to make estimates of the total area of the FPGA. The FGC flow (explained in chapter II of this dissertation) allows for the generation of schematics for the FPGA fabrics, that can then be configured with the ASIC functionality, and simulated to obtain measurements of power consumption, which can then be compared to the ASIC implementations.

Unlike studies comparing FPGAs and ASICs in the past, this chapter will leverage the low-power FPGA design that was recently taped out by the Robust Low Power VLSI group. It includes circuit design choices that are explored and recommended by this dissertation, as well as in other FPGA publications from the Robust Low Power VLSI group at the University of Virginia ([28][4][3][24]). In order to simulate these designs effectively, the FGC flow is used to configure and generate custom FPGA fabrics with circuit elements that match physical low-power FPGA that has been taped out.

Unfortunately, the FGC flow will not provide accurate area estimates for the FPGA circuits that are generated. The VTR flow provides area estimates for the routing and the logic of the mapped FPGA, but those estimates come from assumptions about underlying circuitry for the FPGA that are no longer true when the final FPGA product has differing circuit elements. Thus, in order to make a more accurate estimate, we instead take the total area of the taped-out FPGA, and determine the area per LUT per routing track. This allows for an estimation of the area based on an existing, physical design, as opposed to assumptions from the VTR tool. In reality, total FPGA area scales with the number of LUTs and the number of routing tracks differently, and further analysis would allow for the determination of different factors to multiply the numbers of both LUTs and routing tracks, to get a better estimate of the area. For this dissertation, however, we weigh the

contributions to the area of the LUTs and the routing tracks equally for simplicity. The FPGA we taped out and the calculations for the area factor used in this analysis are shown in Figure 6.4. The FPGA we built has 512 LUTs, 84 routing tracks, and takes up a little more area than 4 mm^2. Thus, the factor comes out to just less than 1×10^{-4} mm^2.

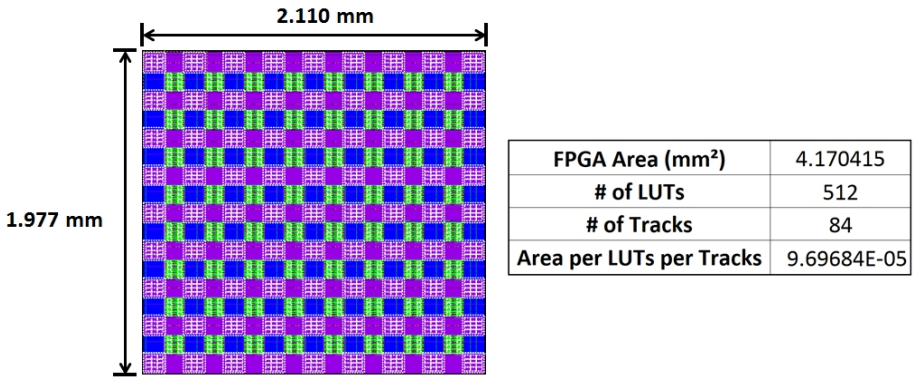

Figure 6.4: Taped out FPGA fabric, and calculations for area factor used in determining the area of mapped FPGAs using the FGC tool-flow.

Figure 6.5 shows the area for the FPGA implementations of the ASIC blocks. As expected, the area increases for each block. Unfortunately, the area consumption for many of the implementations are too high to justify implementing those algorithms on an embedded FPGA inside of an SoC, getting as high as 1000× the area of the ASIC implementations. The RR AFib block has the smallest area increase when implemented in an FPGA, but that is still a ∼20× increase in area. Conversely, if using ASICs, for that same area overhead incurred from adding a small FPGA to the SoC that is large enough to implement an RR AFib block, the designer could add 2 of each of the BSN ASIC blocks. Thus, attempting to use an embedded FPGA to implement ASIC blocks on an SoC seems quite infeasible from an area consumption perspective.

This overhead is even more apparent, and arguably makes implementations of embedded FPGAs more infeasible, when addressing the power consumption of these embedded FPGAs. Figure 6.6 illustrates the power consumption of the FPGA implementations of the BSN algorithms alongside their ASIC counterparts. These estimates come from the VersaPower tool included with VTR. We see here that the power consumption of the blocks is astronomically high for these blocks, increasing power consumption by a minimum of 37,800× for the RR-AFib block, and as much as 576,000× . One matter of saving grace could be that the VersaPower tool uses vastly different assumptions about the underlying circuits, compared to the FPGA architecture that we taped out, and used for the area comparison. Implementing these blocks on generated schematics, and simulating those,

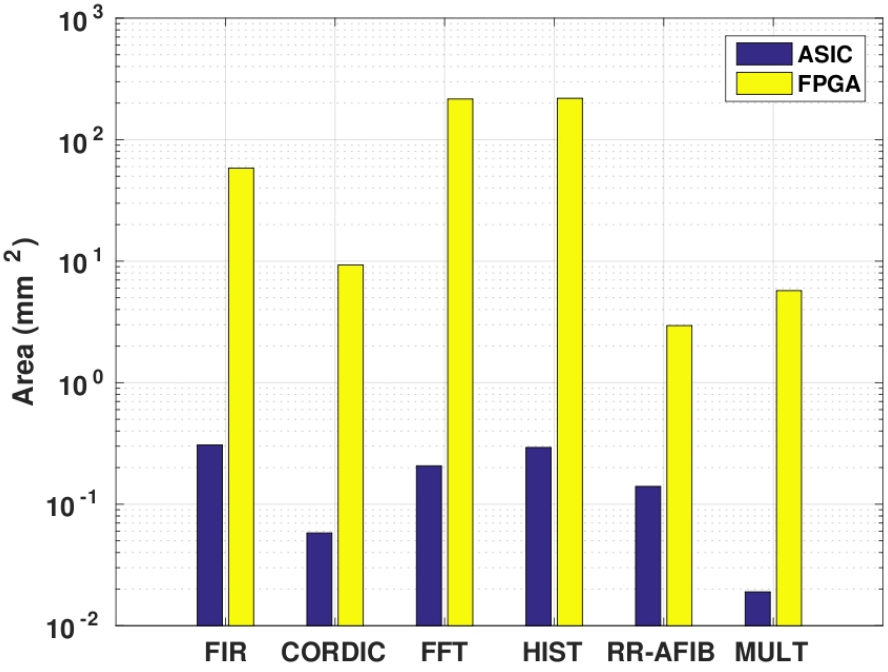

Figure 6.5: Area of ASIC and FPGA implentations of BSN algorithms. Area increases for FPGA implementations range from 20× for the RR-AFib block to over 1000× for the FFT block.

would provide a much better power consumption estimate for each block. However, it would take power reductions of almost 100,000× for FPGA implementations of these algorithms to make sense, which is unlikely to be achieved from more accurate models.

Initial area estimates assumed bi-directional routing in the FPGA fabric. However, because the VersaPower tool only allows for uni-directional routing, each of the blocks required re-mapping using bi-directional routing. An interesting outcome was that the area of mapping reduced as a result of using bi-directional routing. Figure 6.7 displays the area comparison between ASIC implementation and FPGA implementations using uni-directional routing. Even though the taped out FPGA technically has bi-direction routing switches, none of the I/Os of logic blocks are bi-directional, and uni-directional mappings will work just as well. Using uni-directional routing in the VTR tool greatly reduces the number of routing channels required to route designs, probably due to the tool being designed with uni-directional routing in mind. The range of area overheads for ASIC blocks reduces from 20×-1000× to 8×-520×. However, at some point, some of the functionality is almost assuredly ruined. In the mapping of the FIR block, for example, 32 of the 51 inputs were "wiped" away because they "weren't driven" according to VTR. Thus, that mapping is not quite beleivable. The uni-directional implementations overall are much lower overall, and in fact reduce some of the ASIC implementations to some-what reasonable area overheads.

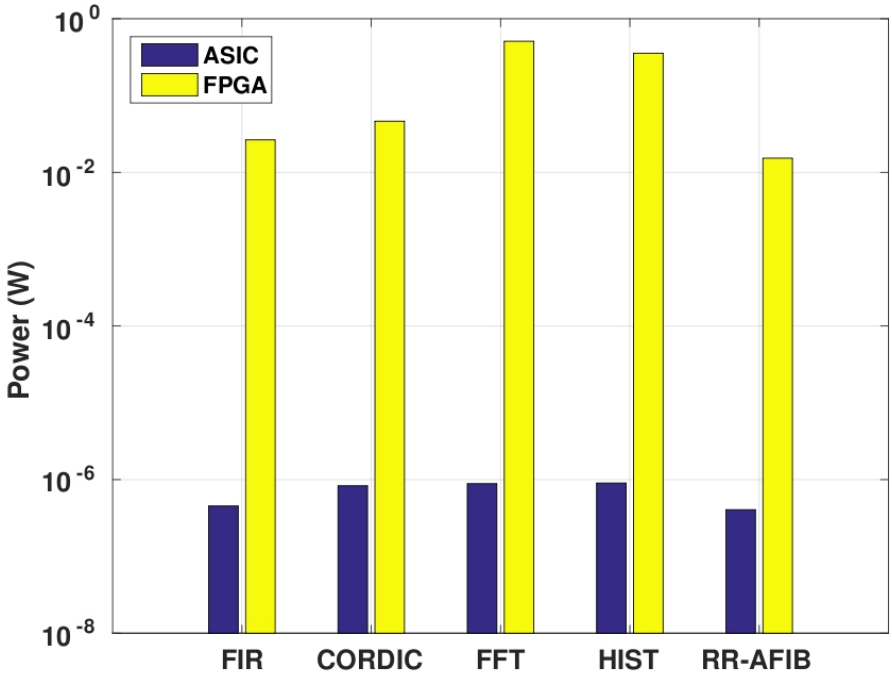

Figure 6.6: Power of ASIC and FPGA implentations of BSN algorithms, as reported by the VersaPower tool. The overhead in power consumption that is reported is astronomical (as high as 576,000× .

In the next section of this chapter, I will explore an additional use for an embedded FPGA, namely built-in self test for other portions of the SoC.

6.5 Using purely CLB fabrics

The exploration of using embedded FPGAs for implementing SoC accelerators shown above presumes that the embedded FPGA is a purely-CLB fabric. However, most FPGAs designed today include some number of hard-IP blocks, making the FPGA fabric more able to attack larger problems. Including blocks like DSP accelerators, arithmetic units, and block memories (which most commercial FPGAs have, but most academic FPGAs do not), combined with the low-power FPGA circuitry discussed in this dissertation, will create a platform that will better address the potential of using embedded FPGAs as replacements for SoC accelerators. Improvements in the FGC flow, allowing for the flexibility to add new IP blocks, will allow for custom-FPGAs to implement such hard-IP blocks.

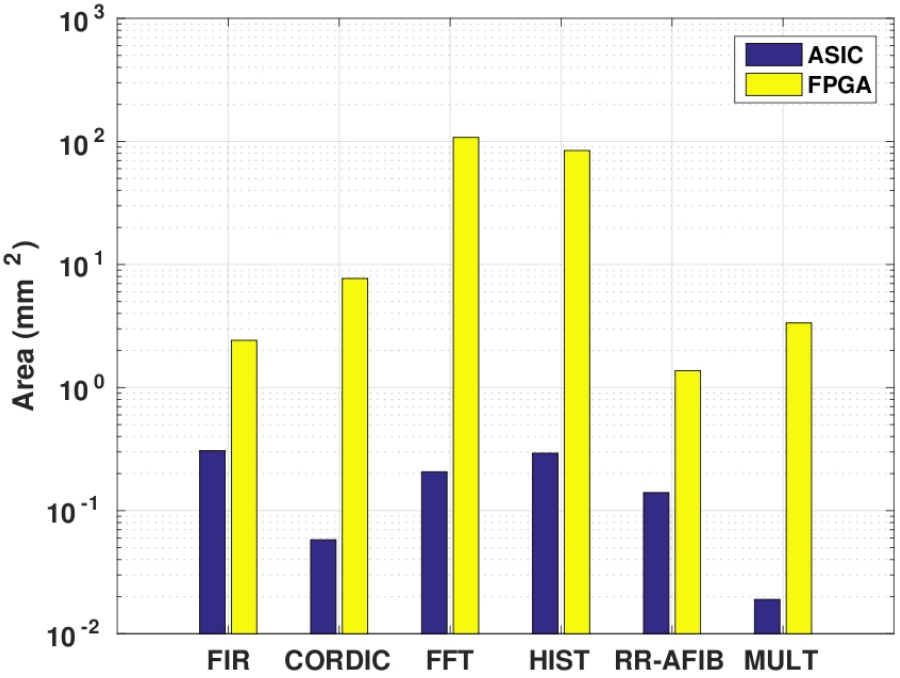

Figure 6.7: Area of FPGA implementations of BSN algorithms using uni-directional routing instead of bi-directional routing. Area is reduced by considerable margins, but is still too high for SoC implementation. There are also questions of functionality loss for the FIR block, specifically, who's overhead is reduced from 188× to 7×.

6.6 Built-In Self Test Mechanisms for SoCs using FPGAs

Embedded FPGA blocks can potentially test other ASIC blocks in the SoC, and even the SoC as a whole. Implementing testing structures with the embedded FPGA would allow ASIC blocks to be built more efficiently, because BIST structures would then not need to be built into the ASIC blocks, thereby removing additional circuitry. Additionally, using the FPGA allows the same test structures to be shared among all of the sub-blocks of the SoC, instead of having specific BIST structures for each ASIC block. Figure 6.8 illustrates the embedded FPGA as a BIST structure for one of the ASIC blocks in the SoC.

Figure 6.9 illustrates how an FPGA fabric can be used to test an ASIC block in further detail. A portion of the FPGA can be configured as an automatic test pattern generator (ATPG) to generate inputs for the ASIC block, either in a deterministic fashion or pseudo-randomly (through the implementation of an LFSR, for example). The rest of the FPGA can be configured as an output response analyzer (ORA). The response analyzer could be designed various ways. It could be designed to simply check for transitions in the output, to verify that the block is alive and functionality. When testing the functionality of a block, the response analyzer can be designed to

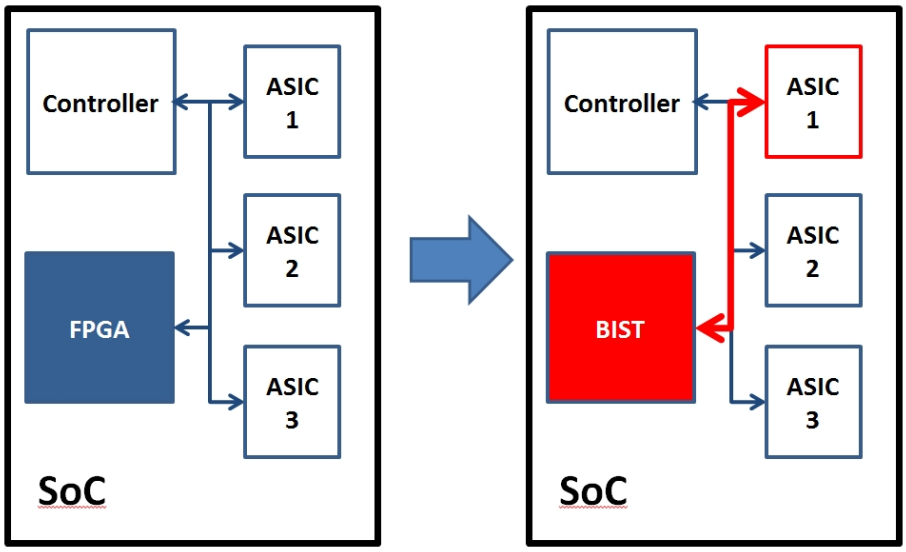

Figure 6.8: Configuring the FPGA as a BIST structure for the SoC. The FPGA can communicate with one or more of the blocks in the SoC (in this example, ASIC 1).

send an alert if the block gives an incorrect output. This would require the user to have a knowledge of the proper outputs, and to configure the analyzing LUTs as a NAND gate, where the inputs are the generated inputs and the expected outputs.

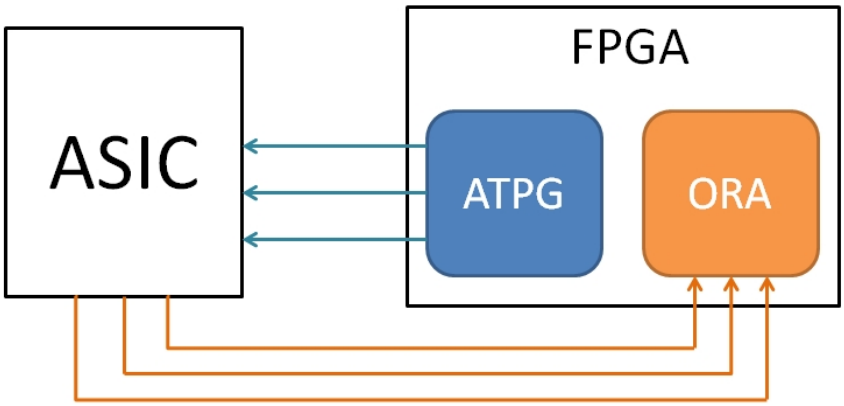

Figure 6.9: FPGA deployed as a BIST structure. Part of the FPGA is configured as an automatic test pattern generator (ATPG) to send inputs to the ASIC block. Other portions of the FPGA are configured as output response analyzers (ORA) to determine whether the block passes the test.

To determine the size of the FPGA needed to test ASICs on the SoC, we first need to determine what circuits would be used in the FPGA to realize the ATPG and the ORA functionality. To simply output constants, a single LUT is required for each constant to give to the blocks. Therefore,

the size of the necessary ATPG would be equal to the number of inputs. For the ORA, the minimum number of LUTs required is $\frac{O}{k}$, where O is the number of outputs from the block in question, and k is the number of inputs to the LUT. This would allow the user to see if all of the outputs are correct or not. However, for more smaller resolution (in order to determine exactly which outputs are failing), at least O LUTs are required. Thus, the total number of LUTs needed is equal to the number of inputs and outputs to be tested.

Another option for ATPG is to generate pseudo-random inputs, which greatly reduces the configuration effort for input generation. I choose to implement the pseudo-random pattern generators using LFSRs for their coding simplicity, and because they are widely accepted for ATPG circuits in BIST. Because each BLE includes an LUT and a register, the number of LUTs required to implement the LFSR is equivalent to that of the constant generation. However, because we can map LFSRs using the VTR tool, we can determine the actual channel width of the implementations, and get a more accurate estimate of the area.

Figure 6.10 the area required to test each block as described above. This area estimate uses the same multiplication factor used before, based on the taped out FPGA fabric. The 4-channel FIR requires the most LUTs at 115, and the estimated area for an FPGA implementing the ATPG and ORA structures for that block is 0.29 mm^2, which is smaller than both the FIR block (0.307 mm^2) and the histogram block (0.293 mm^2).. Thus, including an FPGA block that is only the size of an additional ASIC block can provide on-chip BIST structures for each of the ASIC blocks on the chip.

Still a third option is to provide purely exhaustive testing of all of the inputs. Before exploring this option, it is important to note that doing so is infeasible for large blocks due to time constraints. Taking the FIR block as an example, there are a total of 2^{51} input combinations that could be given to the block. If each input combination was given for only 1 ns, it would take a little more than 26 days to provide every input combination. Exhaustive testing is also rarely necessary, as there is usually only a subset of possible or expected inputs. I still explore implementing ATPGs for exhaustive testing for completeness.

Exhaustive input generators can be implemented using binary counters. Figure 6.11 illustrates the area necessary for implementing binary counters for each of the BSN ASIC blocks. These blocks are higher area than the constants and the LFSR, but are still reasonably sized. With less than double the total ASIC area of the SoC, researchers can have an on-chip tester capable of automatically and exhaustively testing each ASIC block in the FPGA.

If the embedded FPGA fabric could function within a stringent power budget, then it could be possible to test the functionality of the SoC in the field, instead of on a bench in a lab. Doing so could

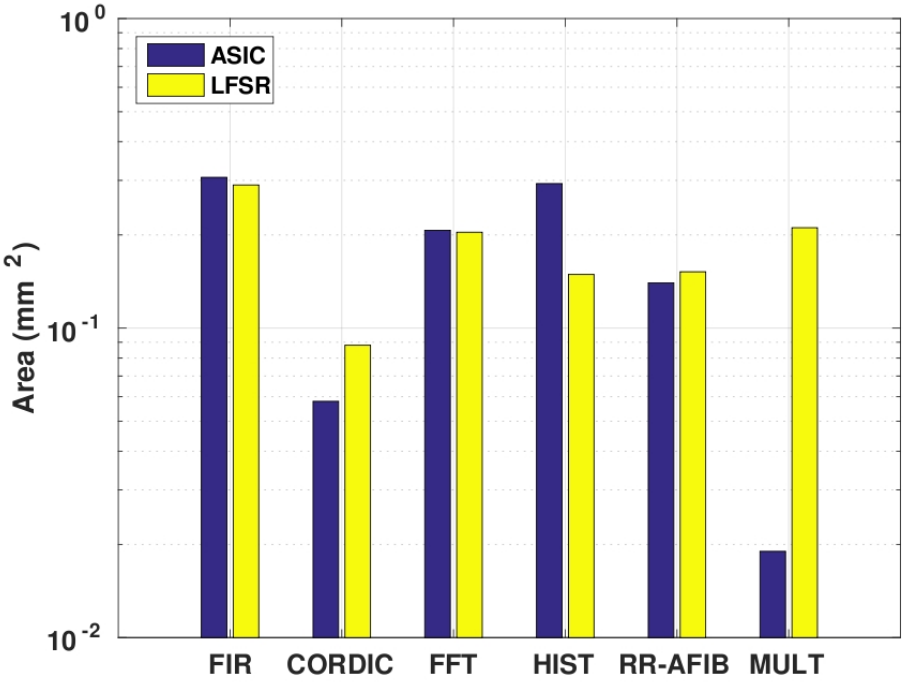

Figure 6.10: Area of ASIC blocks and the FPGAs required to test them. Three out of six testing FPGAs are smaller area than the block it's testing (FIR, FFT, and histogram). Using an FPGA with a similar area as the FIR block would adequately test each of the BSN ASICs.

give valuable insights to designers about how the environment could effect the performance of the SoC as a whole. Because the VersaPower tool heavily over-estimates power consumption for our low-power FPGA fabric, it makes more sense to generate FPGAs using the FGC flow and simulate those. Luckily, the FPGAs for implementing the BIST test structures are small enough for accelerated simulation, which is much more accurate than VersaPower estimates for power consumption.

Figure 6.12 displays the power consumption of the ASIC blocks along side the FPGA implementations of the LFSRs required to test each chip. The power consumption of each block is estimated, but in a more accurate way than the VersaPower estimation. Here, the actual FPGA schematic is generated, but is only simulated for leakage current at 0.5 V. Over the course of many simulations of high-activity circuits at 0.5 V, I have observed that the leakage current of the configurations bits in the FPGA is within one order of magnitude (10×) of the total power consumption. While this estimation is less rigorous in some ways than the VersaPower approach, the answer is guaranteed within an order of magnitude. Figure 6.12 illustrates a large variance in the power overhead of implementing FPGA-based LFSRs as BIST test structures. The FIR block requires the highest overhead for testing using an FPGA. The power consumption of the FIR accelerator is a little more than $\frac{1}{5}$ of the power consumption necessary for the testing FPGA (0.43 μW to 2.01 μW). Other

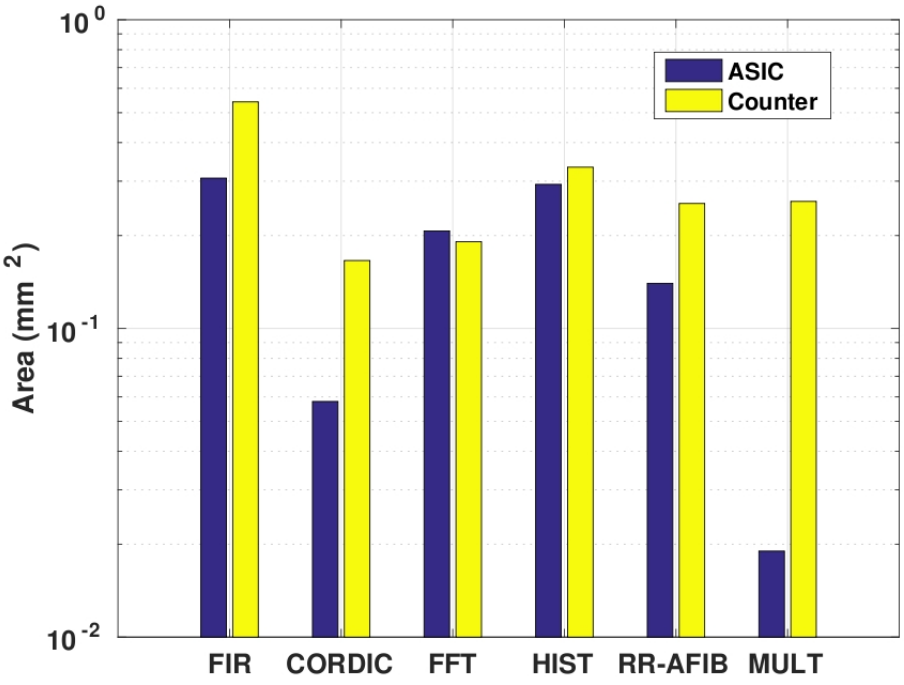

Figure 6.11: Area of ASIC blocks and the FPGAs required to exhaustively testing through binary counting. While these FPGAs are larger, area estimates are still reasonably close to ASIC block area.

blocks (CORDIC, FFT, and histogram) require less power for testing than the block itself.

Figure 6.13 shows the power consumption of the ASIC blocks next to the FPGAs required for exhaustive testing of the inputs. Again, the FIR block requires the most overhead for testing, and the FPGA necessary for its testing is more than 5× larger. The CORDIC, FFT, and histogram blocks can be tested with FPGAs that consume less power than the block.

The discrepancies in the overhead required to test each block comes from the direct proportionality of the power consumption of these FPGA blocks to the number of inputs and outputs. The FIR block, for example, has 3.29× more I/Os than the CORDIC block. Conversely, the power consumption of the FIR block is 54.5% of the CORDIC power. Taking both of these into account results in a 6× reduction in the overhead necessary for the CORDIC than the FIR, even though the power consumption difference between the two blocks is only 2×.

The overall overhead for BIST implementations for the LFSR are illustrated in Figure 6.14 and Figure 6.15. An FPGA large enough to implement LFSRs or constants for testing each ASIC in the SoC is 28.3% of the total ASIC area and 52.7% of the total ASIC power. Once implemented into the SoC, that results in an overhead of 22.1% for area and 34.5% for power. Figures 6.16 and 6.17 illustrate the total overhead required to implement exaustive testing though binary counters.

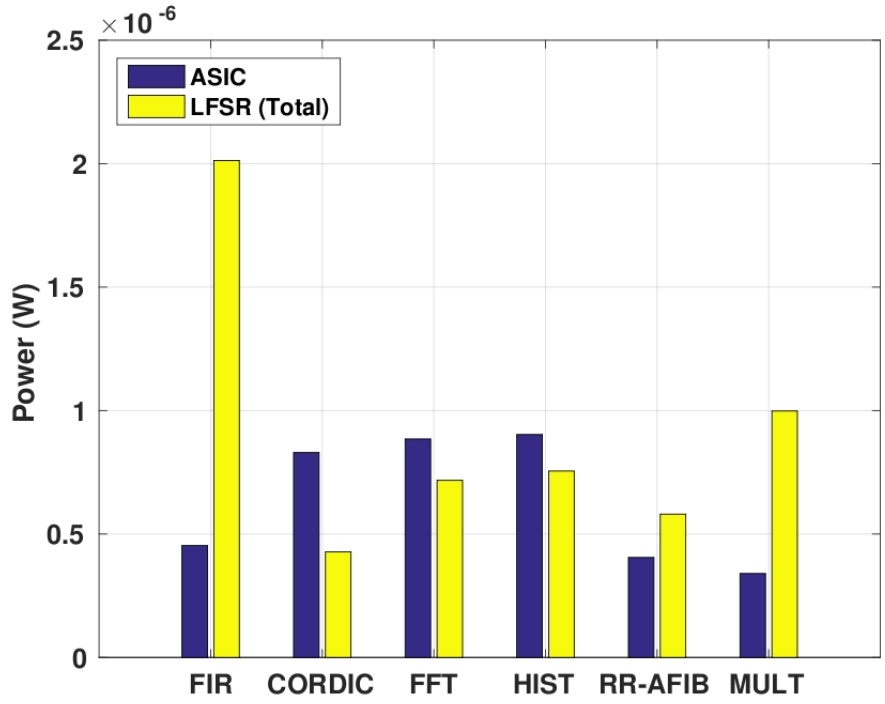

Figure 6.12: Power consumptions of ASIC blocks and the FPGAs required to test them. The FPGA required to test all of the blocks is approximately 5× the area of the FIR block.

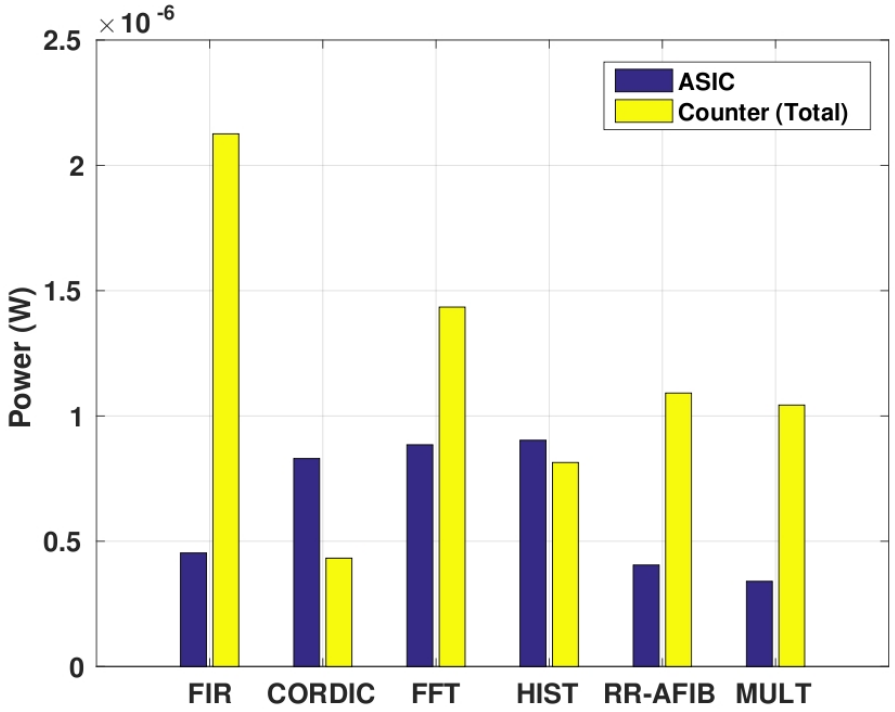

Figure 6.13: Power consumptions of ASIC blocks and the FPGAs required to exhaustively test them.

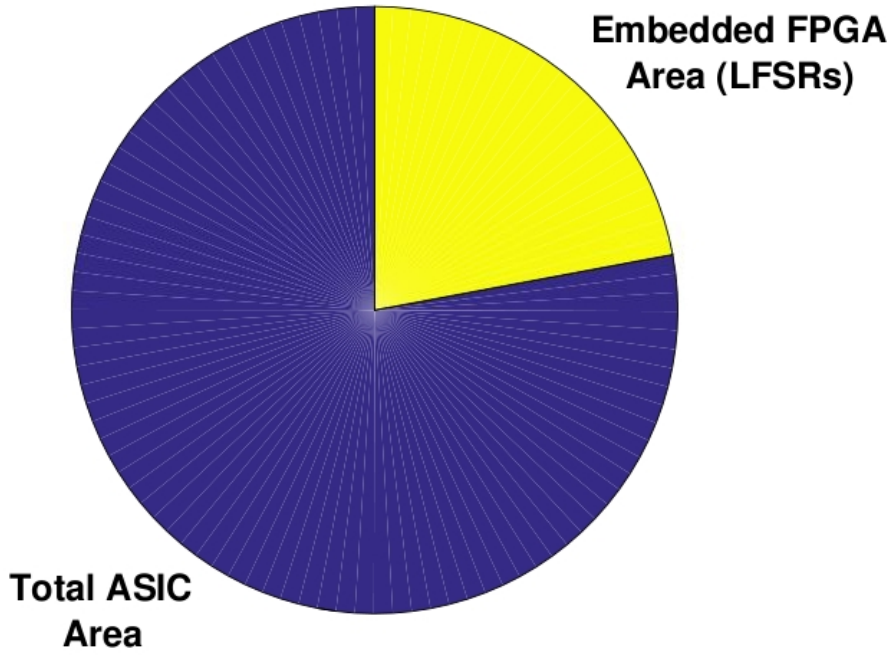

Figure 6.14: Area overhead of embedding an FPGA that could provide constant or pseudo-random inputs for each ASIC block in the SoC and analyze its outputs. FPGA is 22.7% of the total area.

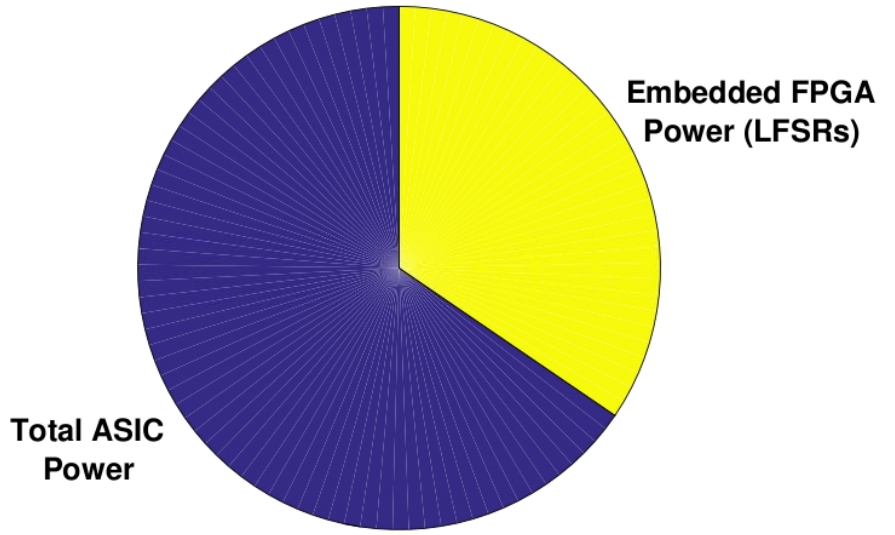

Figure 6.15: Power overhead of embedding an FPGA that could provide constant or pseudo-random inputs for each ASIC block in the SoC and analyze its outputs. FPGA is 34.5% of the total power consumption.

That FPGA would be 52.9% of the total ASIC area and 55.7% of the total ASIC power. Once implemented into the SoC, that results in an overhead of 34.6% for area and 36.8% for power.

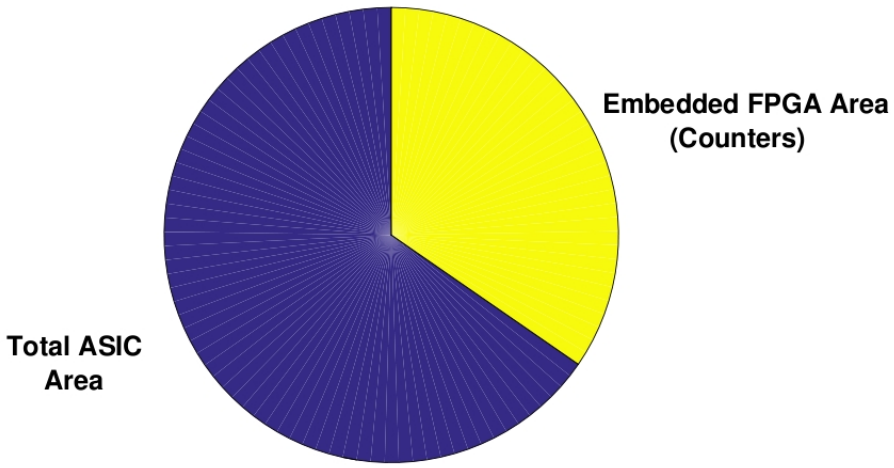

Figure 6.16: Area overhead of embedding an FPGA that could exhaustively test each ASIC block in the SoC and analyze its outputs. FPGA is 34.6% of the total area.

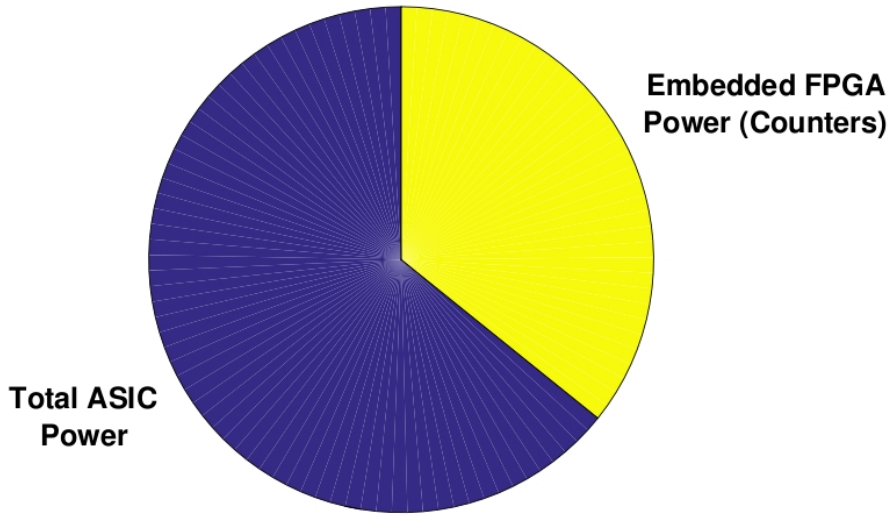

Figure 6.17: Power overhead of embedding an FPGA that could exhaustively test each ASIC block in the SoC and analyze its outputs. FPGA is 36.8% of the total power consumption.

6.7 Conclusions

Embedding FPGAs into SoCs could revolutionize SoCs by providing reconfigurability and flexibility, turning an application-specific circuit implementation into an efficient, flexible platform capable of attacking multiple computing problems. Reconfigurability also allows for additional robustness, allowing the FPGA fabric to replace non-functioning portions of the SoC. The reconfigurable fabric could also function as updated versions of existing ASIC blocks in the SoC. The embedded FPGA fabric can also reduce testing time and cost by implementing on-chip BIST structures, which can

also increase the efficiency of the individual ASIC blocks on the SoC by removing test structures from those blocks.

After studying the area and power consumption necessary for implementing the functions described above, embedded FPGAs for ASIC replacement are unfortunately not feasible for ULP SoCs specifically. The area overhead alone is as much as 1000×, which is far too high for these target applications, as small form factor is an important metric of concern for ULP applications like wireless sensing. Initial power estimates are extremely high (as much as 576,000×!), but are fairly inaccurate.

Using the embeded FPGA as BIST structures for the other blocks in the SoC, however, is very promising. A reconfigurable BIST structure would allow for the chip to reduce overhead (power and area) with regards to testing, and allow for testing in the field once deployed. A portion of the FPGA can be configured with known input test vectors, and another portion as an output response analyzer, which can report failures or successes in the ASIC blocks. For easier configurations, the pattern generation portion of the FPGA can be pseudo-random by implementing LFSRs, or exhaustive by implementing binary counters. While exhaustive testing provides maximum fault coverage, it can be infeasible for large numbers of inputs due to time constraints. Figure 6.18 summarizes the conclusions made in this chapter. According to newer BIST implementations, overheads tend to be less than 10% of total SoC power and overhead ([34][8]). Therefore, while FPGA BIST implementations are promising, additional work is still required to reduce the area overhead to comparable levels to the existing state-of-the-art. While power consumption overheads exist with BIST implementations, the actual power consumption is not a problem during the actual implementation of the BIST testing. A very small percent of the SoC's functional life (less than 1%) is testing, and therefore the power consumption overhead for the testing is amortized over the lifetime of the chip, and is no longer a limitation to its usage. This makes using embedded FPGAs as BIST structures even more intriguing.

6.8 Future Research Directions

Additional analysis is necessary before attempting to use embedded FPGAs in ULP SoC designs. Some of these research directions are described below.

FPGA-Level simulations Power consumption calculations in this chapter are unfortunately inaccurate. CAD tools provide estimations for FPGAs, but do not accurately capture differences in operating conditions and circuit designs specific to custom FPGA designs. The FGC tool, explained in detail in this dissertation, provides schematics for FPGAs, but simulating those schematics is extremely time consuming. SPICE-level simulations of FPGAs this size can take weeks. In order

Application	Potential Benefit	Overhead (vs single ASIC)	Feasibility
BSN Accelerators	- Redundancy for ASICs - Update algorithms	Area: 10-1000x Power: > 10,000x	Low
BIST - Constants	- On-site testing - More efficient ASICs	Area: 22.1% Power: 34.5%	High
BIST - Pseudo-random inputs	- On-site testing - More efficient ASICs - Ease of testing	Area: 22.1% Power: 34.5%	High
BIST - Exhaustive Testing	- On-site testing - More efficient ASICs - Full fault coverage	Area: 34.6% Power: 36.8% - Infeasible for large # of inputs	Medium

Figure 6.18: Conclusions made about embedded FPGAs for in SoCs.

to make accurate power estimates, a combination of modelling that incorporates both circuit-level details and SPICE-level simulation is required.

On-chip configuration mechanisms Embedding FPGA fabrics for implementing BIST structures in SoCs is exciting because it can allow for testing the SoC while it is deployed. However, for that to be possible, the FPGA fabric needs to be configured in the field as well. This presumes that the configuration of that FPGA block is controlled by the SoC, and must be done within the power budget of the SoC.

ULP storage of configurations on-chip An alternative to manually configuring the embedded FPGAs for testing is storing configurations onto the SoC for use later. Commercial FPGAs often have non-volatile memory blocks included with the FPGA fabric to store configurations, but are costly in energy and power. Overheads of storing these configurations need to be explored, as well as automatic configuration strategies, to minimize power consumption. Research is currently being conducted on nonvolatile storage elements for FPGAs ([12]), but this needs to be retargeted for the strict power budgets from ULP applications.

Small-scale reconfigurability It is clear from this exploration that the representation of these ASIC blocks in FPGAs is infeasible. However, increased flexibility for low power sensors is still a powerful notion. Combining ultra-efficient ASIC blocks with a reconfigurable fabric could create a sweet-spot between the efficiency of the ASICs and the flexibility of the FPGA. Small scale reconfigurability has been studied before ([18]), and a solution leveraging the new low-level FPGA fabric research in this dissertation, as well as the low power accelerators described in this chapter,

could result in an effective low-power solution.

Additional uses for embedded FPGAs Other implementations for the embedded FPGA in an SoC make sense for reconfigurability. Encryption algorithms for data transmitted by SoCs may need to change periodically to increase security for highly sensitive information. Implementing encryption with an FPGA allows the user to change the encryption algorithm without re-spinning the hardware of the entire SoC.

Reconfiguring the communication protocols of an SoC could also be very powerful. Instead of requiring multiple ports for communication with different devices, a reconfigurable communications port could potentially reduce power consumption, and would allow the SoC to connect with any device, that may require unforeseen communication protocols.

7 Conclusion

To address the need for ULP, flexible computing solutions for many of today's challenges, this dissertation explores many of the steps required to build and configure custom, ULP FPGAs. This includes creating a framework for rapidly building and testing custom FPGAs, and using that framework to determine optimal circuit-level and architectural parameters for FPGA implementation. This dissertation concludes with a discussion of integrating ULP FPGA fabrics into existing ULP SoC computing solutions, in hopes of addressing some of the present drawbacks of both of the implementations on their own.

The FGC flow described in chapter 3 is a state-of-the-art tool that leverages open-source FPGA mapping software to create a full FPGA-synthesis flow, complete with full schematics, bitstream generation, and a framework for simulation and verification. The FGC flow is the first of its kind to completely close the loop in the design cycle of a custom FPGA. Tools that currently exist either work specifically for commercial FPGAs, or provide one portion of the described flow. This tool has been proposed to allow circuit designers to determine best practices for ULP FPGAs, but the tool can used to create FPGA fabrics designed to meet all types of challenges, including high performance.

Chapter 4 is an exploration of the architectural parameters of an FPGA, and how to choose them so as to minimize power consumption. Because the trend of the FPGA community is more toward high-performance computing than ULP, many of the accepted best practices for FPGA architectures are chosen with performance metrics in mind. Thus, it is worthwhile to see if those practices change much when we revisit the architectures and make reduction in power consumption the sole priority. A design space exploration, which leverages the VersaPower estimation tool in the VTR toolflow, indicates that larger LUT sizes and CLB clusterings then are used in commercial FPGAs minimize power consumption. This chapter also provides a roadmap for continued research in optimizing FPGA architectures for low-power, as many more knobs can be turned to fine-tune FPGA designs.

Chapter 5 is a similar exploration to the architectural exploration, but goes one step deeper to the circuit-level. A similar trend of choosing performance over power efficiency effects circuit-design choices. In revisiting these circuit design choices, it becomes clear that circuit topologies normally avoided for lack of performance are actually better for reducing power consumption. 5T bitcells, all minimum sized, are shown to be the best choice for configuration bits. The 5T cell minimizes leakage at low voltage, data retention voltage, area, and control signal overhead, while maximizing hold stability and system-level robustness. The 5T cell also has high robustness to variability and relatively low write delays at low voltage, though not best-in-class. Chapter 3 also explores inter-

connect sense amp topology. Low-swing pass-gate interconnects with all buffers removed have been shown to minimize power consumption in FPGA interconnects. However, using a low-swing buffer designed to flip at about 30% of the supply voltage proves to be the most power- and energy-efficient choice for a sense amp that restores the swing at the inputs of logic blocks in the FPGA. Finally, this chapter also explores two different CLB architectures, both of which were previously suggested for low-power operation, and determined the "break-even" points between them. Depending on the overall FPGA architecture, it is sometimes better to use multiplexer-based CLBs with varying levels of multiplexer depopulation. This is generally true of architectures with smaller clustering. When there is large clustering in the CLBs, the mini-FPGA architecture, where CLBs are connected by a network of routing channels, switch boxes, and connection boxes, reduces power consumption.

Chapter 6 is a discussion of the feasibility of using embedded FPGAs in ULP SoCs. Prior work describes the potential benefits of using FPGAs on SoCs. They provide additional flexibility, robustness, and update-ability that would not be present otherwise. However, when looking ULP applications specifically, it becomes clear that FPGA fabrics cannot implement many of the ULP sensing algorithms with adequate efficiencies. Thus, using stand-alone FPGA fabrics as a back-up for ASICs on an ULP SoC is infeasible. That does not mean that there is no use for embedded FPGAs in SoCs. The exploration conducted in chapter 6 highlights that the embedded FPGA can be used as a BIST platform for the SoC with a modest 30% power overhead. Optimizing the BIST algorithms, circuit topologies, and FPGA architectures further will bring that overhead down, and make using embedded FPGAs for BIST more feasible on ULP SoCs. Additionally, these comparisons use fullly-CLB FPGA fabrics, which I know are not optimal implementations. As the capabilities of the custom-FPGA generation flow expand, that will allow for more sophisticated FPGA fabrics, which can include ultra-low-power arithmetic accelerators, block RAMS, DSP accelerators, and other blocks, which can even further reduce the gap between FPGAs and ASICs, making embedded FPGAs even more viable.

In the following section, I will conclude this dissertation with a discussion of the high-level impact of the work.

7.1 High Level Impact

Each chapter in this dissertation changes the landscape of ULP FPGA design. FPGAs have historically been overlooked for ULP operation, because of the high cost in terms of power consumption and area for the flexibility that FPGAs provide over more efficient ASIC implementations. However,

this dissertation highlights the validity of using ULP FPGAs as computing solutions.

Figure 7.1 shows the FPGA that was designed using the work in this dissertation, and compares it to the FPGAs highlighted in the motivation of this dissertation. The schematics for this FPGA were generated using the FGC tool flow, and the circuit parameters were chosen based on the work done in this dissertation. Architectural improvements are not included in this design. The FPGA designed has measured (from physical chip testing) static leakage and simulated active power consumption far lower than the other FPGAs while having capacities that fall within the range of all of the commercial parts. By expanding on the tools and the work in this dissertation, further reductions in power consumption can be expected.

FPGA	Size (# of LUTs)	Power (µW)		Config. Bit Topology	Frequency (MHz)
		Static	Active		
Lattice iCE40	384-7680	21-250	just ↓ 1k	SRAM	275
Microsemi IGLOO nano	100-3000	2	400	FLASH	160-250
Ryan et. al. [27]	1134	35	12.5	5T-SRAM	~ 33
Grossmann et. al. [8]	128	8.9	34.6	6T Latch	16.7
Tuan et. al. [32]	1500-15000	46-460	13k-130k	SRAM	244
This Work	**512**	**0.08-0.26**	**0.72**	**5T-SRAM**	**1.67**

Figure 7.1: Comparison of FPGAs, including the FPGA developed using work from this dissertation. This work's FPGA greatly reduces both static and active power consumption compared to the other FPGAs. Including architectural optimizations and continuing to improve the FGC toolflow will result in further reductions in power consumption for FPGAs.

Before the work done in this dissertation, researchers were forced to build FPGAs either by hand, or through a set of unreliable hacks and scripts. The creation of this FGC flow has vastly changed the design process for building a custom FPGA. I have personally worked on building FPGA schematics by hand, and the process can take months. The FGC tool reduces that time to minutes. This gives more than ample time to the designers to explore the unbelievably large FPGA design space, instead of spending most of the design time building the structures. More importantly, custom FPGAs also have to be configured by hand. As a point of reference, the FPGA that was recently taped out by myself and others in my research group has over 200,000 configuration bits that need to be set. Setting each of those by hand, for different algorithmic implementations is completely infeasible. The FGC tool now allows users to configure custom-built FPGAs using the readily available and almost ubiquitously used VTR tool. The FGC tool currently supports generating stimulus for simulation and physical configuration, and in the future will support a greater variety of configuration mechanisms. The tool is already used by multiple people in the research group, and efforts will be made to make the tool even more public.

The circuit design exploration in this dissertation provides concrete evidence for and against assertions made by different leaders in the low-power FPGA research community about different FPGA sub-circuits. It also redefines the list of important metrics for FPGA sub-circuits, which will help direct further improvements on these circuit designs. As an example prior work on bitcell topologies in FPGAs neglected to address hold stability, and instead focused on read and write stability, which are far less important for FPGAs.

The architectural exploration chiefly highlights the need for a deeper understanding of the effects of the FPGA architectural choices on power consumption and area. Commercial FPGAs that target high performance tend to have larger logic blocks and clustering. However, the research in this dissertation asserts that these large blocks actually reduce the power consumption of the blocks. Part of me wonders if the companies that stick with smaller logic blocks do so to differentiate their designs, but the discrepancy between my findings and the commercial state-of-the-art introduces additional questions, that will hopefully spur more research into the area.

The discussion of embedded FPGA fabrics opens the door to new collaborations between ASIC circuit designers and FPGA designers for targeting ULP applications. ASIC designers, who have a deeper knowledge of the individual algorithms, might have an inclination as to which, if any, algorithms have high levels of simple parallelism, making implementation of those algorithms using FPGAs more feasible. Likewise, FPGA designers can help SoC designers to create their blocks such that an FPGA could more efficiently test it, by limiting I/Os, for example. The possibility of embedding FPGA fabrics into low power SoCs could bring researchers with different circuit expertise together to solve some of the world's toughest problems.

A Publications

Ayorinde, Oluseyi, et al. "Using island-style bi-directional intra-CLB routing in low-power FPGAs." 2015 25th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2015.

Ayorinde, Oluseyi A., and Benton H. Calhoun. "Circuit optimizations to minimize energy in the global interconnect of a low-power–FPGA." Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays. ACM, 2013.

Qi, H.,**Ayorinde, O.**, Huang, Y., and Calhoun, B. (2015, September). Optimizing energy efficient low-swing interconnect for sub-threshold FPGAs. In 2015 25th International Conference on Field Programmable Logic and Applications (FPL) (pp. 1-4). IEEE.

A.1 Pending Publication

Ayorinde, Oluseyi, He Qi, and Benton Calhoun. "A Rapid FPGA Generation and Configuration (FGC) Tool for Ultra Low Power FPGAs." Computer Design (ICCD), 2016 34th IEEE International Conference on. IEEE, 2016. (Submitted)

B Glossary of Terms

B.1 Acronyms

BL Bit Line

BLB Bit Line Bar

BLE Basic Logic Element

BLIF Berkeley Logic Interchange Format

CLB Configurable Logic Block

FPGA Field Programmable Gate Array

FGC FPGA Generation and Configuration

LFSR Linear Feedback Shift Register

LUT Look-Up Table

PG Pass Gate

Q Target storage node

QB Complimentary storage node

SA Sense Amplifier

SoC System-on-Chip

SRAM Static Random Access Memory

TX Transmission Gate

ULP ultra-low power

VPR Virtual Place and Route

VTR Voltage Transfer Characteristic

VTR Verilog-to-Routing

WL Word Line

B.2 Terms

Benchmark – circuit description (written in verilog for this dissertation) of target circuit to be mapped to an FPGA.

Bit Line – control signal for SRAM cells that hold the new bit values during a write operation, and precharge and discharge during a read operation.

Bit Line Bar – the compliment of the Bit Line signal, which is used for standard 6T SRAM cells for control of and access to the QB storage node.

Basic Logic Element (BLE) – sub-block of the configurable logic block (CLB). It includes one look-up table (LUT), a register, and a multiplexer that sets the functionality of the BLE to be either sequential or combinational.

Configurable Logic Block (CLB) – logic blocks of the FPGA. Each CLB houses one or more basic logic elements (BLEs). The number of BLEs within a CLB is known as *clustering*.

Configuration Bit – distributed memory cells in the FPGA that determine connectivity of the interconnect, as well and logic in the CLBs.

Look-Up Table (LUT) – smallest logical block in the FPGA. A k-input LUT has 2^k memory bits that store the truth table of any k-input boolean function. The values of each memory bit are connected to a k-to-1 multiplexer, whose select signals are the inputs of the LUT.

Static Random Access Memory (SRAM) – memory cells that are commonly used in FPGAs. SRAM cells consist of cross-coupled inverters that hold state, and access transistors for writing and reading values to the cell.

Word Line – control signal for SRAM cells that hold the new bit values during a write operation, and precharge and discharge during a read operation.

Bibliography

[1] M. Abramovici, C. Stroud, and M. Emmert. Using embedded FPGAs for SoC yield improvement. In *Design Automation Conference, 2002. Proceedings. 39th*, pages 713–724, 2002.

[2] E. Ahmed and J. Rose. The effect of LUT and cluster size on deepsubmicron FPGA performance and density. In *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, pages 288–298. IEEE Computer Society Press, March 2004.

[3] Oluseyi Ayorinde, He Qi, Yu Huang, and Benton H Calhoun. Using island-style bi-directional intra-clb routing in low-power fpgas. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–7. IEEE, 2015.

[4] Oluseyi A Ayorinde and Benton H Calhoun. Circuit optimizations to minimize energy in the global interconnect of a low-power–fpga. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 277–277. ACM, 2013.

[5] Vaughn Betz and Jonathan Rose. Fpga routing architecture: Segmentation and buffering to optimize speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 59–68. ACM, 1999.

[6] Microsemi Coproration. *IGLOO nano FPGA Fabric (User's Guide)*.

[7] Altera Corporation. Arria 10 Device Datasheet, 2015.

[8] Bai Hong Fang and N. Nicolici. Power-constrained embedded memory bist architecture. In *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, pages 451–458, Nov 2003.

[9] P.J. Grossmann, M.E. Leeser, and M. Onabajo. Minimum Energy Analysis and Experimental Verification of a Latch-Based Subthreshold FPGA. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 59(12):942–946, Dec 2012.

[10] Steve Guccione, Delon Levi, and Prasanna Sundararajan. Jbits: A java-based interface for reconfigurable computing. In *2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, volume 261, 1999.

[11] Edson Horta and John W Lockwood. Parbit: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (fpgas). *Dept. Comput. Sci., Washington Univ., Saint Louis, MO, Tech. Rep. WUCS-01-13*, 2001.

[12] K. Huang, R. Zhao, and Y. Lian. Racetrack memory-based nonvolatile storage elements for multicontext fpgas. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(5):1885–1894, May 2016.

[13] Xilinx Inc. Applications. `http://www.xilinx.com/applications.html`.

[14] Xilinx Inc. Zynq-7000 All Programmable SoC Overview (Datasheet), 2004.

[15] Texas Instruments. MSP430F21x1 Mixed Signal Microcontroller, 2004.

[16] P. Jamieson, W. Luk, S.J.E. Wilton, and G.A. Constantinides. An energy and power consumption analysis of FPGA routing architectures. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 324–327, Dec 2009.

[17] Jin Hee Kim and J.H. Anderson. Synthesizable fpga fabrics targetable by the verilog-to-routing (vtr) cad flow. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–8, Sept 2015.

[18] Vinu Vijay Kumar and John Lach. Designing, scheduling, and allocating flexible arithmetic components. In *International Conference on Field Programmable Logic and Applications*, pages 1166–1169. Springer, 2003.

[19] J. Lamoureux and S. J. E. Wilton. Activity estimation for field-programmable gate arrays. In *2006 International Conference on Field Programmable Logic and Applications*, pages 1–8, Aug 2006.

[20] Guy Lemieux and David Lewis. Using Sparse Crossbars Within LUT. In *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, FPGA '01, pages 59–68, New York, NY, USA, 2001. ACM.

[21] Fei Li, Deming Chen, Lei He, and Jason Cong. Architecture Evaluation for Power-efficient FPGAs. In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, FPGA '03, pages 175–184, New York, NY, USA, 2003. ACM.

[22] Jie Lian, Lian Wang, and Dirk Muessig. A simple method to detect atrial fibrillation using rr intervals. *The American journal of cardiology*, 107(10):1494–1497, 2011.

[23] Alexandra Poetter, Jesse Hunter, Cameron Patterson, Peter Athanas, Brent Nelson, and Neil Steiner. Jhdlbits: The merging of two worlds. In *Field Programmable Logic and Application*, pages 414–423. Springer, 2004.

[24] He Qi, Oluseyi Ayorinde, Yu Huang, and Benton Calhoun. Optimizing energy efficient low-swing interconnect for sub-threshold fpgas. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–4. IEEE, 2015.

[25] Anup Kumar Raghavan and Peter Sutton. Jpg-a partial bitstream generation tool to support partial reconfiguration in virtex fpgas. In *ipdps*, page 0155. IEEE, 2002.

[26] Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B. Kent, Peter Jamieson, and Jason Anderson. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing, booktitle = Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays. FPGA '12, pages 77–86, New York, NY, USA, 2012. ACM.

[27] A. Roy, A. Klinefelter, F.B. Yahya, X. Chen, L.P. Gonzalez-Guerrero, C.J. Lukas, D.A. Kamakshi, J. Boley, K. Craig, M. Faisal, S. Oh, N.E. Roberts, Y. Shakhsheer, A. Shrivastava, D.P. Vasudevan, D.D. Wentzloff, and B.H. Calhoun. A 6.45 uW Self-Powered SoC With Integrated Energy-Harvesting Power Management and ULP Asymmetric Radios for Portable Biomedical Systems. *Biomedical Circuits and Systems, IEEE Transactions on*, 9(6):862–874, Dec 2015.

[28] J.F. Ryan and B.H. Calhoun. A sub-threshold FPGA with low-swing dual-VDD interconnect in 90nm CMOS. In *Custom Integrated Circuits Conference (CICC), 2010 IEEE*, pages 1–4, Sept 2010.

[29] Lattice Semiconductor. iCE40 Ultra Family Data Sheet. `http://www.latticesemi.com/Products/FPGAandCPLD/iCE40.aspx`, 2015.

[30] K. Siozios, G. Koutroumpezis, K. Tatas, D. Soudris, and A. Thanailakis. DAGGER: A Novel Generic Methodology for FPGA Bitstream Generation and Its Software Tool Implementation. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 165b–165b, April 2005.

[31] SourceTech411. Top FPGA Companies for 2013. `http://sourcetech411.com/2013/04/top-fpga-companies-for-2013`, 2013.

[32] Xifan Tang, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Fpga-spice: A simulation-based power estimation framework for fpgas. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 696–703. IEEE, 2015.

[33] T. Tuan, A. Rahman, S. Das, S. Trimberger, and Sean Kao. A 90-nm Low-Power FPGA for Battery-Powered Applications. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):296–300, Feb 2007.

[34] Xiaoqing Wen and Hsin-Po Wang. A flexible logic bist scheme and its application to soc designs. In *Test Symposium, 2001. Proceedings. 10th Asian*, pages 463–, 2001.