# Power Exploration for Functional Units in Coarse-Grained Reconfigurable Arrays

Nathaniel McVicar
Corey Olson
Jimmy Xu
{nmcvicar, cbolson1, jimmyxu}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500


**UWEE Technical Report**
**Number UWEETR-2010-0005**
September 2010

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: http://www.ee.washington.edu

# The University of Washington, Department of EE Technical Report Series

Nathaniel McVicar
Corey Olson
Jimmy Xu
{nmcvicar, cbolson1, jimmyxu}@ee.washington.edu

Dept of EE, University of Washington at Seattle
Seattle WA, 98195-2500

University of Washington, Dept. of EE, UWEETR-2010-0005
September 2010

**Abstract**

Spatially tiled architectures such as CGRAs are robust architectural choices for accelerating applications in the DSP, scientific computing, and embedded domains. In the embedded application domain in particular, CGRAs offer a low power alternative to FPGAs by providing coarse-grained word-level computation resources as opposed to FPGAs' fine-grained bit-level LUTs. This key difference provides a dramatic decrease in terms of power and area, since many of the interconnect logic needed in FPGAs are effectively eliminated by the word-level granularity of the CGRA.

The effectiveness of CGRAs in the embedded application domain creates a market that call for low power computations that extends through the top level architecture to the level of all individual units of the CGRA. Using various low-power design techniques and tools, we implement multiple methods to ultimately reduce the amount of power consumed by the Functional Unit of the CGRA. In the process, we also document the toolsets we utilized to accomplish our goal. We show an average of 4.24x savings in terms of dynamic and static power consumption of the ALU, while still meeting all requirements set by the MOSAIC project.

## 1 Introduction

Current generation FPGAs are essentially a sea of 1-bit compute units called LUTs, each statically programmed to do one function over and over. This single-bit, static structure was a good match to the glue logic and PLD-replacement tasks they were originally designed to handle. However, this extreme reconfigurability comes at a significant cost in area, performance, and especially power.

Moving away from this bit-level design, we approach the word-based functional units found in Coarse-Grained Reconfigurable Arrays (CGRA). CGRAs are a novel architecture that presents an alternative to current generation FPGAs. They are composed of a sea of word-wide functional units, including blocks such as ALUs and multipliers, as well as distributed memories, register files, and pipelining registers. The logic and routing is statically scheduled,

where a cyclic schedule is used to time-multiplex resources. Thus, a functional unit might execute Add, Sub, Xor, Add, Sub, Xor, and so on in a recurring cycle. This allows the device to make tradeoffs between capacity and performance on a per-mapping basis. Currently the research group led by Prof. Scott Hauck and Prof. Carl Ebeling is working on one such CGRA design, dubbed MOSAIC.

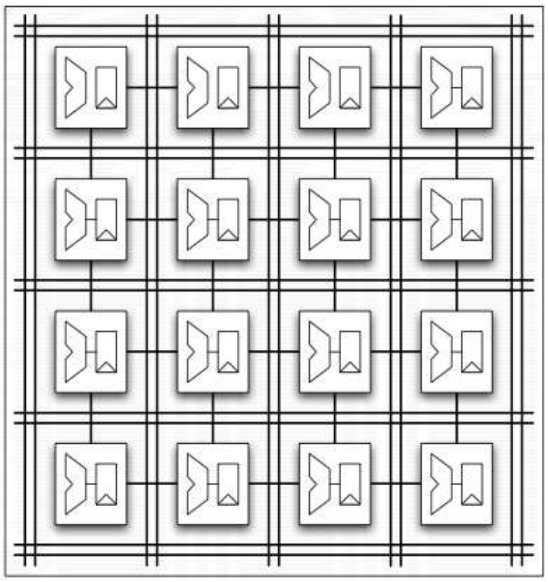

**Figure 1: MOSAIC Hardware Fabric. Each Processing Engine (PE) cluster is complete with a Functional Unit and Crossbar.**

Currently, the MOSAIC functional unit includes a simplified ALU that can perform basic operations, a barrel shifter, a pipelined multiply-add unit, and a data dependent MUX on the inputs of the ALU. The functional unit in MOSAIC is essentially a black box modeled by behavioral Verilog. That behavioral Verilog is synthesized to a model that is functionally correct on IBM's 65nm PDK, but far from being optimized for power and performance, leaving much room for improvement.

## 2 Background

With the limitations of sequential processors being realized, interest in spatial computing is being revitalized in order to deliver performance that follow with the silicon density increases dictated by Moore's law. By using a large number of simple, concurrent, parallel processing elements (PEs) to execute a single application kernel, spatial processors can effectively divide and conquer certain computationally intensive applications. Examples of current spatial architectures include dedicated hardware circuitry known as Application Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA), Graphics Processing Unit (GPU), and Massively Parallel Processing Array (MPPA). To a certain extent, modern multi-core architectures can also be described as sharing traits of spatial computers.

As seen in commercially available ASICs, FPGAs and GPUs, spatial computing has proven time and time again to offer great power and performance advantages over traditional sequential processors [1]. However, the limitations of these architectures has introduced a niche in the spectrum of spatial processors and given rise to the Coarse Grain Reconfigurable Architecture (CGRA).

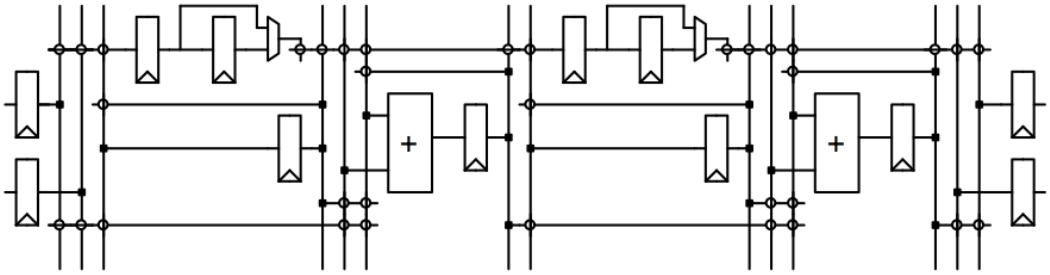


**Figure 2: Pipelined CGRA [2]**

Typically, ASICS provide the best performance and power characteristics of all the architectures mentioned, but have an extremely high cost and require significant design effort. FPGAs on the other hand provide a cheap alternative, but have comparably poor area, power and performance due to the overhead cost of flexibility. The niche filled by CGRAs call for a processor to have the power and performance advantages of an ASIC, in addition to the flexibility of an FPGA. CGRAs meet these criterion by focusing on datapath computation rather than general purpose computation [3]. This allows specialty hardware that only targets the computationally intensive inner-loop of an application kernel.

The hardware fabric of a CGRA is composed of islands of processing elements (PEs), each consisting of a Functional Unit (FU), and a switchbox. Crossbars are used to connect the PEs together, routing data between them. Viewing the CGRA as an evolution on the FPGA architecture, results from FPGA studies indicate that the most power intensive portion of the hardware lies in the routing fabric and logic blocks, with comparably lower percentages being dissipated in the clock network [4].

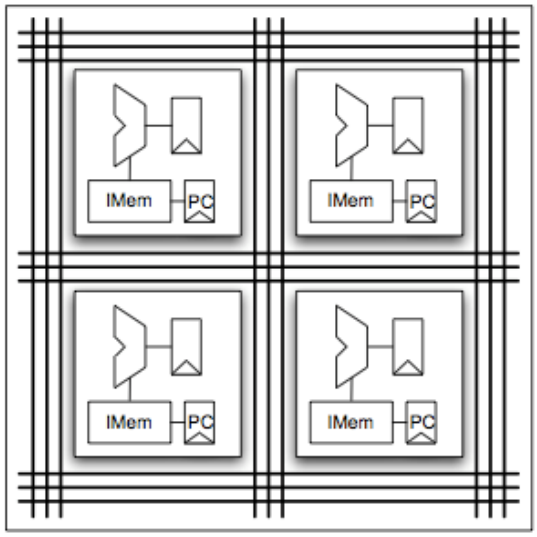


**Figure 3: Island style PEs of CGRAs [2]**

Looking at the Functional Unit of the MOSAIC CGRA, we see that it must support a several categories of instructions, including: Arithmetic , Comparison, Bitwise, Compare A with 0, Compare A with 1, Arithmetic with Immediate, Select with Immediate, and LUT Operations. These instructions are implemented using three separate hardware blocks: Arithmetic and Logical Unit (ALU), Barrel Shifter (BS), and Multiply-Add (MADD) unit. Therefore, to achieve the goal of low power for the Functional Unit and the CGRA, each of these hardware blocks must be individually optimized.

# 3 Behavioral Benchmark

## 3.1 ALU

Prior to commencing our architectural study and implementing the necessary low power techniques, we examined the behavioral benchmarks set in place for the functional unit of the MOSAIC project. These benchmarks were recorded from simple behavioral models, and directly reflect the inadequacies of the synthesis tools.

To better model the power behavior of the FU, we employed two separate methods of measuring power: a probabilistic model that specifies the odds for a given input to switch between a 1 and 0 and a simulation using a given set of input vectors recorded from an actual CGRA application.

For the ALU, the results of these simulations are given below:

**Table 1: Behavioral Benchmark Power Profile of the ALU**

| Type | Probabilistic | Input Vector |
|---|---|---|
| Net Switching Power | 2.18E-03 | 7.64E-04 |
| Cell Internal Power | 2.73E-03 | 9.94E-04 |
| Cell Leakage Power | 5.34E-07 | 5.02E-07 |
| Total Power | 4.90E-03 | 1.76E-03 |

As evident in the results, the primary areas of focus are the switching power consumption and cell internal power consumption, which consume roughly 45 and 55 percent respectively. In comparison, the cell leakage power consumes less than 1% of the total power, and should therefore only be a secondary concern for this study. Examinations into the other units revealed a similar result.

## 3.2 Shifter and MADD

The MOSIAC project did not have any up-to-date behavioral models for the MADD or Barrel Shifter.

# 4 Architectural Design

## 4.1 Functional Unit Top

The top level of the Functional Unit contains the ALU, BS, and MADD units. Initially, we planned for this top level to be able to control power and clock gating, and thought it would serve as a master of sorts, gating off the individual sub-modules when they're not in use.

Our initial attempt to implement this top down power control strategy was met with much success. We were able to fully design and verify a top level controller to meet our desired objectives, and initial investigations into clock gating suggested that this approach would provide huge benefits to overall power consumption.

However, our efforts were curbed when discussions with other members of the MOSAIC team revealed that our expected deliverables did not include a top level unit, and the only way to implement our design into the overall MOSAIC architecture is for us to supply the individual modules as separate entities. This forced us to rethink our design approach and brought us back to the drawing board.

Efforts like clock gating would not have significant value for the ALU, BS and MADD because MOSAIC already employees a similar power saving scheme. Registers in the MOSAIC fabric are used to hold the inputs to the units constant when they are not in use computing new values. This eliminates much of the same extraneous switching power that clock or input gating would.

## 4.2 ALU

The functional unit for the CGRA is a 32-bit system, so a 32-bit ALU is required. This ALU must support 15 arithmetic and logic operations including add, subtract, and, or, xor, invert A, invert B, negative of A, negative of B, increment A, decrement A, absolute value of A, pass A, output 0 and output 1. For the schedule, place, and route (SPR) tools to work correctly, the ALU must be a purely combinational design. In other words there must be no registers in the ALU and therefore a latency of 0 clock cycles. The ALU is also along the critical path of the system, so it's important to operate the ALU as fast as possible. However, due to the number of ALUs that are implemented on the CGRA, it is extremely important to reduce the power consumption as much as possible. With both of these goals in mind, the target frequency for the ALU was set to 1GHz.

Two major techniques can be used when designing the architecture for the ALU. Both techniques must use the 32-bit adder for Add and Subtract operations, but they differ in their implementation of the other functions, such as the XOR, OR, and Invert functions.

The first technique utilizes resource sharing as much as possible. It sets up the inputs to the adder in order to use the adder to accomplish all operations except for the OR operation. Propagate and generate bits must be added as outputs from the adder, but they are already computed during the addition, and they are essentially free outputs. The inputs must have the ability to be able to be set, cleared, or flipped, depending on the operation. An output multiplexer then selects between the output of the adder, propagate, generate, and the external OR gate. This architecture can be seen in Figure 4.

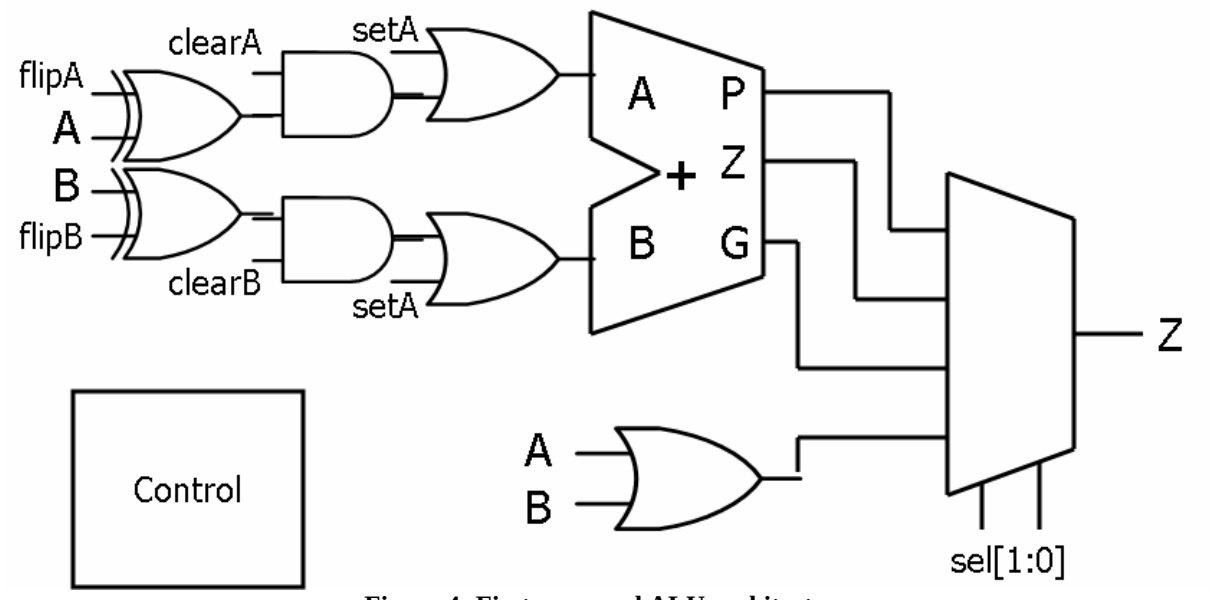


**Figure 4: First proposed ALU architecture**

The second architecture saves power by limiting the resources required to perform an individual operation by adding logic gates to perform the operations that do not require the adder. This technique reduces the amount of switching power by gating the inputs to all the computational blocks except the one that is required for the current operation. An output multiplexer then passes the output of the active block to the output of the ALU. By holding the inputs at the previous value, no switching power is consumed, and the only power burned by the logic blocks that are not in use is leakage power. This architecture requires the same amount of decode logic as the first but must also include latches to hold the inputs at their previous value, thereby increasing the number of cells in the design.

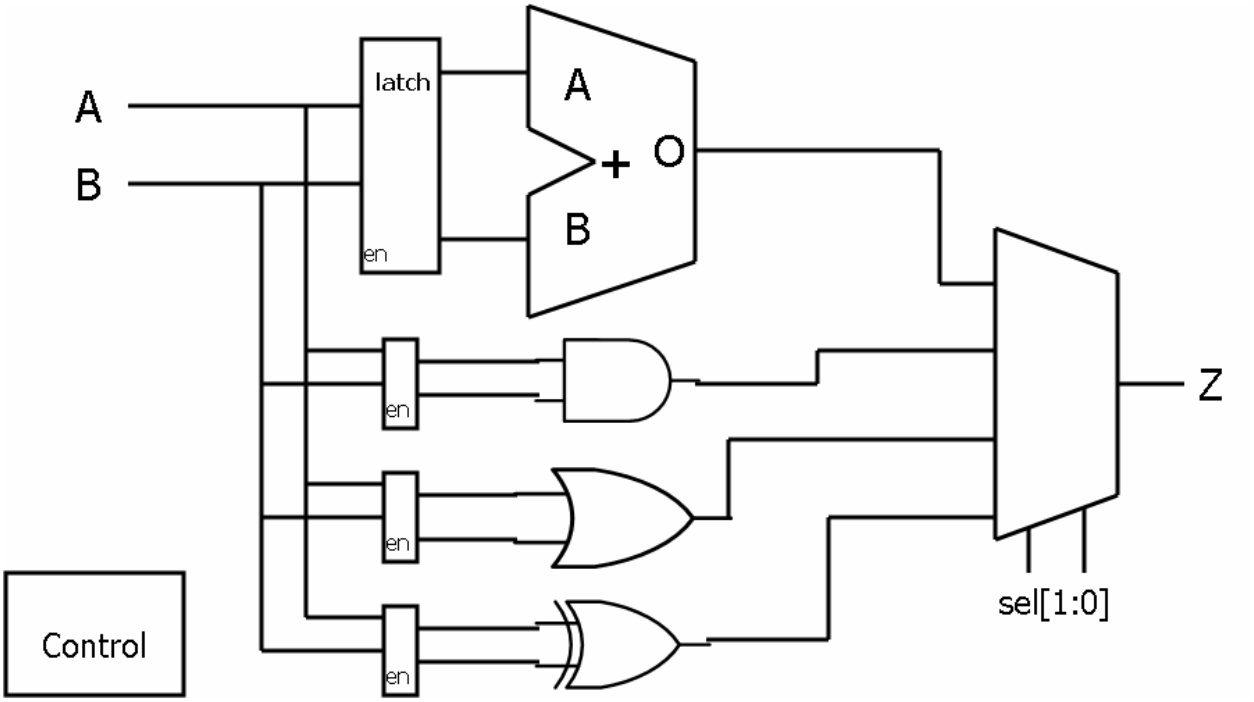


**Figure 5:** Second proposed ALU architecture.

These two ALU architectures were implemented and their power consumption was compared using PrimeTime. The results can be seen in Table 2. The table shows the first ALU design has lower switching, interconnect, and

leakage power. Also, the cell report from design compiler showed the first ALU architecture contained fewer cells than the second design, resulting in the lower leakage measured by PrimeTime. For these two reasons, the first architecture, which performs almost all computations with the adder, was implemented for this project.

**Table 2: Power comparison of 2 ALU architectures.**

| Power Measurement | ALU 1 | ALU 2 |
|---|---|---|
| Net Switching Power | 630 uW | 655 uW |
| Cell Internal Power | 1.14 mW | 1.21 mW |
| Cell Leakage Power | 135 nW | 160 nW |
| Total Power | 1.77 mW | 1.86 mW |

## 4.3 BS

The functional unit requires a barrel shifter that is able to handle bi-directional arithmetic and logical shifting. Recall that the difference between the two shift types is only prevalent while shifting to the right. Arithmetic shifts preserve the sign bit whereas logical shifts shift in zeroes. Similar to the ALU, the barrel shifter is required to be a purely combinational design, due to limitations of the schedule, place, and route tools. Also similar to the ALU, this shifter should be able to run as fast as possible, but because we want a power efficient design, we must limit the maximum operating frequency. Therefore, this barrel shifter will also be designed to operate at a maximum frequency of 1GHz. Because of these requirements, two different architectural approaches can be taken when implementing a barrel shifter. The first approach is an array shifter and the second is a logarithmic shifter.

The array shifter solves this problem by using a large multiplexer for each output bit. Those multiplexers must use all input bits as the select signals. The advantage to this design is the speed at which you can run this barrel shifter. The longest delay through this implementation is only the delay through on large multiplexer. However, the multiplexers consume more area and burn more power than necessary for this problem.

The second architecture is a logarithmic barrel shifter, and it is shown in Figure 6. The log shifter uses a number of stages that is equivalent to the log of the number of output bits, hence the name. At each stage, a small multiplexer is used to select the bits to pass to the next stage, and the input signals are used as the select bits. The first stage sets up the inputs to the barrel shifter depending on the left and logical bits. In order to reuse logic as much as possible, the left shift is converted into an overshift to the right. To do this, the inputs are set up with the input value (X) in the leftmost 32 bits, and the shift bits that are passed to the multiplexers are the 2's complement of the inputted shift bits. For shift left, the first stage gets {X[30:0],31'b0}, arithmetic shift right gets {31{X[31]},X[31:0]}, and logical shift right results in {31'b0,X[31:0]}. The subsequent stages can then be thought of as adjusting the output window to a 32 bit range along those original input bits. This architecture trades speed for lower power and a smaller area than the previous design. Therefore, as long as this design can operate at a fast enough frequency, in this case the frequency is 1GHz, this design will be much more power and area efficient than the first architecture. For this reason, the logarithmic shifter was chosen to be the barrel shifter for the functional unit in this project.

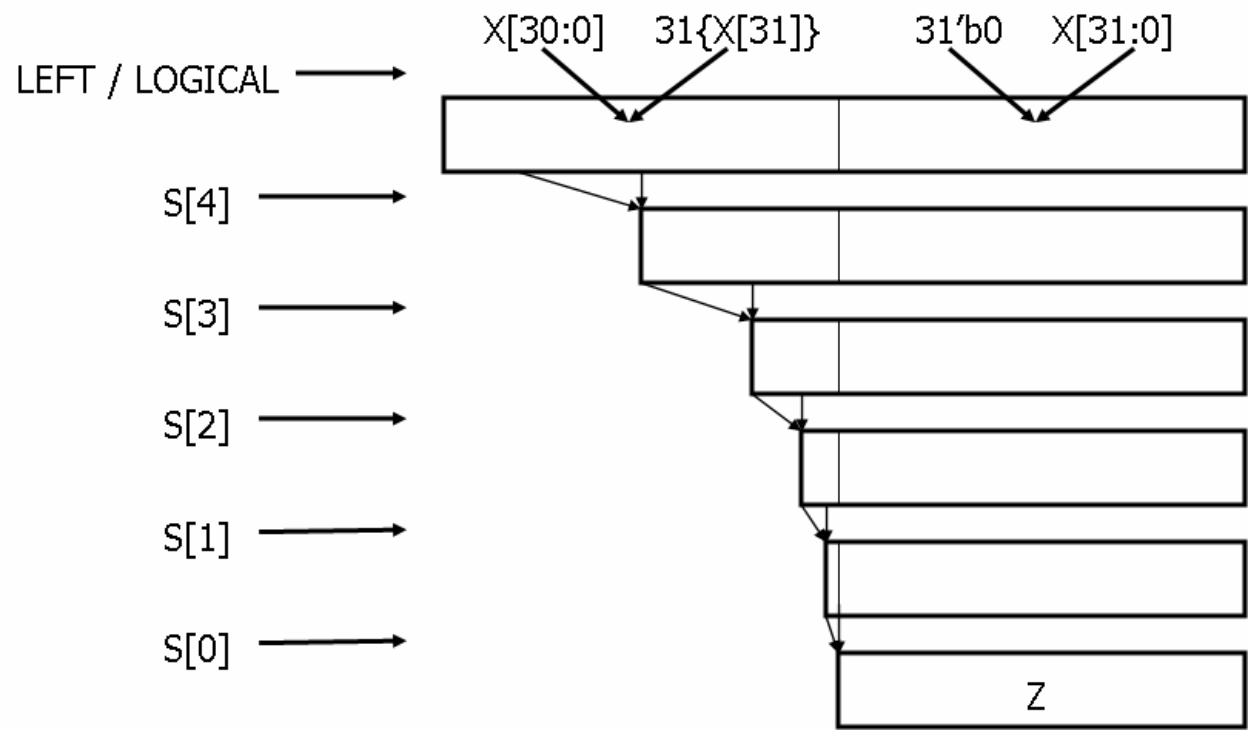


**Figure 6: Logarithmic bi-directional arithmetic and logical shifter. Arrows converging to a point represent multiplexer and the signals on the left are the selects for those multiplexers. X[31:0] is the input to this barrel shifter.**

## 4.4 MADD

The functional unit's ALU only performs logical operations and arithmetic operations based on addition and subtraction. However, many arithmetic heavy applications that are well suited for CGRAs require a multiply operation. The MOSIAC architecture makes use of a multiply-add unit or MADD because this operation is very useful for a number of applications including matrix-multiplication and filters. Additionally, a multiply-add can be implemented at a relatively low additional hardware cost, when compared to a multiply.

An additional design constraint on the MADD, is that a multiplication operation was deemed too complex to perform in a single cycle at MOSAIC 2 speeds. For this reason, the MOSIAC design specifies that the multiplier is a 2-cycle fully pipelined operation. The add input arrives on the second pipeline cycle. The MADD was required to run at 1 GHz, and performs the following function (subscripts are cycle):

$$\mathbf{Z_1 = (A_0 * B_0) + C_1}$$

In light of these constraints, the focus of the design for the MADD was high speed and low power. After extensive research, we determined that the best approach was a modified booth multiplier. This approach is very common, and for the 32-bit multiplier we selected radix-2, as higher radix multipliers use slightly less area and power but are significantly slower [5].

The only unusual aspect about the booth encoding selected is a heterogeneous encoding, where the final partial product is generated using the inverse of the multiplicand. This process is very fast because the inverse can be generated in parallel with the booth encoding, using fast 2's compliment logic. For this to work, the booth encoding of the final three bits of the multiplier is different than the encoding of the other bits. Four encoded signals are generated, instead of the usual three. These are X, 2X, -X and -2X. Then the partial product is generated with a simple multiplexer based on these signals. In cases where none are true, the output should be zero. These four signals are generated as follows:

- **$X = (X_{-1}$ XNOR $X) NOR X_{+1}$**
- **$2X = (X_{-1}$ NAND $X) NOR X_{+1}$**
- **$-X = (X_{-1}$ XNOR $X) NOR \sim X_{+1}$**
- **$-2X = (X_{-1}$ OR $X) NOR \sim X_{+1}$**

This technique replaces the final sign extension bit required by the standard booth encoded multiplier [6] [7]. This is particularly important for a 32-bit multiplier that truncates all of the higher result bits, such as this one, because the last sign extension bit would normally be an input at the 31st bit position, which would require an extra compressor input at the part of the tree where it is most costly.

Other than the partial product generation, the multiplier used in the MADD is standard. The compression of the partial products is done using a Carry-Save Adder (CSA) tree. The CSA tree is very efficient for this standard cell library because its base element is a full adder, for which a cell is available. While more complex compressors, such as (5,5,4) and (7,3), have some performance advantages in a fully custom solution, the fact that they would have to be made out of standard cells eliminate any advantage they would have over a CSA tree.

The input to be added, the C input in the equation above, is introduced in the CSA stage. Because the CSA is a (3,2) compressor, there are some unused inputs, so the C input has an almost zero cost. The only wrinkle is that the input must be introduced after the pipeline registers to produce the desired output. This is not a major issue as the registers are inserted after the first two CSA stages to balance the MADD and achieve the best timing.

For the final adder, a behavioral adder is used in the Verilog. The synthesis tool is sufficiently sophisticated that this produced essentially identical results to those achieved when we tried a Brent-Kung and Kogge-Stone adder. We choose the behavioral adder for simplicity. The final architecture for the MADD can be seen in Figure 7.

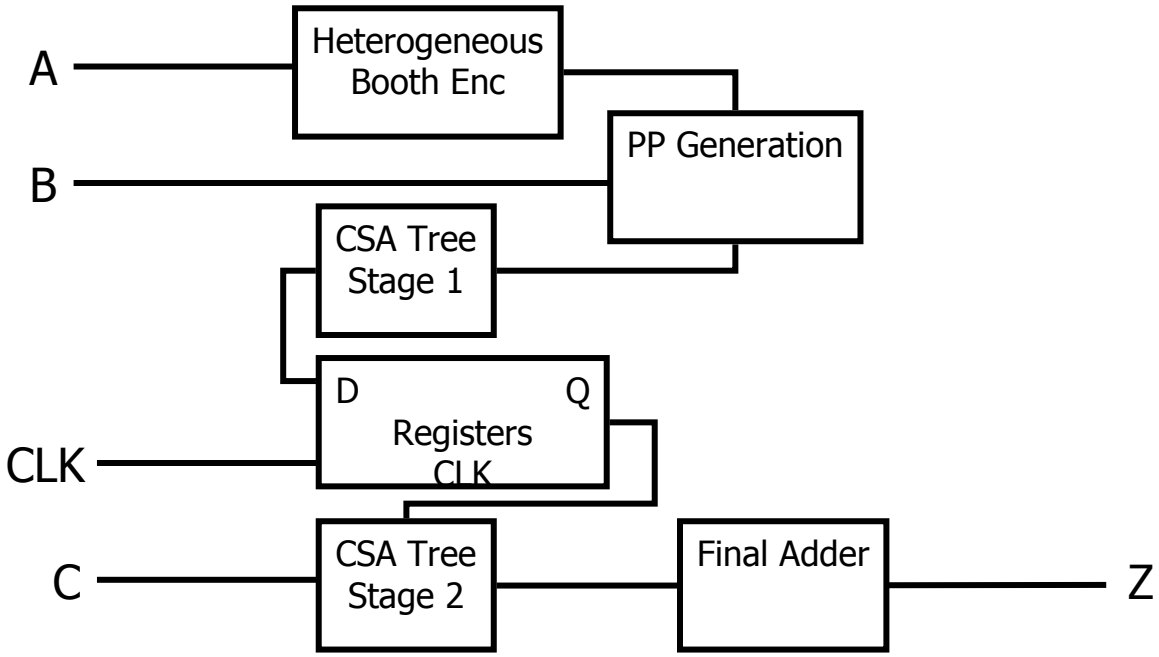


**Figure 7: MADD architecture block diagram**

# 5 Tools

## 5.1 VCS

With any design, the most important step of the design flow is verification. No matter how novel a design may be, it is useless if it doesn't perform as expected. The first step of our verification flow was digital simulation using VCS, which is a Verilog simulator. VCS compiles our verilog design files to a simulation executable, which is then run on a linux machine. Digital verification can take on three methodologies: exhaustive, directed, or random testing. For this project, we obviously didn't have enough time to exhaustively test any of our 32-bit modules, so we combined the latter two methodologies. We exhaustively simulated the control inputs to our designs, such as the shift amount for the barrel shifter or opcode for the ALU, with random data. In other words, we ensured that all control signals function as expected, as long as there are no strange cases that are data dependent. To minimize the probability of missing a data dependent error, we ran multiple iterations involving different data for each control input.

## 5.2 Design Compiler

We used Synposys' Design Compile synthesis tool to convert our Verilog into a structural netlist using the standard cells of the IBM 10LPE PDK. Design Compiler is a widely used synthesis tool in industry, with many separate components that can each be invoked at many stages of the synthesis process to improve the quality of results.

The Design Compiler license available to us on the ACME lab machines included the following components: DC Professional, DC Expert, DC Ultra, FloorPlan Manager, HDL Compiler, VHDL Compiler, Library Compiler, DesignWare Developer, DFT Compiler, BSD Compiler and Power Compiler.

Of these tools, we primarily made use of DC Professional and DC Ultra to perform synthesis. In addition, we compiled our Verilog code using the Presto Verilog compiler found in HDL Compiler and analyzed our code for relevant IP blocks by DesignWare, although none were found. Finally, we used Power Compiler to perform power aware synthesis which, in conjunction with power constraints, leads to much lower power results. This will be discussed in much more detail below.

Our Design Compiler runs were done using a tcl script. Although these scripts are fairly straight forward, they all follow the same general steps, using these important commands:

- analyze – Converts the Verilog files to an intermediate format
- elaborate – Converts this intermediate format to a DC design
- uniquify – Creates a unique cell for each instance for flattening
- set_flatten – Allows optimizations to cross cell boundaries
- compile – Synthesizes the design
- check_design – Verifies the correctness of the design

## 5.3 PrimeTime

To verify the timing constraints and power consumption of our synthesized designs, we used PrimeTime from Synopsys. PrimeTime reads in the synthesized netlist and allows us to verify the critical path delay using the timing annotated standard cell library. It then analyzes the dynamic and static power consumption of the circuit by using the power annotated standard cell library. You can selectively output the timing and power reports. The timing report displays the critical path and its associated delay and reports amount of slack in the design. The power report breaks the power consumption of the design into dynamic and static power consumption. The dynamic power can also be reported in terms of internal switching power and interconnect switching power. Finally, the power report can also break down the consumption in terms of memory, clock network, register, and combinational cells. This allows the designer to understand what areas of the design should be targeted for a reduction in power. For a more efficient design flow, we wrote tcl scripts to easily interact with PrimeTime.

## 5.4 V2LVS

v2lvs is a Mentor Graphics tool designed to create SPICE-like netlists from Verilog files. This tool is explicitly intended to be used for running LVS against a synthesized design that has been run through a place and route tool such as Encounter. However, v2lvs also provides a –i command line option. This creates an HSPICE compatible output, instead of the standard Calibre LVS only format, which uses $PINS.

This option allowed us to use v2lvs to generate HSPICE netlists from our synthesized Verilog netlist. Using these netlists, in conjunction with an HSPICE testbench, we were able to run HSPICE / Verilog-A simulations with techniques such as power gating and VDD scaling that we were unable to test using Prime Time or Verilog simulation.

The process of running v2lvs is very straight forward, as it simply converts the Verilog netlist into HSPICE. For this project, only a few non-standard options were utilized. In addition to the previously mentioned –i command, we also specified the names of the power and ground signals, the include file for the transistor models from the PDK and HSPICE netlists for each standard cell in the PDK. We had to compile this final item from the PDK's database file, as it was not provided.

## 5.5 Encounter

Cadence SOC's First Encounter is a tool that allows the designer to operate on a digital design in many ways. For this project, Encounter gives the ability to import the synthesized verilog netlist, attach power and ground nets to all the standard cells in the design, place and route the synthesized netlist, extract parasitic capacitances of the routed design, and stream out the design in a format that can be imported into Cadence design tools for further modifications, such as adding power gating transistors. The real goal of using encounter though is simply to get a schematic netlist that can be imported into Cadence tools for SPICE simulation. We were able to successfully import and place and route all designs into Encounter, as can be seen in the schematic and layout images of the ALU in Figure 8 and Figure 9.

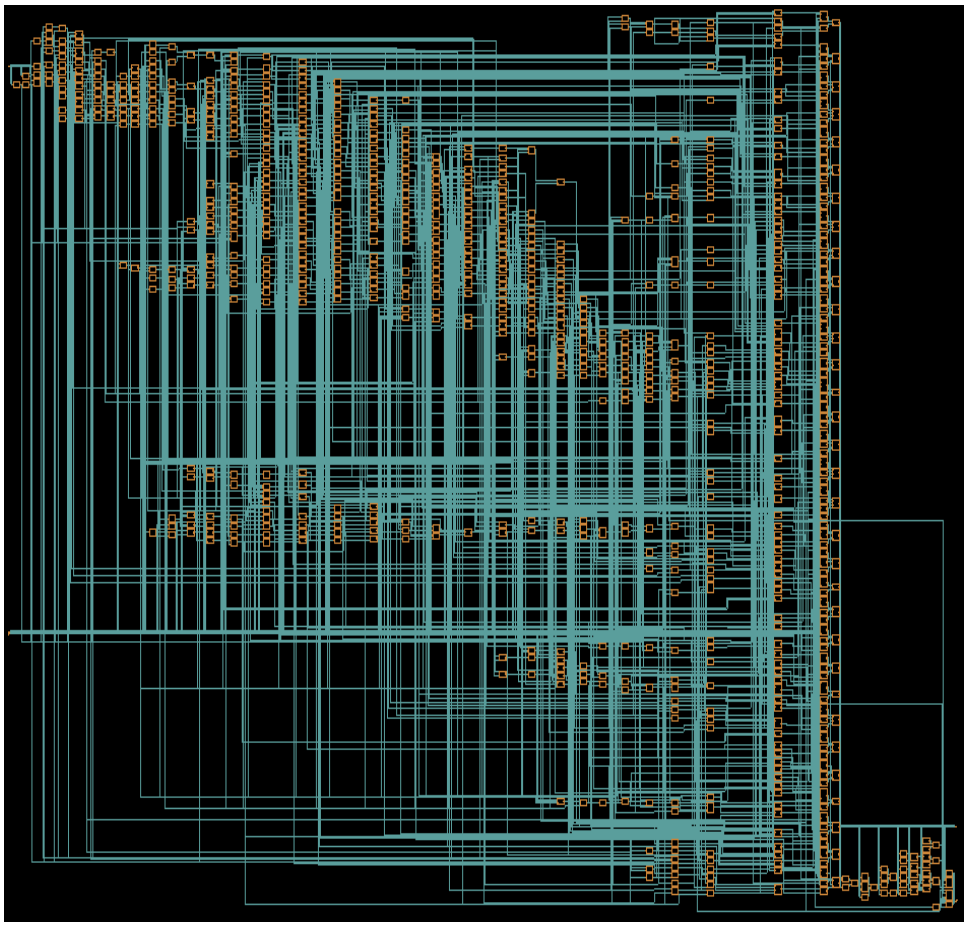

**Figure 8: Schematic view of the ALU using Encounter.**

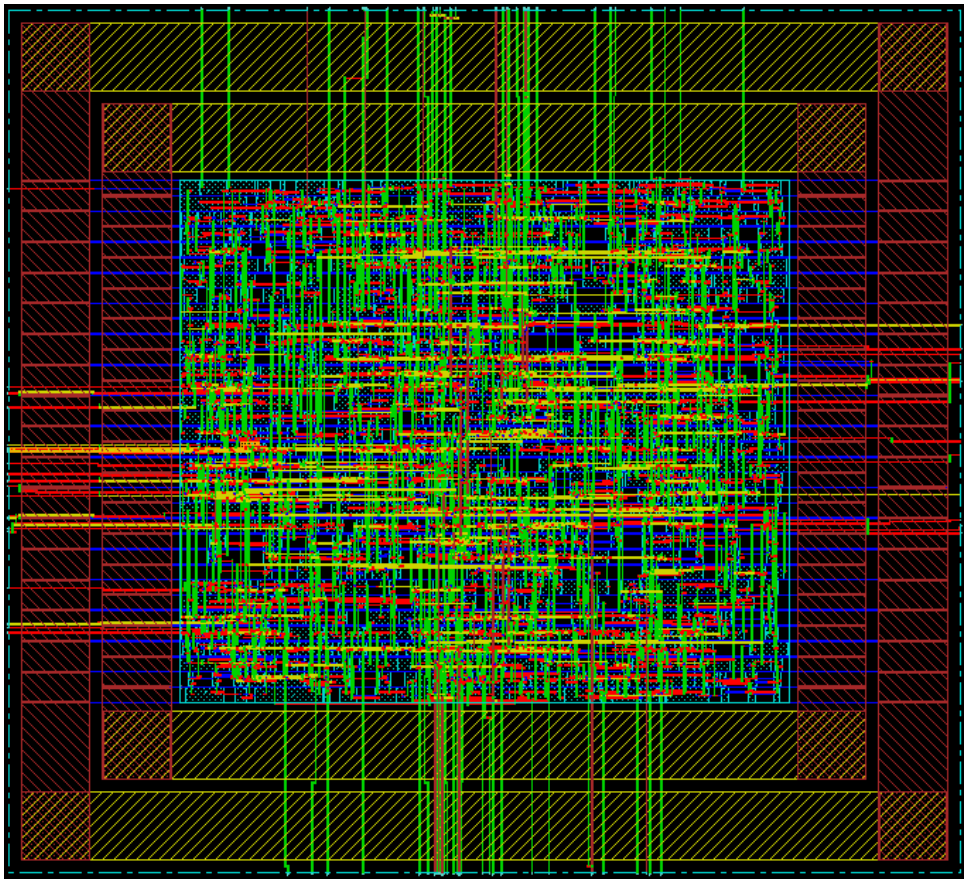

**Figure 9: Layout view of the placed and routed ALU using Encounter.**

While using Encounter, we found two main difficulties that seemed to render the tool useless. First, we were disappointed to learn that Encounter does not have the ability to save any kind of netlist before place and route. Second, after many hours and a large amount of advice from the EE cad TA at the University of Washington, we were unsuccessful at extracting the parasitic capacitances of the routed design, due to the inability to find the necessary capacitance tables for the cmos10lpe PDK. We were however able to stream out the routed design into a gds format, for later import into Cadence design tools, where we could hopefully still extract the parasitic capacitance values.

## 5.6 Cadence

With a working design imported into cadence, we were hoping to be able to export a SPICE netlist to a file, in order to add in the power gating transistors, which cannot be specified in the verilog language alone. Cadence custom design tools allow the designer two options for streaming in a design that applied to our project: gds streamin and importing a verilog netlist. First, we were able to stream in the gds file of the routed design from Encounter. The layout of the shifter can be seen in Figure 10. However, we were unable to successfully stream in the standard cells from the PDK in order to have a complete design that could be run through LVS and extracted.

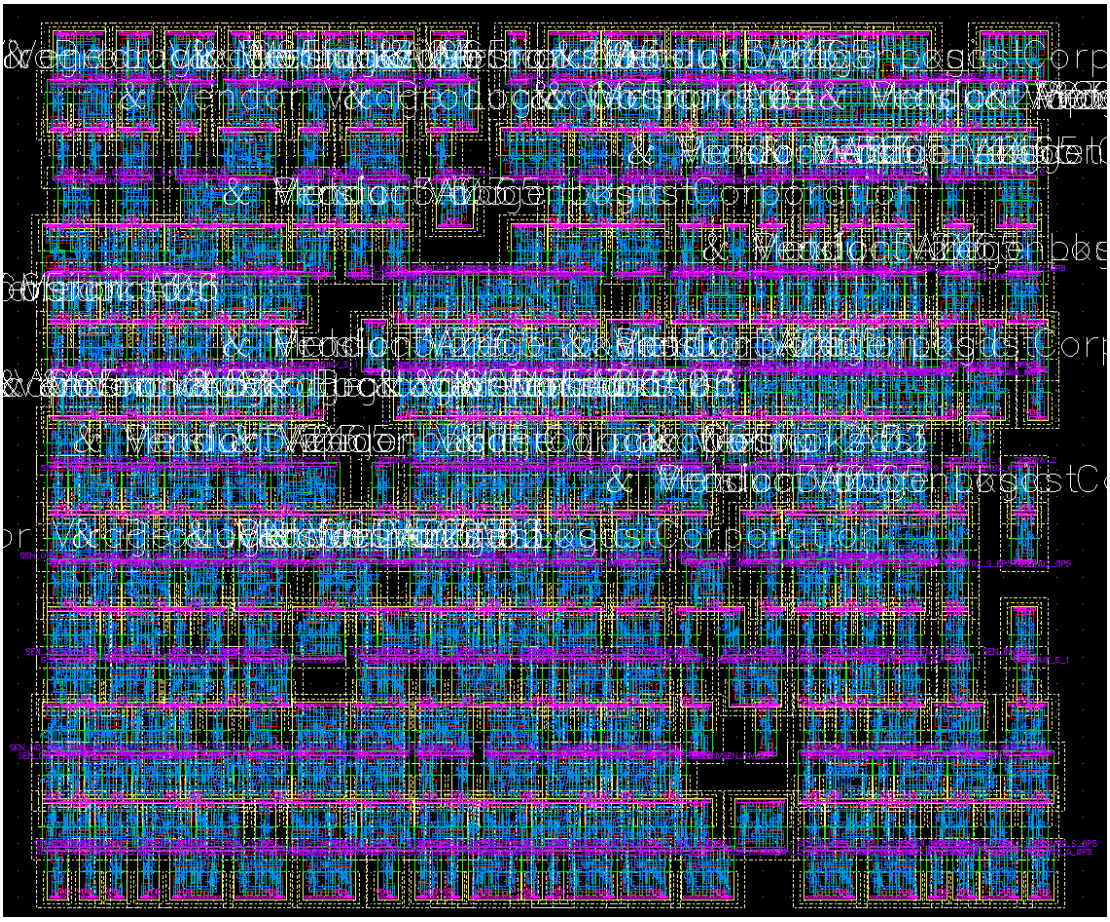

**Figure 10: Layout view of the shifter using Cadence design suite.**

Second, we tried streaming in the original verilog netlist of the design and the standard cells from the PDK. We were able to stream in the schematic views of all the necessary files, but unfortunately were unable to export a SPICE netlist due to an unknown error that could not be resolved by our group or by the EE cad TA. After many hours of debugging to no avail, we abandoned the idea of using both Encounter and Cadence to get the SPICE netlist we required, due to insufficient tools support and inability to identify the necessary files from the IBM PDK.

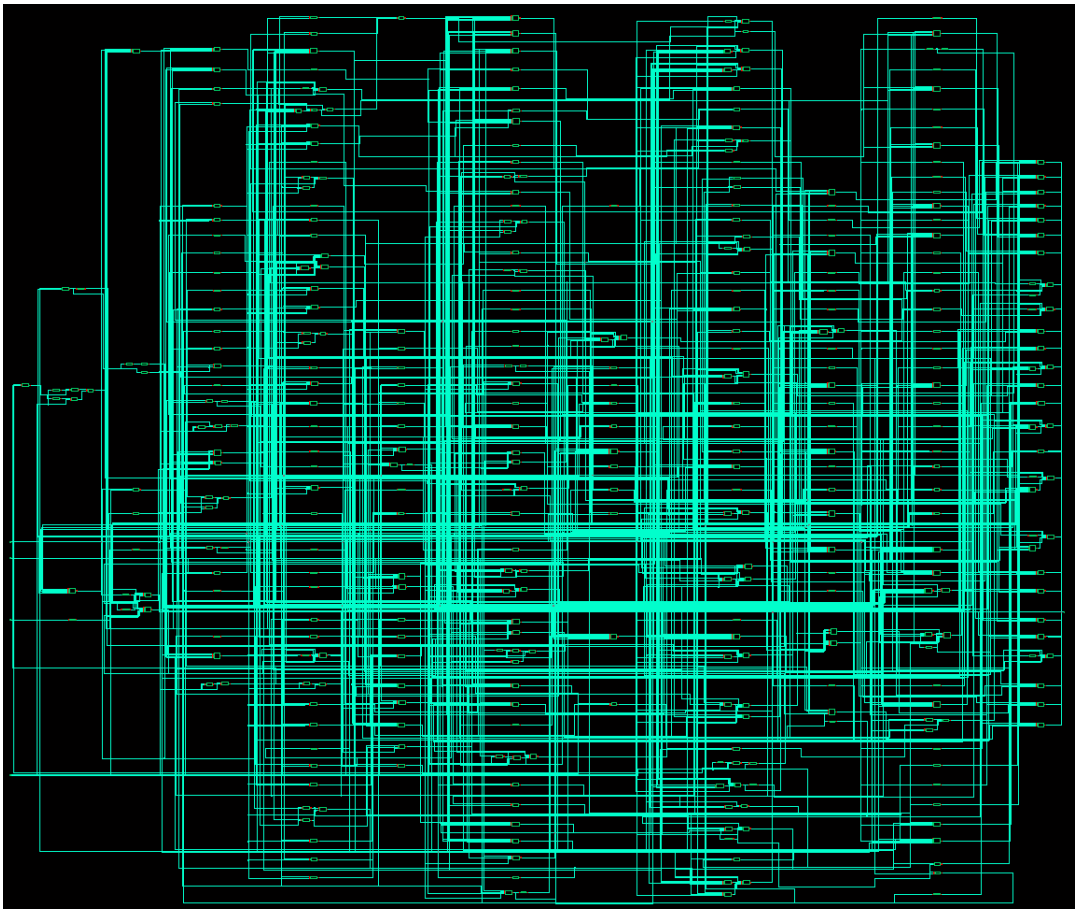

**Figure 11: Schematic view of the shifter using Cadence design suite.**

## 5.7 UPF

The Unified Power Format exists as the industry standard for specifying power intent in low-power design and verification. It is a standard, proposed by Accelera and accepted by the IEEE, which allows the designer to implement low-power techniques such as Multi-voltage domains, power gating, and dynamic voltage scaling.

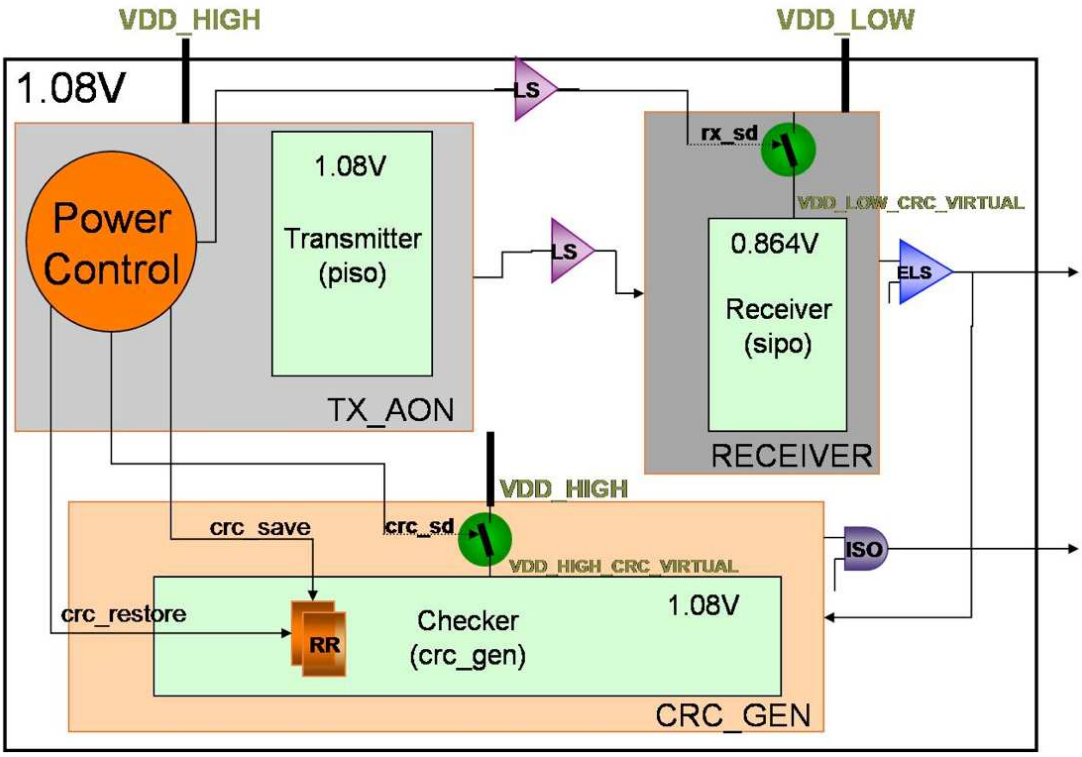


**Figure 12: Example design using UPF [8]**

UPF is designed to be able to be integrated every step of the DV process. Our efforts in particular were focused on integrating it into post-synthesis simulation using PrimeTime. The primary goal of our efforts were to first implement dynamic power gating, then extend our scripts to incorporate dynamic voltage scaling.

While some success was had with creating the power intent using UPF, the lack of documentation and direction prevented us from fully integrating it with PrimeTime. We were able to get our Tcl script to run successfully with the PrimeTime sim, but we weren't able to see any effect on the resulting power from our attempts at power gating and voltage scaling. As a result, we looked into alternative ways to achieve these goals, and started looking into using Encounter and v2lvs.

# 6 Low Power Optimizations

## 6.1 Power Gating

All power consumed in a design is supplied by the VDD net, so one method for greatly decreasing the power consumption in a design is to detach a block from the power supply. If you are able to partition a design into elements that work independently during operation, you may power down those elements that are not needed. One benefit of our functional unit is that it instantiates three modules that work independently based on the operation specified by the instruction. Therefore, we created three voltage islands, with the ALU, barrel shifter, and multiply add units each being independent islands. By creating voltage islands, we can power down an island when that section of the circuit is not needed for computation.

One of the major benefits of the architecture that we are targeting with this project is the knowledge of resource usage at compile time. We can store a disable bit in the instruction memory for each of the modules in the functional unit, instead of having to decode which module requires power based on the instruction. Also, since the SPR tools know the schedule at compile time, they can enable a block for power-up ahead of time if it requires multiple clock cycles to become stable for computation.

This method of saving power is quite risky if timing issues arise, and can also add a lot of overhead in terms of area. With these in mind, we added a pfet to our SPICE netlist that drove the VDD supply for an entire module and was controlled by a power-down signal. We used SPICE simulations to investigate the power-up time for the ALU, and also wanted to determine the size of the pfet required to provide a near instantaneous power-up time.

## 6.2 Voltage Scaling

Voltage scaling is a complex issue. As described in Chandrakasan, Sheng and Broderson's seminal low power design paper [9], lowering the supply voltage for a design will provide power savings at the cost of significantly slower operation. For this reason, we felt that it was important to explore the operation of the ALU, Shifter and MADD at various operating voltages. This is particularly important because not all implementations of the MOSAIC architecture are expected to operate at the same frequency, and the 1 GHz requirement for these modules is more of a soft specification. In addition, as described in the power gating section above, it is conceivable that the three components could be run at different supply voltages for some implementations.

Logic synthesis adds another element to the equation. The design can be synthesized target different frequencies. A design that is synthesized for a high frequency, such as 1 GHz, would be expected to have a low delay. However, the synthesis tool will have to relay on replication to reduce fanout and other techniques that may cause the design to use more power reach the target frequency. Similarly, a design target a low frequency, such as 100 MHz, may use less cells and therefore have a lower power requirement at the same voltage, although it would also be slower. Voltage scaling results are available in the Results section.

## 6.3 Clock Gating

The practice of clock gating is a common low-power design technique used by many synchronous circuits. It essentially reduces switching power by shutting down the clock to the portions of the system that are not in use. Since the clocks drive the state-holding flip-flops in our design, an absence of clock movement will directly results in 0 switching activity in that portion of the system.

In the Unified version of the ALU, both clock and input gating was implemented by an AND function of a set of control flip-flops and the clock signal. The output of this AND function then serves as the clock to a particular clock domain. Each clock domain would have its own set of control flip-flops, thus rendering the operation of different clock domains to be independent, allowing the system design to shut them off individually.

Furthermore, our original design methodology aimed to have the ability to clock gate each of the hardware modules (ALU, BS, MADD) at the top level of the FU. This approach was later deemed unsuitable for implementation in the scope of the MOSAIC project.

## 6.4 Input Gating

Similar to clock gating, input gating is the practice of holding the inputs at the previous value to reduce the amount of transitions seen throughout the system. As the switching power was measured to be roughly 45% of all power consumed by our circuit, we realized that this area required significant focus.

# 7 Final Results

## 7.1 Power

The final results of our low power improvements are shown in the following tables. We present the power results collected from PrimeTime of the finalized architectures with the low-power techniques discussed implemented. For comparison, we also present the behavioral model's measurements using the same measuring techniques. Finally, we present and discuss the improvement we were able to achieve with our low-power techniques.

**Table 3: ALU Power Results**

| | Unified Design | Partitioned Design | Behavioral Model | Improvement |
|---|---|---|---|---|
| Switching Power | 630 uW | 655 uW | 3.55 mW | 5.63x |
| Interconnect Power | 1.14 mW | 1.21 mW | 3.94 mW | 3.46x |
| Leakage Power | 135 nW | 160 nW | 530 nW | 3.93x |
| Total | 1.77 mW | 1.87 mW | 7.5 mW | 4.24x |

For the ALU, we were able to measure the results from both the unified and partitioned design approaches. While each favored different low-power design techniques, the results ultimately show that both approaches produced huge improvements over the behavioral model. With an approximate 4.24x improvement overall, the low-power techniques we implemented definitely produced a noticeable return on reducing power consumption.

Along with the ALU, the final power results for the optimized Barrel Shifter and MADD units are shown in the table below. As we don't have behavioral models readily available for these units, the behavioral model comparison has been left out, although we suspect the improvement should be in line with what we saw from the ALU.

**Table 4: Barrel Shifter and MADD Power Results**

| | Barrel Shifter | MADD |
|---|---|---|
| Switching Power | 538 uW | 3.00 mW |
| Interconnect Power | 439 uW | 4.68 mW |
| Leakage Power | 48 nW | 614 nW |
| Total | 977 uW | 7.68 mW |

## 7.2 Power-up time

To investigate the power-up time of a module, we began with a W=600nm sized PMOS transistor driving the VDD supply for the ALU. The results of that simulation can be seen in Figure 13. The light blue line (VDD) drops during the green line's (PD) high phase, and is unable to recover once PD is de-asserted. This shows that a pfet of

this size is only able to drive the VDD of the ALU to approximately 0.5V, even at DC. It simply is not able to source enough current during the operation of the ALU.

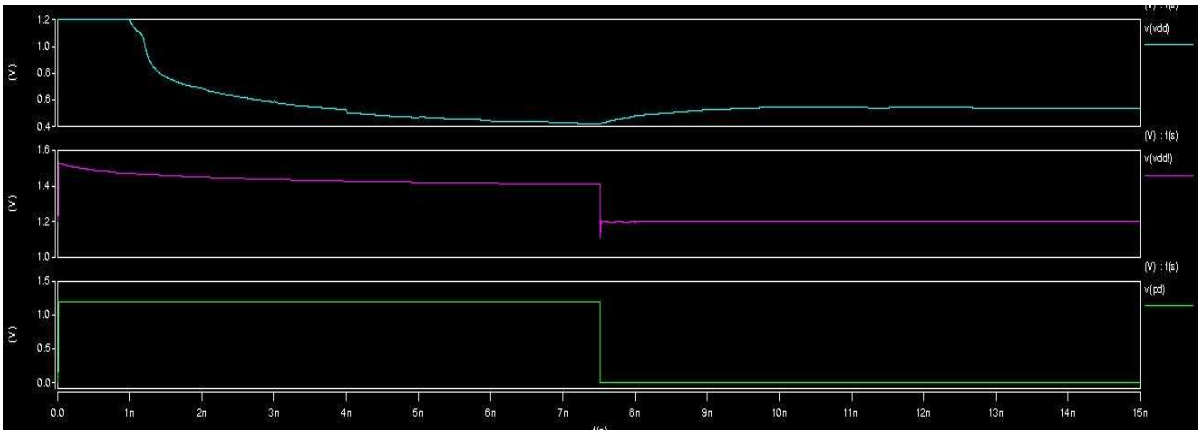


**Figure 13: Power-up and down profile for 0.6 um of power gating transistors. VDD is shown in light blue, and the active-high power-down signal is shown in green.**

The width of the pfet driving VDD was then increased in size to 7.2um and the same simulation was run. The results of the simulation can be seen in Figure 14. Once again, the light blue line is VDD and the green line is the active high power-down signal. This simulation shows the VDD line is very unstable during the time when the ALU is active. This is due to the inability of this size pfet to source enough current to run the ALU. Therefore, the pfet must be further increased.

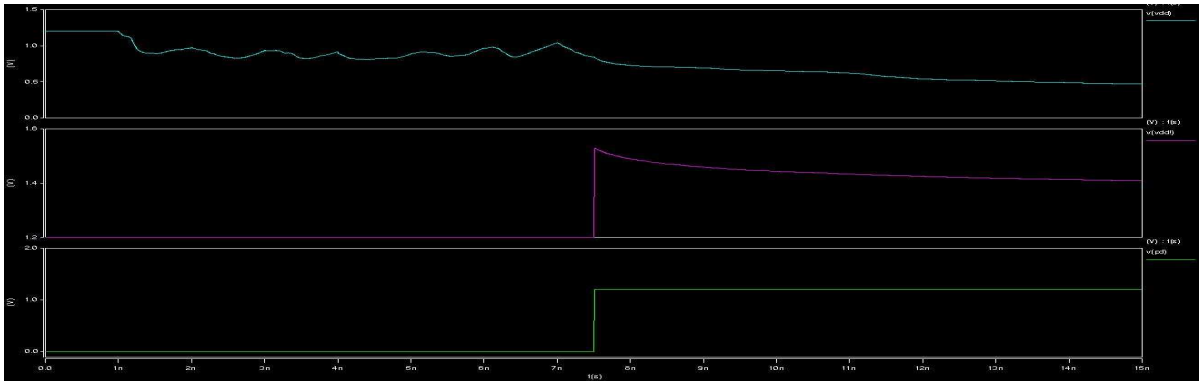

**Figure 14: Power-up and down profile for 7.2 um of power gating transistors. VDD is shown in light blue, and the active-high power-down signal is shown in green.**

The width of the pfet driving VDD was then increased in size to 72um and the same simulation was run. The results of the simulation can be seen in Figure 15. In the image, the yellow line is now VDD and the purple line is the power-down signal. VDD is slightly unstable before the power-down signal is asserted, due to the pfet still not being able to source enough current to operate the ALU. VDD floats during the power-down time, and quickly rises once PD is de-asserted. VDD is within 90% of its stable value within 1ns of the de-assertion of power-down. However, after power-down, the VDD line clearly oscillates with a peak-to-peak voltage of approximately 100mV. This occurs in 1ns increments, and is due to the changing of input values, causing the ALU to calculate the new output. This shows that this pfet is not strong enough to power-up the ALU in 1 clock cycle or to keep a stable VDD for the ALU during normal operation.

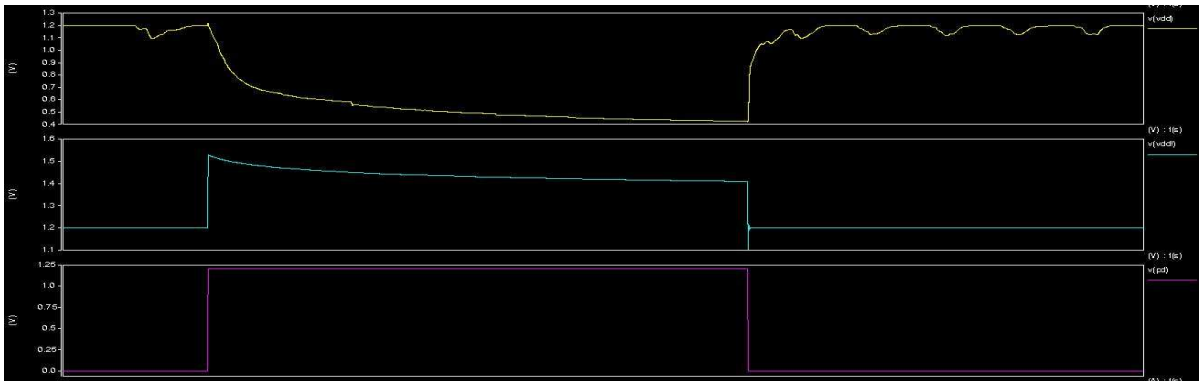

**Figure 15: Power-up and down profile for 72 um of power gating transistors. VDD is shown in yellow, and the active-high power-down signal is shown in purple.**

The width of the pfet driving VDD was then increased in size to 720um and the same simulation was run. The results of the simulation can be seen in Figure 16. In the image, the light blue line is now VDD and the purple line is the power-down signal. From this simulation, we can see two results. First, the response time of VDD after the de-assertion of the power-down signal is now much smaller than a clock cycle. Second, the VDD of the ALU is now held very stable by the pfet during normal ALU operation.

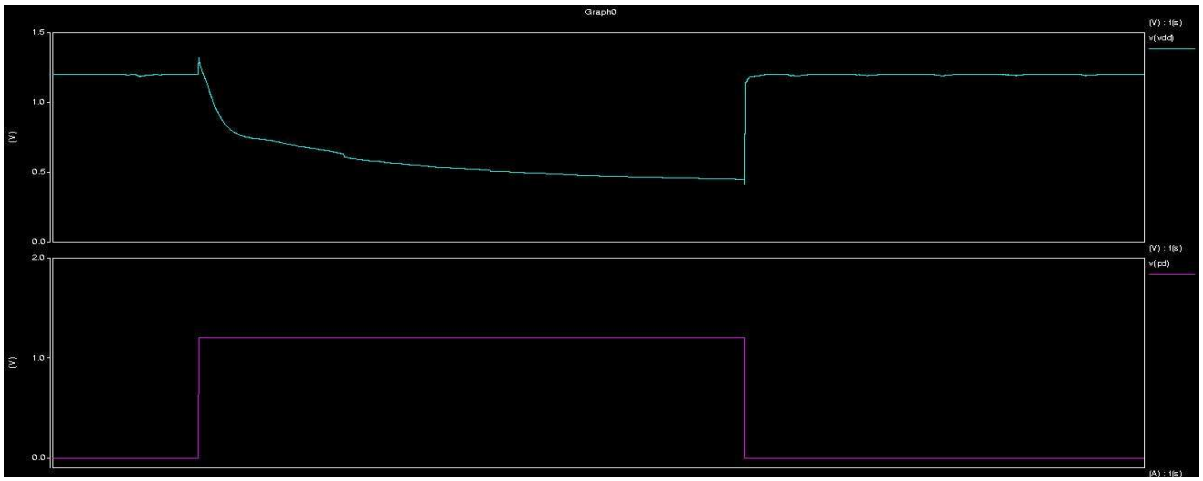


**Figure 16: Power-up and down profile for 720 um of power gating transistors. VDD is shown in light blue, and the active-high power-down signal is shown in purple.**

Shown in Figure 17 is a close up image of the simulation from Figure 16. Here the light blue line is the VDD to the ALU and the purple line is the power-down signal. Also shown here in green is the current supplied by the pfet during power-up. The power-up time for the VDD line to the ALU was measured to be 9.4ps for a pfet width of 720um. This is quick enough that the ALU can be powered up in the same clock cycle that it must be used. However, this power-up causes a short current spike to approximately 2.5mA.

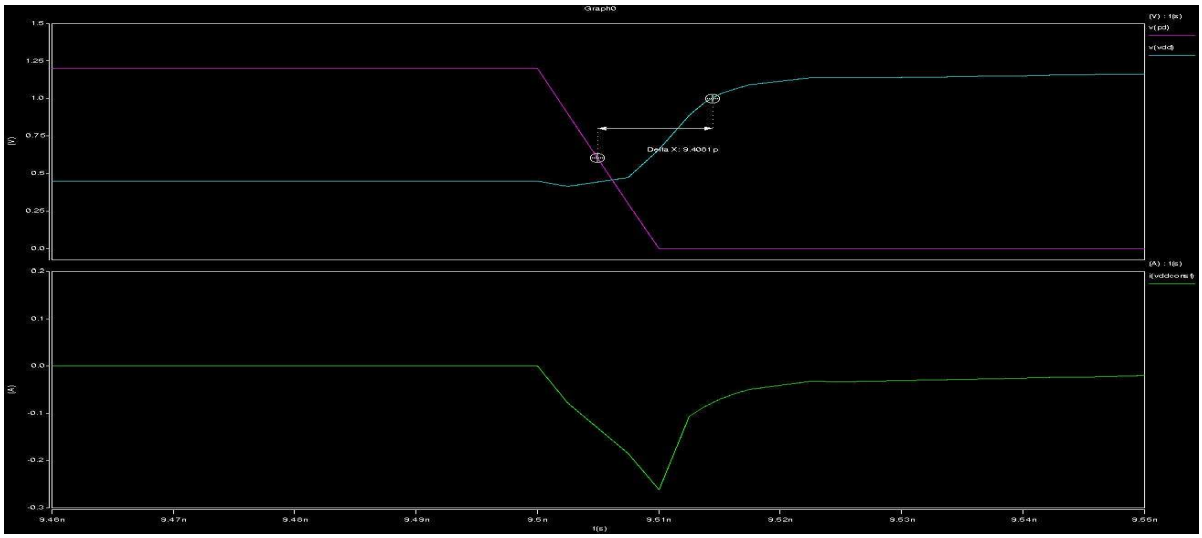


**Figure 17: Power-up time and power-up current for ALU with 720um of power gating transistor. Power-up time is 9.4 ps, and the power-up current draw is approximately 2.5mA.**

The energy required to power-up the ALU can be found by integrating this current spike over the power-up time, and it was calculated to be 14.10 fJ. The average current for the ALU during power down is 3.66 uA and average power was therefore 4.39 uW. The energy consumed by the ALU during a clock cycle while powered down is then 4.39 fJ. The average current of the ALU once a measurement has been completed was 13.59 uA and average power was 16.31 uW. If the ALU is left with gated inputs for one clock cycle, it will consume 16.31 fJ.

If the energy required to power up the ALU plus the energy consumed by the ALU during one clock cycle while powered down is less than the energy of the ALU with gated inputs, than the ALU should be powered down, even if it must be used every other clock cycle. Otherwise, the only reason to power down the ALU is if it is not used at all during the operation of an application. Based on the calculations in the previous paragraph, powering down the ALU is not worthwhile unless it is done for more than one clock cycle. If it is powered down for at least two clock cycles, powering down then back up will consume less energy than gating the inputs and supplying the ALU with power.

## 7.3 Voltage Scaling

Using our HSPICE testbench, we ran a number of simulations for the ALU and Shifter at different supply voltages. These tests were designed to measure delay based on the longest observed path for the given synthesis result and average power consumption at each voltage.

The resulting curve for the ALU, Figure 18, is as expected. Initially, reducing the supply voltage leads to a significant decrease in both power and delay, but as supply voltage falls, the power savings becomes minimal while the delay increase becomes huge. However, this also shows that in cases where delay is not important, power savings of over 75% can be achieved by reducing the supply voltage from 1.2 V to 0.6 V. For some delay-insensitive applications, this may be an acceptable low-power solution.

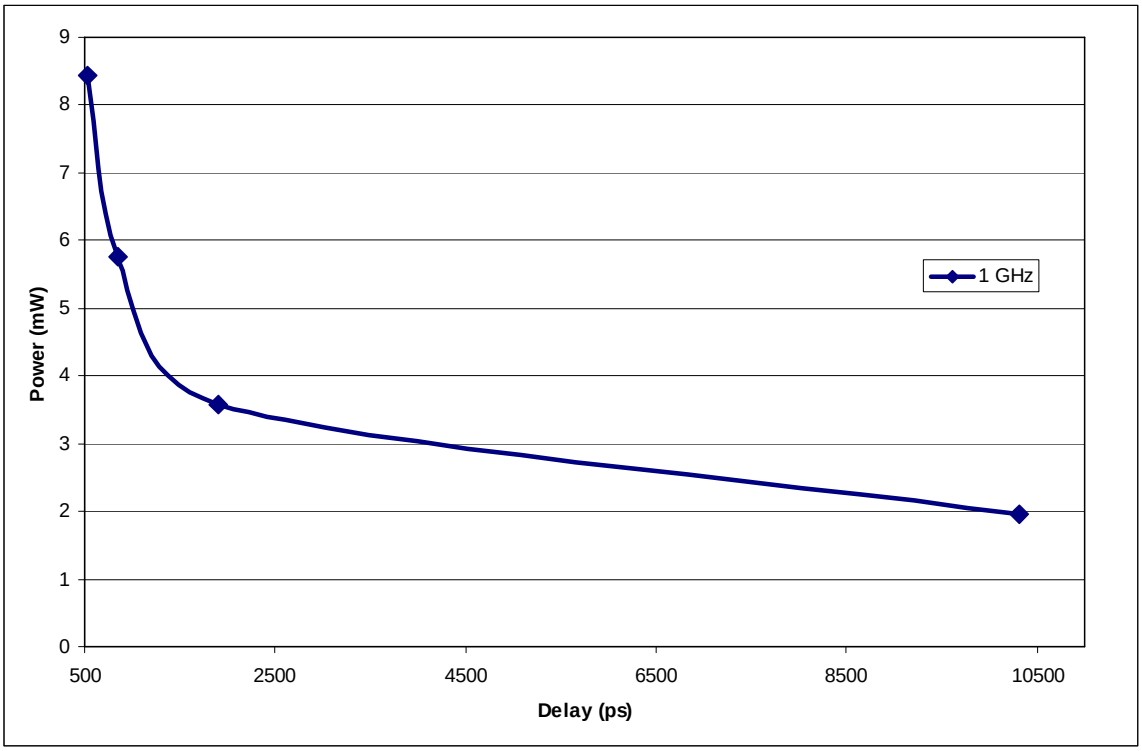


**Figure 18: ALU delay vs. average power at 1.2 V, 1.0 V, 0.8V and 0.6 V supply**

A similar graph is provided for the Shifter in Figure 19. Here, lines are provided showing results for synthesis targeting 1 GHz and 500 MHz. Please note the logarithmic scale for delay. Observe that the design targeting the higher frequency does indeed have a much lower delay at 1.2 V as expected, but also consumes much more power. It is also interesting to observe that at 1.2 V the 500 MHz design consumes less power and has a lower delay than the 1 GHz design does at 1.0 V. In other words, the 500 MHz design is superior in both dimensions at this point.

This result suggests that when targeting a lower frequency, or in delay-insensitive cases, it is best to resynthesize with more relaxed frequency constraints in addition to lowering the supply voltage. Another advantage of this approach is that lower power results can be obtained without introducing level shifters, which burn power reducing the total power savings.

The 200 MHz synthesis result is slightly better than the 500 MHz result in terms of both delay and area. At first this may seem surprising, but it is likely that the shifter has can trivially run at above 500 MHz, so synthesis requires very little effort at these target frequencies. At this point, random variation between the synthesis runs may dominate.

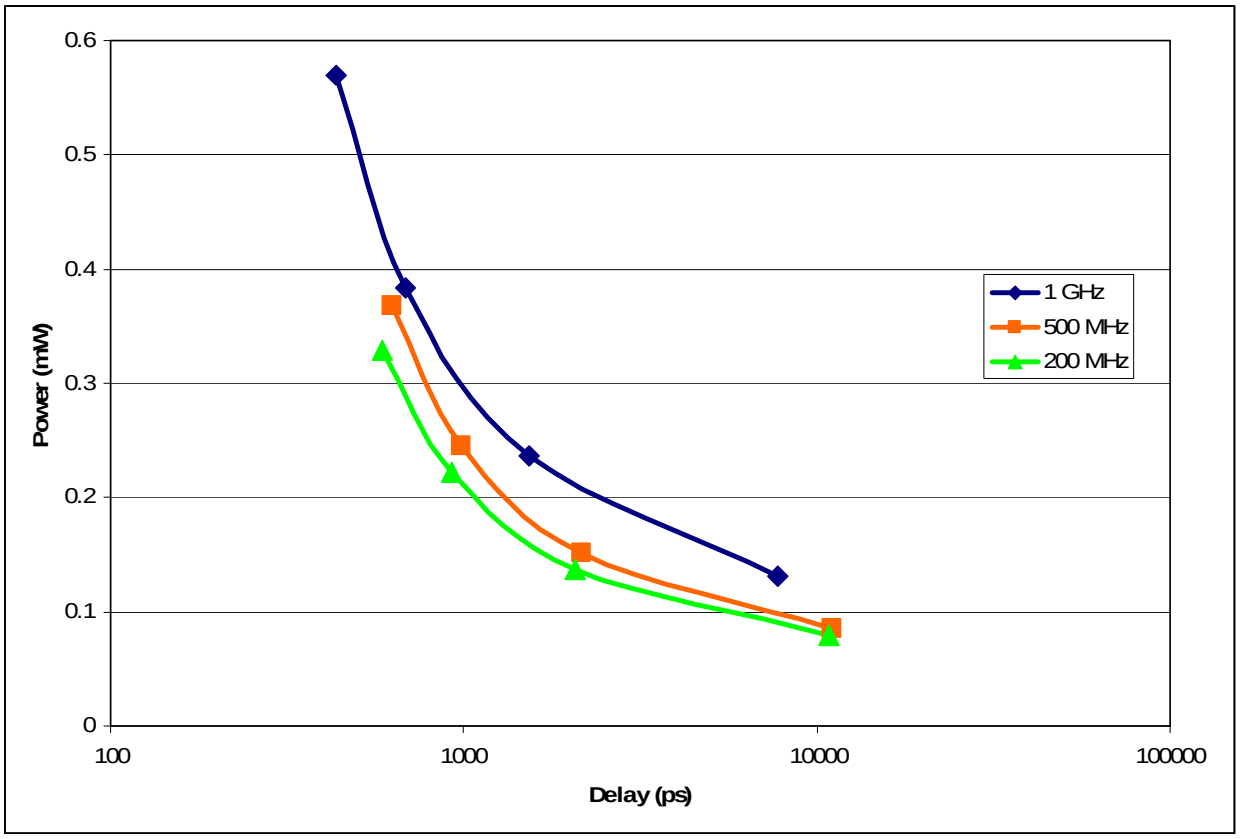


**Figure 19: Shifter delay vs. average power at 1.2 V, 1.0 V, 0.8 V and 0.6 V supply for various synthesis targets**

## 7.4 Synthesis

Logic synthesis is an extremely complex process. Design Compiler attempts to produce the best implementation of the design using the standard cells provided by the PDK, given the input constraints. There are many tuning knobs available in any synthesis tool, and there is no single best value for any of these knobs for all designs. In addition, the default values are frequently arbitrary, and may not be the best values for any design at all. For these reasons, the exact options passed to synthesis can have a significant impact on the quality of results.

To investigate this, we choose to compile the MADD with three different sets of the options. The first option, referred to as the naïve option, makes no attempt to optimize for power. Instead, this option uses the normal compile command and a 1 GHz constraint on frequency. The second option is identical to the first, but dynamic power is constrained to 5 mW, using the set_max_dynamic_power command.

In our tcl scripts, synthesis is completed in two steps. The first compile step does most of the synthesis work. A second incremental compile step finalizes the design and makes incremental improvements, potentially at a greater effort level. The third compilation option used replaces the second compile step with a compile_ultra command. This command makes use of Design Compiler's more advanced DC Ultra module.

Compiling the MADD with these three scripts shows a very significant reduction in total power, as can be seen in Table 5. These power numbers were generated using a probabilistic switching model in Prime Time. Observe that constraining power reduced power usage by 32.8% and that utilizing the more advanced compilation technique reduced power by an additional 37.6%, for a total power reduction of 58.0%.

**Table 5: Power results at 1GHz for various synthesis scripts**

| | Internal | Switching | Leakage | Total |
|---|---|---|---|---|
| Naïve | 11.2 mW | 7.16 mW | 1.07 uW | 18.3 mW |
| Constrained | 7.77 mW | 4.56 mW | 0.59 uW | 12.3 mW |
| Ultra | 4.68 mW | 3.00 mW | 0.61 uW | 7.68 mW |

Given the observations about the nature of synthesis parameters above, these results are not surprising. It is also important to note that the fact that small tweaks to the synthesis tcl script can have such a huge impact on the results makes comparisons between various runs unreliable. For this reason, we have been very careful to only make direct comparisons to between power numbers generated with similar synthesis scripts throughout this report. The reader should be wary when comparing numbers from different sections of the report, as they may not be comparable due to this sort of difference.

Another interesting observation comes from comparing the sources of power usage between these three options. Because the number of registers doesn't change between the options, some areas of power such as clock distribution become much more significant as the total power decreases. This result can be seen in Table 6. When considering these results, please note that we are not making the claim that our Naïve method is the worst possible synthesis option, as it is actually fairly good. In addition, the Ultra method is most likely not the best possible result.

**Table 6: Breakdown of contribution of each power group at 1 GHz for various synthesis scripts**

| | Clock | Register | Combinational |
|---|---|---|---|
| Naïve | 9.95% | 13.0% | 77.05% |
| Constrained | 12.7% | 14.8% | 72.5% |
| Ultra | 27.4% | 12.9% | 58.5% |

## 7.5 Area

To accomplish power-up of the ALU in less than 1 clock cycle, 720 um of a PMOS transistor must be added to the design. This is clearly unrealistic, unless a better pfet could be found to source more current per micron in order to act as a power transistor. That being said, other techniques, such as allowing a full clock cycle during power-up could be used, thereby reducing the size of pfet needed as the power transistor. Also, if the ALU were to only be statically powered down, the size of the pfet could be greatly reduced. The pfet simply needs to source enough current for the normal operation of the ALU and does not have to source the current required for the dynamic power-up.

# 8 Additional Optimizations and Explorations

## 8.1 Behavioral Model Power Consumption

Up until this point, we have been comparing the power consumption of our low power functional unit component designs to those of the original models. However, further investigation revealed that the original component models were not power efficient. The original models do not compare favorably to behavioral models for any of the components. In addition, an improved ALU behavioral Verilog model shows significant improvement over the original ALU model used. The instruction set for the ALU is also slightly different.

**Table 7: Power consumption of low power design and pure behavioral models**

| | Shifter | ALU | MADD |
|---|---|---|---|
| Low Power Design | 0.686 mW | 3.17 mW | 7.68 mW |
| Behavioral | 1.07 mW | 1.70 mW | 21.3 mW |

As anticipated, the Shifter and MADD are significantly better than the pure behavioral design. However, the corrected ALU low power design is much worse than the behavioral design. This is due both to the slightly modified instruction set, as well as incorrect synthesis results for the ALU being reported in section 7.1.

## 8.2 Improved ALU Design

The fact that the "low power" ALU uses 86% more power than a pure behavioral design suggests that there is room for further improvement. It is worth noting that a significant portion of the area, and power, used by the ALU designs in section 4.2 is in the adder component. However, the adder is not in use for many of the instructions. In order to take advantage of this, we placed latches in front of the adder in the ALU, so that the inputs to the adder would not change for instructions where the output of the adder is irrelevant. After making this adjustment changes, the low power ALU design consumes 0.562 mW of total power, a 67% improvement over the behavioral design. Because of its superiority, this design will be used going forward.

## 8.3 Input Latches

In some cases entire ALUs, Shifters or MADDs may be present in a functional unit but not in use. As demonstrated in section 7.2, if the units are only idle for a small number of cycles, power gating is not practical. An alternative approach in these cases is input latches. The inputs are latched, and the latches are only enabled when new outputs are required. In this way, changes to the module inputs are prevented from wasting power when the output is irrelevant. However, since the latches consume some power, there will be an increase in power in the cases where the latches are enabled. The power increases, assuming latches are enabled constantly are as follows.

**Table 8: Power consumption of input latches**

| | Shifter | ALU | MADD |
|---|---|---|---|
| No Input Latches | 0.686 mW | 0.562 mW | 7.68 mW |
| Input Latches | 0.824 mW | 0.651 mW | 8.09 mW |
| Increase in Power | 20.1% | 15.8% | 5.34% |

Because of the high cost of input latches, particularly on the smaller modules, careful evaluation of a particular architecture is required to determine if they are cost effective for that architecture. In general, input latches make the most sense on the MADD, both because they have a smaller relative cost and because a MADD is more likely to be unused for short periods by some applications.

# 9 Discussion

## 9.1 Future Work

Although the work we have completed is beyond what is required for the purposes of the MOSAIC project at this point, there are some additional avenues to explore. Most significantly, we could investigate low-power techniques for some of the nearby related components of the MOSAIC architecture. These include the crossbar, interconnect network and register files.

Focusing only on the functional unit modules, we could attempt to employ more extreme low-power techniques. These include aggressive synthesis optimization, further architectural optimizations and a deeper investigation into ideal supply voltages and required operating frequencies. Additionally, running longer simulations on representative data sets with extracted parasitic could give more exact power numbers. However, we feel that these would all bring diminishing returns at this point and any additional effort would be better spent focusing on other areas of the MOSAIC system.

## 10 Conclusion

In this paper we explored various low-power design techniques such as clock gating, power gating, input gating, voltage scaling and architectural improvements, and implemented them in the Functional Unit of a Coarse Grain Reconfigurable Array.

We demonstrated that with the aforementioned low-power techniques implemented, we are able to reduce the total power consumption of the various pieces of the FU by over a factor of four. This includes the Arithmetic Logic Unit, which is capable of performing instructions from several categories of instructions, including: Arithmetic, Comparison, Bitwise, Compare A with 0, Compare A with 1, Arithmetic with Immediate, Select with Immediate, and LUT Ops. In addition, a Barrel Shifter and MADD unit were implemented and the above low power techniques applied, with similar results to the ALU. In addition, we were able to further reduce ALU power by latching the inputs to the internal adder.

Other low power techniques, including power gating and input latches were explored. These techniques all have potential benefits, but they also come at significant cost. For this reason, their usefulness depends on specific architectural details and application space of the functional units, and we can not recommend them in the general case.

Finally, we also recorded the experience we had with various tools throughout our design process. We believe this to be beneficial not only to future work, but to other designers following in our footsteps.

## 11 References


[1] Jimmy Xu, Nikhil Subramanian, Adam Alessio, Scott Hauck, "Impulse C vs. VHDL for Accelerating Tomographic Reconstruction," fccm, pp.171-174, 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, 2010

[2] Brian Van Essen, Aaron Wood, Allan Carroll, Stephen Friedman, Robin Panda, Benjamin Ylvisaker, Carl Ebeling, and Scott Hauck, "Static Versus Scheduled Interconnect In Coarse-Grained Reconfigurable Arrays", in Proceedings of 2009 IEEE International Conference on Field Programmable Logic and Applications, pages 268-275, Aug. 31 2009-Sept. 2 2009

[3] Stephen Friedman, Allan Carroll, Brian Van Essen, Benjamin Ylvisaker, Carl Ebeling, and Scott Hauck, "SPR: an architecture-adaptive CGRA mapping tool",In Proceedings of 2009 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Feb. 2009, 191-200

[4] Allan Carroll, Stephen Friedman, Brian Van Essen, Aaron Wood, Benjamin Ylvisaker, Carl Ebeling, Scott Hauck, "Designing a Coarse-grained Reconfigurable Architecture for Power Efficiency", Department of Energy NA-22 University Information Technical Interchange Review Meeting, 2007

[5] Cherkauer, B.S.; Friedman, E.G.; , "A hybrid radix-4/radix-8 low power, high speed multiplier architecture for wide bit widths," *Circuits and Systems, 1996. ISCAS '96., 'Connecting the World'., 1996 IEEE International Symposium on* , vol.4, no., pp.53-56 vol.4, 12-15 May 1996

[6]Li-Rong Wang; Shyh-Jye Jou; Chung-Len Lee; , "A well-structured modified Booth multiplier design," *VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on* , vol., no., pp.85-88, 23-25 April 2008

[7] Kang, J.-Y.; Gaudiot, J.-L.; , "A Simple High-Speed Multiplier Design," *Computers, IEEE Transactions on* , vol.55, no.10, pp.1253-1258, Oct. 2006

[8] Magic Blue Smoke blog. http://synopsysoc.org/magicbluesmoke/2008/01/functional-simulation-using-upfcontd

[9] Chandrakasan, A.P.; Sheng, S.; Brodersen, R.W.; , "Low-power CMOS digital design," *Solid-State Circuits, IEEE Journal of* , vol.27, no.4, pp.473-484, Apr 1992