ASSURE Final Report

Task Number: 2.5

Project PI(s): Mark C. Johnson

Contact email(s): mcjohnso@purdue.edu

Performing University: Purdue University

Date: August 3, 2020

1. Accomplishments

1.1 Task goals

This project aimed to achieve the following 1) Prepare students with experience in SoC design and as potential recruits for NSWC Crane. 2) Install and test a 90nm SOI design flow to enable design and fabrication. 3) Tapeout at least two SoC designs to be sent for fabrication. 4) Test both designs at Purdue and Crane NSWC. 5) Release the open-source System Verilog code for a RISC-V based SoC to be made available for use by Crane NSWC and the broader academic community.

1.2 Major activities

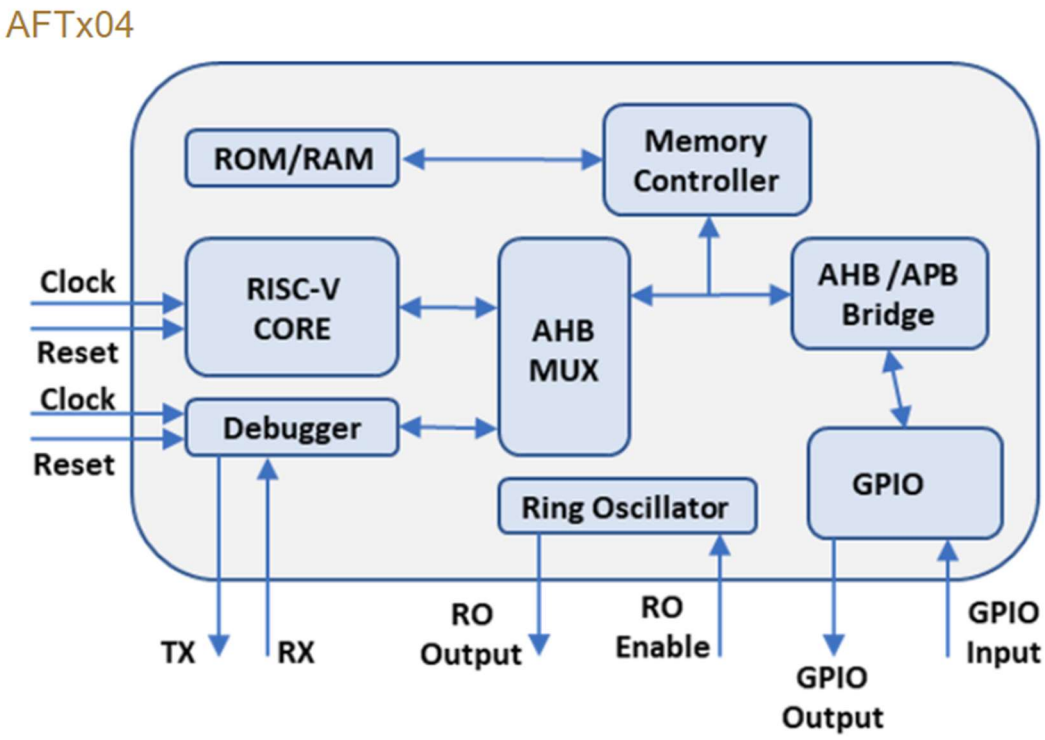
Summary

From January 2018 through July 2020, the SoCET Team has accomplished the following. In August 2018, the team successfully **taped out the AFTx04 System-On-Chip**, and the fabricated chips were tested in Fall 2019. A **mico-controller architecture optimized for machine learning, SparCE[1] [1],** was implemented to take advantage of matrix convolutions which contain a fair amount of zero entries and was included in the SoCET team's second tapeout. **Non-Symmetric, CMOS Implemented Polymorphic logic gates**, based off of Dr. Appenzeller's proposed ASSURE task 1.1, were created to demonstrate the ability to camouflage a gate's functionality from attempts to reverse engineer a gate level netlist. **Layouts for Electromigration Test Structures**, designed by Dr. Peter Bermel's research team, were implemented and included in the SoCET Team's second tapeout. In February 2020, **AFTx05 was taped out**; however, the fabricated chips will not be delivered until Spring 2021. A **JTAG Interface** was made to improve the time required to write a program into SRAM, as well the debugging features our SoC offers; it will be available in AFTx06. A **Phase-Locked Loop** was designed to eliminate the need for an external clock, so that the next, MIT Lincoln labs fabricated, SoC will operate at a higher clock frequency. A **Platform-Level Interrupt Controller** was created to offer interrupt functionality as a step towards maturing the feature set of the next chip iteration, AFTx06.

[1] S. Sen, S. Jain, S. Venkataramani and A. Raghunathan, "SparCE: Sparsity Aware General-Purpose Core Extensions to Accelerate Deep Neural Networks," in IEEE Transactions on Computers, vol. 68, no. 6, pp. 912-925, 1 June 2019, doi: 10.1109/TC.2018.2879434.

AFTx04 (1st SoC) August 2018 Tapeout

Figure 1: Top level Diagram of AFTx04 (1st SoC)



The diagram details the architecture of the AFTx04, the first chip which the team taped out in August 2018, using the MIT Lincoln Labs 90nm FDSOI Process Design Kit. The fabricated chips were delivered in September 2019 and were functionally verified at NSWC Crane. This chip iteration included the replacement of a previously used ARM M0 core in favor of a RISC-V created by SoCET members. Students installed and wrote PDK compatible design flow scripts to create the layout for a RISC-V based System-on-Chip. Once fabricated the chips were sent to NSWC Crane for functional and environmental testing.

Features of AFTx04:

- AMBA 3.0 AHB-lite Bus
- Flip-Flop based on-chip SRAM (512 bytes)
- RISC-V processor (1st AHB Master)
  - RV32I Instruction Set Architecture
  - Pass Through Cache
  - 2 stage Pipeline
- UART debugger (2nd AHB Master)
- AHB – APB Bridge
- 8 pin GPIO
- 32kHz Ring Oscillator

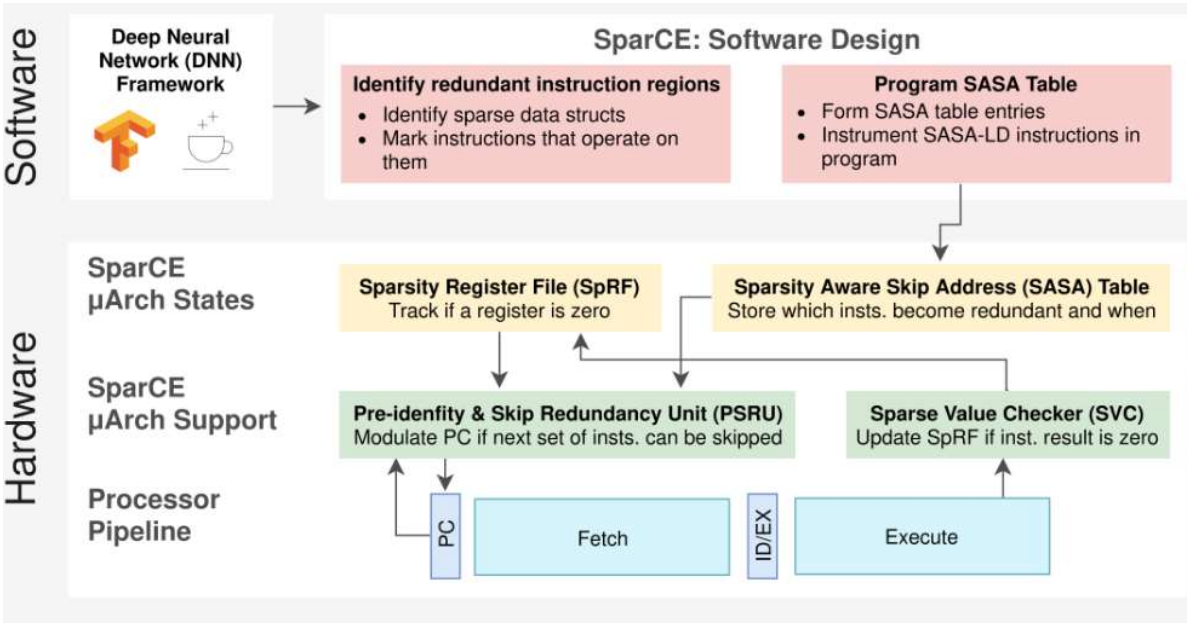
SparCE Machine Learning Architecture

SpareCE is an on-chip machine learning architecture that utilizes sparsity in convolution arithmetic to allow extraneous instructions to be skipped. The architecture has been designed to improve both the speed and power consumption of this common machine learning calculation. Students Vadim Nikiforov and Chan Weng Yan designed the module as it was described in the paper *SparCE: Sparsity Aware General-Purpose Core Extensions to Accelerate Deep Neural Networks*[2][1], with the intention of demonstrating the capabilities of the architecture on an ASIC implementation.

Possible applications include the following:

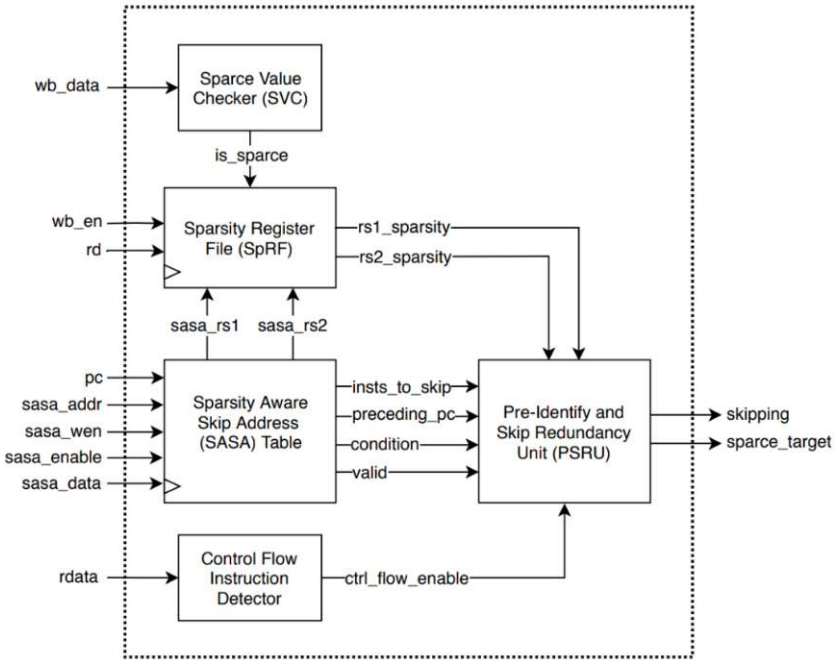
- Fault detection in mechanical devices through accelerometer data
- Object detection using low resolution IR camera data
- Command recognition via microphone input

Figure 2: SparCE Block DIagram



[2] S. Sen, S. Jain, S. Venkataramani and A. Raghunathan, "SparCE: Sparsity Aware General-Purpose Core Extensions to Accelerate Deep Neural Networks," in IEEE Transactions on Computers, vol. 68, no. 6, pp. 912-925, 1 June 2019, doi: 10.1109/TC.2018.2879434.

Figure 3: SparCE Design Architecture



The design architecture of the SparCE module is shown in the figure above. The blocks within the dotted boundary illustrate the interaction between functional blocks. Signals outside the boundary are inputs from and outputs to the rest of the pipeline.

- The **Sparsity Register File (SpRF)** is used to dynamically track which registers in the processor's register file contain zero values. The SpRF contains one entry corresponding to each register in the register file.
- The **Sparce Value Checker (SVC)** checks if the value is zero and updates the SpRF correspondingly, when an instruction that writes to a register completes,
- The **SASA table** is a cache-like block with associative memory structure which stores the information required to skip a region. Each entry contains the PC of the instruction prior to the skippable region, a field which stores the condition for the region to be skippable, and the number of instructions that can be skipped.
- The **Pre-identify and Skip Redundancy Uni (PSRU)** uses the SASA table to identify and skip redundant instruction regions. For each instruction, we check if its PC contains an entry in the SASA table. An entry in the SASA table indicates that the instruction following the current instruction is the start of a potentially skippable region. In this case, the PSRU checks the SpRF to identify if the registers indicated in the SASA table entry are currently zero. If so, it increments the PC to the end of the redundant instruction sequence, thereby skipping instructions. If not, the pipeline proceeds to execute instructions in program order.
- The **Control Flow Instruction Detector (CFID)** decodes the instruction in the decode stage rather than waiting for the instruction to be decoded in the execute stage. This allows control flow instructions to have higher precedence than skipping.

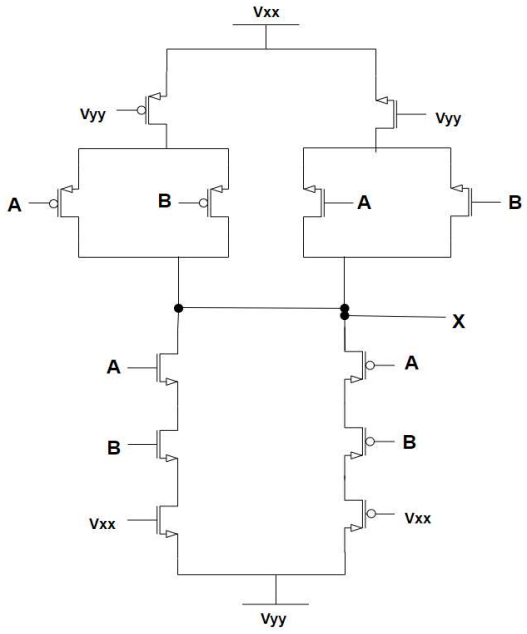
Non-symmetric CMOS Implementation of Polymorphic Logic

In ASSURE Task 1.1: TMD FETs for secure circuits through polymorphic logic gates, Dr. Jeorg Appenzeller proposed the use of TMD FETS[3][2] to implement compact polymorphic logic cells as a solution to protect intellectual property from counterfeit and trojan injections. The MIT-LL 90nm FDSOI available for this project does not support fabrication of TMD FETS without some modification of the fabrication process, so an alternative approach was taken. Intrigued by the idea of polymorphic gates, SoCET students Isaiah Grace, John Martinuk, and Brian Graves created a CMOS, non-symmetric implementation of Dr. Appenzeller's polymorphic logic concept. A gate was made to function as a NAND gate or a NOR gate, depending on the voltages applied to the power rails. Another gate was created that can behave as an XOR gate or as a Buffer.

Specifications for the Polymorphic NAND/NOR Gate:

- Power rails of the gate, Vxx and Vyy, operate at 0V and 1.2V
- Inputs pins, A and B, have a range of 0V - 1.2V
- Output pin X will has a range of 0V - 1.2V
- When Vxx is 1.2V and Vyy is 0V, the gate takes inputs A and B and yields a NAND output, X
- When Vxx is 0V and Vyy is 1.2V, the gate takes inputs A and B and yields a NOR output, X

Figure 4: Polymorphic NAND/NOR Schematic

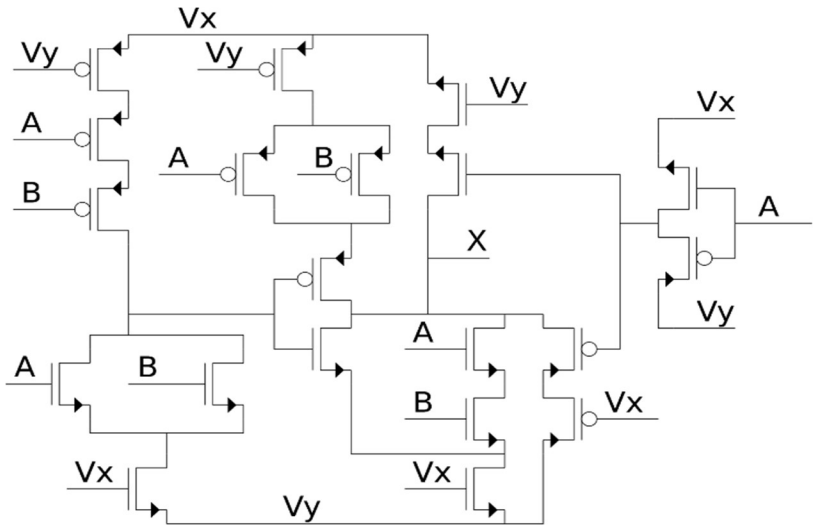


[3] S. Das and J. Appenzeller, "WSe2 field effect transistors with enhanced ambipolar characteristics," Applied Physics Letters 103, 103501-1-5 (2013).

Specifications for the Polymorphic XOR/BUF Gate:

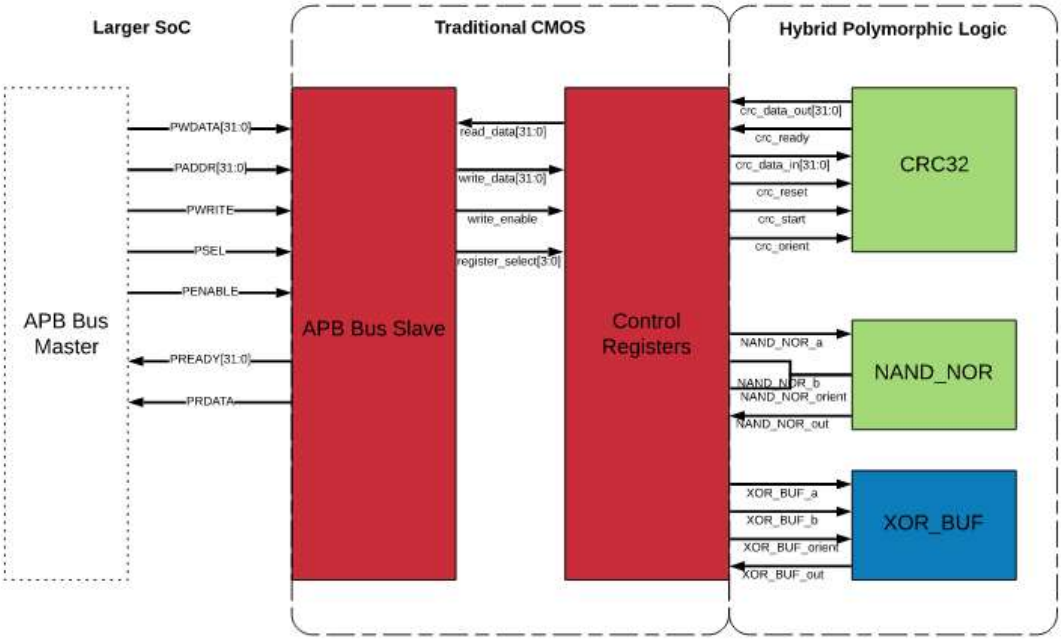
- Power rails of the gate, Vx and Vy, operate in a range of 0V - 1.2V.
- Inputs pins, A and B, have a range of 0V - 1.2V.
- Output pin X has a range of 0V - 1.2V.
- When Vx is at 1.2V and Vy is at 0V, the gate takes inputs A and B and yields an XOR output.
- When Vx is at 0V and Vy is at 1.2V, the gate acts as a buffer.

Figure 5: Polymorphic XOR/BUF Schematic



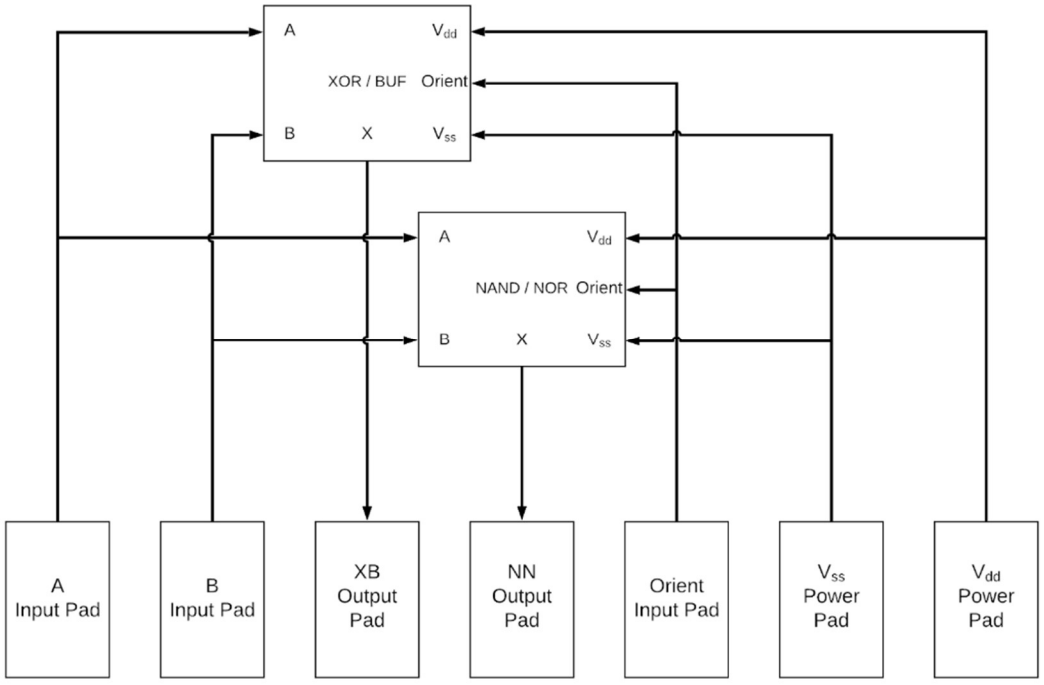
These cells were used to create a 32-bit CRC module with a configurable polynomial. This implementation was chosen to showcase the polymorphic cells' ability to camouflage the CRC polynomial being used. The APB slave interface can also provide inputs, and read outputs, which go to/ come from a single NAND/NOR cell or a single XOR/BUF cell to verify the cells' functionality in a small scale digital design.

Figure 6: Top Level Diagram of Polymorphic CRC Module (APB Accessible)



Additionally, a test structure which only interfaces with IO pads to independently test the functionality of a single fabricated NAND/NOR gate, as well as an XOR/BUF gate was developed.

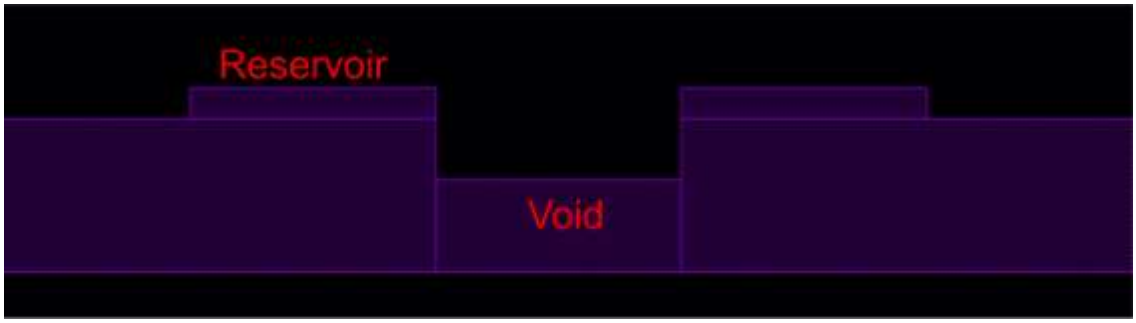
Figure 7: Diagram of Polymorphic Standalone Test Structure (IO Pad Accessible)



Layout for Electromigration Test Structures

The layouts for the electromigration test structures described in ASSURE Task 2.2 were implemented based on specifications derived by Dr. Bermel's group. The structures are currently in the process of being fabricated with AFTx05. Electromigration negatively impacts timing and could potentially create an open circuit as the cross-section area of the wire decreases over long periods of current being applied to the metal. Dr. Bermel's team is investigating methods for measuring electromigration with the use of voids and reservoirs:

Figure 8: Example of an Electromigration Test Structure



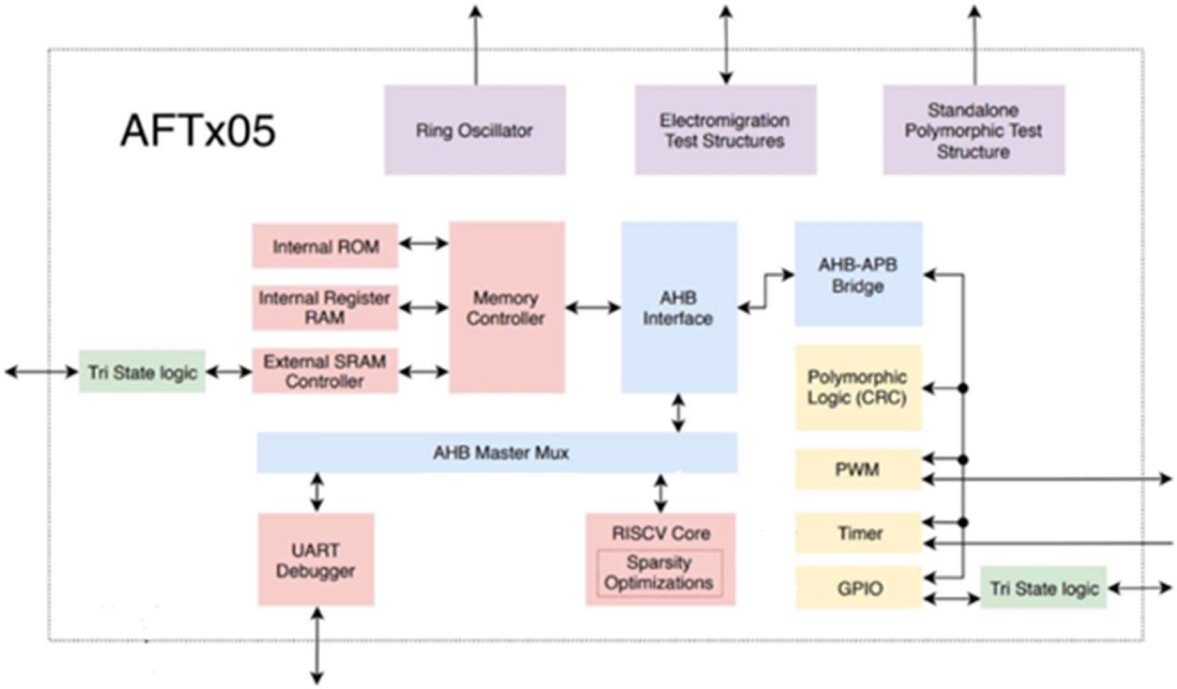
The ions from the reservoir will drift into, and gradually fill, the adjacent voids which will be thermally imaged with a microscope. In order for these devices to be visible to the microscope, the test structures were placed near the top surface of the upper layers of AFTx05 SoC, without any metal layers obstructing the view of reservoirs and voids.

The layout requirements for these electromigration test structures were the following:

- Ensure high currents could be delivered to the test structures.
- Ensure the test structures could be imaged with use of a microscope.
- Create the layout for a set of 10 electromigration test structures, with a minimum of 4 sets. placed in the final chip layout (40 total structures).
- Ensure that the inclusion of these test structures didn't affect the rest of the AFTx05 design.

AFTx05 (2nd SoC) February 2020 Tapeout

Figure 9: Diagram of AFTx05



AFTx05 is SoCET's 5th chip iteration and was taped out on February 18th, 2020. It was the second chip fabricated with MIT Lincoln Labs' 90nm FDSOI design kit and significantly expanded the scope, as well as the feature set, of AFTx04. AFTx05 is an SoC based around a RISC-V single-core processor supporting RV32I Spec. v.2.1.

Between AFTx04 and AFTx05 the following features we added to the design:

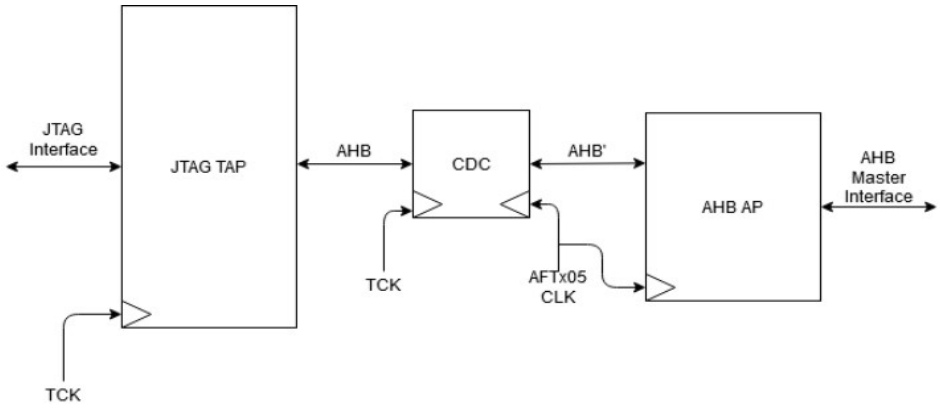
- Sparsity-exploiting processor optimizations targeted at machine learning workloads
- Custom Electromigration test structures
- Polymorphic Logic Test Structure (IO accessible) and Polymorphic CRC Module (APB Accessible)
- Pulse Width Modulation Module
- Timer Module
- External SRAM Interface
- SCAN Flip-Flops & ATE Interface

Moreover, the RTL source files used in the design, as well as the design flow scripts used with them, have been published to the public github repository: https://github.com/Purdue-SoCET/AFTx05_Public . The files are open-source and free to use by the larger academic community and NSWC Crane.

JTAG Interface

JTAG is an extensible serial standard used for board-level IC testing which will be available for AFTx06, the next chip iteration. Common extensions include support for device programming, memory inspection and software debugging, all of which are found on most commercial microcontrollers. The initial implementation of JTAG for SoCET included the **JTAG Test Access Port (TAP)**, the mandatory instructions and a subset of optional instructions from the IEEE 1149.1 standard, and a custom extension to interact with the AHB-Lite bus. The JTAG module is comprised of 3 major components: The aforementioned TAP, the AHB Access Point (AHB AP) created to allow interfacing with the on-chip AHB bus, and the Clock Domain Crossing (CDC) modules designed to transfer data between the JTAG clock domain and the SoC clock domain.

Figure 10: Top Level Diagram of JTAG Interface



The **Clock Domain Crossing (CDC)** portion consisted of 2 CDC FIFO modules based off of the Sunburst CDC design [3][4], and a simple synchronizer for capturing the error flag from the AHB AP.

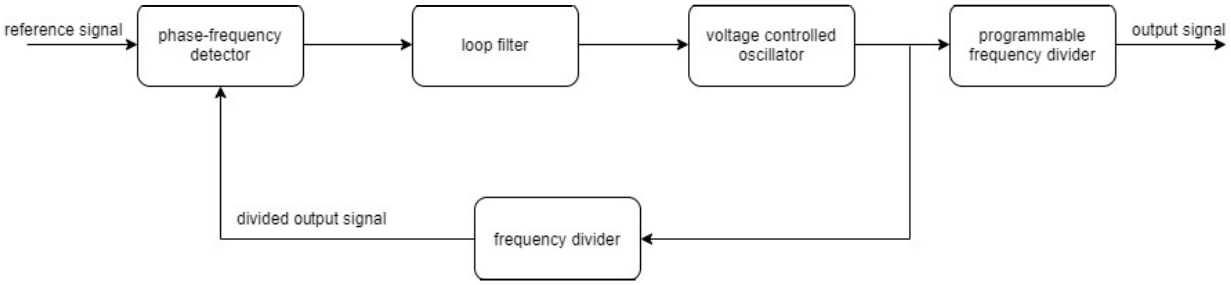
The **AHB Access Point (AHB AP)** uses a 37-bit instruction in order to perform read and write operations on the SoC bus. The instruction format was based off of a JTAG debugger design from Texas Instruments [4]. A normal read/write operation would require 2 such instructions: the first to set the target address, and the second to set the target data and start the bus request. To better accommodate device programming, an optimization allowing an auto-increment after each write was included, which removed the addressing step in consecutive read or write operations.

[4] C. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," Sunburst Design, Provo, UT, USA, 2002.

Phase-Locked Loop

Students Evelyn Ware and Matthew Olinde aimed to produce a Phase-Locked Loop (PLL) design that can provide a steady output signal, with a tunable frequency, to be used as the SoC's clock. The component will be available for the next chip fabricated with MIT Lincoln Labs. The output signal from a PLL provides a more stable signal than an oscillator since it is less susceptible to temperature changes and noise and can provide an output signal that has a much higher frequency than the reference signal.

Figure 11: Phase-Locked Loop Flow and Connections



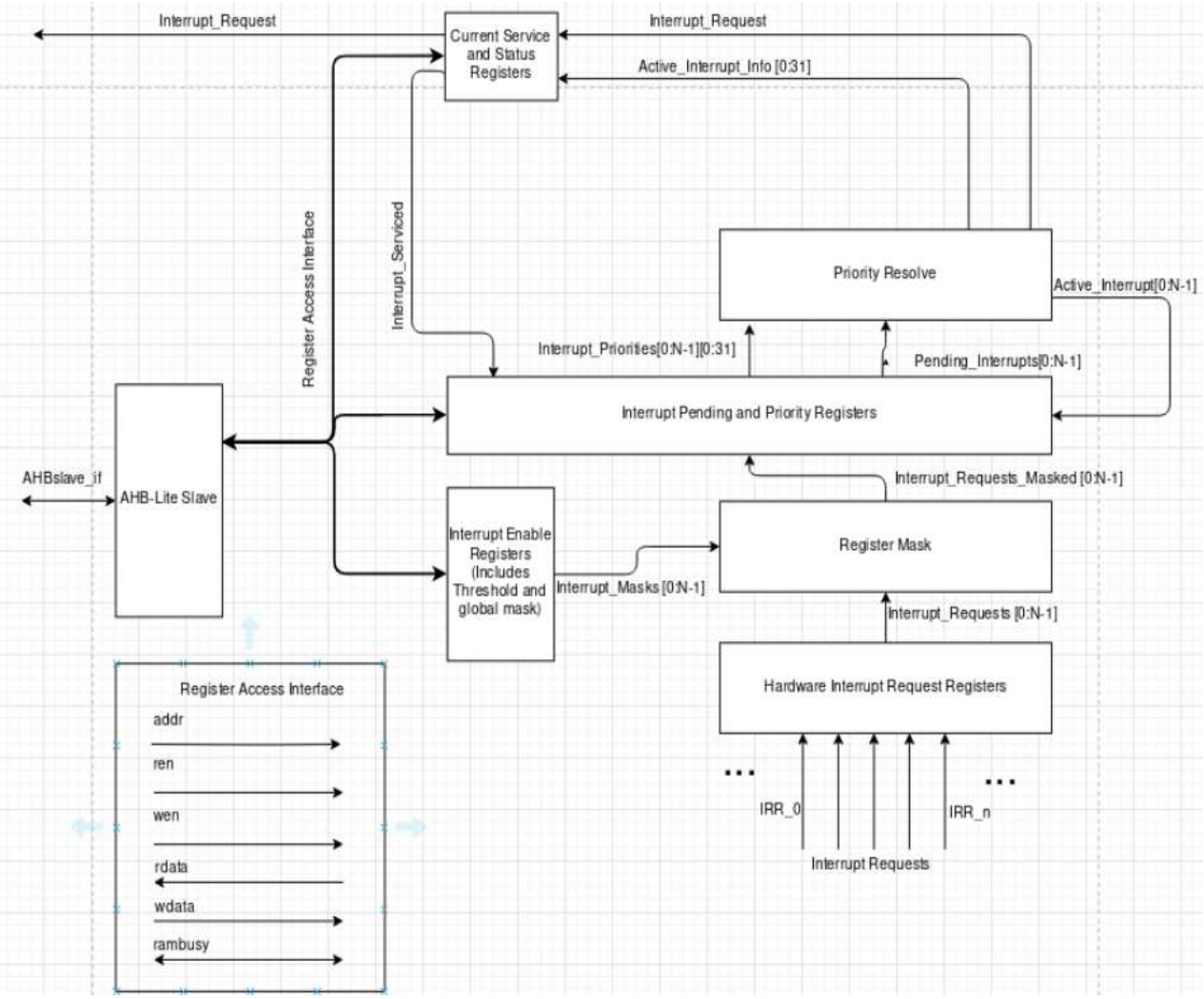
A PLL has three main building blocks: 1) a **phase detector** which compares a reference signal to the output signal and determines phase difference, 2) a **loop filter** to filter out high frequencies from the phase detector output to provide a DC signal based on the phase difference, and 3) a **voltage controlled oscillator** which generates an output signal at a certain frequency based on the input voltage level from the loop filter. With a frequency divider added to the circuit, nearly any output frequency can be generated using a single reference signal.

The phase-locked loop will be supplied with a 32kHz reference signal that comes from a, SoCET designed, ring oscillator. The range of possible output frequencies was targeted for 10Mhz - 65Mhz.

Platform-Level Interrupt Controller

The Platform-Level Interrupt Controller (PLIC) is a standard interrupt management protocol for managing timer, software, and external interrupt communications by reading a memory-mapped register. The interrupt controller will be included in AFTx06, the next chip that SoCET will tapeout. It should be noted that there are different priority levels for different user modes (U, S, H, M). Currently, M mode (or machine mode) is the only privilege level being configured because this is a mandatory privilege level for the hardware platform. The PLIC takes in hardware interrupt requests and serves them, along with an interrupt ID, to the processor. The Interrupt Controller is currently built for an address width of 32 bits and stores information in 32-bit registers.

Figure 12: Top Level Communications Between PLIC, the External Modes, and the Processor



The **Interrupt request registers** translate hardware interrupt requests into pulses to be sent to other components in the submodule. **Register mask** prevents masked interrupts from triggering an interrupt request. The **interrupt enable register** handles logic controlling which registers are masked; it handles status registers related to masking individual interrupts as well as interrupt masking when the disable low priority interrupts module is enabled. The **interrupt pending and priority registers** module handle

registers controlling interrupt priorities as well as the interrupt pending registers. The interrupt priority registers indicate the priority of each hardware interrupt channel and the interrupt pending registers indicate which registers are in the queue to be serviced. The **interrupt priority resolve register** handles sending the highest priority interrupt index to the status registers for the CPU to read.

1.3 Specific milestones and status

	Milestone	Status
1	Acquire 90nm SOI design, and determine if 90nm SOI is likely to support polymorphic logic.	Complete
2	Complete RTL for the Interrupt Handler and Floating Point Unit.	In Progress
3	Familiarize team members with 90nm SOI design flow and create scripts capable of tapping out a small SOI test chip.	Complete
4	Preliminary layout of first SoC.	Complete
5	Functional verification of first SoC.	Complete
6	Tape-out 1st SoC.	Complete
7	Hardware testbed for 1st SoC.	Complete
8	Propose changes for 2nd SoC	Complete
9	Package dies for 1st SoC	Complete
10	Functional, environmental test of 1st SoC	Complete
11	Design revisions for 2nd SoC	Complete
12	Functional verification of second SoC	Complete
13	Tape-out 2nd SoC	Complete
14	Hardware testbed, package dies for 2nd SoC	In Progress
15	Functional, environmental test of test 2nd SoC; deliver devices to Crane and other Purdue teams.	In Progress
16	Release open-source RISC-V Processor (2nd SoC).	Complete

1.4 Significant results

Summary

AFTx04 (1st SoC) August 2018 Tapeout: The SoC was successfully fabricated and tested at Crane; some of the chips had a short on the UART's tx pin. The PCBs have been fabricated and delivered; however, their assembly has been delayed until Purdue lab procedures are updated to prevent the spread of COVID-19.

SparCE Machine Learning Architecture: The module was functionally verified and included on the AFTx05 tapeout with portions of the boot-up self test ensuring the skip table's functionality on the fabricated chip. Our simulation based benchmark led us to believe that, for the average machine learning program, 20% of instructions were able to be skipped with the help of the architectural modifications. It is approximated that 20% of power consumption was saved with the use of SpareCE, but physical power analysis still needs to be conducted to verify this.

Non-Symmetric, CMOS Implemented Polymorphic logic gates: The gates were PEX simulated and signed off to be included in the AFTx05 tapeout. Both the APB interface for the polymorphic configurable CRC module, as well as the standalone test structure were tapeout and will be functionally verified when the AFTx05 chips are delivered.

Layouts for Electromigration Test Structures: 80 total test structures were placed on the taped out chip layout, and 10 unpacked dies will be delivered to Dr. Peter Bermel's group for a total of 800 test structures to be used as their dataset.

AFTx05 (2nd SoC) February 2020 Tapeout: The tapeout deadline was delayed when the foundry timing libraries for the standard cell's spice models were about 10% inaccurate. The fabricated chips will not be delivered until Spring 2021, so testing and verification of the physical chip could not be conducted before the end of July 2020. However, when the chips are delivered, they will be tested with both functional and scan test vectors.

JTAG Interface: The JTAG interface was not integrated on the AFTx05 SoC in time for the tapeout; however, it will be included in the team's next chip iteration, AFTx06. Currently the module is in the process of getting signed off by the UVM team.

Phase-Locked Loop: The layout has been PEX simulated and signed off; it will be included with the next chip that is fabricated with MIT Lincoln Labs' PDK. Another round of DRC and LVS checks will have to be done when Lincoln Labs releases their next kit, and some components may require adjustments if the target clock frequency of the next SoC changes from 50Mhz.

A **Platform-Level Interrupt Controller:** The module will be included in the team's next chip iteration and is currently in the process of getting signed off by the UVM team. Additionally, the software libraries, which facilitate manipulation of the peripherals, will have to be updated to support the use of interrupts.

AFTx04 (1st SoC) Tapeout August 2018 Tapeout

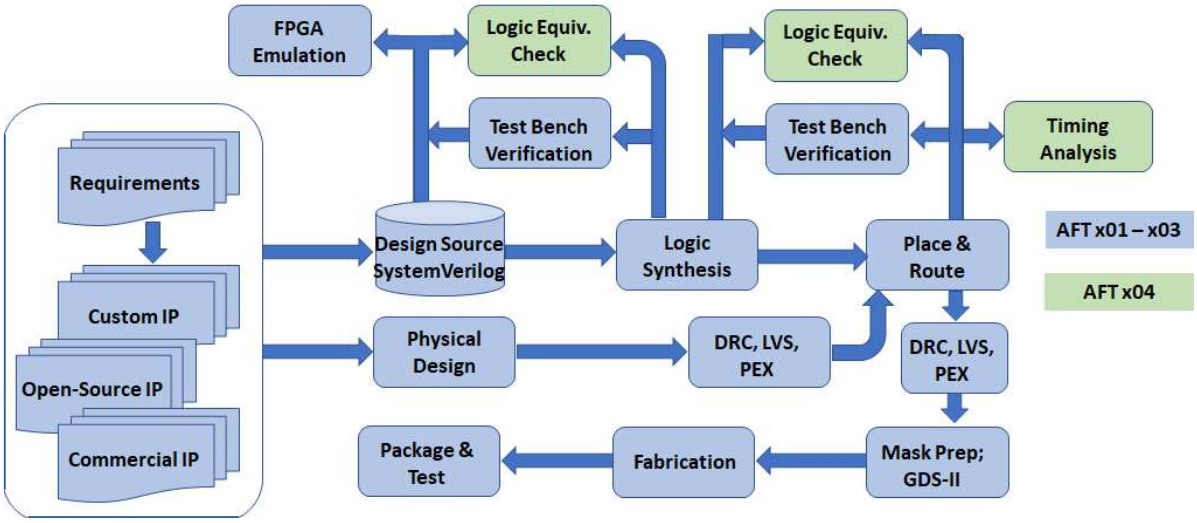
The project achieved a successful 1st tapeout of the AFTx04 SoC in August 2018. This was a simple design iteration, in terms of diverse peripherals; however, most of the digital design members pooled their efforts into the development of a single 50MHz 2-stage RISC-V core which follows the RV32I instruction and replaced a previously used ARM M0 processor. Additional component of the chip include the following peripherals:

- UART debugger
- 8 pin GPIO
- 32kHz Ring oscillator

AFTx04 Design Flow

Figure 13 details the design flow used to tapeout AFTx04. While some of these steps are standard in most fabrication runs, it is important to note that our flow was built by first time users of EDA software and guided by Matt Sale's team at NSWC Crane. Most undergraduate students never get the opportunity to practice anything beyond logic synthesis in their coursework, while the SoCET students were able to experience firsthand what it takes to turn RTL code into a signed off layout.

Figure 13: Design Flow for AFTx04



Typically, MIT Lincoln Labs fabricates small (relatively speaking) analog and mixed signal designs which do not require long wires greater than 100um in length. Consequently, when AFTx04 was going through design rule checks, a large sum of tungsten via corrosion violations were reported due the long wires present. The solution, for the time being, was to insert buffer cells which resulted in over 80% of the design consisting of buffer cells.

AFTx04 Fabrication, Packaging, and Testing

The AFTx04 chips were delivered to Crane in September 2019. A sample of the set was used with an Advantest V93000 SoC tester to be verified with the same functional test vectors that were used to sign off the post place-and route netlist before tapeout. The majority of the chips yielded correct, expected outputs; however, there were a few which had a short on their UART's tx pin.

The following are the AFTx04 layout submitted to MIT Lincoln Labs (Figure 14), without metal fill showing, the fabricated, wire bonded die (Figure 15), and the packaged AFTx05 chip (Figure 16).

Figure 14: Submitted AFTx04 Layout

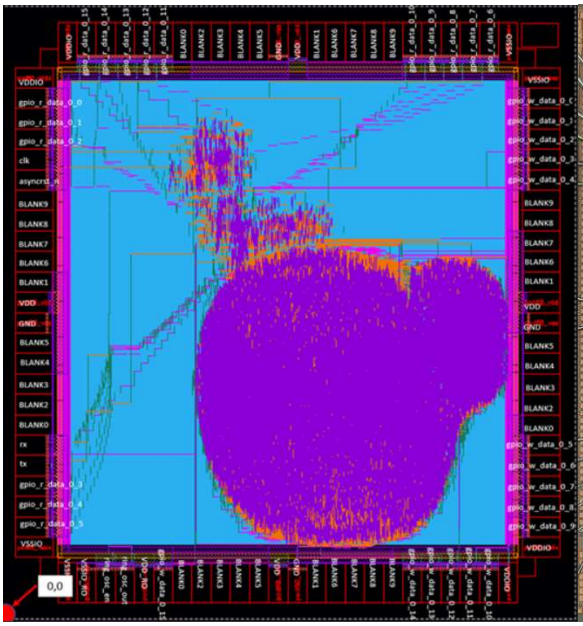


Figure 15: Picture of Wire Bonded AFTx04 Die

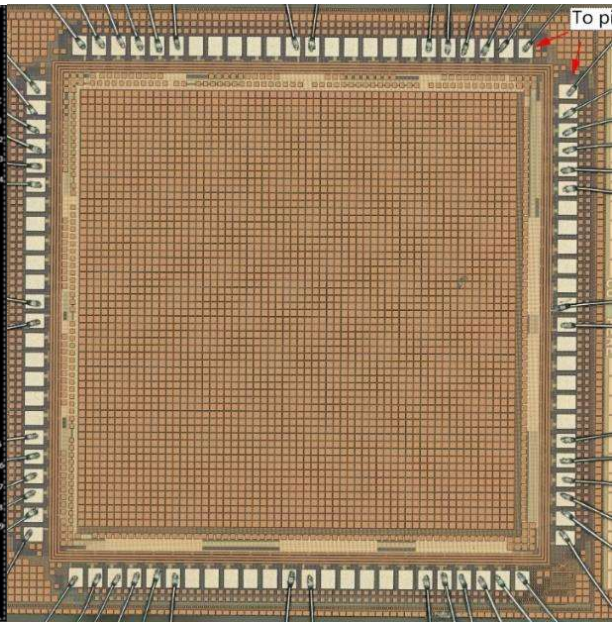
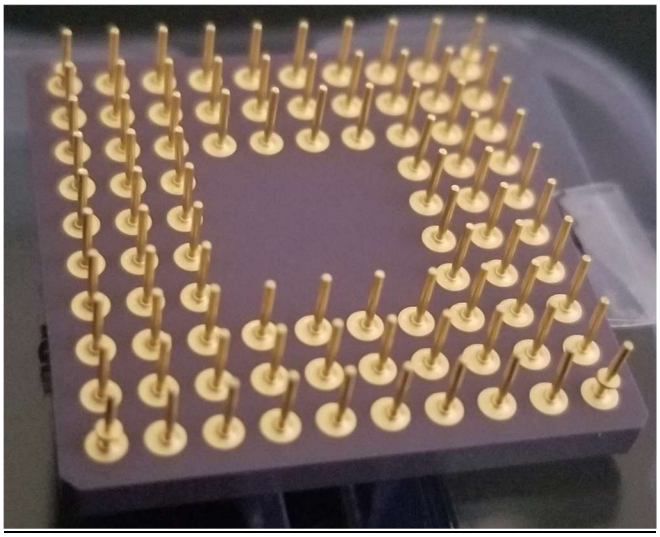


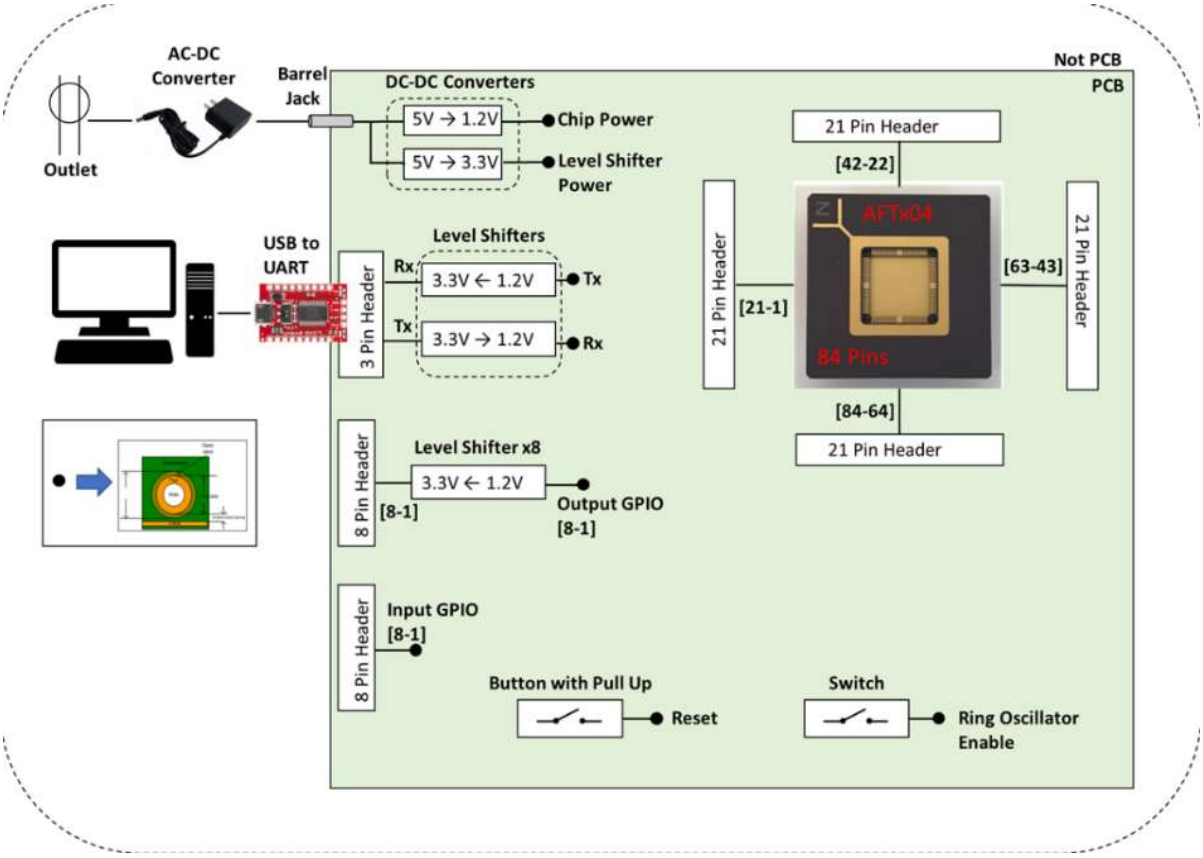
Figure 16: Picture of Packaged AFTx05 Chip



PCB Bring-Up

The PCB design for AFTx04 went through three revisions and the boards were delivered in June 2020. Figure 17 outlines the plans for the board that will connect AFTx04 to the outside world, as a general-use RISC-V microcontroller.

Figure 17: PCB Diagram

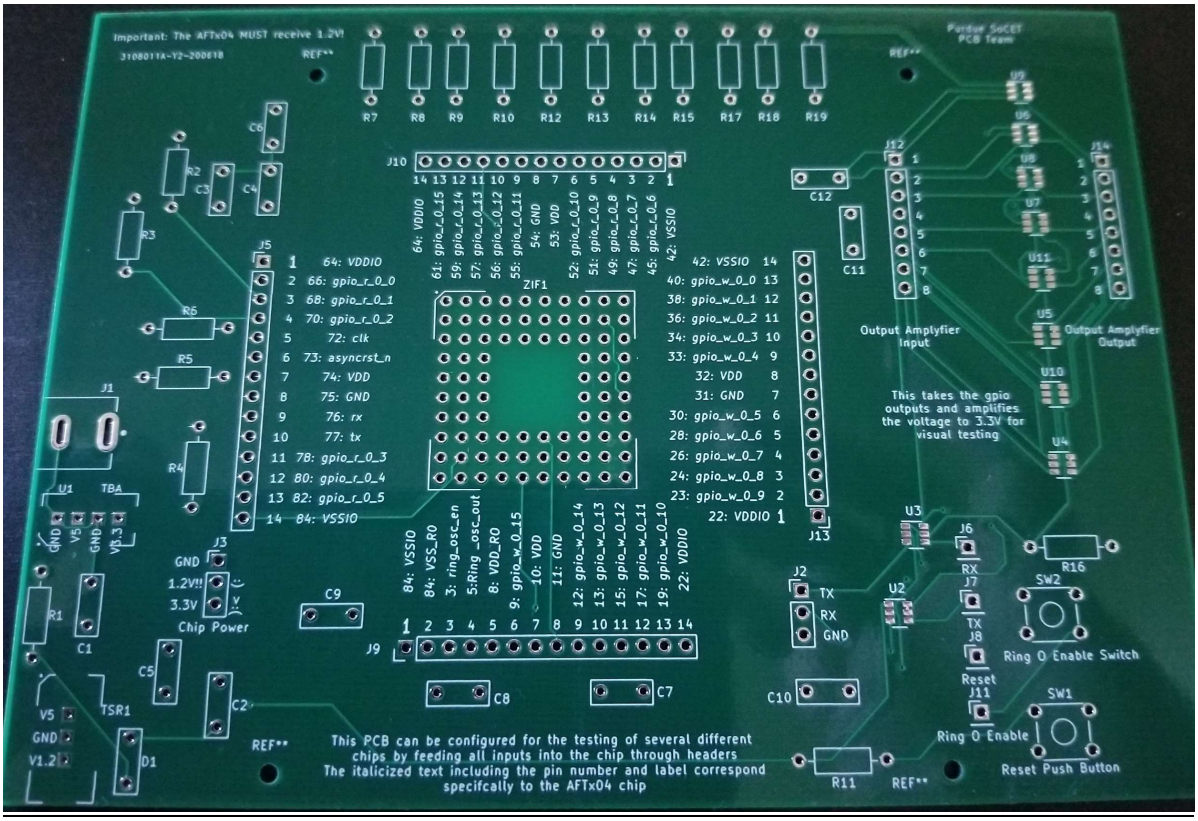


Some important notes regarding the AFTx04 chip, related to the design of the PCB:

- The GPIO pins are NOT bidirectional, so there are separate input GPIO and output GPIO pins.
- Since the voltage supply of the AFTx04 chip is limited to 1.2V, it is necessary to level shift the voltage before interfacing with any other circuits since 1.2V is not a standard voltage level.
- AFTx04 was designed with a 10MHz target clock, but it's critical paths should allow up to 50MHz.
- The self-test procedure on AFTx04 will pass the voltage on the input GPIO to its corresponding output GPIO.

Figure 18 is a picture of one of the AFTx04 PCBs. The boards will be assembled and soldered, once Purdue lab space and procedures have been updated to prevent the spread of COVID-19.

Figure 18: Picture of Delivered AFTx04 PCB



The boards will have the following elements soldered to them within the next few months:

- Interface for Clock signal from a signal generator
- LED subcircuit for the output GPIO pins
- Switch subcircuit for reset, ring oscillator enable and input GPIO
- 1.2V to 3.3V level shifters for interfacing with other circuits (USB-UART, Led subcircuit)
- Interface for wall power supply
- Voltage regulators/ buck converters for voltage supplies
- USB-UART circuit for interfacing with computer
- ZIF (zero insertion force socket) for mounting chip
- Headers for isolating chip from peripheral circuits
- Reset button

SparCE Machine Learning Architecture

Functional Verification of SparCE

The RISC-V reference emulator used by the team does not contain our sparsity optimizations, so it was not possible to verify any new functionality that the sparsity optimizations offer over the baseline processor. However, the reference emulator was used to ensure that the processor functions identically as a regular RISC-V processor, when sparsity optimizations are not used.

Functional verification was used to sign off the machine learning architecture. The following are the SparCE related self-test cases included in the ROM boot code to verify functionality in the fabricated SoC:

Figure 19: Table of SparCE Functional Test Cases

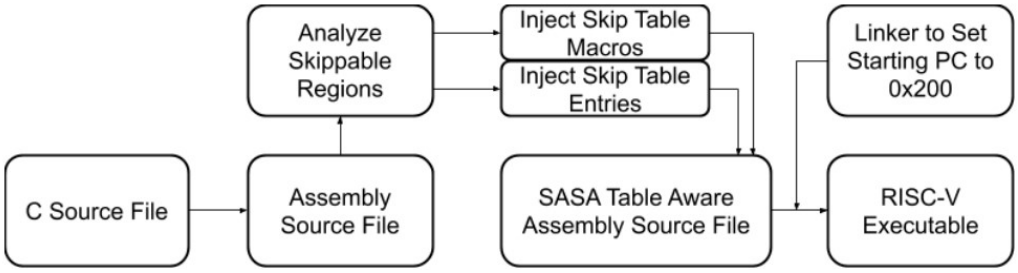
Test Case ID	Test Description
SparCE_Self_00	Basic test to ensure skipping occurs.
SparCE_Self_01	Basic test to ensure that skipping does not occur when registers have not been initialized after reset.
SparCE_Self_02	Test to ensure that each range of skipping works for the chip. Since this test case is written in a way such that different skip ranges apply to the same PC, this will also test the SASA table's collision/replacement policy.
SparCE_Self_03	Test to ensure that the instruction before the SASA preceding PC will be executed. If the instruction modifies the condition register, it WILL affect the skipping conditions because the values are forwarded from the execute stage.
SparCE_Self_04	Test to ensure that the instruction at the SASA preceding PC will be executed. If the instruction modifies the condition register, it will NOT affect the skipping conditions because it cannot be forwarded from the fetch stage.
SparCE_Self_05	Test to ensure that control flow instructions which are taken at the preceding PC will have precedence over skipping. Includes test cases where jumps/branches and skips have the same destination.
SparCE_Self_06	Test to ensure that control flow instructions which are not taken at the preceding PC will have precedence over skipping. For example, if the branch is not taken, then it cannot skip either. Includes test cases where jumps/branches and skips have the same destination.
SparCE_Self_07	Test to ensure that when SASA table is disabled, write to SASA table works but outputs are invalid. When the SASA table is enabled, the program skips normally.

SparCE_SASA_00	This test loops through the SASA table to check that every entry is not valid after reset. It loops through the SASA table and attempts to fetch data from each index. The table should always output its data as invalid
SparCE_SASA_01	This test ensures that entries are loaded correctly to the SASA table and that reading loaded values function properly as well. It loops through every possible entry in the SASA table. Since these are consecutive tests, there should be no collisions, and every value should be readable immediately after writes.
SparCE_SASA_02	This test ensures that when a SASA entry has the same program counter that already exists in the SASA table, it will update the existing entry instead of writing to the other set. This should work regardless of associativity except for direct mapped configuration.
SparCE_SASA_03	This test ensures that when the SASA table reaches full capacity, it forces the original entry out of the table. This should work regardless of associativity except for a direct mapped configuration.
SparCE_SASA_04	This test ensures that when the SASA table reaches full capacity, it forces the LRU entry out of the table. This should work regardless of associativity except for a direct mapped configuration.
SparCE_SASA_05	This test ensures that when the SASA table is disabled, writes to the SASA table is still possible but outputs of the SASA table will be invalid. This test first disables the SASA table (by writing a non-zero value to the SASA configuration register at SASA_addr + 4) writes to the SASA table to full capacity and tries to read from the SASA table. Output should be invalid.
SparCE_SASA_06	This test ensures that when the SASA table is disabled, writes to the SASA table is still possible, re-enable the SASA table and outputs of the SASA table will still be valid.
SparCE_SASA_07	This test ensures that only PC <= 0xFFFC 0000 is allowed to skip. The current SASA table is designed for a 256kB instruction memory. Since we decided to include an external SRAM with 2MB, this feature prevents PC > 0xFFFC 0000 to cause skipping due to the LSB collision.
SparCE_CFID_00	Basic test to ensure that control flow instructions (e.g. branch and jumps) are detected.
SparCE_PSRU_00	Test to ensure that the PSRU does not tell the core to skip when SASA data is invalid.
SparCE_PSRU_01	Test to ensure that the unit functions as intended for every combination of conditions and sparsity inputs.

SparCE_PSRU_02	Test to ensure that the unit correctly calculates the target address for every insts_to_skip value.
SparCE_PSRU_03	Test to ensure that skipping is correctly suppressed when encountering a control flow instruction on the preceding_pc.
SparCE_SpRF_00	Test to ensure that registers are correctly set to 0 (except for the 0 register).
SparCE_SpRF_01	Test to ensure that writing to the registers functions as usual.
SparCE_SpRF_02	Test to ensure that registers are not written to when the enable bit is not set.
SparCE_SpRF_03	Test to ensure that the SpRF correctly reports sparse values from values that are being written in flight.
SparCE_SVC_00	Basic test to ensure that the sparsity in a register is detected.

Toolchain for Utilizing SparCE Architecture

Figure 20: Compiler and Skip Table Generation Workflow



These are the steps which are required to compile a RISC-V executable that includes the SASA Table:

1. Compile the C source files into assembly source files using riscv64-unknown-elf-gcc
2. Run each assembly source file through the compiler to find skippable regions (multiplication instructions' addresses)
    a. The compiler tool first analyzes each assembly source to determine the location of the skip table
    b. Once the regions are determined, the tool adds macros to the assembly file for loading in the skip table
3. Run the riscv64-unknown-elf-gcc compiler to produce an executable binary. When compiling to the executable, a linker file must be provided so the AFTx05 knows the entry point into the code.

In simulation, the sparsity optimizations were found to decrease the number of instructions performed and therefore power draw by 20% nominally across a variety of input data sets. Physical power analysis is required to verify this estimate. This indicates that the sparsity optimizations are functioning as expected. It is also estimated that the sparsity factor of the input data will increase performance exponentially. As such, the results suggest that 20% improvement can be expected on average, but upwards of 50-60% improvement can be achieved under certain circumstances.

Non-symmetric CMOS Implementation of Polymorphic Logic

Individual Cells and Standalone Test Structure (IO Accessible)

The polymorphic cells were successfully included in the AFTx05 (2nd SoC) Tapeout. Both the polymorphic designs were verified with DRC & LVS checks and had their PEX netlists simulated to verify the cells' timing characteristic. The layouts for the NAND/NOR and XOR/BUF gate are shown below.

Figure 21: Layout of NAND/NOR gate

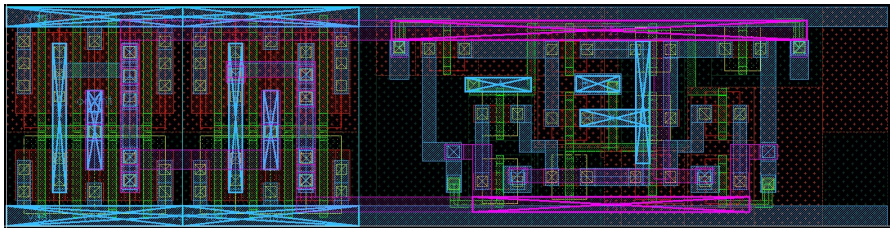
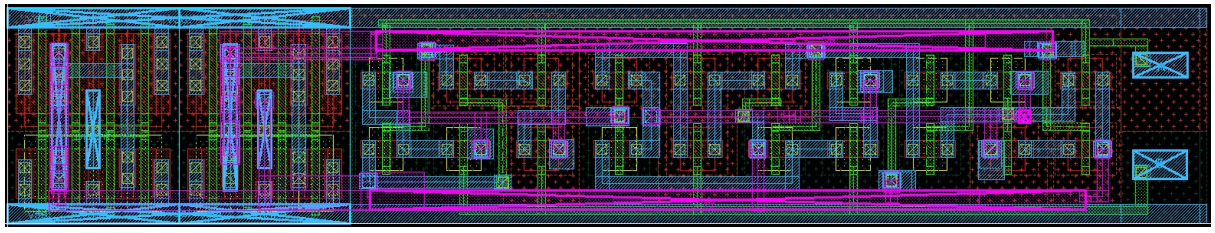


Figure 22: Layout of XOR/BUF Gate



Below are screenshots of the transient simulations which demonstrate correct functionality of both cells. The test cases are denoted at the top of the diagrams.

Figure 23: Transient simulation of NAND/NOR gate

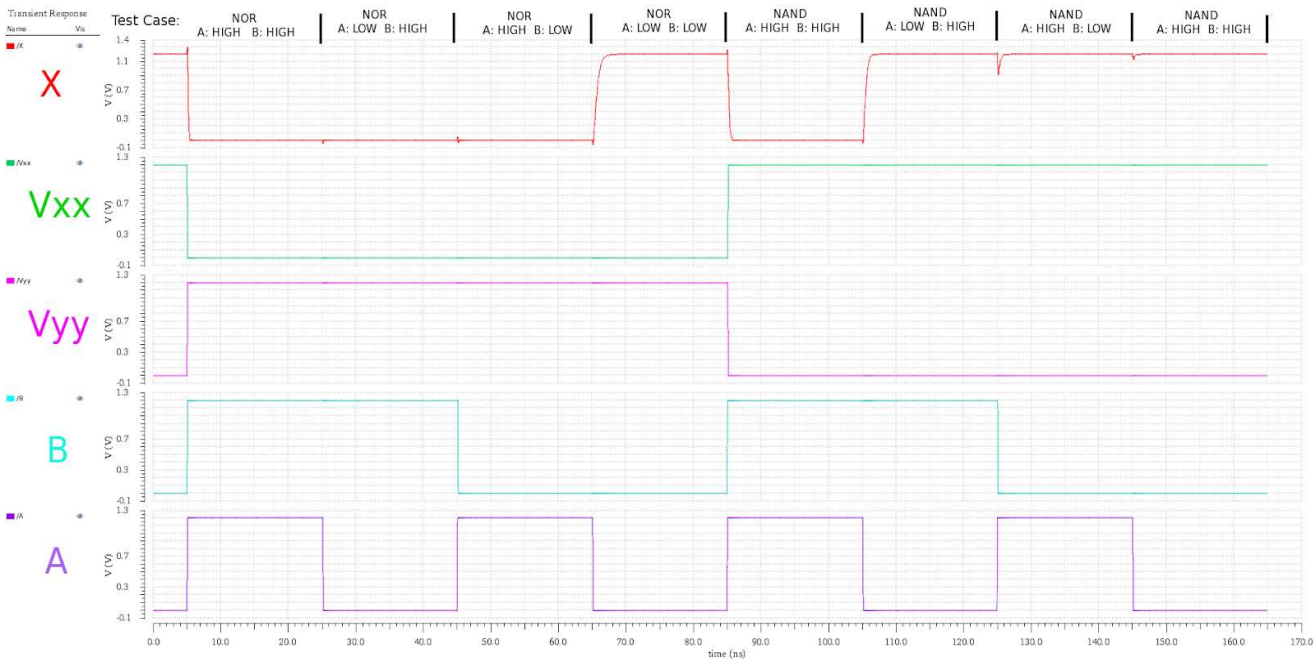
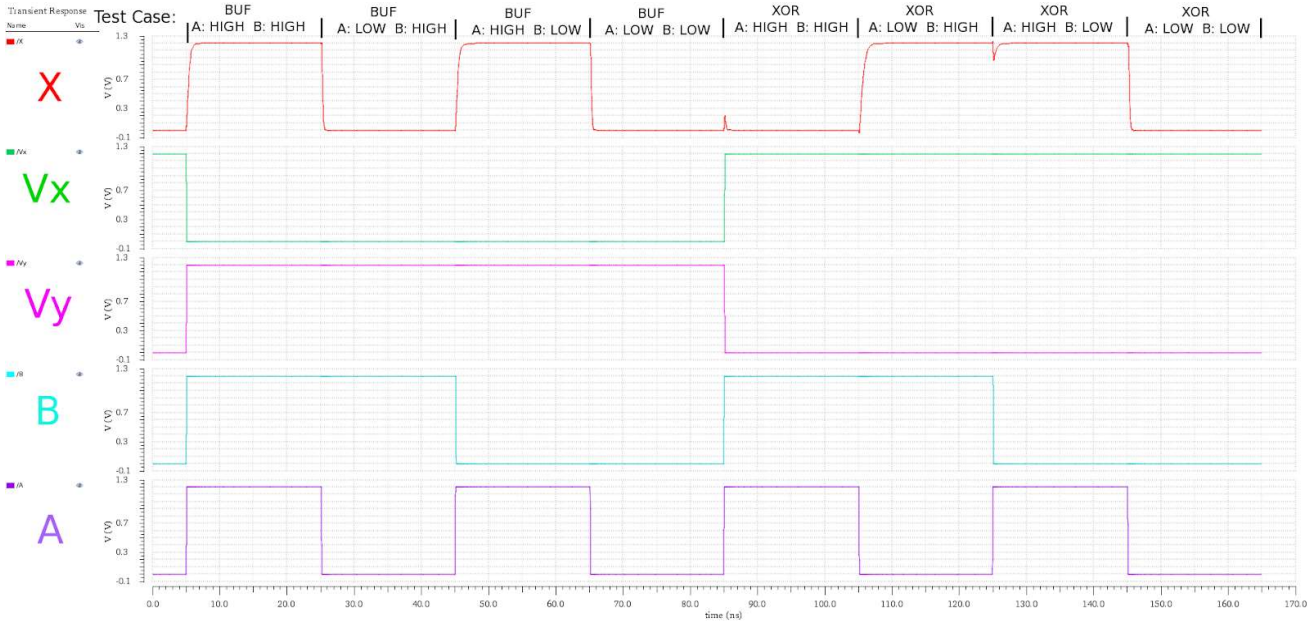
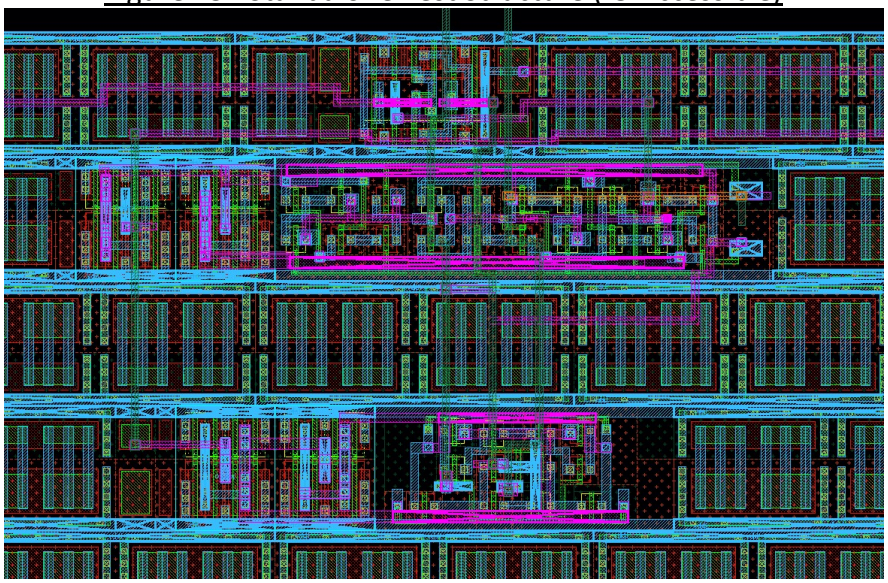


Figure 24: Transient simulation of NAND/NOR gate



The independent test structure was incorporated in the 2nd SoC's Tapeout. Below is a screenshot of the two cells within the AFTx05 layout which will functionally verify the design of the individual cells.

Figure 25: Standalone Test Structure (IO Accessible)



Polymorphic CRC Module (APB Accessible)

The APB peripheral was successfully incorporated with the other APB interfaces in the AFTx05 tapeout. The following is the register map of the APB Slave interface for the 32-bit Polymorphic CRC module:

Figure 26: APB Register Map for Polymorphic CRC module

Base Address: 0x80030000			
Offset	Register name	Read/Write	Reset value
`0x00`	NAND_NOR_CONTROL	R/W	`0x00000000`
`0x04`	NAND_NOR_INPUT	R/W	`0x00000000`
`0x08`	NAND_NOR_OUTPUT	R	`0x00000001`
`0x0C`	XOR_BUF_CONTROL	R/W	`0x00000000`
`0x10`	XOR_BUF_INPUT	R/W	`0x00000000`
`0x14`	XOR_BUF_OUTPUT	R	`0x00000000`
`0x18`	CRC_CONTROL	R/W	`0x00000000`
`0x1C`	CRC_CONFIG	R/W	`0x00000000`
`0x20`	CRC_STATUS	R	`0x00000000`
`0x24`	CRC_INPUT	R/W	`0x00000000`
`0x28`	CRC_OUTPUT	R	`0x00000000`

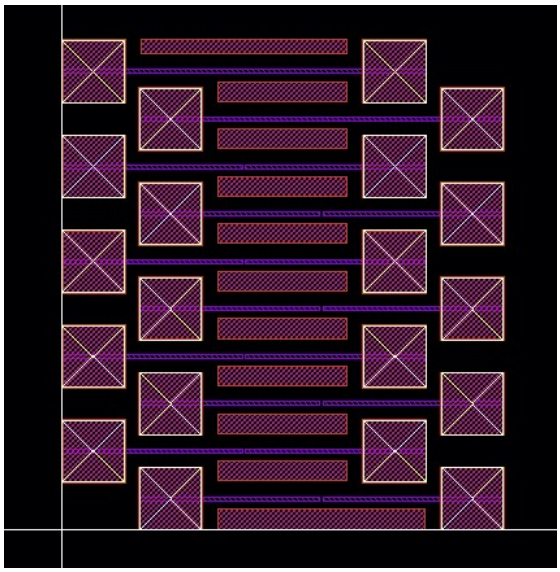
The following are some sample instructions for using this APB slave interface:

1. write 0x4C11DB3 (decimal = 79764915) to **CRC_CONFIG**
2. write 0x11122210 (decimal = 286401040) to **CRC_INPUT**
3. write 0x2 to **CRC_CONTROL**
4. write 0x42501202 (decimal = 1112543746) to **CRC_INPUT**
5. write 0x1 to **CRC_CONTROL**
6. keep reading **CRC_STATUS** until 0x01 is received (should take 32)
7. write 0x24FCC0 (decimal = 2424000) to **CRC_INPUT**
8. write 0x1 to **CRC_CONTROL**
9. keep reading **CRC_STATUS** until 0x01 is received (should take 32)
10. write 0x4222A65C (decimal = 1109567068) to **CRC_INPUT**
11. write 0x1 to **CRC_CONTROL**
12. keep reading **CRC_STATUS** until 0x01 is received (should take 32)
13. write 0x0000 (decimal = 0000) to **CRC_INPUT**
14. write 0x1 to **CRC_CONTROL**
15. keep reading **CRC_STATUS** until 0x01 is received (should take 32)
16. read **CRC_OUTPUT** (this is your checksum) it should be 0xDFBAF47C (decimal = 3753571452)
17. Repeat steps 2 -12
18. write 0xDFBAF47C (decimal = 3753571452) to **CRC_INPUT**
19. write 0x1 to **CRC_CONTROL**
20. keep reading **CRC_STATUS** until 0x01 is received (should take 32)
21. read **CRC_OUTPUT** (this is your checksum) it should be 0x0000 (decimal = 000000)

Layout for Electromigration Test Structures

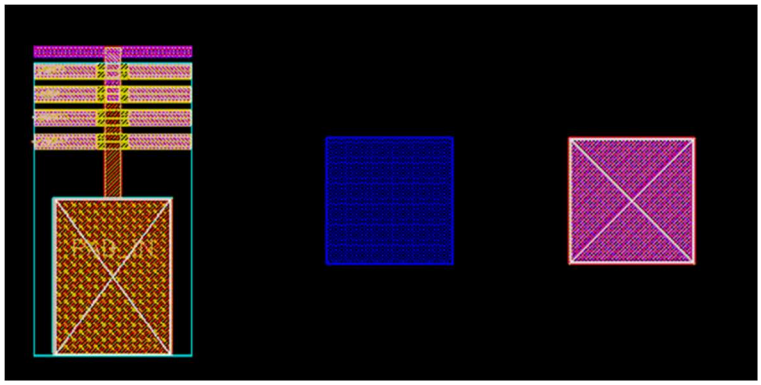
A set of ten different electromigration test structures were designed with variations in wire width, reservoir height, and void shape & depth.

Figure 27: Top View of a Single Set of 10 Electromigration Test Structures



There were three approaches evaluated when determining how to probe the electromigration test structures. Standard IO pads were first considered to deliver the current to the test structures; however, this would have severely limited the total number of test structures placed since each set of devices requires 20 pads. The second approach was to use the MPAD layer in the middle of the die as a pad to probe metal layer 5, as described in the 90nm FDSOI design kit manual. Unfortunately, The MPAD layer was unavailable for the wafer that our chips would be fabricated on. The final approach chosen was similar to the second, in that we would be using pads in the middle of the die to probe down to metal layer 5; however, the top layers MTK1 (Thick Metal), MRF1 (RF Metal), were used with vias going down to metal layer 5 (where the test structures reside). The white 'X' inside the square is the overglass cut.

Figure 28: An IO Pad, MPAD Layer, and Probe Pad Used (Left to Right)



Two of the devices will be utilized as control test structures which consist of normal wires (no voids or reservoirs), connected between two probe pads. One has a wire width of 5um and the other control structure is 2.5um in width. The following are screenshots of the non-control test structures (Note: In Figure 25, the test structures are the purple horizontal lines, and in the figures below the wires are shown vertically; this was done only for formatting purposes).

A portion of the test structures only had voids with the following depths ranging from 1um - 3um.

Figure 29: 2.5um-C1
(Void 1um Deep)

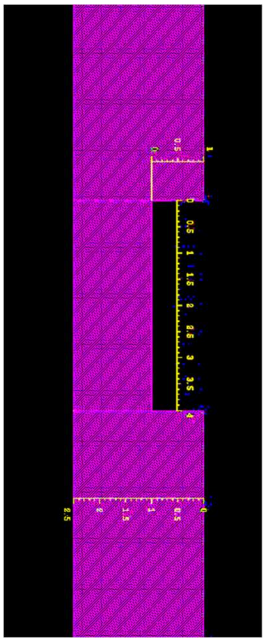


Figure 30: 5um-C2
(Void 2um Deep)

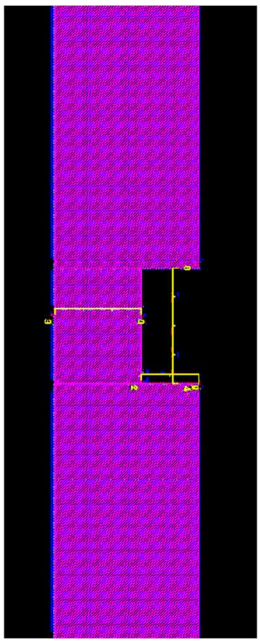
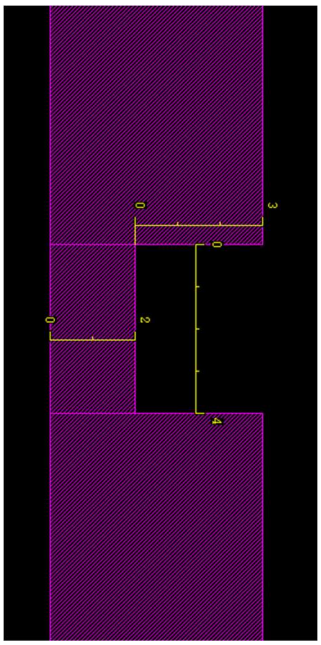


Figure 31: 5um-C3
(Void 3um Deep)



The following structures contained both rectangular voids and reservoirs.

Figure 32: 2.5um-CR1
(Void 1um Deep w/ Reservoirs)

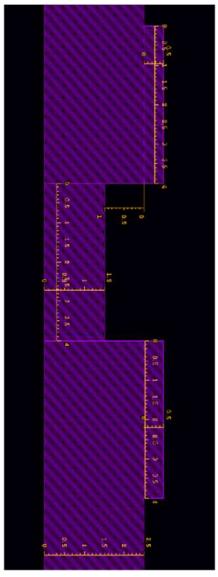


Figure 33: 5um-CR2
(Void 2um Deep w/ Reservoirs)

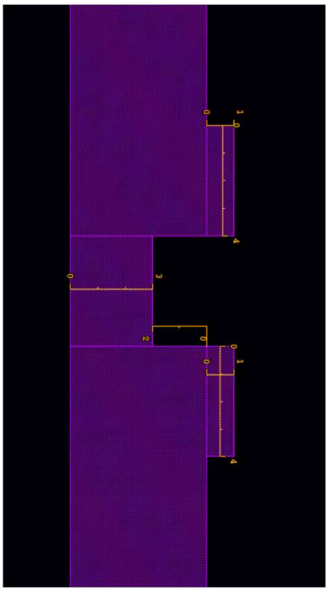
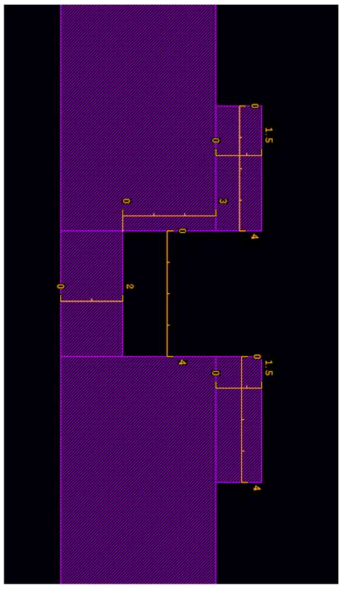


Figure 34: 5um-CR3
(Void 3um Deep w/ Reservoirs)



Two of the devices had triangular reservoirs and voids in a shape.

Figure 35: 5um-C2T
(Triangular Void 2um Deep)

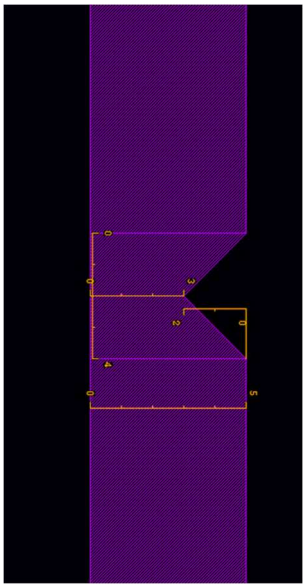
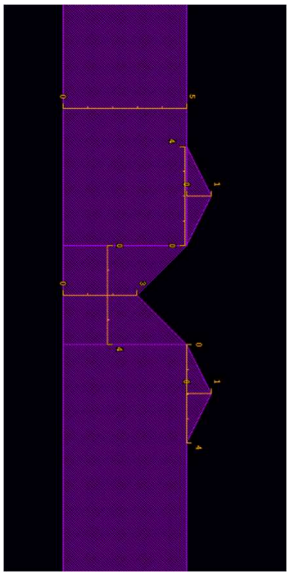
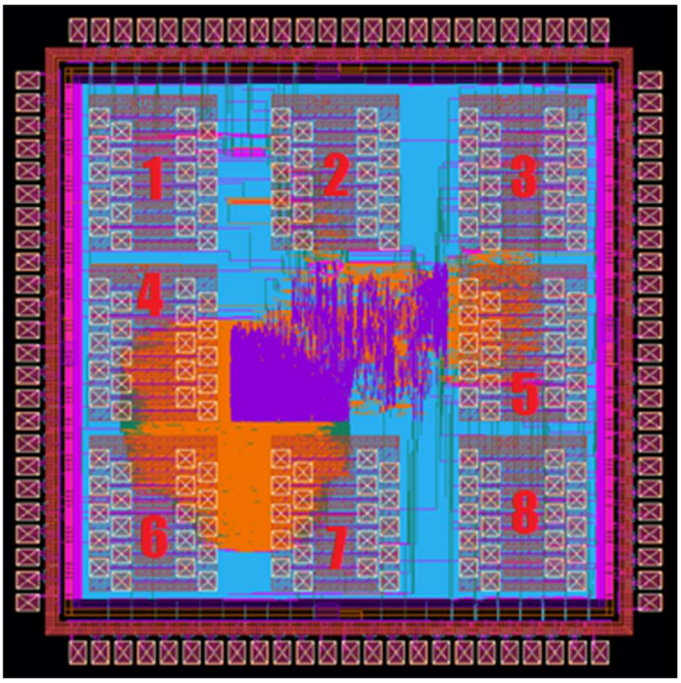


Figure 36: 5um-CR2T
(Triangular Void 2um Deep w/ Reservoirs)



A total of 80 electromigration test structures (8 sets) were able to be placed in the AFTx05 (shown in Figure 33). It is expected that 10 chips will be delivered to Dr. Bermel's group for testing within 6 - 12 months, for a sum of 800 test structures that will be used to build a dataset.

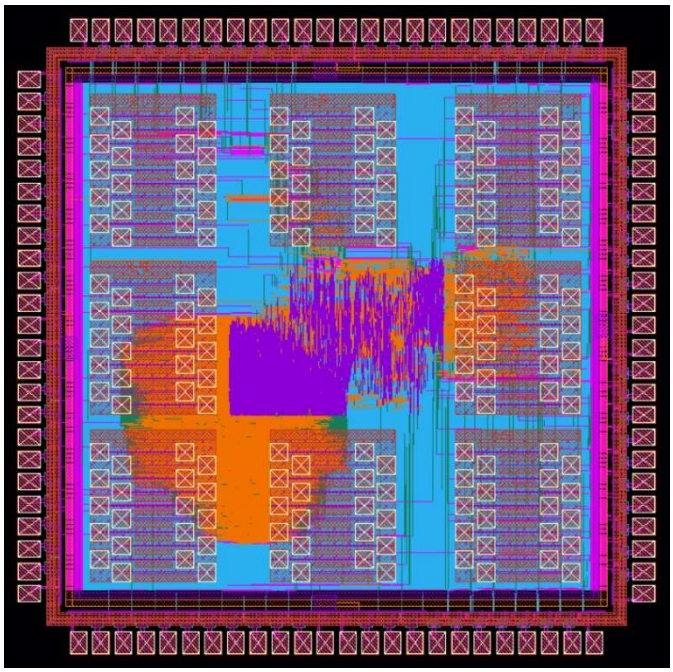
Figure 37: The 8 Sets of Placed Electromigration Structures



AFTx05 (2nd SoC) February 2020 Tapeout

Tapeout Submission

Figure 38: AFTx05 Layout Submitted for Tapeout

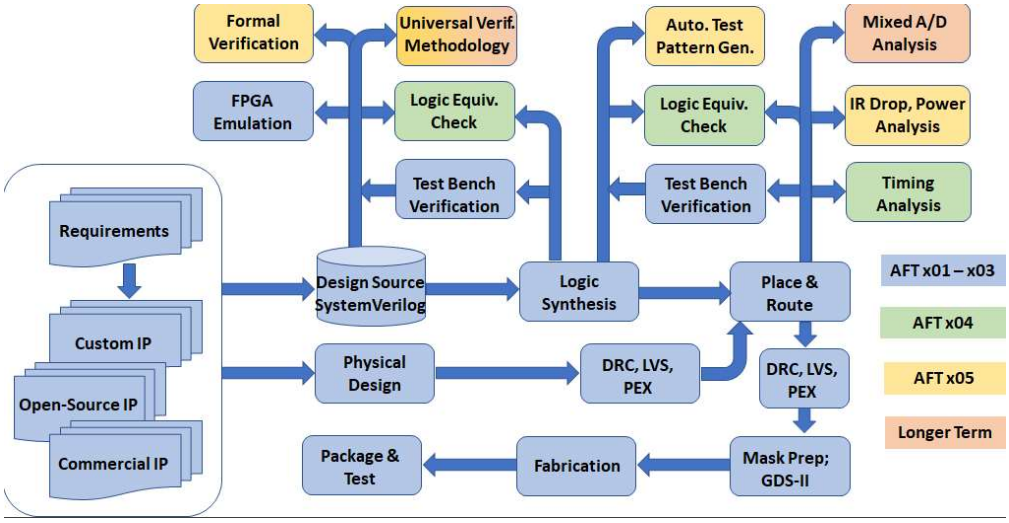


The team completed tapeout of the AFTx05 SoC in February 2020; however, the chips will not be delivered for an estimated 6 - 12 months. The tapeout deadline was initially set for November 2019; however, an inaccuracy in the timing characteristics for the standard cell's spice models was discovered which resulted in a tapeout delay until MIT Lincoln Labs updated the spice models. Testing and verification of the physical chip could not be conducted before the end of July 2020. However when the chips are delivered they will be tested with both functional and scan test vectors at NSWC Crane.

AFTx05 Verification and Design Flow

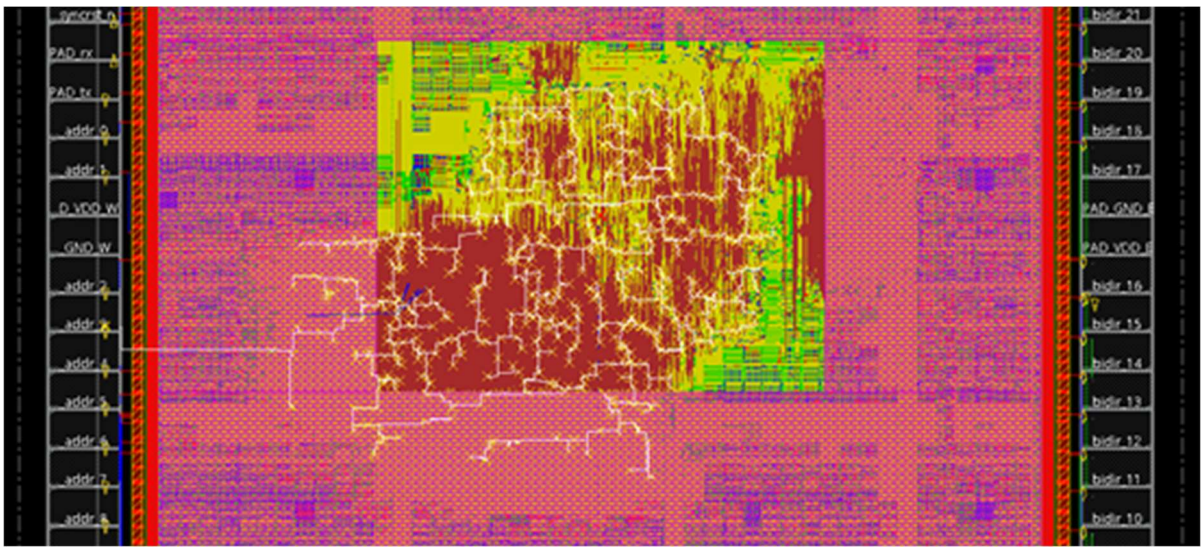
In this design cycle, Universal Verification Methodology (UVM) was pursued by the verification team, as suggested by Matt Sale from NSWC Crane. The AHB-APB Bridge, UAT debugger, as well as the GPIO were able to be signed off in time for the AFTx05 tapeout. In order to address the knowledge gap most of our verification members faced while trying to build their environments, a scaled down tutorial was made.

Figure 39: Design Flow used for AFTx05



The design flow scripts were updated to be compatible with the December 2019 release of the MITLL 90nm FDSOI PDK. The most significant change was the inclusion of the IO pad's liberty files, which enabled the IO pad frame to be created during place and route; it was previously created in Cadence Virtuoso and treated as an analog macro during PnR. Additionally the AFTx05 mapped netlist required post-synthesis modifications to work with IO pads, otherwise 20mm, unconstrained wires would be placed and introduce unacceptable timing delays. The PDK compatible scripts have been published along with the RTL source code at: https://github.com/Purdue-SoCET/AFTx05_Public .

Figure 40: An Example of a Single 20mm Long Wire (Highlighted in White)



Software Capabilities

The scripts used to facilitate the compilation of C/ C++ files into programmable 32 bit words of instructions and data which will run on the RISC-V core and interact with AFTx05's peripherals have been included in the published GitHub repository: https://github.com/Purdue-SoCET/AFTx05_Public/SW-LIBS . Additionally, future users will be able to take advantage of the SparCE machine learning architecture and generate the necessary skip tables to decrease the instructions, as well as the power, required to execute their programs. These scripts which generate the skip tables are included in the same repository as the previously mentioned C/C++ compiler scripts.

The software team created C libraries to facilitate a programmer's manipulation of the peripherals of AFTx05 with the following functions:

GPIO Functions:

- **gpio_enable_input**(unsigned int pins) - Sets the respective gpio 'pins' to be configured as inputs
- **gpio_read_input**(unsigned int pins) - Returns the value on the gpio pins given by 'pins'
- **gpio_enable_output**(unsigned int pins, unsigned int pin_outputs) - For the gpio 'pins', they are configured as outputs and the value is set to the corresponding 'pin_outputs' value.
- **gpio_set_output**(unsigned int pins, unsigned int pin_outputs) - For the gpio 'pins', the value is set to the corresponding 'pin_outputs' value

PWM Functions:

- **pwm_set_frequency**(unsigned int channel, unsigned int frequency) - Sets the period and the duty cycle for the 'channel' based on the given 'frequency'. By default, this sets the duty cycle to be 50%.
- **pwm_set_period**(unsigned int channel, unsigned int period) - Sets the 'period' for the 'channel'.
- **pwm_set_duty**(unsigned int channel, unsigned int duty) - Sets the 'duty' for the 'channel'.
- **pwm_disable**(unsigned int channel) - Disables the 'channel'.
- **pwm_enable**(unsigned int channel) - Enables the 'channel'.
- **pwm_set_active_high**(unsigned int channel) - Sets the active value to high for the 'channel'.
- **pwm_set_active_low**(unsigned int channel) - Sets the active value to low for the 'channel'.
- **pwm_set_align_left**(unsigned int channel) - Sets the active duty to be at the beginning of the period for the 'channel'.
- **pwm_set_align_center**(unsigned int channel) - Sets the active duty to be in the middle of the period for the 'channel'.

Timer Functions:

- **timer_enable**() - Enables the timer module counter.
- **timer_disable**() - Disables the timer module counter.
- **timer_set_output_action**(unsigned int channel, unsigned int output_action) - Set the 'output_action' for the specified 'channel'.
- **timer_set_input_capture_edge**(unsigned int channel, unsigned int capture_edge) - Set the 'capture_edge' for the specified 'channel'.

- **timer_set_prescaler**(unsigned int pre_div) - Set the prescaler ('pre_div') value for the timer clock
- **timer_set_output_compare**(unsigned int channel, unsigned int output_action, unsigned int interrupt_enable, unsigned int value) - Set the specified 'channel' as an output compare with the given 'output_action', comparing the 'value', and setting the 'interrupt_enable' (1 or 0).
- **timer_set_input_capture**(unsigned int channel, unsigned int capture_edge, unsigned int interrupt_enable) - Set the specified 'channel' as an input capture with the given 'capture_edge', and setting the 'interrupt_enable' (1 or 0).
- **timer_read_input_capture**(unsigned int channel) - Returns the timer value for when the input was captured.
- **timer_clear_interrupt**(unsigned int channel) - Clears the interrupt for the specified 'channel'.
- **timer_enable_cf**(unsigned int channels) - Forces the comparison for the specified 'channels'.
- **timer_enable_tov**(unsigned int channels) - Enables toggle on overflow for the specified 'channels'.
- **timer_disable_tov**(unsigned int channels) - Disables toggle on overflow for the specified 'channels'.
- **timer_read_count**() - Returns the current value of the timer.

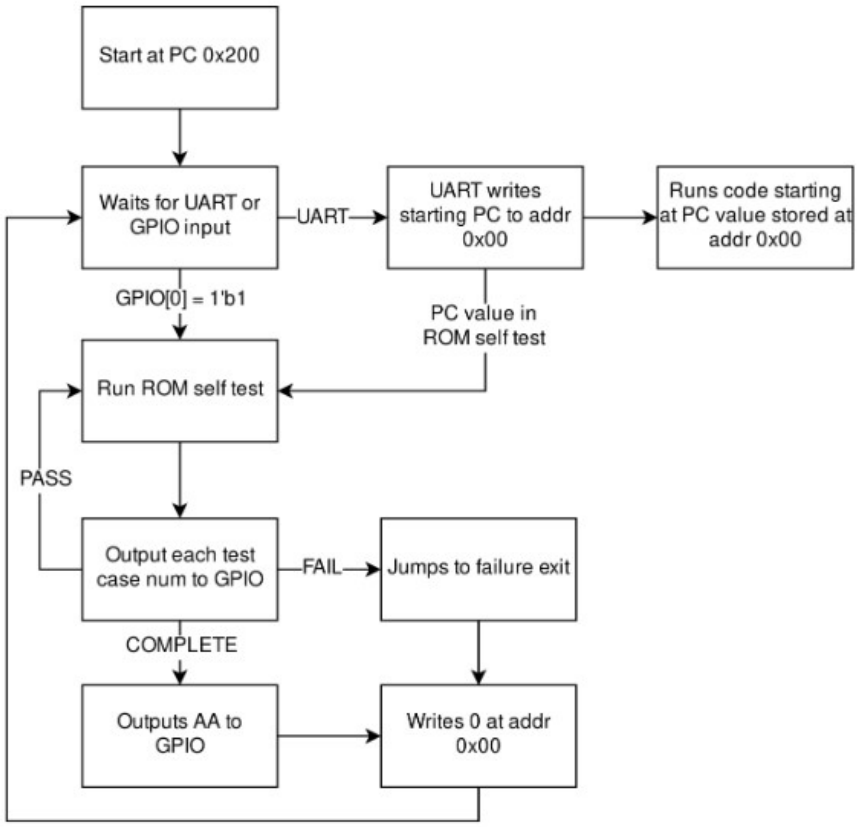
Poly CRC Functions:

- **crc_start**() - Starts the CRC generator with the current register setup.
- **crc_reset**() - Resets the CRC module.
- **crc_set_polynomial**(unsigned int polynomial) - Set the polynomial for the CRC generator.
- **crc_set_input**(unsigned int input) - Set the input for the CRC generator.
- **crc_ready**() - Returns 1 if the CRC generator is done or ready to begin.
- **crc_output**() - Returns the resulting value from the CRC generator.

How to Use AFTx05

Once the processor has been powered on / reset the following boot-up code is executed:

Figure 41: Boot-Up Flow Chart



If a functional test of the chip is desired, a self-test can be run by applying a logic HIGH signal to GPIO pin 0 and resetting the chip, either with the asynchronous reset pin or synchronously with the CORE_RESET UART command. If the test yields the correct, expected outputs, then the value of 0xAA will be output across the GPIO pins, with pin 7 as the most significant bit and pin 0 as the least significant.

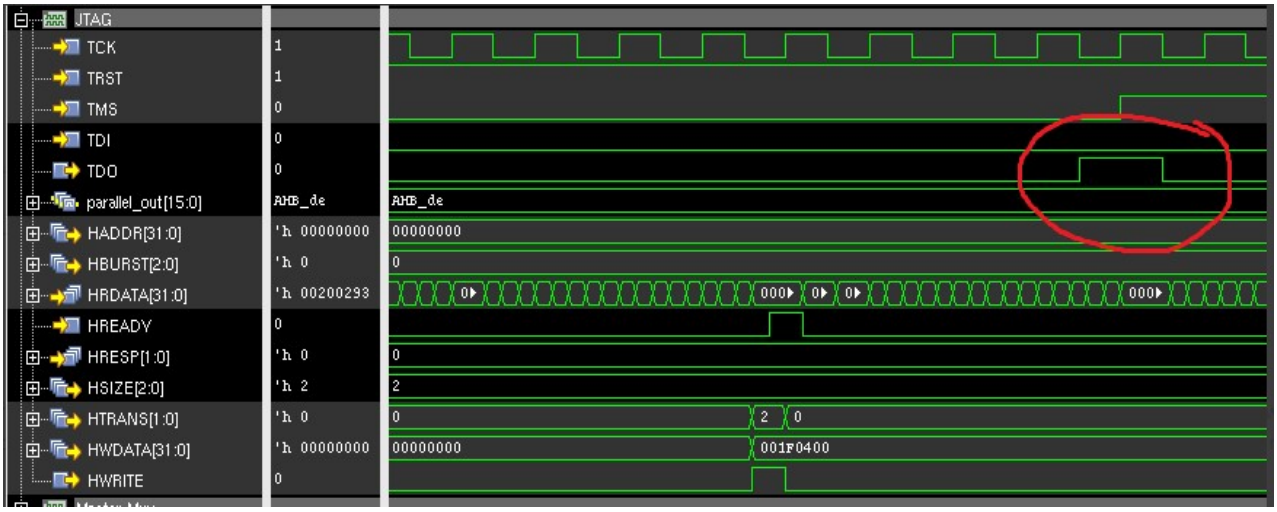
In order to run user code on AFTx05, the program must be written, through the UART, into either the on-chip SRAM (address range: 0x00008000 - 0x000083FF), or the external SRAM (address range: 0x00008400 - 0x001F_FFFF). Once the program has been written, the UART must write, to address 0x00000000, the location that the program counter should jump to: the first written instruction's address. If the result of the program's execution is not as expected, the UART can be used to debug and explore the memory, as well as the peripherals.

JTAG Interface

The JTAG module was not ready to be included in the AFTx05 tapeout; however, it will be an AHB bus master in the next chip iteration, AFTx06, to enable programming over the AHB bus. At a minimum, the current JTAG implementation requires at least 4 pins (preferably 5 to include the optional asynchronous reset) to operate successfully. Current testing has the JTAG running at about 10MHz, which allows for programming many times faster than the current M0 Debugger would permit.

Presented below is a waveform demonstrating a successful AHB write. It writes the value 0x001F0400 to ADDR 0x0 (located in BOTTOMRAM, note endianness is flipped). Notice also that the JTAG sends the intended 0 → 1 to indicate a successful write.

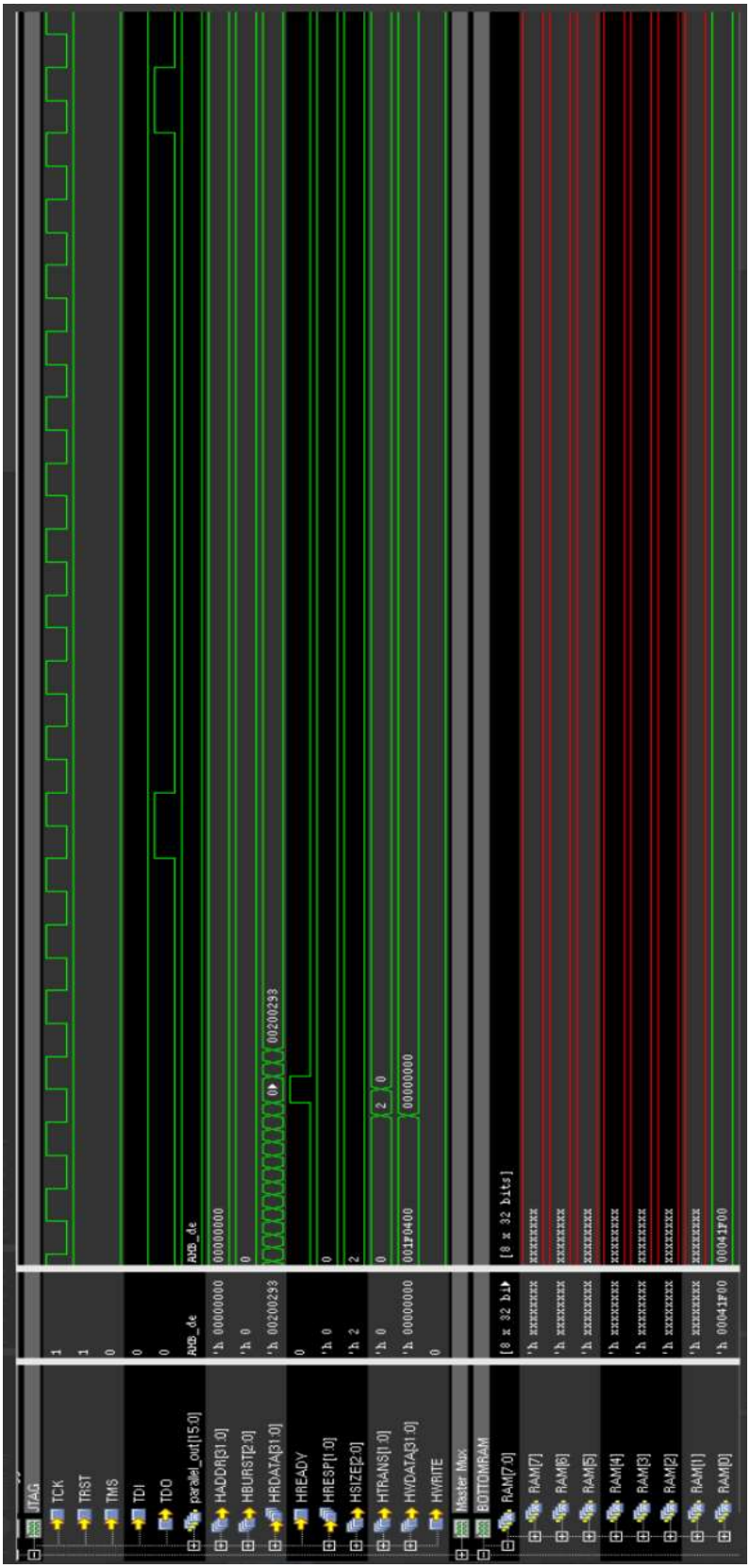
Figure 42: Simulation of JTAG Writing to BOTTOMRAM



Finally, in Figure 43 one can observe that the JTAG read back that same data. In TDO, you see that it sends the leading 1, followed by 10 cycles of 0s, followed by a 1 (indicating that it's reading back the 11th lowest bits of 0x001F0400).

Currently a UVM environment Is being utilized to verify the functionality with the newly included JTAG interface. This module will be taped out in November 2020, if no foundry delays occur.

Figure 43: Simulation of JTAG Reading from BOTTOMRAM

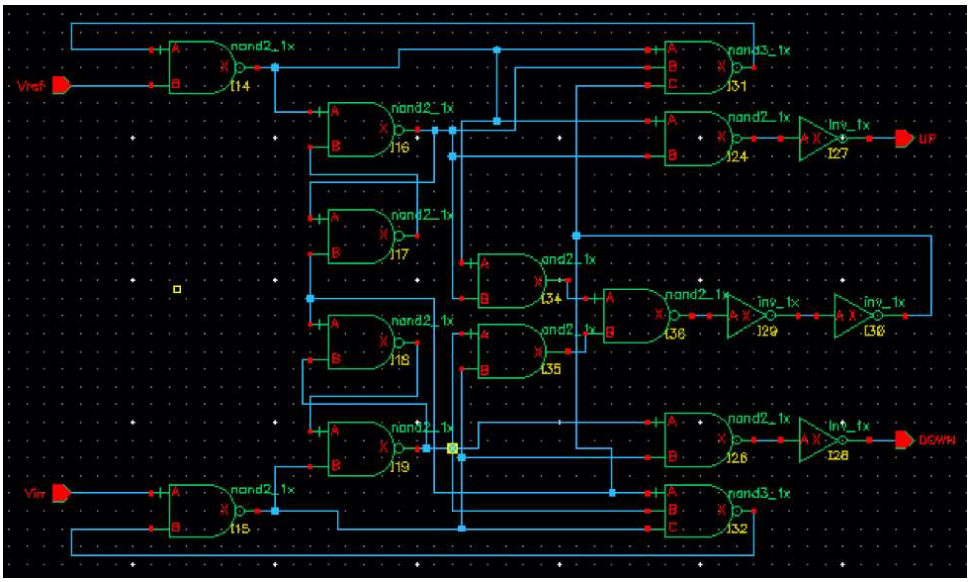


Phase-Locked Loop

Phase Frequency Detector

The phase detector topology chosen was a phase-frequency detector. It provides a wider lock frequency range over a phase-sensitive detector but requires the addition of a charge pump to output the correct voltage level. The PFD circuit is shown below.

Figure 44: Phase Frequency Detector Circuit

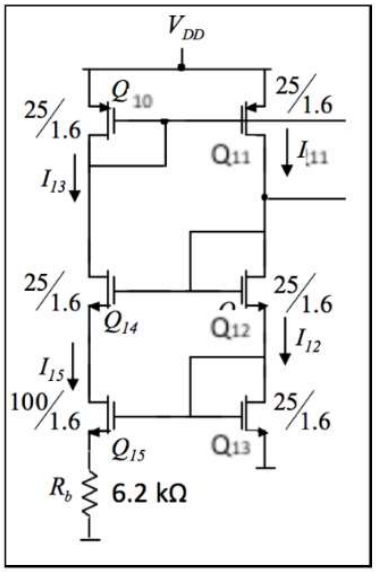


Charge Pump

This design was finished, and a current source of ~1uA was achieved. The following requirements were taken into consideration when designing the charge pump:

- The current Source must be built first to drive the charge pump. This current must be (~1-10 uA) or lower. The current source must have a low current due to the nature of the closed loop transfer function. A lower current used to drive the charge pump allows for smaller values used in the Loop Filter's components.
- A switched circuit must be built (comparable to the one referenced), with the switches controlled by the “UP” and “DOWN” signals of the Phase Detector.
- The currents from the current source and switch circuit must match almost exactly, any sort of mismatch can cause the charge pump to work incorrectly or not function at all. A mismatch in current sources between the UP and DOWN switches can cause phase error, so matching transistor sizes to keep drain currents equal is necessary.

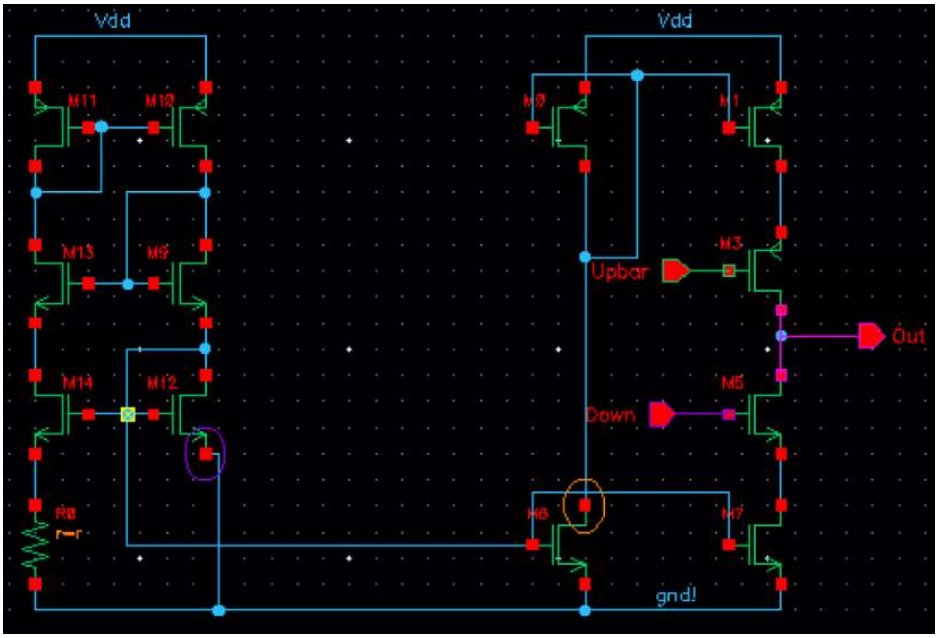
The following approach was taken to design the current source (The non-constant values on the equations to the right were chosen as shown on the circuit to the left):



$$g_{m13} = \frac{2\left(1 - \sqrt{\frac{(W/L)_{13}}{(W/L)_{15}}}\right)}{R_b}$$

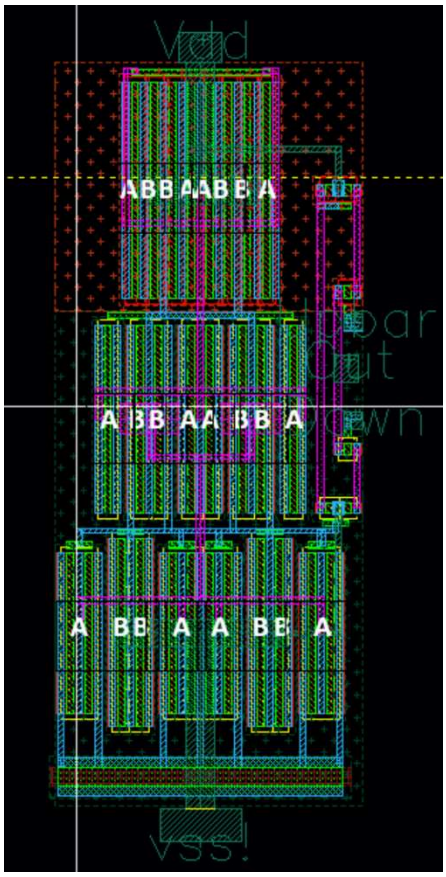
$$I_{13} = \frac{g_{m13}^2}{2\mu_n C_{ox}(W/L)_{13}}$$

Figure 45: Charge Pump Schematic



The charge pump has various current mirrors present which made having the same threshold voltages for the transistors desirable. The ABBA|ABBA common centroid technique was implemented for this layout. The input pairs/current mirrors were split into equal parts and arranged as shown in Figure 39.

Figure 46: Charge Pump Layout



Voltage Controller Oscillator

The Voltage Controlled Oscillator is used to set the output frequency, as well as range of frequencies achievable by the PLL. The design used was a current starved oscillator, which allows the frequency of the oscillator to be controlled using the Loop Filter's output. The design for the oscillator has various transistor sizes to provide the drain currents and parasitic capacitances required to set the delay of the circuit to specification. This design establishes a drain current of ~5uA and a frequency of 10MHz. The frequency gain, which is described as the change in frequency over the change in control voltage, was around 259.6MHz/V.

Considerations taken while designing the VCO:

- Wanted the output frequency to match the input frequency (10MHz) at a control voltage of ~500mV. This is to allow for tunable frequency in both directions.

- We also want a very high frequency gain, described as the change in frequency over the change in control voltage. This should be anywhere near the magnitude of hundreds of MHz/V or GHz/V to allow a wide tuning range.

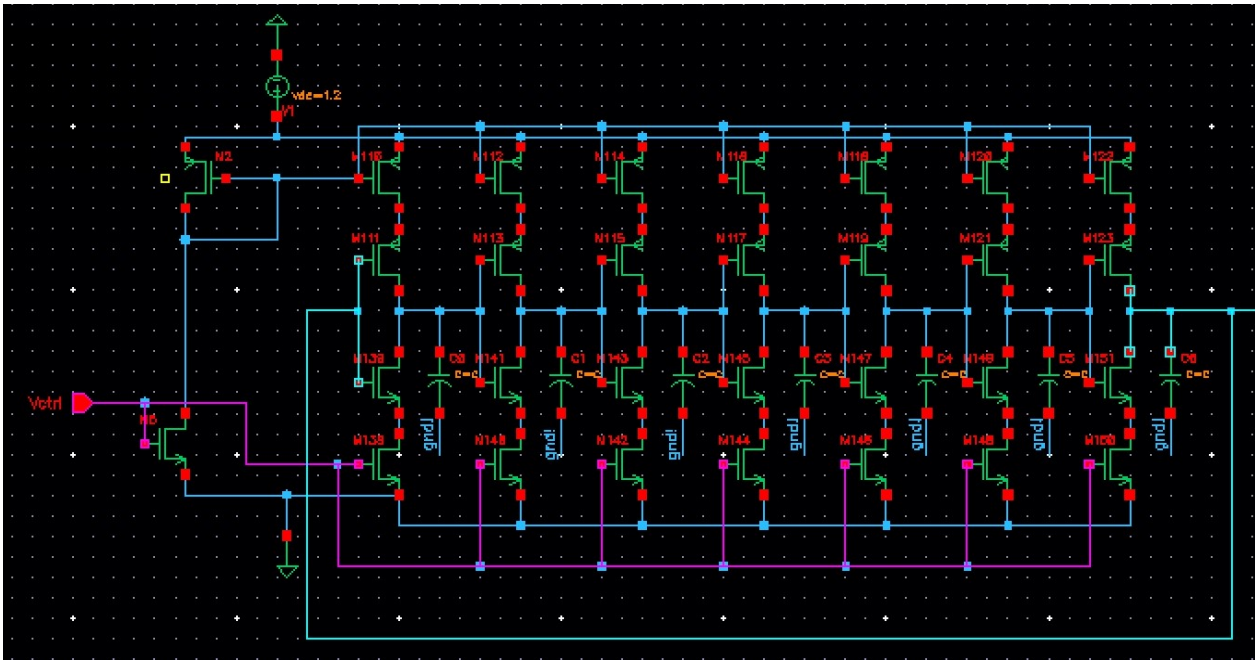
The VCO design was guided by the following equations:

$$f_{osc} = \frac{1}{N(t_1 + t_2)} = \frac{1}{N \cdot C_{tot} \cdot VDD}$$

$$C_{tot} = C'_{ox}(W_p L_p + W_n L_n) + \frac{3}{2} C'_{ox}(W_p L_p + W_n L_n)$$

The only variable required to solve for was the number of stages of inverters. In order to achieve an output frequency of ~65MHz, a total of 43 stages was required, with no load capacitance between them.

Figure 47: Voltage Controller Oscillator Schematic



The "frequency gain" of the VCO can be tested by applying a changing Vctrl input from 0-1.2V with a step size of 0.1V. The output looks like the following:

Figure 48: Frequency Vs. Time [ns] Demonstrating the Range of Stable Frequencies

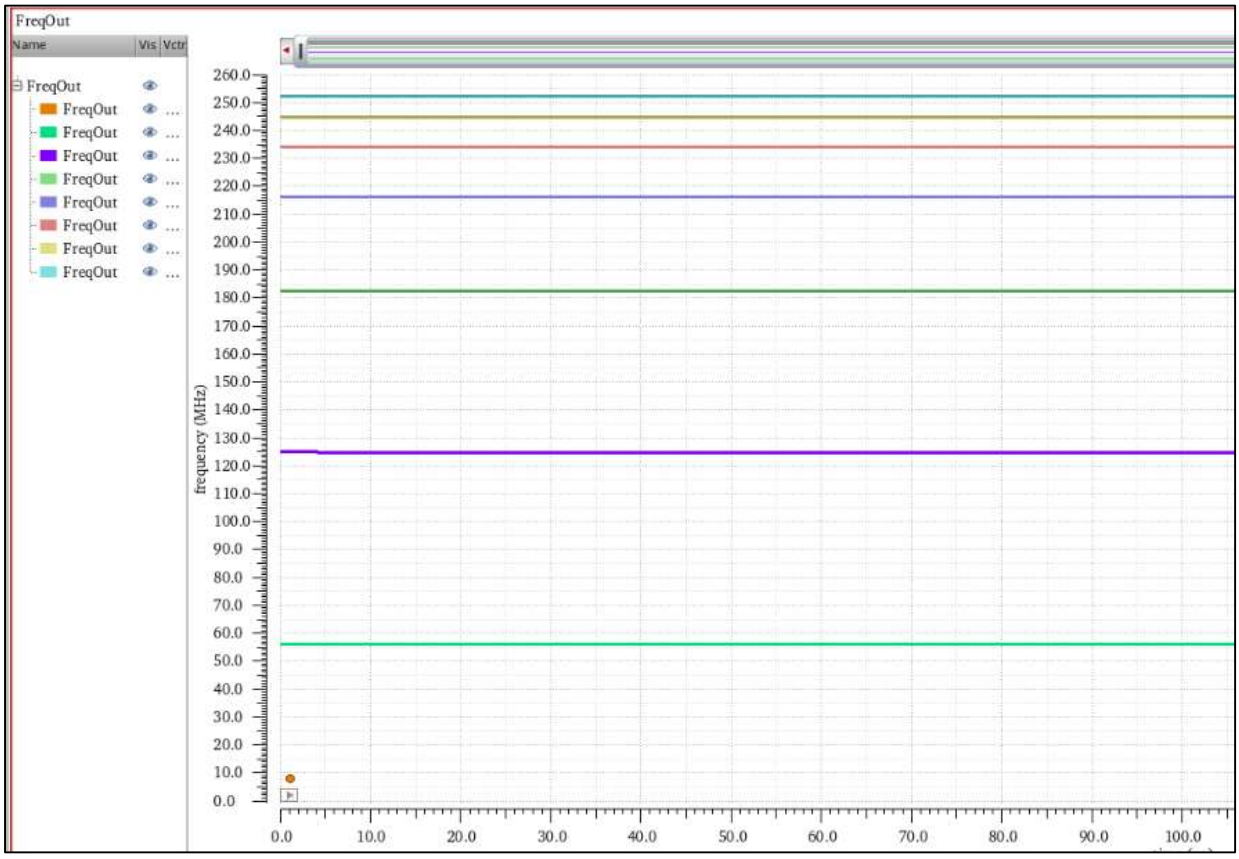
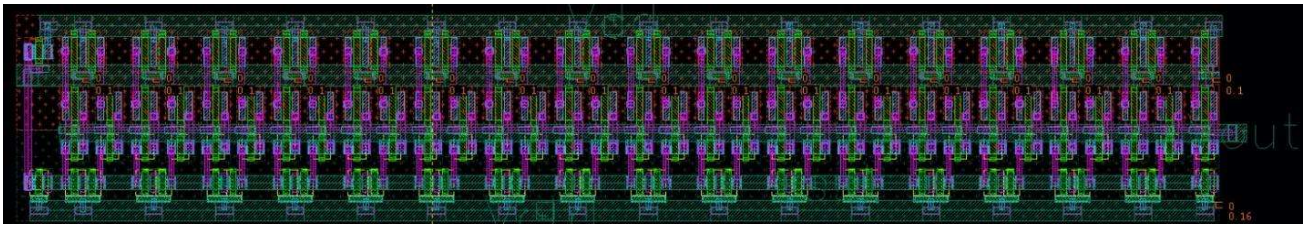


Figure 40 shows the frequency at 1.2V is ~250MHz. The frequency at 0.4V is ~8.5MHz. Using this we calculated the frequency gain of the VCO ($\Delta f/\Delta V$).

The layout for the VCO is very wide, but not very long. It would be better to apply a common centroid technique here; however, with a total of ~166 transistors total (without splitting them), this would be a difficult task. It becomes a challenge since the transistors are already at minimum sizing and couldn't be able to be split further. Consequently, the VCO was designed to be as compact as possible to minimize parasitics; however, the common centroid technique was not utilized for this component.

Figure 49: Voltage Controlled Oscillator Layout



Loop Filter

The Loop Filter was designed last and used to establish the stability of the closed loop transfer function and smooth out the output signal. This was designed to be a 2nd order Loop Filter, with the first resistor and capacitor utilized to define the pole/zero for stability, and the 2nd capacitor to smooth the signal & reduce jitter. The final values for these components are unrealistic in terms of size for a 32KHz -> 10MHz application, but are reasonable for 2.1MHz -> 10MHz.

When designing the loop filter, the following open loop transfer function was referred to:

$$G(s) = (K_{vco}/s)I_{cp}F(s)/M$$

- **Kvco:** Frequency gain of the VCO. This can be calculated as the change in frequency over the change in control voltage input into the VCO.
- **Icp:** This is the magnitude of the current from the current source of the charge pump.
- **F(s):** Transfer function of the Loop Filter.
- **M:** Number of Frequency Division.

While the Voltage Controlled Oscillator and Charge Pump can be built independently and optimized, the Loop Filter required to be tuned along with the final components of the PLL. In order to break this down into a set of more specific values, the following requirements were outlined for the Loop Filter:

$$H(s) = \varpi_n^2 (1+ s/\varpi_z) / (s^2+2s\zeta\varpi_n + \varpi_n^2)$$

where

$$\varpi_n = \text{natural freq} = \text{sqrt}(K_{vco}I_{cp}/MC_1)$$
$$\varpi_z = \text{stabilizing zero} = 1/RC_1$$
$$\zeta = \text{damping} = (RC_1/2)*\text{sqrt}(K_{vco}I_{cp}/MC_1)$$

- **Zeta** must be $0.45 < x < 1.4$.
- **C1 is** typically the largest but must range from ~10-400pF due to size constraints.
- **Natural Frequency** can be set to <~1/20 of the reference frequency, and is typically 1MHz -> 20MHz, which creates a problem with our 32KHz reference.

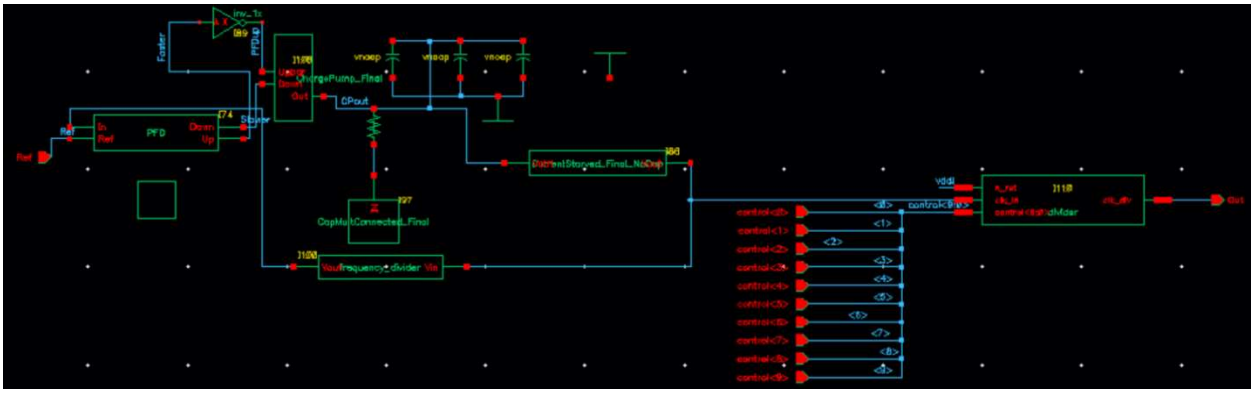
- **R1** is set with the equation relating Zeta to R1 and should be less than ~20KHz.
- **C2** is a smoothing capacitor and should be C1/50 < C1 < C2/20 to increase stability, but limit jitter.

These values are particularly large when dealing with a range from 32KHz input to 10MHz input, due to the nature of the first equation for the natural frequency. Even though Icp should be low and Kvco large, the gap causes the 1st capacitor to be in the magnitude of nanoFarads, rather than picoFarads/femtoFarads which are more acceptable.

Due to the size of the C1, the layout for the loop filter was done simultaneously with the final layout of the PLL in order to fit everything into a compact square.

Full Implementation

Figure 50: Schematic of Phase Locked Loop



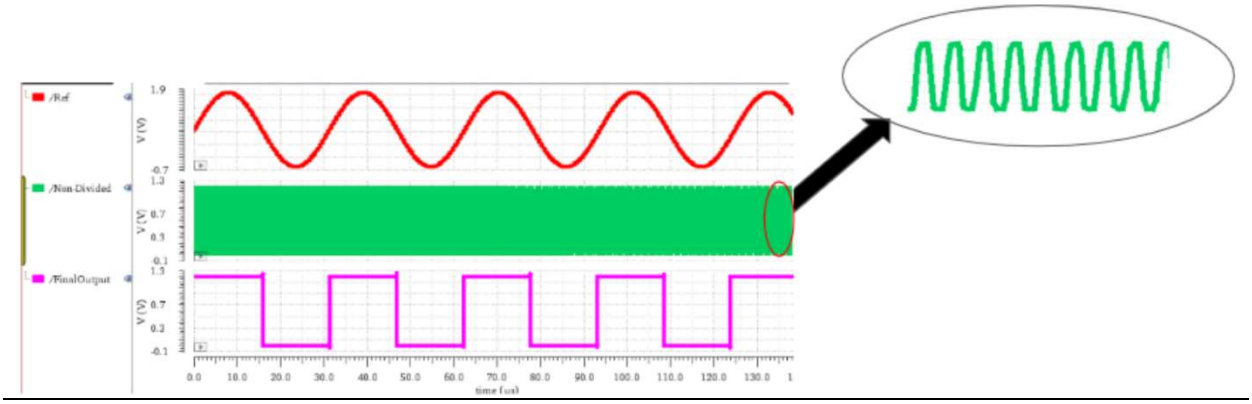
The Phase-Locked Loop will be included in the next chip fabricated with MIT Lincoln Labs; the full implementation in shown Figure 42.

Comments about the design of the Phase-Locked Loop's full implementation:

- The transistor circuit after the PFD is the charge pump.
- The loop filter capacitors are placed as 3 in parallel because they are separate in the layout, and does not cause the circuit to behave any differently.
- The "decap" in the bottom left is a symbol to recognize the array of decoupling capacitors placed in the layout of the PLL.

From an input reference frequency of 32kHz, the following signal was generated:

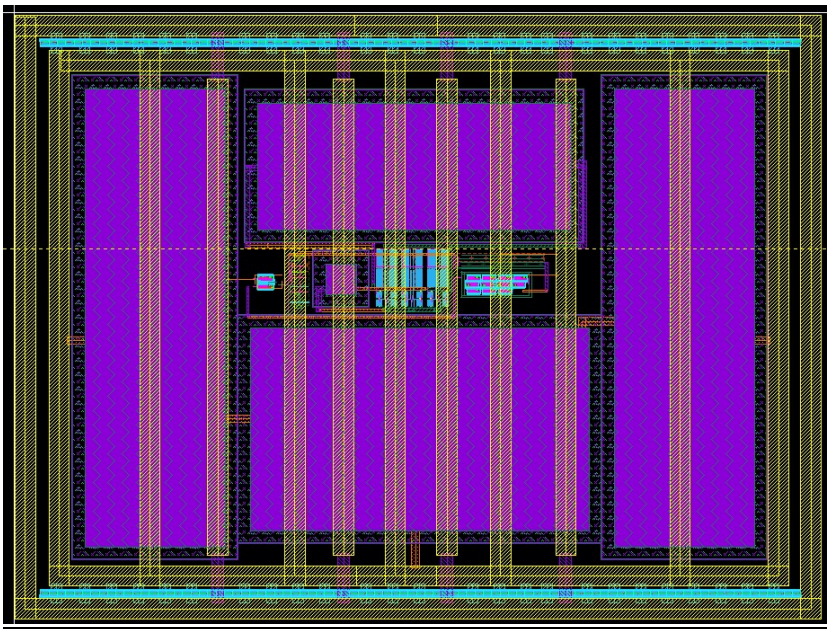
Figure 51: Waveforms for Vref, Non-Divided Output, and Final Output



In green is the ~10MHz (8MHz) signal that was output by the Phase-Locked Loop. The red is the input reference signal of 32kHz, and the purple is the frequency divided output signal to be input to the Phase Frequency Detector. As can be seen in these plots, the PLL produces a steady, nearly sinusoidal waveform, at much higher frequency than the reference signal.

When developing the layout for the top level PLL, the design was made to be as square in shape and compact as possible. In order to do this, C1 from the loop filter had to be split into 3 sections, and the rest of the PLL was placed in the middle of those sections.

Figure 52: Layout of the Phase-Locked Loop



Lock Speed

Lock speed can be determined by plotting the frequency of the output from the frequency divider and observing at what time the signal stably reaches the reference frequency. This lock speed is slower with increase with parasitics, which is why simulations using the PEX netlist yields a slower lock speed than the ideal simulation. To estimate the locking speed of the PLL, the following equation was used to calculate a locking speed of around a hundred microseconds.

$$1/(\zeta * \omega_n)$$

Figure 53: Ideal Simulation

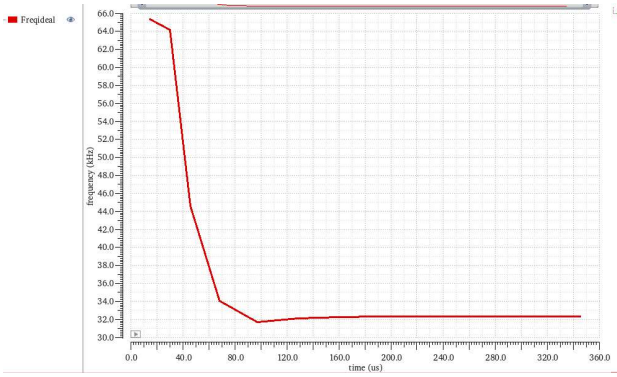
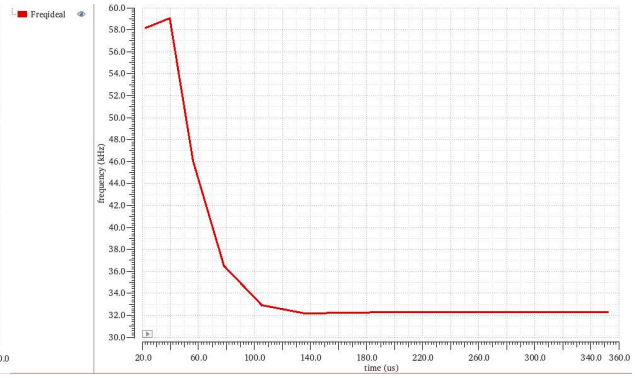


Figure 54: Simulation with Parasitics



The ideal simulation reaches ~32kHz in around 110us, and the parasitic simulation takes around 140us. Considering the target clock frequency is 20ns, the parasitic simulation's lock speed was deemed acceptable.

Jitter

Jitter is a measurement of the variability for how close the output clk rises compared to the reference signal. To get an actual measure of jitter, the average was taken of the time between these rising edges. Below are the jitter outputs for the ideal and parasitic simulations respectively.

Figure 55: Ideal Jitter Simulation

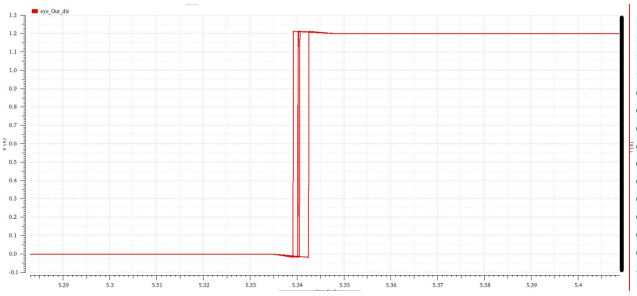
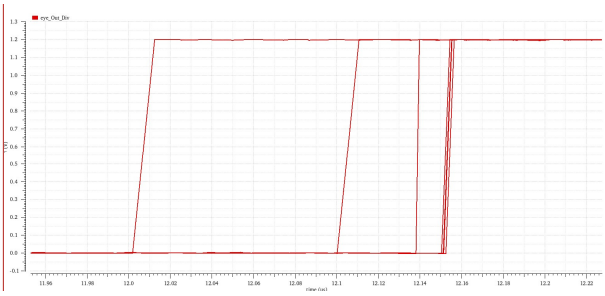


Figure 56: Jitter Simulation with Parasitics



The jitter for the ideal simulation was measured to be ~0.83ps and for the parasitic simulation, ~37.91ps. While this seems like a significant difference; however, it is negligible when taking into consideration that the reference clock period is on the scale of tens of microseconds (10^6 times larger) and the target output period is 20ns.

Platform-Level Interrupt Controller

The first revision of the PLIC was not functionally ready in time for the 1st Tapeout and development was put on hold until Spring 2020 when two students resumed its progression. This next revision is near completion in its integration with the rest of the SoC design and will be included in the next chip iteration, AFTx06. The following is a list of tasks for this project and their current statuses.

Figure 57: Table of PLIC Related Tasks and Corresponding Statuses

Task	Status
Discovered prior documentation and code regarding prior interrupt integration.	Complete
Composed a complete RTL of Control unit within priv to better visualize the categories of interrupts and exceptions.	Complete
Composed starter test benches to test the priv unit within RISC-VBusiness.	Complete
Completed skeleton file for the external interrupt Assembly file.	Complete
Expanded the pending and status registers of core to allow external interrupts.	Complete
Modified the combinational logic within the priv_1_11_control.sv file to allow external interrupts to be registered into the status register.	Complete
Verified the full functionality of the Assembly external interrupt file.	In progress; currently investigating uncertainties about the clear and swap signals.
Combined the functionality of RISC-VBusiness with SoCET Public.	Incomplete; integration with the rest of the RISC-V must be completed first.
Integrated the interrupt_controller module to Top level	Complete
Modified the previous error test case	Complete
Created interface for the interrupt controller module	Complete
Modified Top Level file to be able to arbitrate the request to interrupt controller	Complete
Documented the detail of each test case	Complete
Added extra test case to test the functionality under actual load	In progress; creating additional write and read corner cases

1.5 Key outcomes

1) Prepared students with experience in SoC design and as potential recruits for NSWC Crane.

This project was able to provide 95 students, from Spring 2018 – Summer 2020, with VLSI and chip building experience. Two members were recruited by NSWC Crane, and require minimal time to bring up to speed. Additionally we were able to open the eyes of our undergraduates to aspects of SoC development that they would have otherwise missed while working on their bachelors. The following are examples of these aspects. Most undergraduate classes don't offer detailed content and practice for working with EDA software; however, our design flow students get familiarized with these tools. It's common that a computer engineer's first job will be in UVM, and our verification students already possess the background knowledge and practice to build these environments. Not every student is able to take a compiler course; however students that join the software team are still able to gain experience working with the RISC-V toolchain. Most digital design courses offered at the undergraduate level are at the peripheral or processor scope, but working on the digital design team provides our students with experience working on a full scale SoC. Most undergraduate computer engineers do not get experience developing analog components which actually get fabricated, but our analog team offers the opportunity to build standard cells, ADCs, PLLs, and more to be included in a tapeout. We have bolstered our graduating students' qualifications to secure employment in the VLSI field, as well as the expertise they have to offer the projects they join. We have sent them off as engineers equipped with applicable experience and familiar with the demands of today's efforts to advance computational technologies.

2) Installed and tested a 90nm SOI design flow to enable design and fabrication.

Our team created design flow scripts and tutorials, to facilitate the design and fabrication of the AFTx04 and AFTx05 chips. The capabilities/ content of these tools are listed below:

- Logic Synthesis
- Mapped Functional Verification
- Logic Equivalence Check (LEC)
- Place and Route (PnR)
- Static Timing Analysis
- Post-PnR Functional Verification
- IR Drop and Power Analysis
- Design Rule Check (DRC)
- Layout Vs. Schematic (LVS)
- Parasitic Extraction (PEX)
- Automatic Test Pattern Generation (ATPG)

Additionally polymorphic gates were implemented with MIT Lincoln Labs' PDK. The characteristics of the FD-SOI technology, particularly when reverse biasing, allowed the gate's logical function to be controlled by the polarity of the power rails. These gates were non-symmetric and CMOS based; however, another version of the PDK will be made to implement the symmetric cells using Schottky-barrier transistors.

Evaluation of the September 2019 PDK revealed a couple of issues pertaining to the IO pads, which included a new bidirectional pad. The SPICE simulation of the bidirectional pad did not match the Verilog model provided. Moreover, signals going out from the core to the pad were inverted, while signals going from the pads to the core were not inverted. Both of these issues were resolved in the February 2020 update to the kit.

3) Taped out, fabricated, and tested 1st SoC design.

The AFTx04 SoC was taped out in August 2018 and the chips were delivered in September 2019. Some of the packaged chips were tested at NSWC Crane. Using functional test vectors with Crane's Advantest V93000 SoC tester, the fabricated chips were able to be functionally verified. It's important to note that a few of the chips that were put through the tester were reported to have a short on the UART's tx pin.

The AFTx04 chips will be mounted on PCBs and evaluated once the design is assembled. The PCB boards, and some of the components have already been delivered; however, the board cannot be assembled until lab space has been cleared with Purdue's developing COVID-19 prevention policies. The board will be soldered together and evaluated within the next 5 months.

This tapeout was able to provide our Spring 2019 members with the rare opportunity to create hardware verified digital and analog designs, as well as insight for the design flow steps required to fabricate an SoC which functions as intended. Moreover, the AFTx04 chip was a larger design than most which go through MIT Lincoln Labs. A large magnitude of DRC violations were discovered, due to the size of AFTx04, relating to tungsten via corrosion located on long wires with minimal connecting vias. The workaround used to meet the tapeout deadline was to insert buffers to break up these long wires, although this resulted in over 80% of the design consisting of buffer cells. This incentivized MIT Lincoln Labs to refine their process which increased their PDK's maximum wire length from 100um to 800um.

4) Taped out 2nd SoC design

The AFTx05 SoC was originally planned to be taped out in November 2019; however, MIT LL discovered a ~10% inaccuracy in their hspice models' timing characteristics. Consequently, the tapeout deadline was pushed to February 2020, which delayed our timeline for the distribution and evaluation of the fabricated SoC. While the physical hardware cannot be verified until the chips are delivered, it is worthwhile to note the additional verification measures implemented in this design compared to the 1st SoC. This iteration included some UVM signed off modules, scan chains with test patterns generated, and design reviews with engineers from both commercial and military teams such as Cisco Systems and NSWC Crane.

The PCBs which will host the packaged chips have been developed and are expecting to be delivered within the next few months. However, the AFtx05 chips' estimated delivery is sometime in Spring 2021. A portion of these packaged chips will go to NSWC Crane to be tested with their SoC tester, although this iteration of chips will be verified with both functional and scan test vectors. Ten of the unpackaged dies will be sent to Dr. Bermel's electromigration research group to probe the 800 test structures with high density currents and study the aluminum/ titanium ion migration with their thermal microscope.

This tapeout iteration provided our members with a new opportunity this time around: to center the focus of their curriculum required senior design projects to be SoCET based over the span of two semesters. Typically, the scope of projects which are feasible to finish in a semester are limited in terms of usefulness and quality. The students which have taken advantage of this opportunity were able to create the Phase Locked-Loop, the SparCE optimizations, the polymorphic gates, and the JTAG interface.

5) Released the open-source System Verilog code for a RISC-V based SoC to be made available for use by Crane NSWC and the broader academic community

The AFTx05 chip is available to download at https://github.com/Purdue-SoCET/AFTx05_Public . The intention of its publication is to serve both as a template for users new to SoC development and as a quick, easy-to-modify build for researchers in need of a RISC-V SoC to use with their research. Our design is currently being used, by another Purdue research group, to implement a value-similarity optimization with similar benefits as the SparCE architecture.

What makes this publication unique to other open-source RISC-V builds is the inclusion of design flow scripts. Permission has been obtained from Cadence Design Systems to include scripts for RTL simulation, synthesis, place-and-route, ATPG, and LEC. Additionally, we have made arrangements with MIT Lincoln Labs to archive the PDK used to tapeout AFTx05. This way anyone new to using EDA tools will be able to run the scripts, with minimal modification and no errors, to rebuild our SoC from its RTL to it's Post Place and Route layout with ATPG.

6) Collaborate with academic and industry teams' research ventures.

The SoCET team found opportunities to join efforts with Concertal Systems and Dr. Peter Bermel's electromigration research group. These collaborations were initiated to both expand the scope of available projects to our members and provide knowledgeable students to facilitate the efforts of these outside teams.

Most groups would be apprehensive to rely on the help of undergraduates with heavily funded projects, but our track record has demonstrated that our skilled members are able to make meaningful contributions. Our team was able to supply Dr. Bermel's group with the layouts for their electromigration structures, and also included these layouts in the AFTx05 tapeout, eliminating the requirement for the group to seek and pay a foundry to fabricate their structures.

Furthermore, our students have been exploring Concertal System's tools as an alternate method for integration of IP blocks produced by the team. Concertal provides an additional avenue for open-source dissemination of work from the team and facilitates the re-use of our IP in other SoC designs. Concertal has helped us ensure that our IP is in a form which future users can easily repurpose for their own design's requirements. We have worked with Concertal to create a prototype design, using open source IP from this project and Concertal tools, to create an SoC which will be demonstrated with brushless motor control applications.

2. Products

GLSVLSI '19 Paper: System-on-a-Chip Design as a Platform for Teaching Design and Design Flow Integration [5][5]

This paper claims that most undergraduate VLSI curriculums are taught in unconnected pieces which complicate the creation of manageable, semester-long projects that reflect the microelectronic design experience. The solution explored is the System-on-a-Chip Extension Technologies (SoCET) group: an undergraduate design group modelled after industry, aiming to emphasize the integration and cooperation required across multiple disciplines in SoC development. This is a multi-semester project that gives undergraduates the rare opportunity to iteratively improve, prototype, fabricate, and test an SoC. Students on the team are divided into smaller groups which focus on specific SoC design tasks. These tasks are connected by a near industry grade design flow forcing students to address system and design flow integration issues. This framework enables students to approach engineering as an integrative process and learn the relationships between seemingly separate disciplines.

[5] J. Covey and M. Johnson, "System-on-a-Chip Design as a Platform for Teaching Design and Design Flow Integration," in *GLSVLSI '19: Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI 2019, Tysons Corner, VA, USA, May 9-11, 2019,* pp. 249–253.

3. Training and Professional Development

Workforce development is one of this project's core goals. Through SoCET, 95 students gained experience in VLSI & SoC design over the duration of ASSURE task 2.5. These students possess skill sets which will facilitate their transition into industry with minimal time required to be brought up to speed at their next place of work. These skill sets vary according to which of the five subteams a student has joined. 1) The digital design team writes the System Verilog source code that describes the hardware of the SoC. 2) The software team develops methods and code to test & demonstrate the functionality of the SoC design. 3) The physical & analog design team is responsible for the physical implementation of the digital and analog design. 4) The verification team develops the UVM (Universal Verification Methodology) framework for functionally verification. 5) The post silicon team handles the packaging, PCB test platform design, and fabricated SoC testing.

The following is the list of students which have made significant, documented contributions to the team from Spring 2018 through Summer 2020, as well as a brief description of their accomplishments.

	Student Name	Term of Participation	Summary of Accomplishments
1	A J Gregorian	Fall 2019 - Spring 2020	● Compiler compatible with SparCE optimizations
2	Aditya Chakraborty	Fall 2018	● First revision of DAC
3	Aeson Akhras	Spring 2019	● ALU
4	Andrei Aldea	Fall 2018	● PCB design for IC test beds
5	Ankeet Annapur	Spring 2019	● Assisted UVM team
6	Atif Niyaz	Fall 2019 – Spring 2020	● Compiler compatible with SparCE optimizations
7	Ben Dyer	Spring 2020	● Worked with Analog team
8	Bharath Mukundan	Fall 2018	● Assisted UVM team
9	Blake Wilson	Summer 2018 – Fall 2019	● Software team lead ● ROM Creation Script ● Build environments (simulation, syn, pnr)
10	Bojun Huang	Fall 2018	● GPIO test circuit (PCB) ● USB to UART converter schematic

11	Brady Malcomson	Spring 2020 – Summer 2020	● Currently being trained
12	Brian Graves	Fall 2019	● Polymorphic XOR/BUF gate
13	Brandon Wu	Spring 2020	● Assisted Compiler team
14	Brian Helfrecht	Fall 2019 – Spring 2020	● Compiler compatible with SparCE optimizations
15	Brian Ko	Spring 2020 – Summer 2020	● FPGA emulation of AFTx05
16	Chan Weng Yan	Spring 2019 – Fall 2019	● NVDLA research ● SparCE optimization
17	Chandan Bothra	Summer 2019 – Fall 2019	● ATPG for static and dynamic Faults ● Initial Concertal Work
18	Chris Priebe	Fall 2019 – Summer 2020	● Updated and reorganized documentation ● Intro to SoCET student ● Updating SPI RTL
19	Chun Tao	Summer 2019	● Studied OpAmp Design
20	Cole Nelson	Fall 2019 – Summer 2020	● Digital team lead ● JTAG
21	Cole Stecyk	Fall 2019	● Updated UART UVM environment
22	Dali Lai	Spring 2020 – Summer 2020	● Updated Oscillator
23	David Castley	Spring 2018	● Inverter Virtuoso/ Calibre tutorial ● Ring Oscillator
24	Dotun Akinnola	Summer 2019	● Second revision of SPI RTL
25	Enes Shaltami	Spring 2020 – Summer 2020	● Interrupt Controller Integration ● Fixed and integrated FPU
26	Evan Miller	Spring 2019	● Assisted UVM team
27	Evelyn Ware	Fall 2019 – Spring 2020	● Phase Locked Loop
28	Fred Owens	Fall 2019 – Spring 2020	● JTAG
29	Geoff Cramer	Spring 2018	● APB UVM driver and sequencer
30	Haoming Duan	Summer 2020	● AHB Bus Master redesign
31	Himank Kothari	Spring 2018	● Assisted PCB team

32	Hsin-Han Yu	Fall 2018	● UVM scoreboard
33	Huy Minh Tran	Fall 2019 – Spring 2020	● Script for USB communication with FT232R chip ● Intro to SoCET student ● Updating SPI RTL
34	Hyunoh Song	Fall 2019	● Third revision of DAC
35	Isaiah Grace	Spring 2019 – Fall 2019	● APB slave interface for Poly CRC module ● Studied Floorplanning ● Learned formal verification
36	Itsuki Sakamoto	Spring 2019	● First revision of Timer UVM
37	Jacob Covey	Fall 2018 - Summer 2020	● Research Assistant ● Coordinated overall operation of team ● Physical design team lead ● Second revision of DAC ● Estimate for IR Analysis of AFTx04 ● GLSVLSI 2019 design flow paper ● Wire Bond Specs for AFTx04 ● Scan Chain insertion
38	Jake Stevens	Spring 2018 – Summer 2020	● Digital Lead ● One of two designers for RISCV core.
39	James Zampa	Fall 2019	● Machine learning benchmark environment
40	Jing Yin See	Spring 2020 – Summer 2020	● Intro to SoCET student ● ADC
41	Jingchen Lei	Fall 2018	● APB UVM Scoreboard ● Tutorial for account setup
42	Joe Nasti	Spring 2018 – Fall 2018	● UVM environment for the GPIO ● First revision of FPU
43	John Martinuk	Spring 2019 – Summer 2020	● Design Flow lead ● IR Drop Analysis (Static and Dynamic) ● Power Analysis (Static and Dynamic) ● Polymorphic NAND/NOR gate ● Electromigration Test Structure Layout ● Backend design flow for 2nd SoC tapeout
44	Karthik Maiya	Fall 2019 – Spring 2020	● JTAG
45	Keshav Raheja	Fall 2018	● Updated UVM APB sequences

46	Kevin Mi	Spring 2020	● Assisted Analog Team
47	Liangyu Chen	Fall 2018	● Second revision of AFTx04 PCB
48	Luis Haddock	Spring 2019	● Updated self-test simulation environment ● DAC
49	Luis Materon	Spring 2019	● Updated ISA self-test script
50	Luke Kok	Spring 2019 -Summer 2020	● First revision of SPI UVM environment
51	Manik Singhal	Fall 2018 – Fall 2020	● Verification team lead ● Backend design flow for 1nd SoC tapeout
52	Marco Garcia	Spring 2019	● UART UVM environment
53	Matt Olinde	Fall 2019 – Spring 2020	● Phase Locked Loop
54	Matthew Waldren	Spring 2020 – Summer 2020	● Intro to SoCET student ● AFTx04 PCB ● AFTx05 PCB
55	Michael Seaborg	Summer 2018 – Fall 2018	● PCB Component Selection and Pinout for AFTx04 ● OpAmp Revision 1 ● Low Voltage Current Mirror Analysis ● Simple Current Mirror Design & Analysis
56	Michel Brandao Raskin	Summer 2020	● Floorplanning ● ADC/APB module
57	Minh Tran	Fall 2019	● USB communication with FT232R chip
58	Naazneen Rana	Summer 2020	● Currently being trained
59	Nicholas Haythorn	Fall 2019 – Spring 2020	● Compiler compatible with SparCE optimizations
60	Niraj Menon	Fall 2018 - Spring 2020	● gcc compiler setup ● SPIKE setup ● Updated toolchain used for compiler environment
61	Noelle Crane	Fall 2018	● AFTx04 Self-Test Code and simulation
62	Noureldin Hendy	Spring 2019	● Assisted software team
63	Oliver Krefta	Spring 2020 – Summer 2020	● Intro to SoCET student ● I2C

64	Patrick May	Spring 2018	● Interrupt Controller
65	Peyton Young	Spring 2020	● Assisted Analog team
66	Radhika Poddar	Fall 2019	● Updated synthesis script ● Concertal IntelliConX compatible RISC-V
67	Raghul Prakash	Spring 2020	● PWM UVM environment
68	Rajat Arora	Summer 2019 – Spring 2020	● Timer UVM environment
69	Ruoyi Chen	Spring 2020	● Interrupt Controller Integration ● Updating machine learning benchmark
70	Ruth Zhong	Spring 2019	● Third Revision of OpAmp
71	Sanghoon Han	Spring 2020	● FPGA emulation of AFTx05
72	Sean Hsu	Spring 2020	● Fixed and integrated FPU
73	Sean Hwang	Spring 2020	● FPGA emulation of AFTx05
74	Shaunak Robin Oswal	Spring 2020 – Summer 2020	● Worked on Concertal Motor Driver Design
75	Shivam Sharma	Spring 2018	● AFTx03 PCB
76	Xianmeng (Simon) Zhang	Spring 2019 – Spring 2020	● Updated compiler and SPIKE simulator ● JTAG
77	Stephanie Ro	Spring 2018	● Updated Virtuoso Layout Tutorial
78	Tucker Swan	Fall 2019	● Machine Learning benchmark environment ● Developed software libraries for AFTx05
79	Vadim Nikiforov	Spring 2019	● Digital Lead ● SparCE optimization
80	Victor Le	Spring 2020	● Intro to SoCET student ● I2C
81	Vivekanandan Kulumani Rajarajan	Fall 2019 – Spring 2020	● Logical Equivalence Checking ● Functional Verification of RISC-V core
82	Wayne Chen	Spring 2020	● DAC
83	Xe Jin Chan	Spring 2018	● Assisted UVM team
84	Xinlue Liu	Spring 2020 – Summer 2020	● Fixed and integrated the FPU

85	Yi Feng Wang	Spring 2018	● AFTx03 Packaging
86	Yiming Li	Spring 2020 – Summer 2020	● Compressed Scan Chain Implementation
87	Yiming Ma	Spring 2018 – Spring 2019	● AFTx03 PCB ● UVM APB driver, sequencer, comparator, predictor, and scoreboard
88	Young Joo Moon	Summer 2020	● Currently being trained
89	Youtian Chen	Fall 2018	● OpAmp revision 2
90	Yupei Cao	Fall 2019 – Spring 2020	● SPI UVM environment
91	Yuqing Fan	Summer 2020	● Currently being trained
92	Zhao Xing Lim	Spring 2018	● Assisted UVM team
93	Zhengsen Fu	Summer 2020	● FPU UVM environment ● UVM tutorial
94	Zhewen Pan	Fall 2019	● AHB-APB bridge UVM environment
95	Zihan Liu	Summer 2020	● I2C UVM environment

4. Impacts

Disciplinary Impacts

A group of students sought to implement the SparCE optimization for machine learning acceleration, detailed in the paper [2], within the project's 2nd SoC. They succeeded in their goal to get their implementation into the tapeout and when the chips are delivered, the power savings will be measured. This will provide hardware verification for the architecture proposed in the paper and further support its legitimacy. Previously the merit to the paper's findings was purely simulation/ emulation based; however, the evaluation of the speed and power savings for the ASIC implementation shall demonstrate physical results.

Another ASSURE group, led by Dr. Peter Bermel, required wafer space, as well as the layout for their Electromigration structures to be created by one of the SoCET students. Their structures will be fabricated and returned to the research group in approximately a year. Without the SoCET's contribution, the progression of their study might have been delayed, and could have potentially increase the project's expenses.

Human Resources

Once a student has expressed an interest in joining the SoCET team, they are sent a questionnaire that collects their relevant experience and interests. During a team leads meeting, the student is assigned a project based on the available set of beneficial projects, their skillset, and the hours per week & number of semesters they're interested in committing to SoCET work. The students would be asked to go through tutorials applicable to the subteam they joined. Students report to their team leads every week and their participation was typically linearly related to the number of SoCET credit hours they were registered during the semester. This would range from 1 to 3 credit hours commitment per semester which is the equivalent to 3 to 9 hours of SoCET work per week.

The skills which students hone during their time involved with SoCET allows for their swift entry into the SoC/VLSI industry and research. We were able to determine the employment status of 47 former SoCET members and found 27 are engaged in SoC/VLSI related work, 14 are in software engineering positions, and 6 are in other kinds of engineering positions. Employers of those in SoC/VLSI related positions include Intel (7), IBM (2), Broadcom (2), Apple (2), Bosch Australia, Cadence, Johns Hopkins Applied Physics Lab, Lenovo / Motorola Mobility, Micron, NXP Semiconductor, Nokia, Northrop-Grumman, Qualcomm, Shure Inc., and Texas instruments.

Infrastructure / Institutional Resources

The design flow and build scripts we have developed have been used as examples / skeleton files by other Purdue Research teams. This ranges from helping reduce the time required to place-and-route a large design to helping fabricate analog devices on MIT Lincoln Lab's 90nm PDK. Our documentation and scripts can guide other research teams to achieve the following tasks:

- Logic Synthesis
- Mapped Functional Verification
- Logic Equivalence Check (LEC)
- Place and Route (PnR)
- Static Timing Analysis
- Post-PnR Functional Verification
- IR Drop and Power Analysis
- Importing a GDS Stream into Virtuoso
- Design Rule Check (DRC)
- Layout Vs. Schematic (LVS)
- Parasitic Extraction (PEX)
- Automatic Test Pattern Generation (ATPG)
- Generating Functional Test Vectors
- Compiling C code (with ML optimizations) into the RISC-V ISA equivalent
- FPGA emulation

Infrastructure / Information Resources

The AFTx05 chip is available to download at https://github.com/Purdue-SoCET/AFTx05. This repository includes the design flow scripts required to rebuild our SoC from its RTL to its place-and-routed layout. Additionally, users that contact MIT Lincoln Labs for the archived February 2020 version of the PDK will be able to run the scripts, with minimal modification and no errors.

Furthermore, the following tutorials were designed to familiarize incoming students with the software tools they'll be using by practicing with a small scale example of what their assigned task will encompass. To request a copy of these materials, please email John M. Martinuk (jmartinu@purdue.edu).

Intro to Physical Design: Instructions for making an inverter using the MITLL PDK in Cadence Virtuoso and performing with DRC and LVS checks with Mentor Graphic Calibre.

Intro to Analog Design: Practice with small signal analysis on amplifier & Op Amp designs.

Intro to PCB: Tutorial on the mechanics of creating PCB layout using KiCAD.

Intro to Digital Design: An introduction to combinational and sequential circuit design described with systemVerilog, as well as the discrepancies between source and mapped simulations.

Intro to Design Flow: A guide and explanation for the scripts the team uses for synthesis, place-and-route, simulation, and verification of a flex counter.

Intro to UVM: Guide for a student creating their first UVM environment with aggregated information from Mentor Graphic's Verification Academy and ChipVerify to provide sufficient background information.

Technology Transfer

All of the open-source systemVerilog files and have been published on github (https://github.com/Purdue-SoCET/AFTx05) to the world under an apache 2.0 license. We are transferring the repository into the hands of academia and NSWC Crane to use as a quick, easy to modify template for projects which require a RISC-V SoC. At the moment our architecture, particularly the SparCE architecture, is being used to implement a value-similarity optimization with one of Dr. Anand Raghunathan's research groups.

What makes this publication unique to other open-source RISC-V builds is the inclusion of our design flow scripts. Permission has been obtained from Cadence Design Systems to include scripts for RTL simulation, synthesis, place-and-route, ATPG, and LEC. Additionally, we have made arrangements with MIT Lincoln Labs to archive the PDK used to tapeout AFTx05. This way anyone new to using EDA tools will be able to run the scripts, with minimal modification and no errors, to rebuild our SoC from its RTL to it's signed off layout.

The design flow scripts which were published with Cadence Design Systems' permission and will be transferred to users new to using EDA tools and/ or MIT Lincoln Lab's PDK. In order for the script to be useful, users will still need a Cadence license and contact MIT LL for the December 2019 revision of the PDK (with the February 2020 IO pad files). Future users would only be able to tapeout with the latest PDK, but if they get access to the archived version mentioned, the design flow scripts in the git repository would provide people design flow scripts which require minimal debugging.

Once the chips for the AFTx05 SoC have been fabricated, 10 unpacked dies will be transferred to Dr. Bermel's research team. The pads in the middle of the die will be probed with high current densities to be driven through the 80 total electromigration structures and observed with a thermal microscope to derive a method for measuring circuit lifetime like an odometer.

This technology is also being used with Concertal Systems. An IntelliConX compatible RISC-V was published to their web platform for free use to reduce the cost of Concertal System on Chips which require a processor; the other available core was from ARM and required a license. Additionally the AFTx05 processor is serving as the core for a brushless motor controller proof of concept to highlight the efficiency and mobility of Muz Motion's novel motors.

5. Changes / Problems

The turnaround time for fabrication turned out to be approximately a year. While the 2nd SoC was successfully taped out, the batch of chips will not be delivered for another 6 - 12 months. This prevented the project from achieving milestones 13 (Hardware testbed, package dies for second SoC) and 14 (Functional test 2nd SoC, deliver devices for environmental testing to Crane and other Purdue teams) by the end of July 2020. However, this does not mean the milestones have been abandoned; they will be fulfilled when the chips are delivered

Additionally, the original November 2019 tapeout was moved to February 2020 when it was discovered that the spice models for the standard cells were inaccurate; the paths through the cells were about 10% faster than what the spice models were simulating.

The FPU and interrupt handler were postponed for a few semesters until a more experienced team of students (still undergraduates) could be assigned to the task. Development was resumed in Spring 2020, and the module was integrated with the reset of the design, as well as signed off by our UVM team in Summer 2020.

The on-chip "RAM" is implemented with flip-flops instead of actual RAM cells. This work around was used since MIT LL cells, compatible with a memory compiler, were not ready in time for the tapeout dates.

The 1st SoC consisted of mostly buffer cells (over 80%). This was due to a maximum wire length restriction that was put in place to avoid tungsten via corrosion. The 2nd SoC did not have such a large amount of buffer cells. This was thanks to MIT Lincoln Lab refining their process to accommodate larger, digital designs.

6. Special Requirements

In order for the design flow scripts, within https://github.com/Purdue-SoCET/AFTx05_Public, to work properly, the following EDA software and PDK releases must be used:

- Cadence Incisive 15.2
- Cadence Genus 18.1
- Cadence Innovus 18.1
- Cadence Conformal 18.1
- Cadence Modus 18.1
- MIT Lincoln Labs 90nm FDSOI PDK - December 2019:
    - MITLL90_STDLIB_8T (2019.12.20)
    - MITLL90_IOPads_5_1_1 (2020.01.27)
    - Technical Contact: Pascale Gouker (pgouker@ll.mit.edu)

7. References

[1] S. Sen, S. Jain, S. Venkataramani and A. Raghunathan, "SparCE: Sparsity Aware General-Purpose Core Extensions to Accelerate Deep Neural Networks," in IEEE Transactions on Computers, vol. 68, no. 6, pp. 912-925, 1 June 2019, doi: 10.1109/TC.2018.2879434.

[2] S. Das and J. Appenzeller, "WSe2 field effect transistors with enhanced ambipolar characteristics," Applied Physics Letters 103, 103501-1-5 (2013).

[3] C. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” Sunburst Design, Provo, UT, USA, 2002.

[4] *JTAG Programmer Overview for Hercules-Based Microcontrollers*, Texas Instruments, Dallas, TX, United States, Nov. 2015, Accessed on: September, 4th, 2019. [Online]. Available: https://www.ti.com/lit/an/spna230/spna230.pdf

[5] J. Covey and M. Johnson, "System-on-a-Chip Design as a Platform for Teaching Design and Design Flow Integration," in *GLSVLSI '19: Proceedings of the 2019 on Great Lakes Symposium on VLSI, GLSVLSI 2019, Tysons Corner, VA, USA, May 9-11, 2019*, pp. 249–253.