

Open-Source Radio Microcontroller for Fabrication

DESIGN DOCUMENT

Team Number: 27

Client/Adviser: Dr. Henry Duwe

Team Members/Roles:

Ibrahim Shenouda – Analog Architecture Developer

Noah Thompson – Analog Architecture Developer

Nathan Stark – Digital Architecture Developer

Nolan Eastburn – Digital Architecture Developer

Ethan Kono – Security Architecture Developer

Will Custis – Security Architecture Developer

Team Website: <https://sdmay25-27.sd.ece.iastate.edu/>

Revised: 5/4/2025/v3

Executive Summary

The goal of this project is to create a radio microcontroller unit (MCU) with all created artifacts being open-source and tailored to those with only basic engineering experience. Engineers and hobbyists can then use our design as a learning tool to investigate radio frequency (RF) design. This makes our radio MCU unlike all the others we found on the market, which are closed source. Closed source designs make it quite difficult for a user to understand how the unit works, which is what makes our design important for users who want to learn how a radio MCU works. Based upon this description, our design must meet the following key requirements:

- All created artifacts (design documents, test document, code, etc.) must be open-source
- All testing and design documentation must be understandable to an individual with a basic knowledge of circuits and embedded systems
- The design must allow for wireless communication using an open standard wireless communication protocol
- The design must be fabricable
- The design must interface to peripherals (servos, sensors, motors, etc.)

As requested by our client, we used the Caravel harness provided by the Efabless corporation to implement our design. The Caravel harness contains some basic functional units like a management core and I/O access to interact with our design in the user area. The Caravel harness and the design we put in the user area is laid out in the Skywater 130nm PDK, which Efabless has the means to fabricate in. This means we could fabricate the Caravel harness with our custom design in it to implement this MCU. Unfortunately, Efabless shutdown in March 2025, meaning we could not fabricate our design. We continued development and testing to prove our design is viable to fabricate.

For the design, we split it into two primary sections: digital and analog. The digital section contains all the digital components that will execute instructions, transfer data, and interface with peripherals. The analog section will contain all the analog circuits required for RF communication. For the digital section, we are going to be implementing a RISC-V core, a DFF RAM block, a wishbone crossbar, GPIO and I2C peripheral access, and an AES security accelerator. For the analog section, we will be implementing a frequency synthesizer to create stable carrier frequencies for RF communication. These components alone will not be enough to enable RF communication. There is simply too much work for a single senior design team to implement a fully functional radio MCU. What we can do is establish top-level architecture and implement a subset of the RF module designed to meet our requirements with future design teams building out the remaining components.

Using the Efabless toolchain we were able to simulate our designs when laid out to verify that they meet our design requirements. While there will not be chips to learn from, our designs can still be used as a valuable learning tool because we have proven our design is able to be fabricated and provided thorough documentation in the form of this document, and our GitLab page. Even with fabrication uncertain, we hope that future teams can continue development to enable RF communication.

Learning Summary

Development Standards & Practices Used

We considered several engineering standards from IEEE. These include IEEE 802.15.1, 802.15.4, and 1481-2019. IEEE 802.15.1 outlines the requirements for the Bluetooth communication protocol, IEEE 802.15.4 outlines the requirements for the Zigbee communication protocol, and IEEE 1481-2019 outlines standards for verifying integrated circuit designs. While our project implementation does not contain everything necessary to make a functioning radio, the standards helped provide guidance when implementing the system to make sure that it can be easily extended to make a functional radio. In addition to these standards, we also utilized best practices taught in courses and gained from experience in industry, such as self-checking testbenches to confirm digital designs work as intended. Regular testing caught errors earlier in the design process, mitigating repetitive work.

Summary of Requirements

- The MCU shall be implemented using the Efabless Caravel Harness (constraint).
- The MCU shall implement the Zigbee communication stack.
- The MCU shall contain a radio subsystem.
- The MCU shall contain two independent RISC-V cores to execute user programs.
- The MCU shall contain two DMA engines.
- The MCU shall contain the following standard peripherals:
 - GPIO
 - I2C
 - SPI
- The MCU shall have software libraries that provide access to hardware functions.
- The MCU shall have a programming interface.
- The RF module shall be able to transmit over the 915MHz ZigBee broadcast band with a 1MHz channel width.
- All Wishbone masters and slaves will use a 10 MHz reference clock which will be shared with the management SoC.
- 2 KiB of DFF RAM shall be provided for data storage.
- 2 KiB of DFF RAM shall be provided for instruction memory for the RISC-V processor.
- The MCU shall have a testing interface that enables a user to fully test the unit.
- The digital MCU peripherals shall be tested in a computer simulator to ensure correct behavior prior to fabrication.
- The MCU shall have a test plan to evaluate the characteristics of the system.

- Each piece of the RF module will be tested by sending their outputs to analog GPIO pads to read the waveforms. Some internal signals will be connected to the pads via transmission gates for testing.
- All the artifacts produced throughout the design of the MCU shall be open-source (constraint).
- All the documentation shall be intuitive and understandable by individuals with a basic understanding of circuits, digital logic, and MCU usage (constraint).
- The design will fit into a die area of 2.92 mm x 3.52 mm (constraint).

Applicable Courses from Iowa State University Curriculum

The following courses taught at Iowa State are relevant to this project.

| Course | Relevance |
|-----------|---|
| CPR E 281 | Basics of digital logic design and HDLs |
| CPR E 288 | Embedded systems programming in C |
| CPR E 381 | More advanced digital hardware design, CPU architecture |
| E E 201 | Basics of electrical circuits |
| E E 230 | More advanced electrical circuits and systems |
| E E 330 | Integrated circuit design, testing, and layout |
| E E 465 | Digital VLSI |
| EE435 | Analog VLSI |
| CYB E 331 | Cryptography |

New Skills/Knowledge acquired that was not taught in courses

| Skill | Relevance |
|------------------|--|
| PLL design | Key component in analog portion of the design |
| ASIC fabrication | Design is going to be fabricated, design of ASICs was taught, but not fabrication specific |
| NGSpice | Simulator used for analog components |
| OpenLane | Hardening tool to convert HDL code into digital logic layout |

Table of Contents

| | | |
|--------|---|----|
| 1. | Introduction..... | 13 |
| 1.1. | Problem Statement | 13 |
| 1.2. | Intended Users | 13 |
| 1.2.1. | ChipForge Co-Curricular | 14 |
| 1.2.2. | Faculty | 14 |
| 1.2.3. | Radio Hobbyists..... | 14 |
| 2. | Requirements, Constraints, And Standards | 14 |
| 2.1. | Requirements & Constraints | 15 |
| 2.1.1. | Long Term Requirements | 15 |
| 2.1.2. | May 2025 Requirements..... | 16 |
| 2.2. | Engineering Standards | 17 |
| 2.2.1. | Importance of Engineering Standards..... | 17 |
| 2.2.2. | IEEE 802.15.1 - Bluetooth Standard | 17 |
| 2.2.3. | IEEE 802.15.4 - LR-WPAN Standard (Zigbee) | 18 |
| 2.2.4. | IEEE 1481-2019: Integrated Circuit (IC) Open Library Architecture (OLA) | 18 |
| 2.2.5. | Incorporation into Project Design | 18 |
| 3. | Project Plan | 18 |
| 3.1. | Project Management/Tracking Procedures | 18 |
| 3.2. | Task Decomposition | 19 |
| 3.2.1. | Research and Design | 19 |
| 3.2.2. | Hardware Implementation | 20 |
| 3.2.3. | Testing | 21 |
| 3.3. | Project Proposed Milestones, Metrics, and Evaluation Criteria | 22 |
| 3.3.1. | Fall 24 Milestones | 22 |
| 3.3.2. | Spring 25 Milestones | 22 |
| 3.4. | Project Timeline/Schedule | 23 |
| 3.4.1. | Gantt Chart for Fall 2024 | 23 |
| 3.4.2. | Gantt Chart for Spring 2025..... | 24 |
| 3.4.3. | Plan Changes..... | 24 |
| 3.5. | Risks and Risk Management/Mitigation | 25 |
| 3.5.1. | Planned Risks | 25 |

| | | |
|--------|---|----|
| 3.5.2. | Realized Risks | 26 |
| 3.6. | Personnel Effort Requirements | 27 |
| 3.6.1. | Planned Personnel Effort Requirements | 27 |
| 3.6.2. | Deviations From Planned Personnel Effort Requirements | 29 |
| 3.7. | Other Resource Requirements | 31 |
| 4. | Design | 31 |
| 4.1. | Design Context | 31 |
| 4.1.1. | Broader Context | 32 |
| 4.1.2. | Prior Work/Solutions | 33 |
| 4.1.3. | Technical Complexity | 37 |
| 4.2. | Design Exploration | 38 |
| 4.2.1. | Design Decisions | 38 |
| 4.2.2. | Ideation | 39 |
| 4.2.3. | Decision-Making and Trade-Off | 49 |
| 4.3. | Final Design | 51 |
| 4.3.1. | Overview | 51 |
| 4.3.2. | Detailed Design and Visual(s) | 51 |
| 4.3.3. | Functionality | 64 |
| 4.3.4. | Areas of Concern and Development | 64 |
| 4.4. | Technology Considerations | 65 |
| 5. | Testing | 65 |
| 5.1. | Unit Testing | 65 |
| 5.1.1. | Digital | 65 |
| 5.2. | Interface Testing | 67 |
| 5.2.1. | Digital | 67 |
| 5.2.2. | Analog | 67 |
| 5.3. | Integration Testing | 67 |
| 5.4. | System Testing | 67 |
| 5.4.1. | Digital | 67 |
| 5.4.2. | Analog | 67 |
| 5.5. | Regression Testing | 68 |
| 5.5.1. | Digital | 68 |

| | |
|---|-----|
| 5.5.2. Analog..... | 68 |
| 5.6. Acceptance Testing | 68 |
| 5.6.1. Digital | 68 |
| 5.6.2. Analog..... | 69 |
| 5.7. Security Testing | 71 |
| 5.8. Results | 71 |
| 5.8.1. Wishbone Crossbar | 71 |
| 5.8.2. DFF RAM | 72 |
| 5.8.3. User Area RISC-V Core | 72 |
| 5.8.4. I2C Peripheral | 74 |
| 5.8.5. Wishbone Register File | 76 |
| 5.8.6. PLL Divider | 77 |
| 5.8.7. VCO..... | 84 |
| 5.8.8. Charge Pump | 87 |
| 5.8.9. PFD/Charge Pump | 87 |
| 5.8.10. Level Shifters | 88 |
| 6. Implementation and Design Analysis | 90 |
| 6.1. Wishbone Crossbar | 90 |
| 6.2. VexRISC-V Core | 92 |
| 6.3. DFF RAM | 92 |
| 6.4. I2C Controller | 92 |
| 6.5. AES Subsystem | 93 |
| 6.5.1. Open-Source AES Options | 93 |
| E-Fabless AES..... | 93 |
| 6.4.2 Selected Option | 94 |
| 6.6. PLL | 94 |
| 6.6.1. Fractional Divider..... | 94 |
| 6.6.2. Voltage Controlled Oscillator (VCO)..... | 98 |
| 6.6.3. Charge Pump | 101 |
| 6.6.4. Phase Frequency Detector (PFD) | 102 |
| 6.6.5. Level Shifters | 103 |
| 6.6.6. Loop Filter | 104 |

| | | |
|---------|---|-----|
| 6.6.7. | Integrated PLL | 105 |
| 7. | Ethics and Professional Responsibility..... | 105 |
| 7.1. | Areas of Professional Responsibility/Codes of Ethics | 106 |
| 7.2. | Four Principles | 107 |
| 7.3. | Virtues | 109 |
| 7.3.1. | Responsibility | 109 |
| 7.3.2. | Respect..... | 109 |
| 7.3.3. | Flexibility | 109 |
| 7.3.4. | Individual Virtue Assessment | 109 |
| 8. | Closing Material | 111 |
| 8.1. | Summary of Progress | 111 |
| 8.2. | Value Provided | 112 |
| 8.3. | Next Steps | 112 |
| 9. | References | 112 |
| 10. | Appendices..... | 113 |
| 10.1. | Operation Manual | 113 |
| 10.1.1. | Creating, Compiling, and Executing Programs for the User Area RISC-V Core | 114 |
| 10.1.2. | Using the I2C Module | 115 |
| 10.2. | User Journey Map | 116 |
| 10.3. | Security Analysis | 116 |
| 10.3.1. | Attacks & Vulnerabilities in Microcontrollers | 117 |
| 10.3.2. | Countermeasures and Solutions for Vulnerabilities in Microcontrollers | 118 |
| 10.3.3. | Attacks & Vulnerabilities in Radio Frequency Modules..... | 119 |
| 10.3.4. | Countermeasures and Solutions for Vulnerabilities in Microcontrollers | 119 |
| 10.3.5. | Conclusion | 120 |
| 10.4. | PLL Current Starved VCO Circuit Analysis | 121 |
| 10.5. | PLL Buffer Insertion Theory | 123 |
| 10.6. | PLL Charge Pump Circuit Analysis | 125 |
| 10.7. | Code | 128 |
| 10.8. | Team Organization | 128 |
| 10.8.1. | Team Members | 128 |
| 10.8.2. | Required Skill Sets for Your Project | 128 |

| | | |
|---------|--|-----|
| 10.8.3. | Skill Sets covered by the Team | 128 |
| 10.8.4. | Project Management Style Adopted by the team | 128 |
| 10.8.5. | Initial Project Management Roles | 128 |
| 10.8.6. | Team Contract | 129 |

Table of Figures

| | |
|--|----|
| Figure 1: Research and Design Task Decomposition | 19 |
| Figure 2: Hardware Implementation Task Decomposition | 21 |
| Figure 3: Testing Task Decomposition | 22 |
| Figure 4: Fall 2024 Gantt Chart | 23 |
| Figure 5: Spring 2025 Gantt Chart | 24 |
| Figure 6: First Order Delta Sigma | 41 |
| Figure 7: Dual Modulus Divider..... | 42 |
| Figure 8: PLL Diagram | 43 |
| Figure 9: Phase Frequency Detector..... | 43 |
| Figure 10: Phase Frequency States | 44 |
| Figure 11: PFD Gain Without Dead Zones | 44 |
| Figure 12: PFD gain with Dead Zone | 45 |
| Figure 13: Dead Zones Jitter | 45 |
| Figure 14: Ring Oscillator..... | 46 |
| Figure 15: Current-Starved Ring Oscillator | 46 |
| Figure 16: PLL Charge Pump & Loop Filter..... | 47 |
| Figure 17: $3/2$ Inverter Path..... | 48 |
| Figure 18: Full Design - Multi-Year | 51 |
| Figure 19: Spring 2025 Design | 52 |
| Figure 20: PLL Design | 52 |
| Figure 21: Fractional N Divider Design..... | 53 |
| Figure 22: T Flip Flop | 54 |
| Figure 23: Accumulator Architecture | 59 |
| Figure 24: Loop Filter | 60 |

| | |
|--|----|
| Figure 25: Digital Layout..... | 69 |
| Figure 26: Fractional N Divider..... | 70 |
| Figure 27: Wishbone Crossbar Testbench Waveform | 71 |
| Figure 28: DFF RAM Writes | 72 |
| Figure 29: DFF RAM Reads..... | 72 |
| Figure 30: RISC-V Core Instruction Fetching | 73 |
| Figure 31: RISC-V Core Data Memory Write | 73 |
| Figure 32: Management Core Data Memory Read | 74 |
| Figure 33: I2C Two Writes | 74 |
| Figure 34: Oscilloscope I2C Readings..... | 75 |
| Figure 35: I2C I/O Expander Testbench with Seven Segment Displays..... | 76 |
| Figure 36: Wishbone Register File Write and Read | 77 |
| Figure 37: Wishbone Register File Arbitration | 77 |
| Figure 38: Prescaler Testbench | 78 |
| Figure 39: Waveform of N=3 Test Results..... | 79 |
| Figure 40: Waveform of N=2 Test Results | 79 |
| Figure 41: Prescaled Clock Divider Results | 80 |
| Figure 42: F=7 and N=1 Accumulator Results | 81 |
| Figure 43: Full Divider Testbench..... | 81 |
| Figure 44: CLK_IN = 928MHz, N.F =92.8 Accumulator Results..... | 82 |
| Figure 45: N = 3 at 928MHz with tt corner | 83 |
| Figure 46: Prescaler Frequency Sweep Code | 83 |
| Figure 47: LV VCO Frequency vs Vctl | 85 |
| Figure 48: LV VCO Duty Cycle vs Vctl | 85 |
| Figure 49: HV VCO Frequency vs Vctl..... | 86 |
| Figure 50: HV VCO Duty Cycle vs Vctl | 86 |
| Figure: 51 PFD Input..... | 88 |
| Figure 52: Sample 2x2 Wishbone Crossbar Architecture | 91 |
| Figure 53: Prescaler XScem Schematic..... | 94 |
| Figure 54: Final Prescaler Layout..... | 95 |
| Figure 55: Counter XScem Schematic | 96 |
| Figure 56: Divider XScem Schematic | 96 |

| | |
|--|-----|
| Figure 57: Accumulator XSchem Schematic | 97 |
| Figure 58: Accumulator Design..... | 98 |
| Figure 59: Divider XSchem Schematic | 98 |
| Figure 60: LV VCO Final Schematic | 99 |
| Figure 61: LV VCO Output Buffer | 99 |
| Figure 62: LV VCO Final Layout | 100 |
| Figure 63: HV VCO Final Schematic | 101 |
| Figure 64: HV VCO Output Buffer | 101 |
| Figure 65: Charge Pump Schematic..... | 102 |
| Figure 66: PFD Schematic with Reset Delay..... | 102 |
| Figure 67: 3.3V to 1.8V Level Shifter Schematic..... | 103 |
| Figure 68: 1.8V to 3.3V Level Shifter Differential Schematic..... | 104 |
| Figure 69: 1.8V to 3.3V Level Shifter Current Mirror Schematic | 104 |
| Figure 70: Second Order PLL Loop Filter Schematic..... | 105 |
| Figure 71 Integrated PLL | 105 |
| Figure 72: User Journey Map | 116 |
| Figure 73: Current Starved Oscillator | 121 |
| Figure 74: Unbuffered Voltage Controlled Oscillator Schematic..... | 122 |
| Figure 75: Buffered Current Starved Voltage Controlled Oscillator Schematic..... | 123 |
| Figure 76: Conventional Charge Pump Schematic | 125 |
| Figure 77: Charge Pump Output Stage..... | 125 |
| Figure 78: Charge Pump Middle Stage | 126 |
| Figure 79: Charge Pump Current Reference (First Stage) | 127 |

Table of Tables

| | |
|--|----|
| Table 1: Planned Personnel Effort Requirements | 27 |
| Table 2: Actual Personnel Effort Requirements | 29 |
| Table 3: Broader Design Context | 32 |
| Table 4: Market Research | 34 |
| Table 5: Prescaler Truth Table | 55 |

| | |
|--|-----|
| Table 6: Programmable Counter Truth Table | 56 |
| Table 7: Routh-Hurwitz Criterion | 61 |
| Table 8: Sweep Prescaler Results | 84 |
| Table 9: LV VCO Full Results | 85 |
| Table 10: HV VCO Full Results | 87 |
| Table 11: Current vs output voltage at process corners | 87 |
| Table 12 PFD/Charge Pump Full Results | 88 |
| Table 13: 3.3V to 1.8V Level Shifter Simulations | 89 |
| Table 14: 1.8V to 3.3V Level Shifter Differential | 89 |
| Table 15: 1.8V to 3.3V Level Shifter Current Mirror Simulations | 90 |
| Table 16: Open-Source AES Encryption Options | 93 |
| Table 17: Areas of Professional Responsibility and Codes of Ethics | 106 |
| Table 18: Four Ethical Principles | 107 |
| Table 19: Address Map for Wishbone Crossbar | 114 |
| Table 20: Register Summary for I2C Module | 115 |

1. Introduction

1.1. PROBLEM STATEMENT

Many radio microcontroller units (MCUs) exist on the market today; however, their designs are closed source. This makes it difficult for users of radio MCUs such as the ISU ChipForge co-curricular, faculty, and radio hobbyists to learn about radio microcontrollers without reverse engineering the unit due to the designs not being publicly available. It is important that they learn about radio communication now as wireless connections are becoming more common compared to their wired counterparts. To address this, our team has designed an open-source radio MCU. This will provide anyone who wants to learn about how a radio MCU works with all the documentation and implementation details, including how we designed and implemented in our unit. In addition, our implementation can be fabricated on the Skywater 130nm PDK, which allows users to physically use and analyze the radio MCU on top of being able to look at design documents. Having this radio MCU be open-source enables users to make their own modifications to the design, which is something an individual cannot do with closed source units. This further promotes learning and creates opportunities for individuals to be innovative with the base radio MCU design.

1.2. INTENDED USERS

Our user journey map for an ISU EE201 Student is included in the appendix.

1.2.1. ChipForge Co-Curricular

Chip Forge is a co-curricular at Iowa State University (ISU) that primarily focuses on the design and fabrication of chips through the Efabless Corporation. The members of Chip Forge consist of undergraduate and graduate students at ISU as well as faculty advisors. All members of Chip Forge have an interest in chip fabrication and desire to learn more about it through project development and experimentation. Based upon their interests, the ChipForge members need an open-source radio MCU that they can see the designs for, fabricate the design, and then use the MCU in the lab. This will allow them to dive deep into how a radio frequency (RF) subsystem works on an MCU as well as use this MCU has a component in any projects they work on. This provides a much better learning experience than attempting to reverse-engineer an existing closed source radio MCU, which would prove to be quite difficult and not clear. Finally, we hope that the ChipForge members can take the radio MCU design and build upon it to meet their needs. Since the original design can be fabricated, the ChipForge members can make whatever modifications they see fit for their applications and learn tons about radio MCUs along the way.

1.2.2. Faculty

Some courses at Iowa State University utilize microcontrollers for labs, such as CPR E 288. Once fabricated and tested, faculty could use the open-source MCU for lab assignments in place of existing options. Faculty teaching these courses need a reliable way to teach students about MCUs with good documentation and tooling support. Good documentation and tools are essential since this will likely be used for undergraduate students. Documentation and tools would also reduce the time that faculty need to spend changing existing lab experiments should they adopt the new MCU. There are several potential advantages of the open-source MCU over existing ones. These include the ability to tailor the hardware to course requirements, such as adding additional functionality not commonly implemented in other commercially available products. Additionally, for cybersecurity focused courses, faculty could provide students the opportunity to study in detail a wireless device from the hardware level in the lab, which would be a valuable learning experience not able to be replicated with closed-source designs.

1.2.3. Radio Hobbyists

Radio hobbyists are usually interested in unique features and documentation/tooling and need MCUs that enable them to communicate with other devices in an easy way without a lot of setup. If one examines commercial MCUs that sell well amongst hobbyists, they are typically low-cost, have good tools and documentation, and have several connectivity features that allow hobbyists to connect them to common devices. Due to the low volume of the production, competing on price will be difficult, but tools and documentation is going to be a central focus of the project, and unique features are possible due to the open-source nature of the project. Furthermore, for hobbyists with a larger budget, our design could serve as a starting point to develop their own MCU tailored to their specific application.

2. Requirements, Constraints, And Standards

Due to our project scope being too large for a single ISU senior design team, we decided to scope our requirements into two groups: Long term requirements and May 2025 requirements. All the

requirements in these two sets need to be implemented for our project to solve the problem described in our problem statement. Therefore, the union of these two requirement sets make up the full system requirements for our project. The long term requirements will be implemented by a future senior design team and the May 2025 requirements were be implemented by our senior design team by May of 2025. Requirements marked as “constraints” are constraints given by our advisor/client. May 2025 requirement sections marked as “partial” indicate that only some of the requirements for that section are May 2025 requirements and the remaining ones are under long term requirements.

2.1. REQUIREMENTS & CONSTRAINTS

2.1.1. Long Term Requirements

2.1.1.1. Functional Requirements

- The MCU shall implement the Zigbee communication stack.
 - The MCU shall be able to connect to Zigbee-compatible devices.
 - The MCU shall be able to generate the Message Authentication Code (MAC) for encrypted data.
- The MCU shall contain a radio subsystem.
 - The radio subsystem shall support multiple operating frequencies.
 - The radio subsystem shall support multiple modulation schemes.
 - The radio subsystem shall be able to transmit and receive information to/from other radios.
- The MCU shall contain two independent RISC-V cores to execute user programs.
- The MCU shall contain two Direct Memory Access (DMA) engines.
- The MCU shall contain the following standard peripherals:
 - Configurable timers, Serial Peripheral Interface (SPI), and a Universal Asynchronous Receiver/Transmitter (UART).
 - The peripherals must have an easy interface for use and configuration (constraint).
- The MCU peripherals shall be memory-mapped and accessible to the RISC-V processors.
- The MCU shall have software libraries that provide access to hardware functions.
- The MCU shall provide support for loading user programs for the independent RISC-V cores.
 - The programming interface shall be over a serial connection to the MCU.
 - The programming interface shall have a stand-alone application that users run on their PCs to program the MCU.
 - The programming interface shall be intuitive to use, such that an individual with basic MCU programming knowledge can easily program the MCU (constraint).
- The RF module shall be able to transmit over the 915MHz ZigBee broadcast band with a 1MHz channel width.
 - The input signal from the processor will go through a Digital to Analog Converter (DAC) before being modulated with the carrier signal using quadrature phase shift keying
 - The modulated signal will be sent to a power amplifier and transmitted from an antenna.

2.1.1.2. *Testing Requirements*

- The MCU shall have a testing interface that enables a user to fully test the fabricated unit.
 - The testing interface should provide electrical connections to the following components in the MCU for testing:
 - RISC-V cores, DMA engines, peripheral interfaces, PLL
- The MCU shall have a test plan to evaluate the characteristics of the fabricated system.
 - The test plan shall provide steps to test the MCU peripherals via software.
 - Transmit and receive for the I2C, SPI, and UART
 - Correct divider, counter, and comparator behavior for timer.
 - Correct input/output functionality for GPIO.
- Each piece of the RF module will be tested by sending their outputs to analog GPIO pads to read the waveforms. Some internal signals will be connected to the pads via transmission gates for testing .

2.1.2. *May 2025 Requirements*

2.1.2.1. *Functional Requirements*

- The MCU shall be implemented using the Efabless Caravel Harness (constraint).
 - All digital and analog components shall be compatible with the Skylake 130nm technology that the Caravel platform supports (constraint).
- The MCU shall implement the Zigbee communication stack (partial).
 - The MCU shall be able to encrypt and decrypt data with Advanced Encryption Standard (AES) 128-bit encryption using counter (CTR) mode in accordance with the ZigBee standard.
- The MCU shall contain an independent RISC-V core to execute user programs.
- The MCU shall contain the following standard peripherals:
 - General Purpose Input Output (GPIO) and Inter-Integrated Circuit (I2C)
 - The peripherals must have an easy interface for use and configuration (constraint).
- The MCU peripherals shall be memory-mapped and accessible to the RISC-V processors.
- The MCU shall provide support for loading user programs for the independent RISC-V core (partial)
- The RF module shall be able to transmit over the 915MHz ZigBee broadcast band with a 1MHz channel width (partial).
 - The carrier signal will be generated by an analog Phase-Locked Loop (PLL) frequency synthesizer to 915MHz using a reference oscillator.
 - The PLL will be able to step 1MHz between 902 and 928MHz to land at every frequency band.
 - The PLL shall have a settling time of less than or equal to 170us [ATSAMR30M18A](#)
 - The PLL output will have phase noise less than -91dBm/Hz at a 32 MHz offset.
- All Wishbone masters and slaves will use a 10 MHz reference clock which will be shared with the management System on Chip (SoC).
- 2 KiB of D Flip-Flop Random Access Memory (DFF RAM) shall be provided for data storage.
- 2 KiB of DFF RAM shall be provided for instruction memory for the RISC-V processor.

2.1.2.2. *Non-Functional Requirements*

- All the artifacts produced throughout the design of the MCU shall be open-source (constraint).
- All the documentation shall be intuitive and understandable by individuals with a basic understanding of circuits, digital logic, and MCU usage (constraint).

2.1.2.3. *Testing Requirements*

- The digital MCU peripherals shall be tested in a computer simulator to ensure correct behavior prior to fabrication.
 - Unit tests, system tests, and ad-hoc tests on an ARTY A7-100T FPGA shall be conducted.
- The analog RF MCU components shall be tested in a computer simulator to ensure correct behavior prior to fabrication
 - The PLL will have its divider, PFD, reference oscillator, and VCO simulated and characterized using the layout derived from the caravel board.
 - The wave forms can then be used to confirm the components behave as expected.

2.1.2.4. *Resource Requirements*

- The design will fit into a die area of 2.92 mm x 3.52 mm (constraint).

2.2. ENGINEERING STANDARDS

During the design phase of the project, several engineering standards were examined to help with planning. Two of these standards, IEEE 802.15.4 and IEEE 1481-2019 served a useful purpose during the implementation process. However, IEEE 802.15.1, which defined Bluetooth, ended up not being used.

2.2.1. *Importance of Engineering Standards*

- Engineering standards are important to ensure safety, reliability, and compatibility with other products and protocols. Standards also ensure that the creation and manufacturing of products meets sets of requirements that are deemed necessary to meet several different qualifications.

2.2.2. *IEEE 802.15.1 - Bluetooth Standard*

- This standard defines the physical and medium access control layers for wireless communication. The Bluetooth standard is used worldwide for low power wireless communication devices across short range radio frequency connectivity, which is important for our project as we are implementing a short range and low radio frequency device through the caravel and Efabless process. Its standards will be important to take into consideration since our design implements an RF (radio frequency) module as well.
- This standard was ultimately not used during the implementation phase, since the client decided that Zigbee was a better fit for the project after being presented with our research. This decision was made because Zigbee is an open-source standard, while Bluetooth is not, and Zigbee has the option for a lower operating frequency of ~915 MHz as opposed to Bluetooth's 2.4 GHz frequency. Since lack of prior

experience with these high frequencies was already a significant risk for the project, Zigbee helped to mitigate this by keeping the frequency lower.

2.2.3. IEEE 802.15.4 - LR-WPAN Standard (Zigbee)

- This standard specifies low-rate wireless personal area network operations, network stack, model, and foundational network layers for the physical and MAC networking layers. It is the basis for the ZigBee standard that our project group is utilizing. It also specifies communication for low-rate wireless devices and is intended to provide solid foundations in networking for said wireless devices. This standard applies to our project as the ZigBee protocol outlines the details of operating a ZigBee device so that they are compatible with existing devices. This includes details of the physical and MAC layers of the network stack, which are necessary for reliable communication.

2.2.4. IEEE 1481-2019: Integrated Circuit (IC) Open Library Architecture (OLA)

- This standard provides the analysis for designers to analyze timing, signal integrity, logic behavior, and power consumption across different technologies within a certain accuracy. This standard is relevant to our project since it notes proper timing processes which are essential for the correct operation of the digital components of our microcontroller unit and the RF module.

2.2.5. Incorporation into Project Design

- General principles of power consumption and timing closure will be taken into consideration for our project design and implementation, as it is important for both the MCU and RF module we are designing. The general protocols and principles for Bluetooth and ZigBee will also be utilized and based around because all other low-rate wireless devices follow the same set of requirements and protocols. Incorporating those standards specifically into our project will be necessary across the network stack for connectivity with other devices that use the same protocols and standards.

3. Project Plan

3.1. PROJECT MANAGEMENT/TRACKING PROCEDURES

This project will be managed using a Waterfall approach. This was chosen because we have a small set of deliverables with a lot of interdependence that are required to be completed by a deadline with little flexibility. Waterfall will help make sure that each part of the project (requirements, design, implementation, verification) are all completed in order. Additionally, documentation is a key part of the project, and Waterfall methodology typically results in more comprehensive documentation than Agile, the other approach that was considered. This stems from the fact that Agile focuses on functionality over documentation as defined in the Agile Manifesto. Since the project will span multiple senior design teams, it is more acceptable to trade off functionality for better documentation, since poorly documented functions would likely cause delays and confusion among future design teams.

The project will use Git as the version control system for code created for the project and a repository hosted on the Iowa State Gitlab server. This was chosen since Git provides a lot of functionality for managing different feature development and dealing with conflicts, and the Iowa State Gitlab is already set up for easy collaboration. Issue tracking will be done using Trello, since it is free and allows for custom categories to easily classify issue status. Project communication is currently utilizing a Discord server since it is free, allows different channels to discuss different aspects of the project, and allows for voice and video calls for remote collaboration.

3.2. TASK DECOMPOSITION

The task decomposition for the project was broken into four main categories, with each having several subcategories to further break down tasks so they can be more easily measured. This will allow for actionable tasks that can easily be assigned to team members so everyone knows what they should be doing.

3.2.1. Research and Design

The primary deliverables expected from this portion of the project are the test plan document and the documentation for the components of the system (both hardware and software). The test plan will outline the bring up tests necessary to confirm the chip functions correctly after it is fabricated. This will need a high level of detail, since these tests will be carried out by people not directly involved with the design process. Documentation for individual components is also important, since the design will be used by a variety of people, including undergraduate students with limited experience, so understanding how each component of the system functions is critical for the chip to be useful.

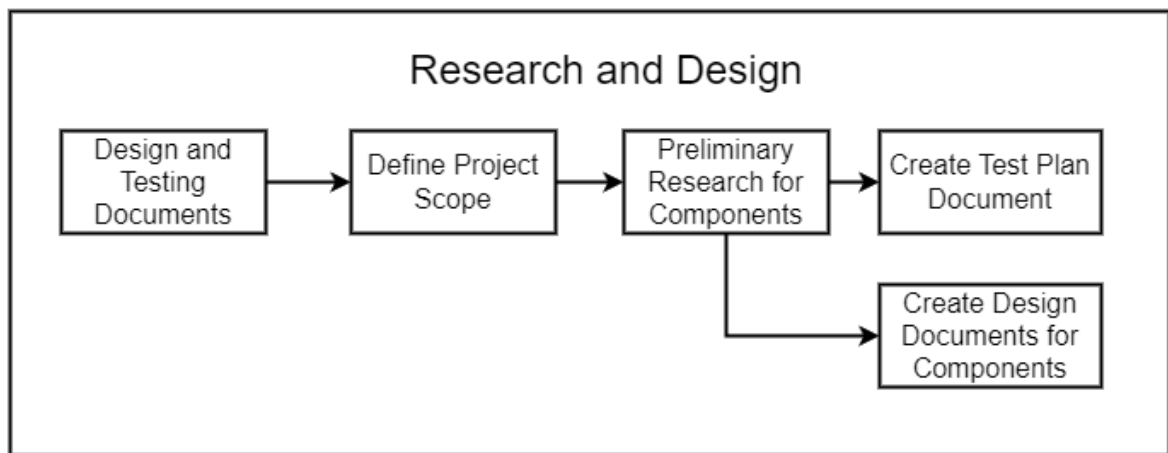


Figure 1: Research and Design Task Decomposition

3.2.2. Hardware Implementation

The hardware implementation is split into two primary categories, analog and digital design. These two will be developed and tested separately and combined during integration to create the final system.

The analog design will primarily consist of a PLL, which in turn is made up of a phase detector, charge pump, low pass filter, voltage-controlled oscillator, and a divider. Each of these components will be designed and tested individually before being integrated together to form the final PLL. In future iterations of the project, the PLL will make up an RF subsystem for transmitting and receiving data.

The digital design will consist of a Wishbone crossbar, RISC-V core, DFF RAM, DMA controller, security acceleration, and external peripherals. The Wishbone crossbar will arbitrate access to memory mapped peripherals in the design and will be created by hand and may change in the future due to concerns about area usage. The RISC-V core will be generated using a project called VexRISC-V that provides optimized cores with good performance along with customization options. This core will be responsible for running user software. The RAM will be implemented using existing DFF RAM macros found on the Efabless marketplace. The DFF RAM will serve primarily as data storage for the RISC-V core, with smaller blocks being used for instruction memory or FIFOs to send/receive data. The DMA controller will help to offload work from the RISC-V core when transferring data to/from various peripherals. This will allow more processing power to be available for user applications. Security acceleration will allow users to encrypt and decrypt data more efficiently than doing it in software by providing hardware capable of performing these operations. Finally, the external peripherals will provide a way for the user application to communicate with other devices using protocols such as I2C, SPI, or UART. This will let users interact with other devices, such as sensors, nonvolatile storage, or other microprocessors.

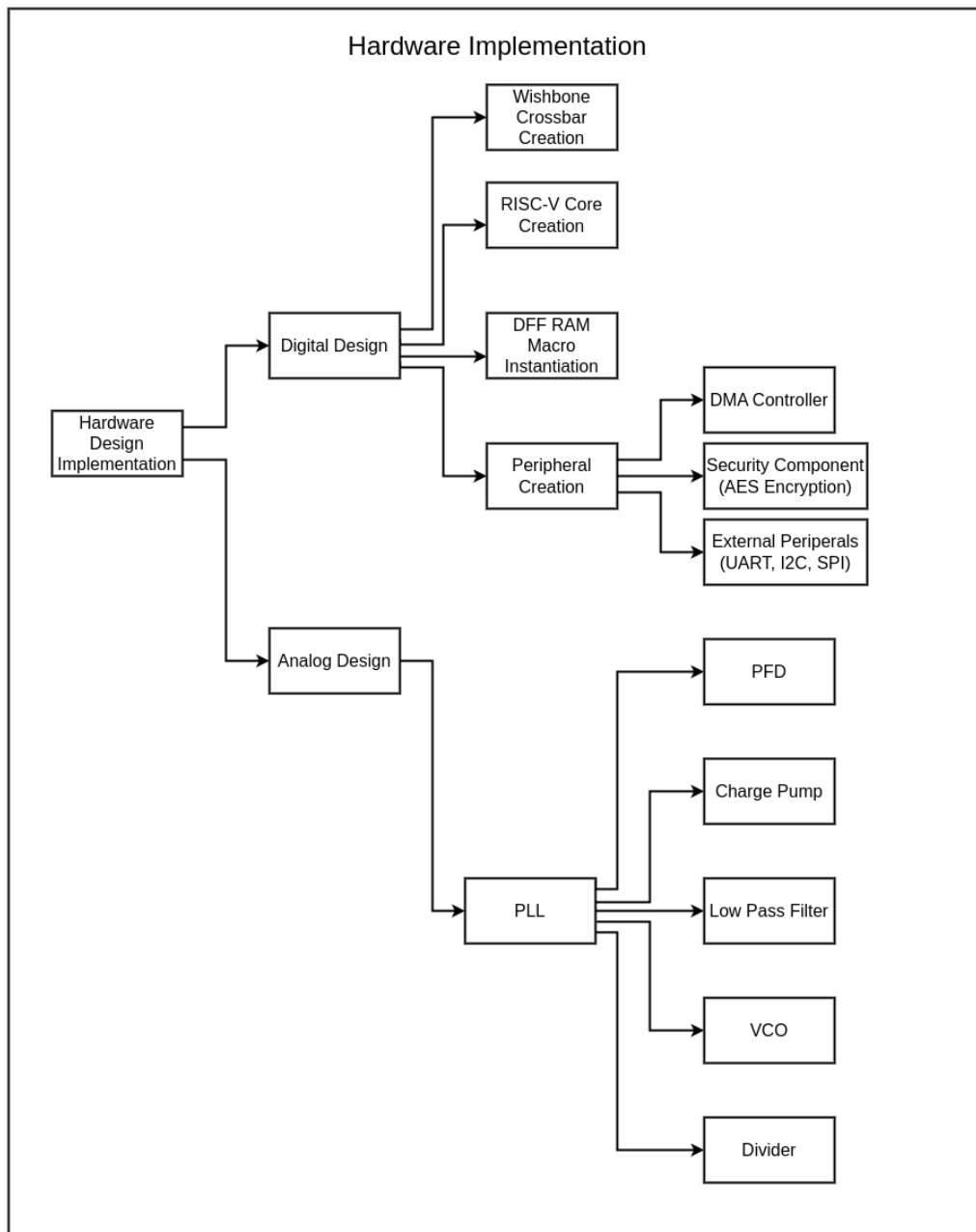


Figure 2: Hardware Implementation Task Decomposition

3.2.3. Testing

Testing of the various components of the design will be essential to ensure that the chip functions properly. There are three primary types of testing that will be performed. Verilog testbenches for the digital peripherals will help to ensure that the blocks work correctly in isolation, which will help to eliminate sources of error during integration. Test C programs for the RISC-V processor will provide a means of system-level testing and will make sure that the system functions as intended. Finally, analog component testing in simulation for the PLL will make sure that the PLL

is functioning as expected and can be tested after fabrication to ensure the fabricated version meets specifications.

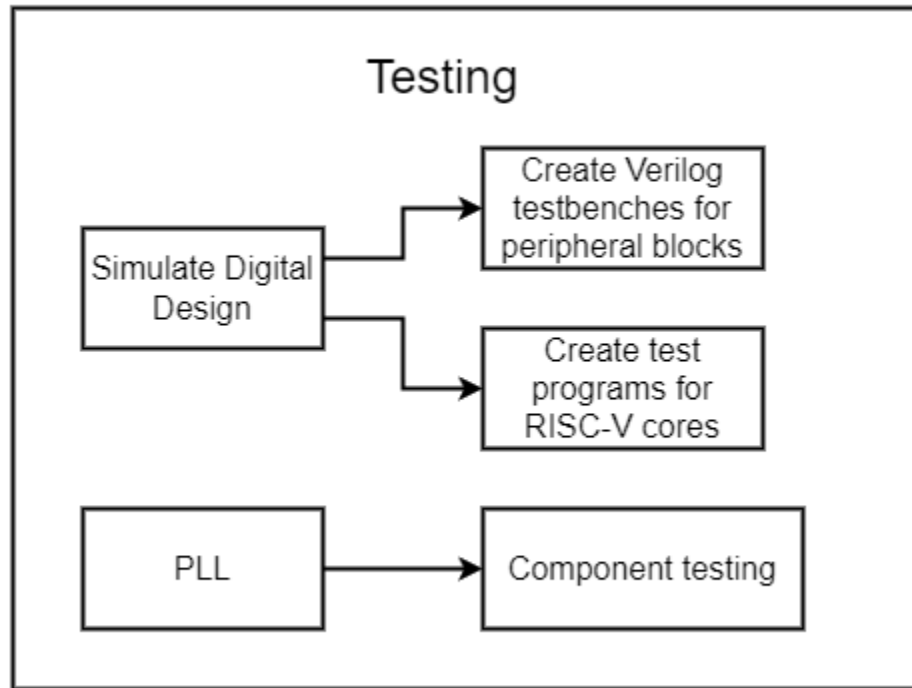


Figure 3: Testing Task Decomposition

3.3. PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

3.3.1. Fall 24 Milestones

- Design document finished
- Initial hardware designs finalized
 - Specific Implementations of each PLL components
 - Baseline implementations of peripherals
 - RISC-V core design
- Initial test plan
 - Defined expected behavior of each component
 - System to test component behavior

3.3.2. Spring 25 Milestones

- Hardware components created
 - All baseline implementations created and building
- Initial testing complete
 - Digital components tested as a system in simulation/on FPGA
- Test plan created

- Test cases for after fabrication to evaluate electrical characteristics and behavior of the device
- Test cases should cover all peripherals and electrical characteristics

3.4. PROJECT TIMELINE/SCHEDULE

Our full project timeline and schedule are captured in our Gantt charts shown on the next pages. Due to the complexity of our project, many of the tasks will need to be shifted to the end of Fall or Spring of next year. We will only have a single deliverable after the Spring semester, which will contain a partial implementation of our design as well as all of the design and testing documentation.

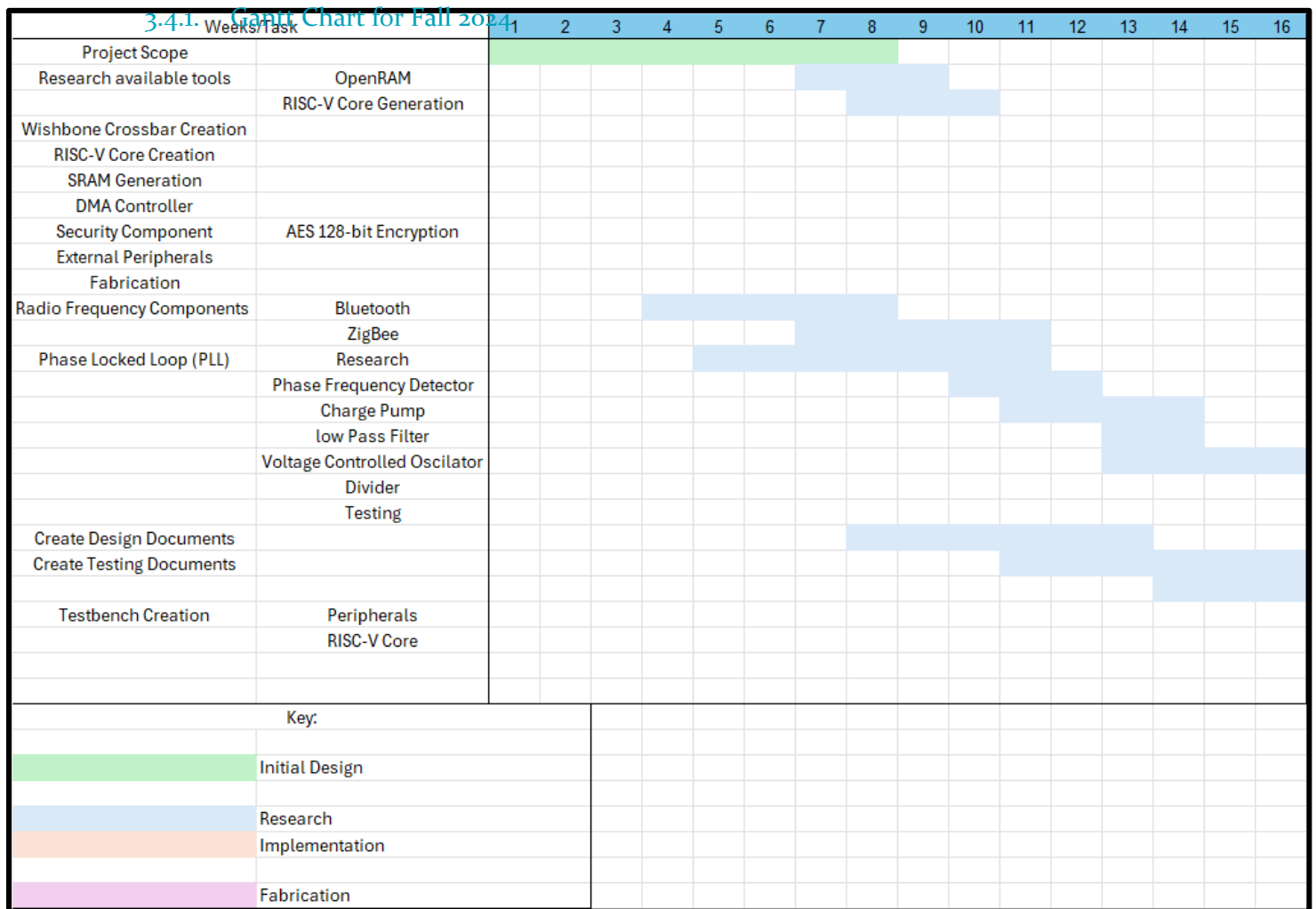


Figure 4: Fall 2024 Gantt Chart

3.4.2. Gantt Chart for Spring 2025

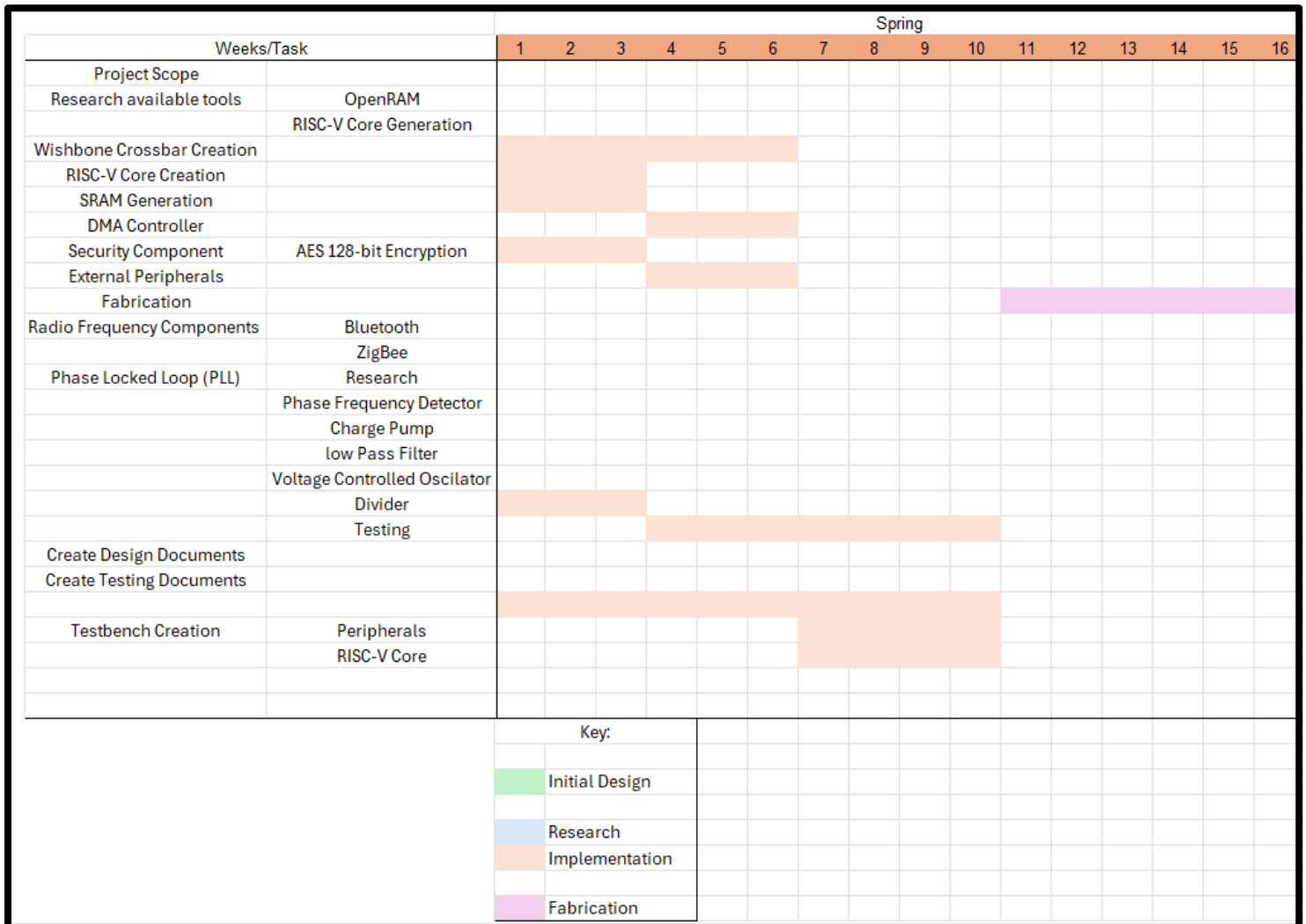


Figure 5: Spring 2025 Gantt Chart

3.4.3. Plan Changes

In late March of 2025, our team was informed that Efabless (the organization that is providing the fabrication of our chip) was shut down due to funding issues. This effectively pushed development time out until May instead of April, giving us more time to work on our design implementation. In addition, due to many realized risks, which will be covered in later sections, many of the tasks listed on the Spring 2025 Gantt chart took much longer. In addition, due to realized risks, we cut the DMA Controller.

3.5. RISKS AND RISK MANAGEMENT/MITIGATION

3.5.1. Planned Risks

3.5.1.1. *Unknown Design Tools*

Risks

- Our team has no experience with Caravel and the Efabless process.
- Some tools have known issues that may cause delays.

Mitigation

- We are Currently working with members of ISU Chip Forge to learn how to design for the Caravel Board and avoid known issues.
- We will use available IP from the Efabless marketplace when possible.

3.5.1.2. *Limited Die Space*

Risks

- The die is 2.92mm x 3.52mm, this means we may not be able to fit everything we want on the chip.
- Challenging to accurately evaluate the amount of space on the die.

Mitigation

- We have created a list of components for a minimum viable product that must be included.
- We can remove lower priority components from the design should die space become an issue.
- If necessary, some components can be accessed via a breakout board.
- Based on our testing, about 300KB of RAM would fill the chip.

3.5.1.3. *High Frequency Components*

Risks

- High frequency component behavior is difficult to predict prior to simulation.
- Simulation may reveal that divider creates too much spurring for the PLL to create a usable signal.

Mitigation

- We plan to characterize each individual component through simulation and use that data to characterize the system. This will allow us to quickly pinpoint a component that is not meeting its requirements and revise its design.
- As a backup, we have investigated an alternative divider design that eliminates spurring at the cost of greater noise.

3.5.2. Realized Risks

3.5.2.1. *Unknown Design Tools*

Risk Realization

- Lack of knowledge on how to use various tools, including digital/analog simulation, synthesis, and layout resulted in longer than expected times to implement various components of the design, such as the Wishbone crossbar and frequency divider.
- Lack of knowledge on how to use tools and time constraints also resulted in a change of plan from using an SRAM generation tool to using RAM implemented with D Flip-Flops. This resulted in increased die usage, which is discussed further in section 3.5.2.2.
- The Efabless corporation experienced financial difficulties and ultimately went out of business, so tool and process support was not as available as we planned. This is discussed in more detail in section 3.5.2.3.

Implemented Mitigation

- Scope had to be modified to deliver a minimum viable product. For the digital side, this meant cutting out the DMA engine as well as the UART and SPI peripherals to spend more time developing and debugging other digital components.

3.5.2.2. *Limited Die Space*

Risk Realization

- As a result of tool issues, SRAM generation was abandoned in favor of a more proven D Flip-Flop approach to memory. This worked as expected functionally but decreased the amount of memory per unit area that was possible.

Implemented Mitigation

- The total memory in the system was reduced to 4 KiB, down from the original plan of 8 KiB.

3.5.2.3. *Efabless Corporation Bankruptcy*

Risk Realization

- The Efabless corporation experienced funding issues and declared bankruptcy. There is limited information available on why exactly this happened, but they will not be able to fabricate anything for the foreseeable future. This risk was not planned for by the team and occurred with little warning.

Implemented Mitigation

- Digital design moved to target an FPGA-based demonstration for the end of the project. While the fabricated chips would not have arrived back in time to demonstrate, more effort was previously being put into trying to harden the design for fabrication. Since fabrication is no longer possible, more effort was put into simulation and synthesis for an FPGA.

- Analog design became simulation only. Unlike the digital portion of the design, there is no straightforward way to examine the analog portion in real system without fabrication.

3.6. PERSONNEL EFFORT REQUIREMENTS

3.6.1. Planned Personnel Effort Requirements

Below is an effort analysis for each of our tasks listed in our Gantt chart:

Table 1: Planned Personnel Effort Requirements

| Task | Hours (estimate) | Justification |
|--|---------------------|---|
| Define Project Scope | 80 | The scope of this project is quite large and there are many different components we can implement. This will take substantial effort. |
| Research OpenRAM | 15 | There exists good documentation for OpenRAM and it utilizes generation scripts. This will only take about a week of effort. |
| Research RISC-V Core Generation | 15 | There exists good documentation for the Vex RISC-V generator and the generation is done via scripts. This will only take around a week of effort. |
| Wishbone Crossbar Creation | 50 | This component is inherently complicated since it must multiplex many different wishbone interfaces. This will take a large amount of effort. |
| RISC-V Core Creation | 50 | Although the core itself will be generated, an interface must be defined for the core such that it can interact with the rest of the design and be programmed. This is complex since we have many components in our design, and we must find a way to supply the core with a program. This will require a large amount of effort. |
| DFF RAM Macro Insanitiation | 40 | The RAM will be implemented using an existing DFF RAM macro on the Efabless marketplace. Since the macro is not wishbone compatible, we will have to write our own wishbone slave interface for it and do any address translation that is needed. This will require a large amount of effort. |
| Create DMA Controller | 60 | The DMA controller is an inherently complex component that will take a decent amount of research to understand how to create. In addition, we will have to create an interface for this component so it can interact with the rest of our design, which is complex. This will take a substantial amount of work. |
| Create AES 128-bit Encryption Hardware | 40 | This hardware will require research to implement, but there is good documentation and |

| | | |
|--|----|---|
| | | known hardware solutions. The main difficulty will be incorporating this into the rest of the design, which is not trivial. This will take a large amount of effort. |
| Create External Peripherals | 80 | We are including many peripherals in our final design, each of which have well documented designs, but can be complicated. Since we are implementing many peripherals and need to create an interface for them so they can interact with the rest of the design, this will require a substantial amount of effort. |
| Fabrication | 0 | With the Efabless shutdown, we can no longer fabricate. |
| Research PLL | 20 | The PLL is inherently complex, but there exists good documentation of hardware implementations. So, this will require a few weeks of effort. |
| Research Phase Frequency Detector | 20 | analog research complexity note The analog design for the RF module will be complicated since many different analog components are required to create each component. Good documentation exists, but many of our team members have not had prior experience designing these analog components. Therefore, it will take a few weeks to fully research each component. |
| Research Charge Pump | 20 | See "analog research complexity note" |
| Research Low-Pass Filter | 20 | See "analog research complexity note" |
| Research Voltage Controlled Oscillator | 20 | See "analog research complexity note" |
| Research Divider | 20 | See "analog research complexity note" |
| Create a Testing Environment for the PLL | 50 | Since the PLL is a complicated component, creating the testing environment for it will be complicated as well. There are many different waveforms we need to see and analyze and it is not trivial to setup an interface to accomplish this. Therefore, this will take a large amount of effort. |
| Design Each PLL Component | 80 | Create each component of the PLL on the Caravel board, then extract its parasitics to be used for simulation. |
| Simulate Each PLL Component | 80 | Simulate each component, testing expected behavior, loop characteristics and signal characteristics. |
| Finalize PLL Design | 50 | Review and revise the design based on simulations. Determine PLL characteristics to be tested after fabrication. |
| Create Design Documents | 80 | The design documents are quite large and will be edited throughout the course. This will take a substantial amount of effort. |

| | | |
|--------------------------------|----|--|
| Create Testing Documents | 80 | The testing documents will be quite large and must contain detailed instructions to test our many components. This will take a substantial amount of work. |
| Create Peripheral Testbenches | 60 | Since we have many peripherals, each of which can be complicated, creating testbenches for the peripherals will take a large amount of work. |
| Create RISC-V Core Testbenches | 60 | Due to the complexity of a RISC-V core, creating a solid testbench will be very difficult. There are many different signals we would need to capture and analyze in a meaningful way. This will take a large amount of effort. |

3.6.2. Deviations From Planned Personnel Effort Requirements

Many implementation efforts took significantly longer than originally anticipated due to issues with tools or bugs encountered during simulation. The actual times for each are outlined in the table below.

Table 2: Actual Personnel Effort Requirements

| Task | Hours (estimate) | Justification |
|---------------------------------|------------------|--|
| Research OpenRAM | 20 | OpenRAM was more challenging to get working than anticipated. Generation was ultimately not functional, and we decided to move to DFF RAM since it was silicon proven and in a convenient macro. It also easily translated to the FPGA. |
| Research RISC-V Core Generation | 20 | Researching the RISC-V core took around the time we expected. We had to learn the VexRISCV tool in addition to some parts of the scala programming language, which resulted in the hours listed. |
| Wishbone Crossbar Creation | 110 | The Wishbone crossbar turned out to be more challenging to create than anticipated. The original approach was to make it generic in Verilog, but due to lack of experience and language limitations, this turned out to not be possible. The approach then pivoted to a Python script, which could write out a Verilog file. |
| RISC-V Core Creation | 40 | After the research was performed, generating the RISC-V core was quite straightforward and integration went smoothly, so our actual hours closely matched our predicted ones. |
| DFF RAM Macro Insanitation | 60 | Instantiating the DFF RAM macro and creating a wishbone interface for it took about how long we predicted. However, we did not take into consideration the added time to configure OpenLane to use this macro in the layout. Due to the realized risk of new tools, this added many hours of work. |

| | | |
|--|-----|--|
| Create DMA Controller | 0 | Due to extra time spent on the crossbar, RAM, and peripherals, this was cut from the project scope due to time constraints. |
| Create AES 128-bit Encryption Hardware | 40 | After conducting more research, we found that it would be easier to use a pre-existing open-source implementation of AES 128-bit encryption rather than creating our own. This still required a significant amount of time to research our options and choose which one would be the best for our project. |
| Create External Peripherals | 55 | I2C was challenging to get working for a variety of reasons. This was due to lack of experience with the protocol and mismatches between the simulator and hardware. |
| Fabrication | 0 | Fabrication was not performed due the Efabless shutdown discussed earlier. |
| Research PLL | 20 | Researching the PLL did not exceed the expected time due to expansive documentation and resources available to learn from. |
| Research Phase Frequency Detector | 20 | Analyzing different PFD structures and what is the simplest to implement using the standard cell library that Xschem provides. |
| Research Charge Pump | 20 | Lack of knowledge and intuition of complex MOS structures and their anomalies. |
| Research Low-Pass Filter | 20 | The loop filter has huge effects on the damping of the PLL transient response as well as most other loop characteristics. Researching and understanding these effects. |
| Research Voltage Controlled Oscillator | 20 | Researching different types of VCO circuit topologies and which can operate at high frequencies did not take much time. However, trying to familiarize myself with characteristics and how it affects the loop dynamics took some time. |
| Research Divider | 20 | Researching the divider proved difficult due to a lack of detailed documentation on implemented designs. This made it challenging to determine what designs could work for our project. |
| Create a Testing Environment for the PLL | 50 | With the given tutorials from Efabless and ISU Chip Forge, we were able to set up the testing environment in the anticipated time frame. |
| Design Each PLL Component | 100 | The unknown tools made component design take far longer than anticipated. We struggled with several errors that slowed progress, and it was difficult to find help because very few people have experience with these tools. |
| Simulate Each PLL Component | 100 | The unknown tools stretched the time taken to simulate components. The lengthy computing time for some of the more comprehensive simulations also accounts for the additional time to simulate. |

| | | |
|--------------------------------|-----|---|
| Finalize PLL Design | 50 | Most components underwent several revisions before being finalized. |
| Create Design Documents | 150 | Significant time was committed to creating thorough documentation to facilitate our project's use as a learning tool. |
| Create Testing Documents | 20 | Creating annotated testing results and processes |
| Create Peripheral Testbenches | 5 | Since only the I2C peripheral was created, the testbench was straightforward to confirm that reads, writes, repeated starts, and stops worked correctly. |
| Create RISC-V Core Testbenches | 15 | This was straightforward after it was figured out how to load programs onto the second core. All connected interfaces on the core were easily confirmed to function properly. |

3.7. OTHER RESOURCE REQUIREMENTS

We require a testing platform to test our design, which is provided by Efabless and ChipForge. Efabless provides documentation on how to use existing open-source tools to simulate our digital and analog designs, which became our main testing approach throughout this project. In addition, the ChipForge co-curricular has provided ARTY A7-100T FGPAs and tools to load the entire Caravel harness with our design to them. This enables us to test our designs in real hardware so we can have greater confidence that our design will work after fabrication. In addition, we require tools to develop the digital and analog designs and layouts. Efabless and ChipForge have provided documentation and configuration for open-source tools that enable the creation of the designs and layouts for our project. These are all the resources we need to implement but not fabricate our proposed design. Originally, we required a means for fabrication, which was provided by Efabless. However, due to the Efabless shutdown, fabrication will not be possible through them. So, we require another means of fabrication, which is not currently known.

4. Design

4.1. DESIGN CONTEXT

All radio MCUs currently available are closed source and use proprietary architectures. This makes it more difficult for members of ChipForge, a co-curricular at ISU, and professors doing research to understand their inner workings and tailor them to their specific needs. Our design focuses on being easy to understand as well as open-source, which will allow users to investigate the implementation and customize it for their own needs if they have access to the fabrication resources. This is also supplemented with in-depth documentation, to make the design easy to understand for people with a basic level of knowledge of electrical and computer systems.

4.1.1. Broader Context

The primary community targeted by our design is college students and professors, since they would likely benefit the most from having access to an open-source radio MCU. Since the product we are creating will likely be lower performance as a tradeoff for being open-source, it is unlikely to impact the microcontroller industry significantly. The biggest societal need being addressed is the need for transparency in microcontroller design, something that is not addressed by many major companies in the space.

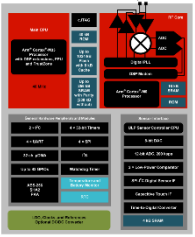

Table 3: Broader Design Context



| Area | Description | Examples |
|------------------------------------|--|--|
| Public health, safety, and welfare | Our project will benefit people who are college students and professors by providing a learning platform that can perform a variety of tasks often asked of MCUs. It is unlikely to have a significant impact on companies or organizations that already have a foothold in the MCU market. | <ul style="list-style-type: none"> • Extensible design • Documentation allows students with basic knowledge to use |
| Global, cultural, and social | <p>The academic community has a strong tradition of sharing resources and information, and the open-source nature of our design helps us to continue this tradition by making our work available to be built on by others.</p> <p>The team at Efabless fostered a community of students and professionals interested in chip design. When Efabless closed, it became apparent how important maintaining that community is. It facilitates the sharing of ideas and expertise among its members and results in more ambitious projects.</p> | <ul style="list-style-type: none"> • Open-source code • Documentation • Contributing to an ISU co-curricular • Chip design community migration |
| Environmental | There are two potential sources of environmental impact for the project, the impact for fabrication and the energy consumption by the device itself after fabrication. Neither of these are likely to be significant, since only a small number will be fabricated, and the final design will consume a small amount of power. | <ul style="list-style-type: none"> • Fabrication impact • Power consumption during operation |
| Economic | Our product will end up being more expensive per unit than existing products for customers, since it is being fabricated in small numbers. This is not something within our control, since small fabrication runs of ASICs are typically | <ul style="list-style-type: none"> • Higher unit cost • Fabrication cost covered by university |

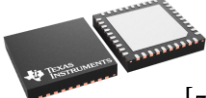
| | | |
|--|--|--|
| | expensive regardless of the process or vendor used. However, funding from the university has been secured to cover these costs, so this will not directly impact the primary users at ISU, only other potential users. | |
|--|--|--|

4.1.2. Prior Work/Solutions

Table 4: Market Research

| Product Services and Design <i>What is the product?</i> | Unique Value Proposition <i>What makes this product unique?</i> | Product Advantages <i>What are the things that provide a leg up?</i> | Product Disadvantages <i>Where might drawbacks exist?</i> | User Pros <i>What do users like about the product?</i> | User Cons <i>What do users NOT like about the product?</i> |
|--|--|---|--|---|--|
|  <p>TI CC1352P [1]</p> | <ul style="list-style-type: none"> - Thread, Zigbee, Matter - Bluetooth - Low power consumption | <ul style="list-style-type: none"> - Low power consumption while supporting Bluetooth - Supports multiple radio protocols | <ul style="list-style-type: none"> - Only use 2.4GHz radio frequencies which can be more than necessary for protocols like Zigbee | <ul style="list-style-type: none"> - Part of simple link system with common simple development environment | <ul style="list-style-type: none"> - Not open-source - Designed for general use - Specific applications can do better |
|  <p>Espressif ESP32 [2]</p> | <ul style="list-style-type: none"> - Bluetooth & WIFI - Microcontroller - Multicore - Low cost | <ul style="list-style-type: none"> - Wide variety of users - Used in low power IoT products - ESP-NOW Protocol | <ul style="list-style-type: none"> - Proprietary CPU architecture, limited support - Documentation can be lacking - 512kB memory: not enough for some memory-heavy applications | <ul style="list-style-type: none"> - Very affordable - Easy to use libraries - Various resources - Collaborative online community for hobbyists - Can program with Arduino IDE | <ul style="list-style-type: none"> - Low Range - Only - High power consumption for some peripherals |

| | | | | | |
|---|--|--|---|--|---|
|  <p>[3]</p> <p>Raspberry Pi Pico W</p> | <ul style="list-style-type: none"> - Wi-Fi and Bluetooth 5.2 support - PIO state machines - MicroPython support - Bootloader allows software to be loaded without a special programmer | <ul style="list-style-type: none"> - PIO state machines allow for flexible peripheral allocation - Dual core to allow one core to handle radio and one to handle application | <p>264 kB memory may not be enough for some applications</p> | <ul style="list-style-type: none"> - Cheap - Good documentation - No need for extra tools for programming | <ul style="list-style-type: none"> - C/C++ SDK setup can be painful relative to other MCUs |
| <p>STM32WB09KEV6 TR</p>  <p>[4]</p> | <ul style="list-style-type: none"> - Has multiple run modes, notably, for low power - ARM Mo processor (well-known architecture) - Multiple supported frequency bands - Designed for ultra-low power | <ul style="list-style-type: none"> - This microcontroller is designed for ultra-low power applications, so it is viable to use when power consumption is a concern - Small form-factor | <ul style="list-style-type: none"> - Has an ARM Mo, which is not super powerful - When in ultra-low power mode, functionality is heavily limited. | <ul style="list-style-type: none"> - Not generally purchased by individuals for hobbyist use | <ul style="list-style-type: none"> - Not generally purchased by individuals for hobbyist use |

| | consumption | | | | |
|---|--|---|---|--|--|
|  <p>[5]</p> <p>TI CC2540F256RHAR</p> | <ul style="list-style-type: none"> - Low energy SoC - True Single-Chip BLE Solution that can run both application and BLE protocol stack | <ul style="list-style-type: none"> - Low-power - True system-on-chip (SoC) - Cost-effective - In-system programmable flash memory | <ul style="list-style-type: none"> - Limited to BLE only - Less powerful processor compared to competitors - Cannot support complex, higher power applications | <ul style="list-style-type: none"> - Cost effective - Low power consumption - BLE implementations/support | <ul style="list-style-type: none"> - Strictly limited to BLE implementations - Not suitable for complex tasks. |

The primary advantage our design will have over these others on the market will be documentation and its open-source nature. Due to process constraints, it is unlikely that our design will be capable of competing with many of these designs on performance, since they mostly use newer processes than what we have available, which are close to two decades old. However, since ChipForge typically deals with smaller programs for learning purposes, this is an acceptable tradeoff, since they would be unlikely to use the extra performance of the other solutions.

4.1.3. Technical Complexity

The project consists of three main engineering design systems, analog/RF, digital, and cyber security. Each system consists of multiple subsystems, each containing one or more components. We are implementing all these components using our understanding of complex circuit design principles, computer architecture hierarchies, as well as hardware and software security standards. For example, the PLL is a closed loop system, in which its output depends on the individual gain and noise levels of each subcomponent as well as the closed loop transfer function. The Wishbone crossbar uses interconnected circuitry to enable communication between N number of masters to M number of slaves. The cyber security component design depends on strong foundations of the AES encryption algorithms. Implementing a fully functioning radio microcontroller from scratch mandates the collaboration of multiple cross functioning engineering teams and years of design and testing. This is why our team will be scoping the project and implementing some of the radio microcontroller systems. This is why our limited implementation does not enable transmit or receive; however, it layers the foundations for other senior design teams to further contribute to this project. Our limited implementation consists of the following systems:

1. Analog/RF subsystem:
 - a. Phase Locked Loop:
 - i. Phase frequency detector (PFD)
 - ii. Charge Pump (CP)
 - iii. Loop filter
 - iv. Voltage controlled oscillator (VCO)
 - v. Divider
 - b. Digital to analog converter
2. Digital subsystem:
 - a. RISC-V processor
 - b. Wishbone bus crossbar
 - c. DFF RAM
 - d. Peripherals
 - i. Timers
 - ii. GPIO
 - iii. I2C
3. Cyber security subsystem:
 - a. AES 128bit encryption

4.2. DESIGN EXPLORATION

4.2.1. Design Decisions

4.2.1.1. *ZigBee Wireless Protocol*

One of the first major decisions our group had to make was what wireless communication protocol we wanted to implement in our design. After researching what protocols some other radio microcontrollers support, we found that Bluetooth and ZigBee were some of the most common with many market options supporting both. After discussion with Professor Duwe and members of the Chip Forge club we concluded that ZigBee would be best for our design. Bluetooth could eventually be added later but we wanted to focus on getting one wireless communication protocol working first. We made this decision due to ZigBee using sub-GHz radio frequencies which made tolerances for design layouts far laxer.

4.2.1.2. *Reducing Scope*

When beginning this project with Professor Duwe we did not know the full scope of what would be required to design a radio microcontroller. So, our first couple weeks of work were researching just that. We quickly came to realize that the scope of this project was much larger than we originally anticipated, and that this project would take multiple teams across a couple of years. We then had to decide which components of our complete design we would try to complete before passing the project on to the next group. We chose to start with implementing one of two RISC-V cores, a wishbone bus crossbar, DFF RAM, one DMA Engine, the flash controller, security acceleration, digital to analog converter, phase-locked loop, reference oscillator, modulator, and some smaller timing components. While we will not have a fully functioning radio microcontroller with just these components, we aim to be able to create a product able to send data wirelessly by the end of our project.

After working through the implementation portion of the project, some additional scope had to be cut. This included the DMA engine and the UART and SPI peripherals. This was done because while they are useful features, they aren't critical to creating a minimum viable product since the running of programs and transferring of data could be done with just the crossbar, RAM, extra core, and I2C peripheral. In reducing the scope, we were able to take components further through the design and testing process than we would have been able to with a larger scope.

4.2.1.3. *AES Encryption*

We knew from the start that we wanted to include some level of security on our microcontroller, not only to secure it, but also so it can be used to teach wireless security. Encrypting data is essential for wireless communication as radio waves are very easy to intercept and read. By making the data unreadable without a key we largely negate this risk. There are many encryption methods that could be used but we decided to use AES as it is what is required by the ZigBee protocol as well as being one of the most popular and secure encryption methods.

4.2.1.4. *PLL Divider*

The design for the PLL divider was more complicated than we initially thought. Because the division is required to operate at 930MHz, a simple programable flip-flop divider won't work. Our research resulted in two potential designs, fractional N, implemented as a first order sigma delta

divider, or integer N, implemented as a dual modulus divider. We decided to use the fractional N divider because it creates less in-loop noise. The reduction in noise results in laxer benchmarks for other components in the PLL.

4.2.1.5. PLL Phase Frequency Detector

The PFD design is the simplest subcomponent of the PLL yet there exist different design implementations. The widely common circuit topology of the PFD uses the XOR, JK flops, or D-flip flops. The XOR implementation is sensitive to the input duty cycle and will lock with phase error if both inputs are not exactly 50% duty cycle. Unlike the exclusive OR implementation, the DFF implementation is not sensitive to the input duty cycle. It is also easier to implement and less noisy than the JK flip-flop implementation. This is the main reason that it has become the industry standard in PFD design.

4.2.1.6. PLL Voltage Controlled Oscillator

Various VCO design circuits are used in PLLs. The main two circuit families are ring and LC oscillators. The ring oscillator is generally easier to implement, requires less area, and has a more tunable frequency range than the LC oscillator but has more phase noise. This is the main reason we decided to implement a current-starved ring oscillator VCO.

4.2.1.7. PLL Charge Pump

The charge pump circuit could be implemented with 4 transistors. However, because it controls the voltage on the VCO, any mismatching in current source/sink will cause the VCO to set a wrong frequency, causing oscillation. The focus of any extra circuitry will be used to charge sharing, and other signal transients.

4.2.2. Ideation

The following describes the solutions presented for our wireless protocol and our PLL divider. Each solution is given a summary explaining its functionality and appeal.

4.2.2.1. Wireless Protocols

With various wireless protocols, we identified potential options based off their widespread use and documentation availability. The ones we found that were most commonly used across a majority of the microcontrollers on the market utilized some form of Wi-Fi, Bluetooth, or Zigbee protocols. The five major options we considered were Zigbee, BLE (Bluetooth Low-Energy), Wi-Fi (Wireless LAN), LoRa, and NB-IoT.

4.2.2.2. Zigbee Wireless Protocol

Zigbee was one of the first options we considered, since it allows for lower frequency rates than other wireless protocols. Specifically, it supports both 2.4 GHz and 915 MHz frequencies, which was more applicable to our design due to the nature of the RF module we are implementing. Zigbee also has relevant documentation regarding the IEEE standard 802.15.4, (since the protocol itself is built on-top of this standard) which specifies the technical standards of low-rate wireless personal area networks.

4.2.2.3. BLE Wireless Protocol

BLE or Bluetooth Low Energy was another option we highly considered working with, because of its low power consumption, as well as its ability to support 2.4 GHz and the various amounts of documentation available. The major downside was that BLE does not support lower frequencies, and although it supports versions of Bluetooth up to Bluetooth 5.0, the lack of low frequency support is ultimately why we choose not to proceed with BLE.

4.2.2.4. Wi-Fi Wireless Protocol

Wi-Fi as an option was considered, as it operates on 2.4 GHz and 5 GHz frequencies and supports various IEEE standards 802.11a, 802.11b, 802.11ac, and more, proving that there are tons of documentations on implementations, protocols, and specifications for wireless communications, making implementation easier. It is also the most widely used wireless communication protocol. Similar to why BLE was not considered, the lack of low-rate frequency support, and ultimately did not fit our projects goals and specifications that utilize an RF module.

4.2.2.5. LoRa Wireless Protocol

The LoRa protocol utilizes radio communication techniques to achieve low power and long-range communication. It also supports sub-gigahertz radio frequency bands and enables long-range transmissions. While it contains a lot of the components we would like in our design, it ultimately doesn't work well in data transmission with low data rates, and slow data transmission. It also would require its own network for deployment, which would limit its accessibility and ease of use for our intended users.

4.2.2.6. NB-IoT Wireless Protocol

The NB-IoT protocol is a low-power wide-area network meant to connect devices that have low bandwidth across long distances. Its low power consumption, ability to connect across long distances, and low frequency support makes it enticing as a design choice, but has limited data rates, latency issues, and coverage limitations (specifically indoors) despite being able to connect across long distances. Its implementation also highly depends on the existing network infrastructure. As our microcontroller is likely being utilized in environments with heavy infrastructure such as large buildings and labs throughout the university, it didn't make sense to implement since it wouldn't fit our intended user's needs.

4.2.2.7. PLL Divider Designs

Of the various options for frequency dividers, we selected the first order delta sigma and the dual modulus dividers because of their ability to operate in the GHz range, their common use in frequency synthesizers, and their relatively simple design.

4.2.2.8. First Order Delta Sigma

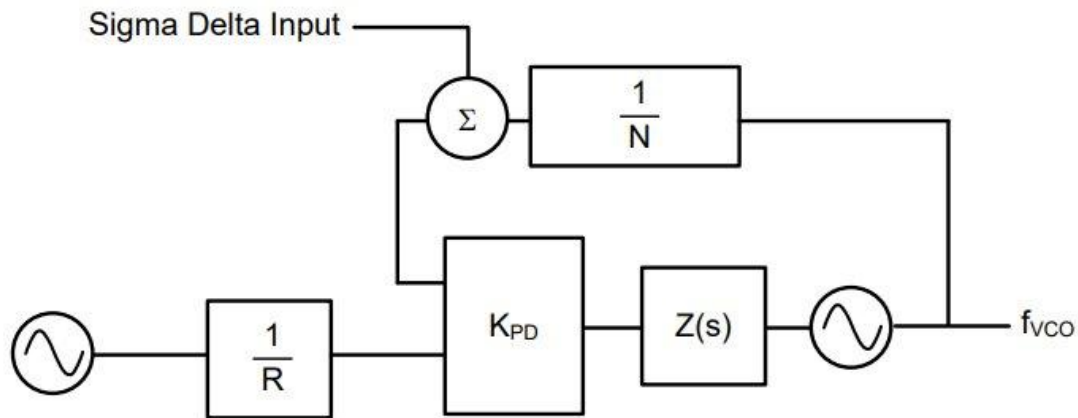


Figure 6: First Order Delta Sigma

The first order delta sigma design uses fractional division. This allows the reference frequency to be higher than the channel width. In our design, it means we can use a 10MHz reference frequency instead of a 1MHz one, which results in a maximum division of 92.8 instead of 928. This reduction in division is what allows this divider to operate at higher frequencies. The fractional division is accomplished by alternating integer division values from a programmable dual modulus divider in a sequence (Sigma Delta Input) that averages to the desired value. For example, dividing by 92.8 would be achieved by dividing by 93 for 4 cycles and 92 for one cycle. This oscillation results in an average division of 92.8, smoothed out by the loop filter.

4.2.2.9. Dual Modulus

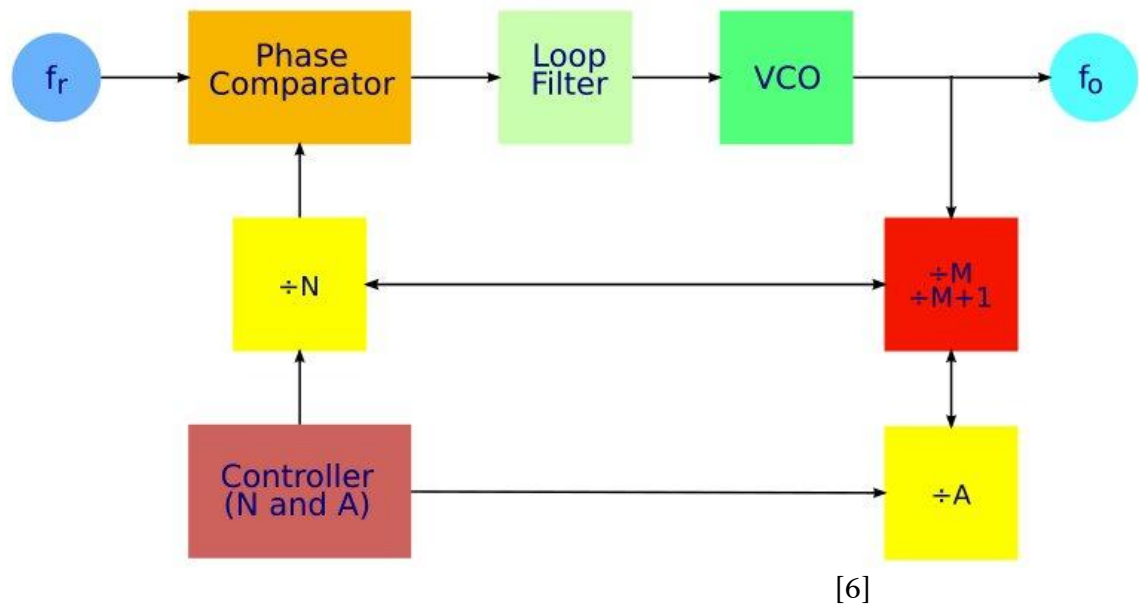


Figure 7: Dual Modulus Divider

The dual modulus design uses integer division which limits the reference frequency to the channel width, 1MHz. To achieve high frequency division, this design uses a fixed prescaler at m and $m + 1$. This prescaler divides the frequency down to a functional operating frequency for a simple programmable flip-flop divider, N , to create the total divisor. Just a prescaler of $M+1$ and N results in a division ratio of $N(M+1)$ because M is fixed the divider would only be create a divisor as a multiple of $M+1$. To allow the divider to create every division ratio instead A second programmable divider A is added where $A < N$. A and N count down at the same time and when $A = 0$ the prescaler is change from $M+1$ to M . This results in a division ratio of $A(M + 1) + (N - A)M$. If $N \geq M$, then every division ratio is possible.

4.2.2.10. PLL PFD Designs

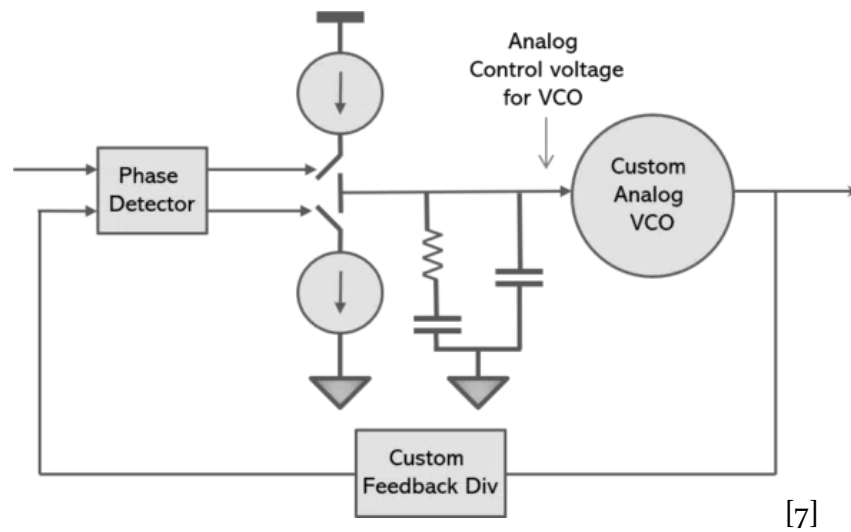


Figure 8: PLL Diagram

The phase frequency detector is the first component of the PLL. It detects the phase error between the reference oscillator signal and the feedback signal from the divider. There are multiple circuit topologies that could be used to make the PFD. However, a D flip-flop implementation is the standard practice in the industry because it is simple to implement, does not depend on the duty cycle, and has a constant gain of $1/2\pi$.

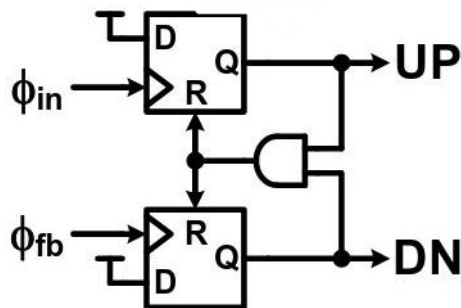
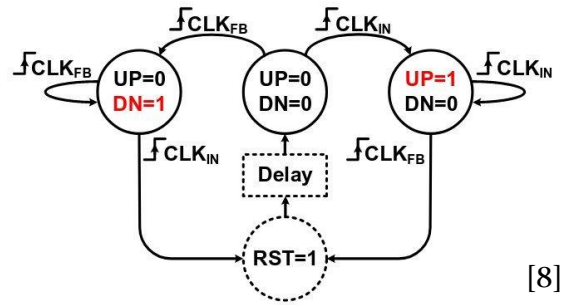


Figure 9: Phase Frequency Detector

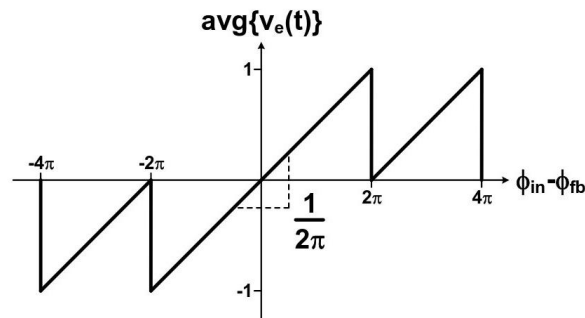


[8]

Figure 10: Phase Frequency States

The UP and DN signals from the PFD will later be used to drive a charge pump, which sources or sinks current to the loop filter, thus dynamically changing the voltage input of the VCO. The UP signal is reflected by a positive 1 because it sources current and the DN signal is reflected by a negative 1 because it is responsible for sinking current from the loop filter.

Because the charge pump and PFD combined should not be able to switch slower than twice the delay from the PFD output to the Charge Pump. This causes a defective region formally known as the dead zone, which can cause lower gain and increased jitter. To avoid dead zones, a delay has been inserted between the RST signal of the DFFs and the AND gate. This will ensure that both UP and DN signals will stay high for transistors to switch properly. As both UP and DN are high, the net current (ICP) passing through the loop filter will be zero, causing no net change to the output frequency.



[8]

Figure 11: PFD Gain Without Dead Zones

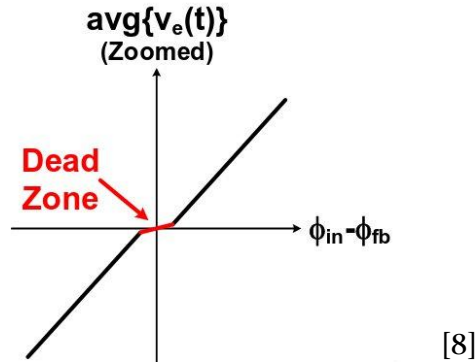
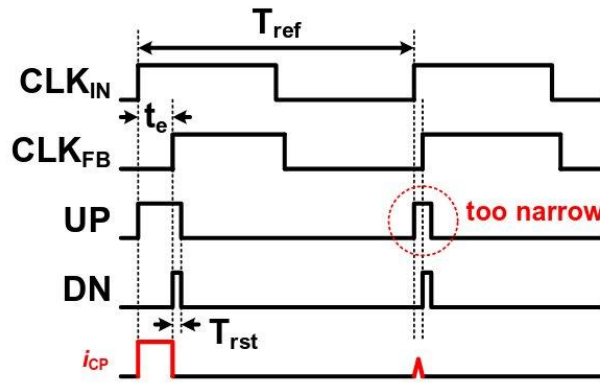


Figure 12: PFD gain with Dead Zone



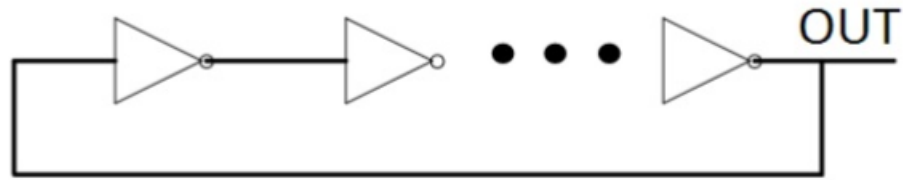
[8]

Figure 13: Dead Zones Jitter

4.2.2.11. PLL VCO Designs

Due to the area restriction of the die and design complexity, we have decided to implement a current starved ring oscillator. This type of VCO is very common among PLL designers because of its small area and wide frequency tuning range.

The current-starved ring oscillator functions the same way as an N stage inverter ring oscillator works. It connects a series of odd number of inverters in series, connecting the output node to the input of the first inverter. Because there are an odd number of inverters, the output and input node constantly change causing an oscillation. This oscillation is directly related to the delay of each of the inverter stages ($f = 1/2NT$) where N is the number of stages and T is the delay per inverter.

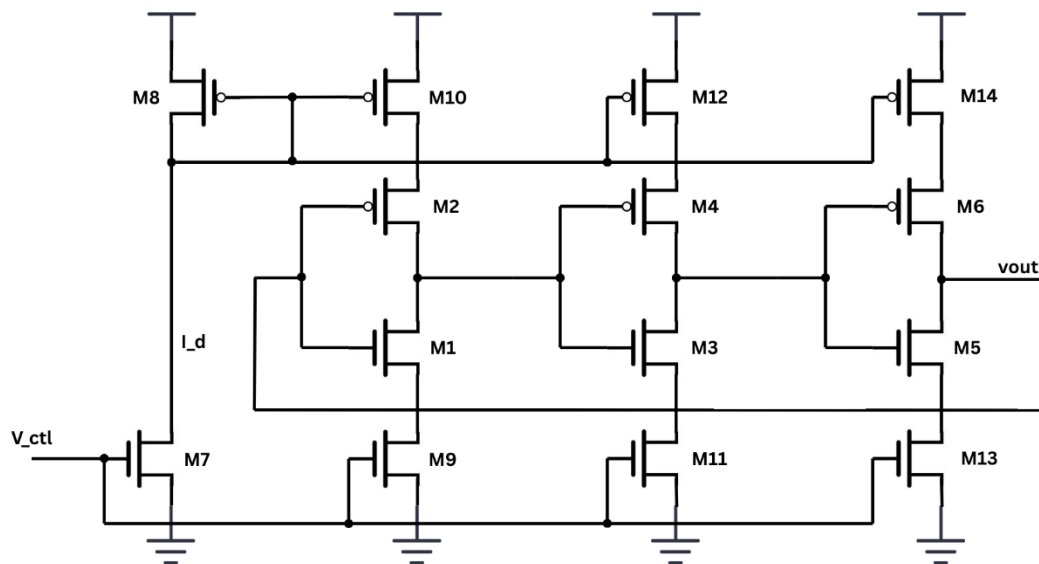


[8]

Figure 14: Ring Oscillator

The current-starved ring oscillator behaves in the same way. However, we add a current mirror, which is controlled by a control voltage to tune the latency of each stage, thus changing the output frequency. The minimum viable operating frequency occurs when the input voltage is less than the threshold voltage of the at the control voltage FET. While the maximum linear operating frequency occurs when the excess bias voltage of PMOS current mirrors M10, M12, M14 are greater than the PMOS threshold voltage.

Given these equations, we plan to design a current starved ring oscillator that can linearly operate at the Sub-GHz ZigBee frequency range. There have been designs within the online community that were able to achieve such a high frequency using the Skywater 130nm process. Refer to appendix 8.3.2 for detailed analysis.



[8]

Figure 15: Current-Starved Ring Oscillator

4.2.2.12. PLL Charge Pump Designs

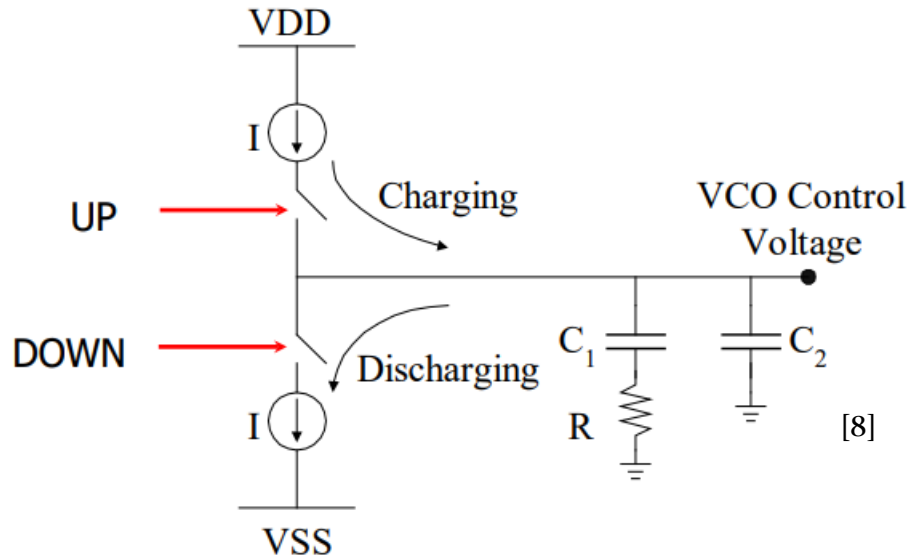


Figure 16: PLL Charge Pump & Loop Filter

The conceptional design of the charge pump component is fairly simple, yet it requires careful timing analysis. The charge pump receives the UP and DN signal from the PFD component and sources or sinks current to the loop filter, thus changing the VCO Control Voltage. Because PMOS is a better current source than NMOS, the transistor that switches the UP signal will be a PMOS transistor preceded by an inverter. However, the transistor switches the DOWN signal will be an NMOS because it is sinking current. Because the added delay in the UP stage causes a mismatch in timing, a series of buffers will be added to both UP and DOWN stages to match that delay.

3/2 Inverter Path

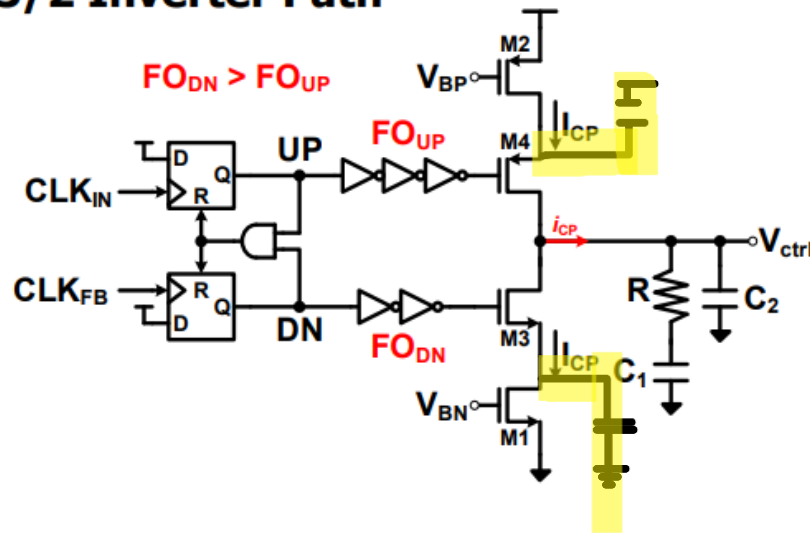


Figure 17: 3/2 Inverter Path

One of the main issues with this design is charge sharing. When both the DOWN or Up switches are off, a zero net current should pass through. However, the highlighted M1 Drain to source capacitors were initially at GND and/or VDD, and is now connected to a different potential (V_{ctrl}). This causes a possible voltage level disturbance at the control voltage node, causing slight fluctuations in output frequency. This results in spurs in the frequency domain. Clock feedthrough and charge injection also introduces irregularities which ultimately result in spurs.

In order to minimize this effect with the current structure, I will shift the switch FETs away from the output voltage by switching M1 and M3 as well as M2 and M4 and redirect the signals accordingly.

This conventional topology is commonly referenced in most research papers and books due to its simplicity. A fully differential charge pump circuit is a more complex structure that aims to reduce these effects, but it is not a feasible implementation for this project.

4.2.2.13. User Area RAM

After the OpenRAM investigation and lack of success with generating SRAM macros, another option of DFF RAM was investigated. This RAM uses many D flip-flops in place of SRAM cells to create a module that has the same functionality as RAM. This comes at a cost of density, with the same amount of memory taking up significantly more space. The exact difference is difficult to calculate, since the OpenRAM was not able to harden successfully. However, the DFF RAM was already silicon proven, meaning that it had already been fabricated successfully, and easily synthesized for an FPGA, which were two substantial advantages. If later teams have more success with generating SRAM, it would also be straightforward to replace the DFF RAM with it. Due to the large upsides of easy mapping to an FPGA and being silicon proven, the downsides of capacity were seen as acceptable, and so the project moved to use DFF RAM for all RAM modules.

4.2.3. Decision-Making and Trade-Off

The following discusses the characteristics considered when deciding what protocol and what divider to use. Each criterion is ordered by importance with each option's relevant information listed. After the criterion, there is a discussion about our decision and how it best meets the criteria.

4.2.3.1. Wireless Protocols

The following characteristics were considered when determining which protocol our microcontroller would follow. Frequency was the largest factor as higher frequencies would be more difficult to implement correctly. Next security was considered, we knew we would be implementing some sort of security for our radio communication so protocols with security requirements that matched our team members' skills sets were prioritized. Finally, the modulation technique is used by the protocol. This was considered last because it is not a part of our scope but will affect teams later on.

Frequency

Zigbee: 2.4GHz worldwide, 902MHz America

BLE: 2.4GHz

Wi-Fi: 2.4GHz, 5GHz

LoRa: 169MHz, 315MHz worldwide, 915MHz America

NB-IoT: Built onto cellular networks, bands between 600MHz and 1700MHz

Security

Zigbee: Advanced Encrypted Standard (AES)

BLE: Encrypted Advertising Data (EAD)

Wi-Fi: Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA)

LoRa: Advanced Encrypted Standard (AES)

NB-IoT: Inherits Security from cellular network

Modulation

Zigbee: Binary Phase Shift Keying

BLE: Gaussian frequency shift modulation

Wi-Fi: Binary Phase Shift Keying, Quadrature Phase Shift Keying, QAM

LoRa: Proprietary spread spectrum modulation

NB-IoT: Orthogonal Frequency-Division Multiplexing

We decided to use the Zigbee protocol. In terms of frequency, Zigbee comes second, allowing the microcontroller to be sub-GHz by using the American bands. For security, Zigbee uses AES, a common security protocol that our team has experience with. Notably, LoRa operates at lower frequencies and uses AES as well. We decided to use Zigbee because of its simpler and open-source modulation scheme. The goal of our project is to be open-source and LoRa uses a proprietary form modulation. Lastly, Zigbee's ability to operate at 2.4GHz has the potential for future teams to modify our design into a multiprotocol radio.

4.2.3.2. PLL Divider Designs

The following characteristics were considered when determining which divider our PLL would use. The most important aspect of the divider was noise. Noise is the defining characteristic of a PLL so minimizing noise through the divider was a priority. Next, we looked at the magnitude of the high-speed division. The greater the divisor at high frequency, the more likely it is for the divider to fail. Lastly, we looked at the complexity of the design. A more complex design takes up more of the limited space on the die.

Noise

First Order Delta Sigma:

20log(10) less phase noise. Fractional spurring. Instantaneous frequency is not accurate.

Dual Modulus:

20log(10) more noise. No spurring. Instantaneous frequency is accurate.

High Frequency Division

First Order Delta Sigma:

Divide by 93 at high frequency

Dual Modulus:

Divide by 31(prescaler only) at high frequency

Complexity

First Order Delta Sigma:

M/M+1 is programmable, range: 90-93. N is programmable, range: 1-10. A is programmable, range 0-9.

Dual Modulus:

M/M+1 is a fixed prescaler, 30/31. N is fixed divider, 30. A is programmable, range: 2-28.

We decided to use the First Order Delta Sigma design. While it has a higher division ratio at high frequency, it is still very low. It is a more complex design, but we decided the lower noise was worth using a more complicated design. The Fractional spurring, caused by the periodic oscillation

between the two divisors, is difficult to quantify, however. Because of this, we plan to use the dual modulus design as a backup design in case the spurring results in an unusable signal.

4.3. FINAL DESIGN

4.3.1. Overview

At a high level, our design is composed of an analog component and a digital component. The analog component for our part of the project is a PLL capable of generating frequencies between 900 MHz and 928 MHz to transmit and receive using Zigbee. The digital component for our project consists of a processor to run user programs, peripherals to interact with other devices, and RAM to store information. These components are interconnected to form the final design.

As outlined in our requirements, there is a full system design that was planned out that encompasses all of what the client wanted for the project, and a smaller subset of components that contains what we implemented this semester. These are shown in the figures in section 4.3.2.

4.3.2. Detailed Design and Visual(s)

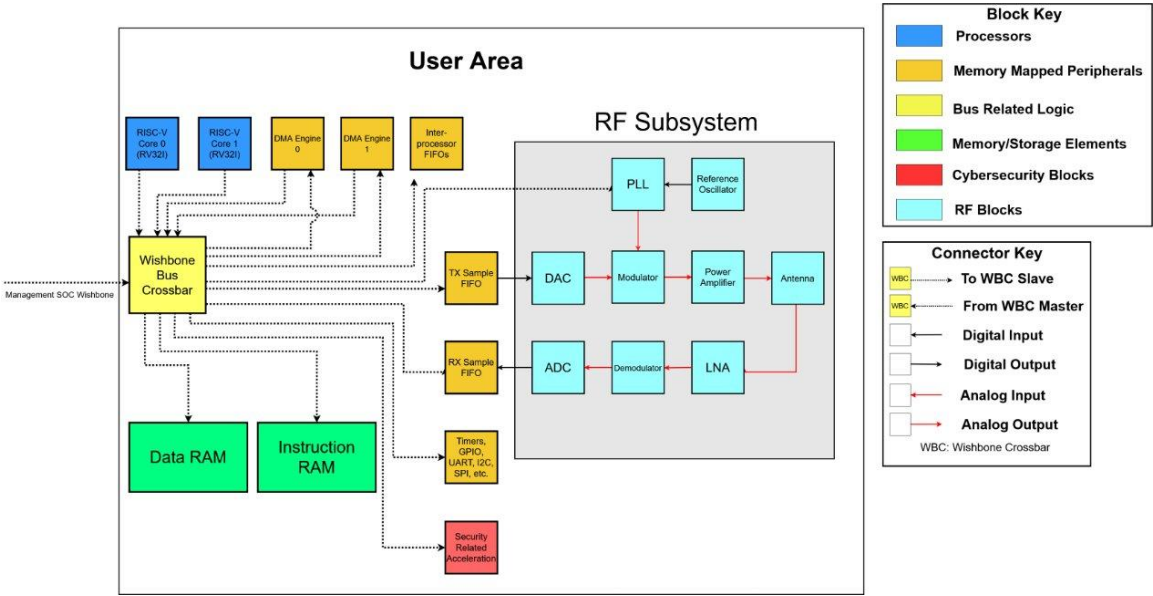


Figure 18: Full Design - Multi-Year

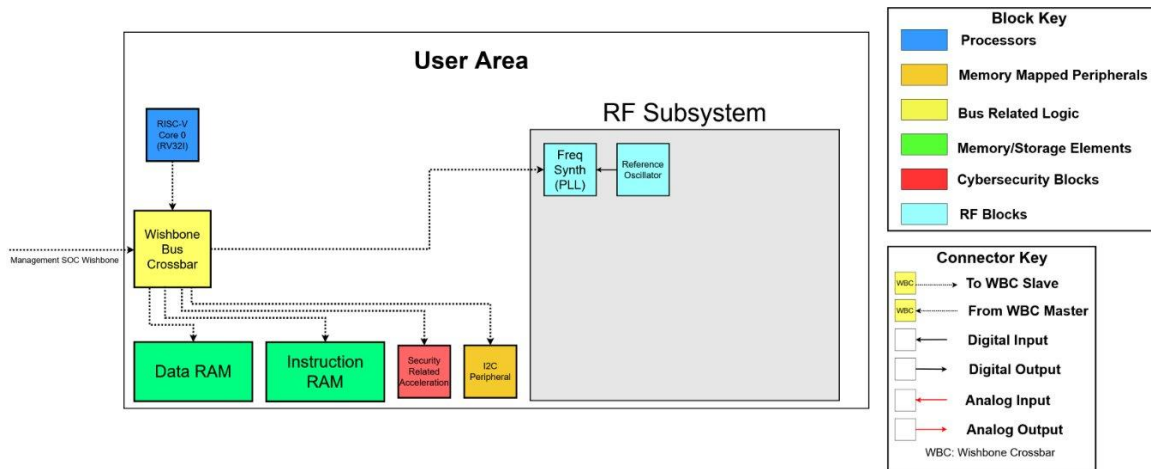


Figure 19: Spring 2025 Design

4.3.2.1. Analog Subsystem

The analog subsystem for this part of the project is a frequency synthesizer implemented as a phase locked loop (PLL) to operate 26 channels from 902 to 928MHz.

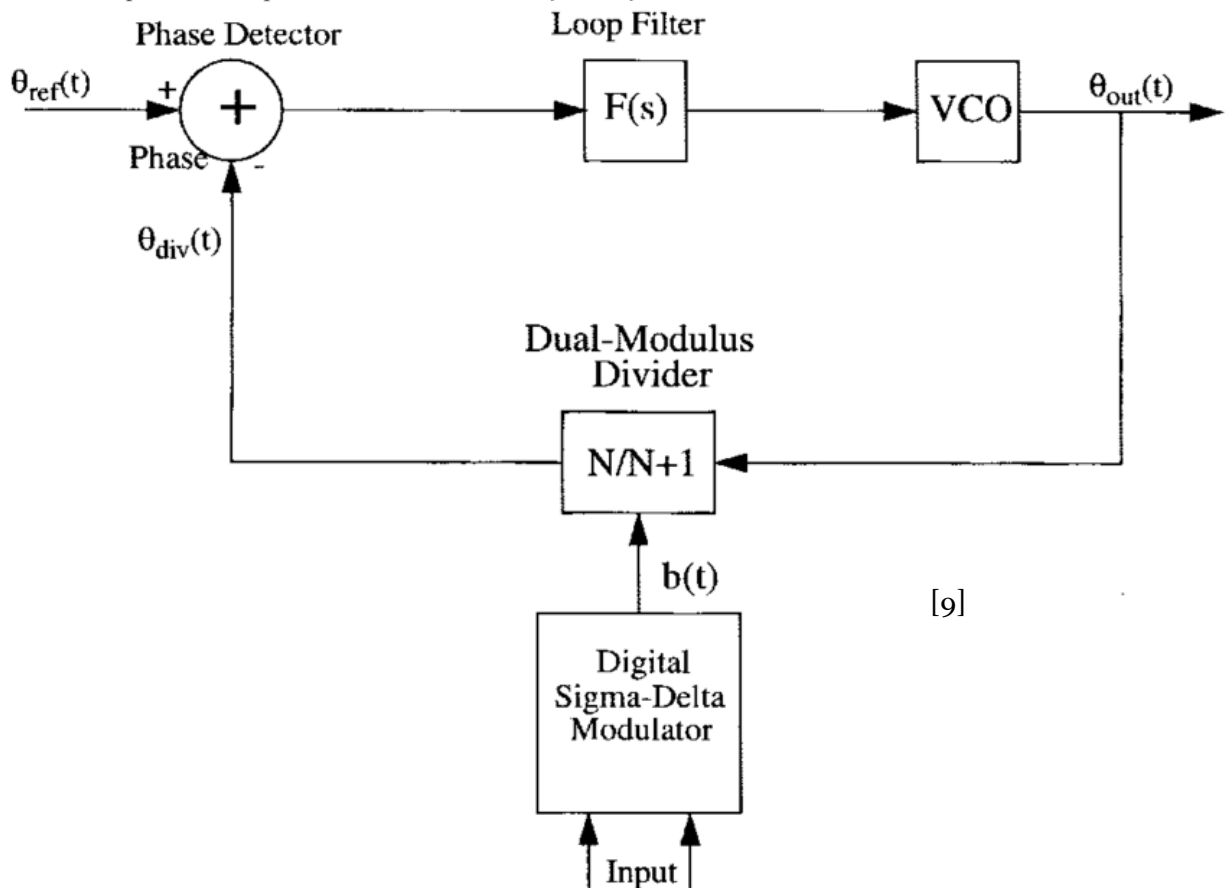


Figure 20: PLL Design

4.3.2.1.1. PLL Fractional N Divider

To achieve fractional division at high frequency, the divider is designed as a dual modulus divider with an accumulator to facilitate delta sigma modulation. Figure # shows the high-level architecture of the divider.

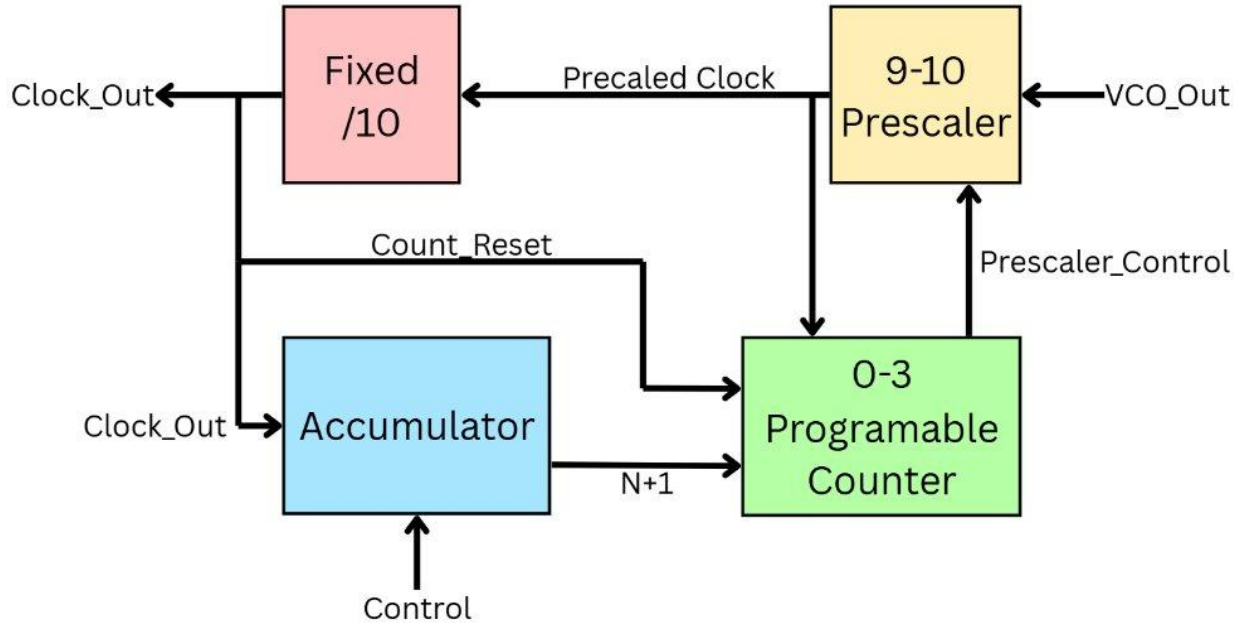


Figure 21: Fractional N Divider Design

4.3.2.1.1.1. Dual Modulus design

The dual modulus divider uses a prescaler, programmable counter, and a fixed divider, as shown in Figure 21.

The prescaler divides the input frequency by either 9 or 10 depending on the control signal from the counter, Prescaler_Control. Its output, Prescaled_Clock then serves as the clock for the fixed divider and the counter.

The counter counts a number of clock cycles (Prescaled_Clock) determined by its control signal, N+1. While counting, Prescaler_Control is set low. After reaching the set count value, the counter sets Prescaler_Control to high telling the prescaler to switch from dividing by 9 to dividing by 10. The output of the fixed divider, Count_Reset resets the counter to 0 to begin counting again.

The fixed divider further divides Prescaled_Clock down to complete the integer portion of the division, outputting Clock_Out which also serves as Count_Reset.

This design results in a denominator value of:

$$M \times (A - N) + N \times (M + 1)$$

Where: M, M+1 = prescaler values, N = counter value, A = fixed divider value

M and A are fixed therefore the denominator has a range of:

$$M \times N \leq M \times A + N \leq (M + 1) \times A, \text{ where } N \leq A.$$

Importantly, incrementing N by one only increments the denominator by one. This allows the divider to achieve its target value while maintaining the minimal step size.

4.3.2.1.1.2. Delta Sigma Modulation

To achieve fractional division, the accumulator varies the control signal to the counter between N and $N+1$ per Clock_Out cycle. This causes the denominator to vary between $M \times A + N$ and $M \times A + 1$. By varying the denominator at specific ratios, the divider can create an average denominator that is between N and $N+1$. As seen in Figure #, the PLL has a second order filter which can be leveraged to integrate the denominator fluctuation into an even average fractional value.

4.3.2.1.1.3. Choosing Values

Zigbee's frequency ranges from 902MHz to 928MHz in 1MHz channels. Our reference frequency is 10MHz. Therefore, the divider must be able to divide by 90.2 to 92.8 with a 0.1 step size. The dual modulus subset must be able to divide by 90, 91, 92, and 93, and the accumulator must be able to vary between N and $N+1$ at a ratio of $\frac{F}{10}$ where $0 \leq F < 10$.

With these constraints, $M \times A = 90$ and $(M + 1) \times A \geq 94$. The nearest integer solution to this system is $M = 9$, $A = 10$. Thus, $(M + 1) \times A = 100$. Since the maximum denominator required is 93, N needs only fill the range: $0 \leq N \leq 3$.

With these values, the dual modulus subset will consist of a 9-10 prescaler, a 0-3 programmable counter, and fixed divide by 10 divider.

4.3.2.1.1.4. Component Design – Prescaler

The prescaler is designed as a 0 to 9 counter and a 0 to 10 counter with Prescaler_Control arbitrating the reset point. This necessitates a four-bit counter using four flip flops. The Sky 130nm PDK is limited to D flip flops, so the design uses an XOR gate at the Data input of the flip flop to create a T flip flop as shown in Figure 22

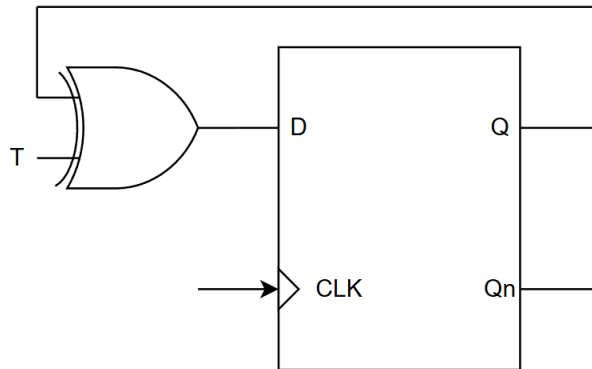


Figure 22: T Flip Flop

As $A \oplus B = A! \oplus B!$, the Boolean equation for D is either $D = Q \oplus T$ or $D = Q! \oplus T!$

Table 5 contains the truth table for the toggle enable, T, for each flip flop label 1-4 where 1 represents the least significant bit. C is the control signal, Prescaler_Control.

Table 5: Prescaler Truth Table

| Prescaler | | | | | | | | | | | |
|-----------|----|----|----|---|----|-----|----|-----|----|-----|----|
| Q4 | Q3 | Q2 | Q1 | C | T1 | T1! | T2 | T2! | T3 | T3! | T4 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | x | x | x | x | x | x | x |
| 1 | 0 | 1 | 0 | 1 | x | x | x | x | x | x | x |
| 1 | 0 | 1 | 1 | 0 | x | x | x | x | x | x | x |
| 1 | 0 | 1 | 1 | 1 | x | x | x | x | x | x | x |
| 1 | 1 | 0 | 0 | 0 | x | x | x | x | x | x | x |
| 1 | 1 | 0 | 0 | 1 | x | x | x | x | x | x | x |
| 1 | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 0 | 0 | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 0 | 1 | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 1 | 0 | x | x | x | x | x | x | x |
| 1 | 1 | 1 | 1 | 1 | x | x | x | x | x | x | x |

Because this component operates at the highest frequency, the design is focused on prioritizing NAND and NOR gates due to their lower gate delay and minimizing the total number of gates. To do this we use T! and Q! for flip flops 1, 2, and 3 when formulating the Boolean equations.

From this table we derived the Boolean formulas as follows:

$$T1! = c! \times Q4$$

$$T2! = Q1! + Q4$$

$$T3! = Q1! + Q2!$$

$$T4 = c \times (Q1 \times Q4 + Q1 \times Q2 \times Q3) + c! \times (Q4 + Q1 \times Q2 \times Q3)$$

T₄ can be rewritten as:

$$T4 = (c * (Q1! + Q4!) \times (Q1! + Q2! + Q3!) + c! \times (Q4! \times (Q1! + Q2! + Q3!)))!$$

This optimizes NAND usage while the total number of gates remains the same.

4.3.2.1.1.5. Component Design – Programmable Counter

The counter is designed to take a two-bit input, c₀ and c₁, setting the count value. The outgoing control signal, max, is low while the counter is counting and high when the counter has reached the target value. There is a reset signal that sets both flip flops to zero. Since the counter only needs to count to three, it uses two flip flops. Table 6 contains a truth table that determines the state of max based on c₁, c₀, reset, and the state of the flip flops, Q₁ and Q₂.

Table 6: Programmable Counter Truth Table

| 0-3 Counter | | | | | |
|-------------|----|----|----|---|-----|
| C1 | C0 | Q2 | Q1 | R | Max |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 |

Max is found to be:

$$max = R! \times (c0 \times c1 \times Q1! + c0 \times Q1! \times Q2! + c1 * Q2!)$$

The term $(c0 \times c1 \times Q1! + c0 \times Q1! \times Q2! + c1 * Q2!)$ in max determines if the counter should be counting. This was used to create equations for the flip flops:

$$D1 = R! \times Q1 \times \text{counting} + (R + \text{counting})! \times Q1$$

$$D2 = R! \times Q1 \times Q2 \times \text{counting} + (R + \text{counting})! \times Q2$$

The first term determines when to toggle from zero to one, the second term determines when to stay at one.

4.3.2.1.1.6. *Component Design – Fixed Divider*

This component is very similar to the prescaler, just lacking the multiplexer for divide by nine. Recall the Boolean equations from the prescaler:

$$T1! = c! \times Q4$$

$$T2! = Q1! + Q4$$

$$T3! = Q2! + Q1!$$

$$T4 = c \times (Q1 \times Q4 + Q1 \times Q2 \times Q3) + c! \times (Q4 + Q1 \times Q2 \times Q3)$$

By removing the c! terms and inverting T! to T, the Boolean formulas for the flip flops become:

$$T1 = 1$$

$$T2 = Q1 \times Q4!$$

$$T3 = Q2 \times Q1$$

$$T4 = Q1 \times Q4 + Q1 \times Q2 \times Q3$$

4.3.2.1.1.7. *Component Design – Accumulator*

The accumulator takes an input, F, and repeatedly adds F to a running count. When overflow occurs, the accumulator increases N by 1 for one output cycle. By controlling when overflow occurs, the accumulator creates the denominator of the fraction. The numerator is the input, F.

Recall that the accumulator needs to create a fraction of $\frac{F}{10}$. Since F is bounded by $F < 10$ the input is limited to four bits. Figure 23 shows a block diagram of the described design.

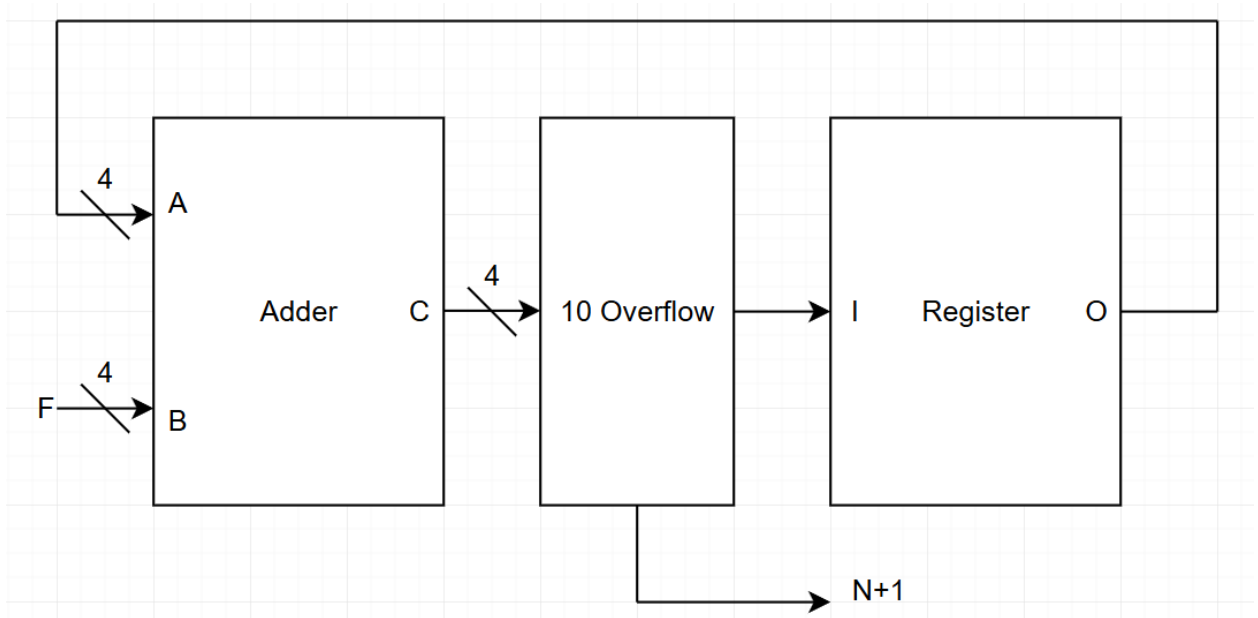


Figure 23: Accumulator Architecture

This design creates the most even distribution of N and $N+1$ at the given ratio. For example, if F is 3, the register will follow this sequence: 3, 6, 9, 2, 5, 8, 1, 4, 7, 0. Including the first cycle, cycles 4, 7, 10 have overflow. This matches the desired ratio of $\frac{3}{10}$ and distributes the variation as evenly as possible.

The adder is implemented as a full adder with access to the carry out for each bit. These are used to create overflow logic. Creating a truth table for the said logic with eight input is not practical, thankfully the limited overflow cases allow for the equations to be created without one. For the following equations $a0$ - $a3$ refers to the adder output bits and $c0$ - $c3$ refers to the carry bits.

$$D1 = a0$$

$$D2 = a1! \times c3 + (a3 + c3)! \times a1 + a1! \times a2 \times c3$$

$$D3 = a1! \times c3 + a1 \times a2 + a2 \times a3!$$

$$D4 = (a1 + a2)! \times a3 + c0 \times c3 \times (c1 + c2)!$$

To determine if overflow occurred:

$$Ov = a2 \times a3 + a1 \times a3 + c3$$

To increment N :

$$N0_out = N0_in \oplus Ov$$

$$N1_out = N0_in \times Ov \oplus N1_in$$

4.3.2.1.2. PLL Transfer Function and Loop Analysis

4.3.2.1.2.1. VCO Transfer Function

The VCO is not linear in terms of output frequency because the output is sinusoidal function, and the input is a DC voltage source.

$$VCO(s) = \frac{f_{out}}{V_{ctl}} = \mathcal{L}\{K \sin(\omega_0 t)\}$$

However, $VCO(s)$ is linear in terms of output phase

$$\varphi_{out} = \int_0^t \omega(t) dt$$

We define a constant K_{vco} and set it equal to $2\pi \frac{df_{vco}}{dV_{ctl}}$

$$\varphi_{out} = \int_0^t \omega(t) dt = \int_0^t 2\pi \frac{df_{vco}}{dV_{ctl}} V_{ctl} dt = \int_0^t K_{vco} V_{ctl}(t) dt$$

$$\varphi_{out}(s) = \mathcal{L}\left\{\int_0^t K_{vco} V_{ctl}(t) dt\right\}$$

$$\varphi_{out}(s) = K_{vco} V_{ctl}(s)$$

$$VCO(s) = \frac{\varphi_{out}(s)}{V_{ctl}(s)} = \frac{K_{vco}}{s}$$

4.3.2.1.2.2. Loop Filter Transfer Function

The transfer function of the loop filter is equal to its complex impedance which is equivalent to the output voltage divided by the input current.

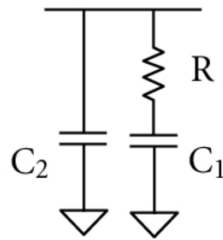


Figure 24: Loop Filter

$$Z(s) = \frac{V_{ctl}}{I_{cp}} = (R + \frac{1}{sC_1}) // \frac{1}{sC_2}$$

$$Z(s) = \frac{sRC_1 + 1}{s(sRC_1C_2 + C_1 + C_2)}$$

4.3.2.1.2.3. PFD/Charge Pump Transfer Function

The gain of the PFD and the Charge Pump is the slope of the line of on-current of the charge pump compared to the total phase.

$$K_{PD} = \frac{I_{CP}}{2\pi}$$

4.3.2.1.2.4. Closed Loop Transfer Function

$$\text{Open Loop TF} = G(s) = K_{PD}(s) * Z(s) * \frac{K_{VCO}}{s}$$

$$\text{The feedback divider TF} = D(s) = \frac{1}{N}$$

$$\text{Closed Loop TF} = H(s) = \frac{G(s)}{1 + G(s)D(s)} = \frac{K_{PD}(s) * Z(s) * \frac{K_{VCO}}{s}}{1 + \frac{1}{N} * K_{PD}(s) * Z(s) * \frac{K_{VCO}}{s}}$$

4.3.2.1.2.5. Stability Analysis

Closed Loop Transfer Function

$$H(s) = \frac{K_{PD}(s) * Z(s) * \frac{K_{VCO}}{s}}{1 + \frac{1}{N} * K_{PD} * Z(s) * \frac{K_{VCO}}{s}}$$

The loop dynamics and stability are hugely affected by the loop filter because it defines the bandwidth.

Substitute Z(s) with the proposed 2nd order filter,

$$H(s) = \frac{K_{PD}K_{VCO}(1 + sRC_1)}{s^3RC_1C_2 + s^2(C_1 + C_2) + s\left(\frac{RC_1K_{PD}K_{VCO}}{N}\right) + \frac{K_{PD}K_{VCO}}{N}}$$

Define a constant K

$$K = \frac{K_{PD}K_{VCO}}{N}$$

Using Routh Hurwitz Criterion to find stability limits compared to the loop filter resistors and capacitors values:

Table 7: Routh-Hurwitz Criterion

| | | |
|-------|-----------------------------|---------|
| s^3 | RC_1C_2 | RC_1K |
| s^2 | $C_1 + C_2$ | K |
| s | $\frac{KRC_1^2}{C_1 + C_2}$ | 0 |
| s^0 | K | 0 |

The sign of the second column is always positive regardless of the values of the resistor and the two capacitors indicating that the loop is always stable.

4.3.2.1.3. PLL Noise Transfer Functions and FOM

Inject noise at the output of each block and get the transfer function of the output relative to that input. We did not implement a reference oscillator divider in our design.

4.3.2.1.3.1. VCO Noise Transfer Function

$$Noise_{VCO} = \frac{1}{1 + G(s)D(s)} = \frac{1}{1 + \frac{1}{N} * K_{PD}(s) * Z(s) \frac{K_{VCO}}{s}}$$

4.3.2.1.3.2. PFD/Charge Pump Noise Transfer Function

To reduce the PFD noise, increase K_{PD}

$$Noise_{PFD} = \frac{1}{K_{PD}} * H(s) = \frac{1}{K_{PD}} * \frac{G(s)}{1 + G(s)D(s)} = \frac{1}{K_{PD}} * \frac{K_{PD}(s) * Z(s) * \frac{K_{VCO}}{s}}{1 + \frac{1}{N} * K_{PD} * Z(s) * \frac{K_{VCO}}{s}}$$

4.3.2.1.3.3. PLL Noise Figure of Merit (FOM)

To compare the noise between different PLLs which operate at various frequencies, this FOM is used to unify a noise metric. PLL_{Noise} is the actual measured noise floor in dBc/Hz over 1Hz, F_{PFD} is the operating frequency of the PFD, and N is the divider value.

$$PLL_{Noise\ Flat} = PN1Hz + 20\log(N) + 10\log(F_{PFD})$$

4.3.2.2. Digital Subsystem

The digital subsystem consists of two RISC-V cores, DFF RAM, a Wishbone crossbar, and peripherals to interface with other devices or accelerate basic functions. Due to constraints, the inter-processor FIFOs, one of the RISC-V cores, and the DMA engines will not be created during our part of the project but are included in the design since they will be part of the system in future iterations.

4.3.2.2.1. RISC-V Core

Several open-source options for generating a RISC-V core were investigated, since implementing this from scratch would be time consuming and be likely to result in errors that could result in non-working chips. The option that was chosen was VexRISC-V, which had several configuration options for the core generation so that it can be tuned to increase performance or decrease die area use. It also natively supports the Wishbone bus, which is the interface that is already used by the Caravel harness processor. This enables easy interconnection between peripherals. The RISC-V core will be clocked using the 10 MHz clock that is also used by the management SoC. This will reduce additional clocking circuitry and make digital design easier by having all Wishbone devices on a common clock domain.

4.3.2.2.2. DFF RAM

DFF RAM is a critical part of the design, since it provides an area for data to be stored both for computation in a user program and for the data transmitted and received via Zigbee. The DFF RAM will be implemented via a macro found on the Efabless marketplace. Since the macro layout has already been completed by the creator, we will not have to do the layout ourselves, which will save us significant time. In addition, the macro has already been tested, so we have confidence that it will work in our final design. However, since the macro is not wishbone compatible out-of-box, we will need to create our own wishbone interface for it. Finally, we are going to instantiate two 2 KiB DFF RAM macros for the user space RISC-V core. One will serve as its data RAM and the other will serve as its instruction RAM.

4.3.2.2.3. Wishbone Crossbar

The Wishbone Crossbar is a module that connects all Wishbone masters, which includes the management area RISC-V core and the user area RISC-V core's instruction and data interface, to all memory mapped slaves in the design. The slaves instantiated in the final design will be an I2C module, two banks of 2 KiB of DFF RAM, a reset controller for the user area RISC-V core, and the AES-128 hardware accelerator. The Wishbone Crossbar arbitrates access to these slaves, ensuring that two masters never attempt to access the same slave at the same time. This prevents data corruption or deadlock as a result of simultaneous access. It does not prevent typical concurrency issues that arise in multithreaded programs, these must still be solved using programming techniques. In addition to preventing simultaneous access to the same slave, the Wishbone Crossbar also allows parallel access to different slaves. This allows for higher throughput, such as allowing the RISC-V core in the user area to fetch instructions from one RAM bank while reading or writing data from the other RAM bank.

4.3.2.2.4. I2C Controller

The I2C controller enables user programs to communicate with peripherals such as sensors, I/O expanders, and memory modules. The controller will act as a master that will communicate to slaves on the I2C bus. To enable the RISC-V cores (both user and management cores) to configure the controller we created a wishbone register file. The register file can contain N 32-bit registers and can be written and read from the wishbone bus. A separate read/write port is provided so that any module (the I2C controller in this case) can directly read and write without having to be on the wishbone bus. The register file for the I2C controller consists of five registers: Control, status, address, write data, read data. The control register configures an I2C operation to be read or write, include start and stop bits, and is used to start the I2C operation. The status register indicates if the commanded I2C operation is still ongoing and it also indicates if the operation ended in a NACK from the slave device. The address register stores the 7-bit I2C slave address that the operation will act on. The write data register stores the 8-bit value that will be written to the I2C slave upon a write operation. The read data register stores the 8-bit value that is read from the I2C slave upon a read operation. By accessing these registers through the separate read/write port on the wishbone register file, the I2C module is provided with all the information it needs to execute I2C operations.

4.3.2.3. Security Acceleration

The security acceleration component will implement the AES-encryption specified by the Zigbee protocol, which will encrypt, and decrypt data transmitted and received on the microcontroller,

allowing for safe and secure transmission of data. It uses 128-bit keys to encrypt/decrypt data using a symmetric block cipher.

4.3.3. Functionality

To use the radio MCU, users will need to connect it to power and connect any devices that they desire to be controlled by the radio MCU. Once everything is hooked up, a computer that contains the code desired to be run on the MCU and the programming software must be connected via a serial interface to the MCU. Then, the user can flash the program they wrote to the MCU, which will provide it with instructions to execute. Once the MCU is running the program, the user can then observe the signals output from the MCU, which are going to devices that the user has plugged into the various outputs of the MCU. Now that the program is running, the user can connect any wireless devices that support Zigbee to the radio MCU for wireless communication. Once the chosen devices are connected, the user will be able to observe the communication between the radio MCU and the devices by how they each respond to incoming signals. Overall, the radio MCU can be used to control and communicate with devices given that it has been powered, the devices have been connected (wired or wireless), and a program with the control code has been flashed to the MCU.

4.3.4. Areas of Concern and Development

If we implement all we have described, our design will fully satisfy the user requirements. We would provide a platform that can be programmed by the user and control devices both wired and wireless. However, our biggest concern is going to be getting all the components implemented in our short timeframe. As seen in the “Personal Effort Requirements”, many of the tasks will take over 50 hours to complete. So realistically, we cannot finish all the tasks in two semesters, but we can finish a select few. The diagram below shows what elements of the design that constitutes our “minimum implementation”, which we believe that we can have finished by next semester.

As shown in Figure 19, our minimum implementation contains many of the digital components in the full design, but our analog section (RF Subsystem) contains much fewer components. With the digital components shown, we provide the user with a processor to execute their program, DFF RAM to store their program and act as memory, and peripherals that they can hook up devices to. This allows the user to control wired devices such as servos, motors, and LEDs. The RF subsystem is not complete, which means that radio communication will not be possible, which does not satisfy the RF communication user requirements. However, we still plan to implement a portion of the full RF subsystem, which then can be used by other teams who tackle this project to get a head start on finishing the RF subsystem. To address these timeline issues, we will be documenting our design and our implemented features well. Then, future senior design teams or individuals from ISU ChipForge can take our design and working components and build off them, approaching the full implementation which satisfies the user requirements.

Finally, another concern we have had while working on this project is not having all the information to create prototypes for our components. To resolve this, we have been finding online publications, talking with faculty members at ISU who have relevant experience, and attending the ChipForge weekly meetings to ask questions about the Caravel Harness and the tooling. With all these resources, we are confident that we can make substantial progress and implement our minimum design implementation.

4.4. TECHNOLOGY CONSIDERATIONS

The primary technology we are using for the design is the Caravel Harness from Efabless. This is a platform that utilizes the Skywater 130nm process and has a management system on chip (SOC) already designed for fabrication. The harness also contains a user space where digital and analog designs can be inserted and are able to communicate with the management SOC as well as the pad frame (where the GPIO pins are). Using the Caravel Harness, our biggest limitations are the lack of documentation and the size limitations of the user area. So far, we have not found great documentation on the Caravel Harness, so all our learning has been done by looking at examples created by both Efabless and ChipForge as well as reading through the source code since the Caravel Harness is completely open-source. For the size limitations, we are limited to a die area of 3mm x 3.6mm, which limits how many transistors we can fit in that space. Since using the Caravel Harness is a constraint for our design, there is not a way to remove these limitations.

Efabless provides four tools for analog design, XScem, Ngspice, Netgen, and Magic, for schematic capture, simulations, LVS, and layout respectively. These tools represent a large part of the unknown tools risk discussed earlier as no members of the team had used these before. These tools were unintuitive to use and difficult to learn with limited documentation. Debugging issues was challenging even when communicating with the team at Efabless because there was often no one who was familiar with the issues we were facing. Ultimately, we were able to complete development on the analog designs, but the tooling challenges stretched out the project timeline.

5. Testing

Testing is an integral part of the project, since the design cannot be altered after being submitted for fabrication. To ensure correct functionality, testing will need to be performed throughout the project and again after fabrication to make sure the implementation matches the design. Digital and analog testing will be performed using computer simulations during the implementation portion of the project. Digital testing will primarily use software during the bringup testing of the project, and analog testing will use a frequency counter and a spectrum analyzer to make sure the PLL is generating correct frequencies with acceptable noise figures.

5.1. UNIT TESTING

Unit testing will focus on individual blocks outlined in the system diagram. These tests will be relatively simple and focus on making sure the blocks function as intended.

5.1.1. Digital

Digital unit testing will be performed using the simulator provided by the Efabless tools. These tests can be automatically checked using a Verilog testbench to automate testing.

5.1.1.1. Wishbone Crossbar

- Verify that address mapping works correctly
- Verify that write/read data are sent to/from slaves correctly
- Verify that two masters can access different slave simultaneously
- Verify that two masters cannot access the same slave simultaneously

5.1.1.2. *VexRISC-V Core*

- Verify that a simple program can be loaded and executed correctly
- Verify that the processor generates correctly formatted Wishbone bus transactions

5.1.1.3. *DFF RAM*

- Verify that word (four bytes) reads and writes can be performed
- Verify that half-word (two bytes) reads and writes can be performed
- Verify that single byte reads and writes can be performed

5.1.1.4. *I2C Controller*

- Verify that the five wishbone register file registers can be written and read
- Verify that I2C reads and writes can be initiated
- Verify that ACK/NACK status is read and stored in the status register when writing to a slave
- Verify that data is correctly written to the I2C bus
- Verify that data is correctly read from the I2C bus
- Verify that the start bit is sent correctly when configured to be included
- Verify that the stop bit is sent correctly when configured to be included

5.1.1.5. *Security Acceleration*

- Verify that configuration and data registers can be read/written via Wishbone bus
- Verify that round keys are stored properly in registers
- Verify that substitute-byte transformation, shift row, and mix column operations compute the correct output
- Verify the key generated by expansion algorithm is properly copied and filled
- Verify the number of cycles encryption takes matches the expected target value

5.1.1.6. *Phase Frequency Detector (PFD)*

- Verify that it can set UP and DN signals as expected.
- Verify that the reset signal is functional.

5.1.1.7. *Charge Pump and Filtering*

- Verify that the transistors are ON for the same delay that is designed in the PFD component. If the output fluctuates, increase the PFD Reset delay as necessary.
- Verify that the VCO control voltage does not change before PFD reset where both UP and DN and high causing current sourcing and sinking of the charge pump.
- Verify that the PFD-Charge Pump gain matches the post-layout simulation gain.

5.1.1.8. *Voltage Controlled Oscillator (VCO)*

- Simulate VCO and measure its gain to confirm voltage required from the PFD. Also measure maximum and minimum frequencies to ensure VCO functions in the necessary range.
- Simulate VCO and measure its phase noise. This measurement can then be used to characterize the expected phase noise for the entire system.

5.1.1.9. *Fractional N Divider*

- Simulate the divider and verify its maximum operating frequency is greater than 928MHz.

- Simulate the divider and measure its output frequency to confirm it functions correctly.
- Measure the simulated divider's spurring and determine if it is necessary to swap to the integer N design.

5.2. INTERFACE TESTING

5.2.1. Digital

The primary interface used for digital components of the design is the Wishbone interface, which is governed by an open standard from OpenCores. Since most of the individual digital components being tested have a Wishbone master or slave interface (or both), helper functions will be created to verify correct Wishbone functionality during simulation. This will be done with Verilog functions that can generate Wishbone transactions as well as check that the transactions performed are the transactions expected. Interface testing on the digital side will be primarily integrated with the unit testing, since the interface is integrated with the units.

5.2.2. Analog

The PLL will use the pad frame for main connection to the inputs and outputs of the PLL as well as two connections for testing its functionality and measuring its characteristics. The testing and the PLL divider controls will be fed from the designed RISC-V processor. Apart from the divider control signals, the PLL is mostly isolated from the rest of the digital and security components. Therefore, after ensuring the functionality of the digital component, the main debugging interface will be through the pad connections.

5.3. INTEGRATION TESTING

The digital components are integrated and tested as a submodule and the analog components are integrated and tested as a submodule. A full system simulation is not feasible due to computing requirements.

5.4. SYSTEM TESTING

5.4.1. Digital

In addition to the unit tests to make sure each individual block functions as intended; additional simulations will be run on the entire digital system. These will be somewhat limited in scope, since simulating large programs would take large amounts of time which would slow down the development process.

- Verify that simple test programs can be run correctly
 - o These are programs that access several peripherals (I2C, DFF RAM, etc.)
- Verify that system does not freeze or enter undesired states during program operation
- Verify that system comes out of reset in the correct manner

5.4.2. Analog

The system testing plan of the PLL mostly consists of the closed loop simulations of the system, ensuring optimal working conditions.

- Measure its output frequency to verify that it outputs the correct frequencies.

- Measure the simulated PLL's phase noise to ensure a useable signal.
- Measure the simulated PLL's lock time.
- Measure the simulated PLL's settling time and confirm that it conforms to the Zigbee standard. The [ATSAMR30M18A](#), a commercial Sub-GHz ZigBee compatible RF microcontroller, has a 170us settle time. We will design the PLL to have the same settle time.

5.5. REGRESSION TESTING

5.5.1. Digital

Regression testing is conveniently enabled by the simulation toolflow. A single command can be issued to automatically run all test cases for digital components to make sure that no functionality has been broken by changes. All tests should be run when making changes to make sure that no regression has occurred.

5.5.2. Analog

Due to the closed loop nature of the PLL, any change to the subcomponents will ultimately affect the overall functionality and characteristics. This is why the analog regression testing will consist of rigorous simulation testing of the closed loop system whenever a characteristic of a subcomponent changes. This will ensure we spot any design mistakes throughout the system integration and testing process before fabrication.

5.6. ACCEPTANCE TESTING

Originally, a test plan write-up for after fabrication was to be completed to help ChipForge members test the chip after fabrication was completed. However, since fabrication became impossible due to the shutdown, acceptance testing shifted to making sure that parameters were met on the digital and analog portions of the design.

5.6.1. Digital

Digital acceptance testing involved synthesizing the design for an FPGA and running sample C programs on the FPGA and analyzing various signals to make sure that peripherals and the Wishbone interface are functioning correctly. The synthesis flow also revealed important information about how the design was being implemented in hardware. This checked for latches, combinational loops, and logic that didn't have a direct hardware mapping. Later stages of the FPGA toolflow check that the design is capable of operating at our 10 MHz target frequency for the system clock. In addition, we ran the digital design through the OpenLane flow to obtain a layout for it. The layout is shown below (the two large rectangles are the DFF RAMs):

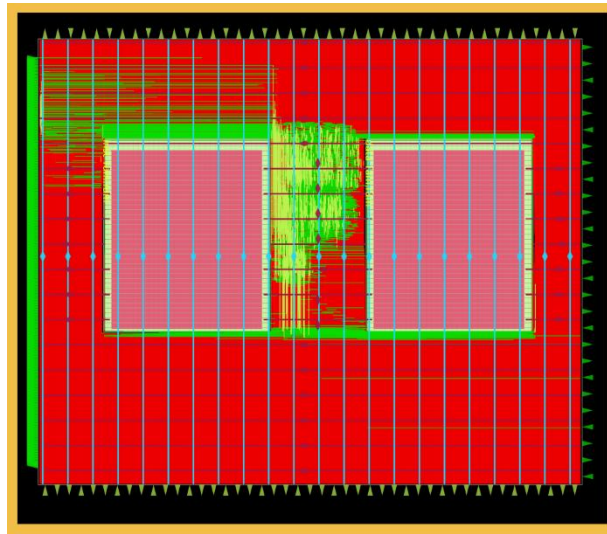


Figure 25: Digital Layout

5.6.2. Analog

5.6.2.1. PLL Testing Architecture

The Fref input signal will be connected to an off-chip crystal oscillator through the pad frame. The loop filter will be connected off-chip which will save area for other necessary components. Changing the loop filter will significantly change the loop dynamics if the values are properly re-configured. Having an off-chip filter allows for individual VCO testing because the user has full control over the VCO control voltage.

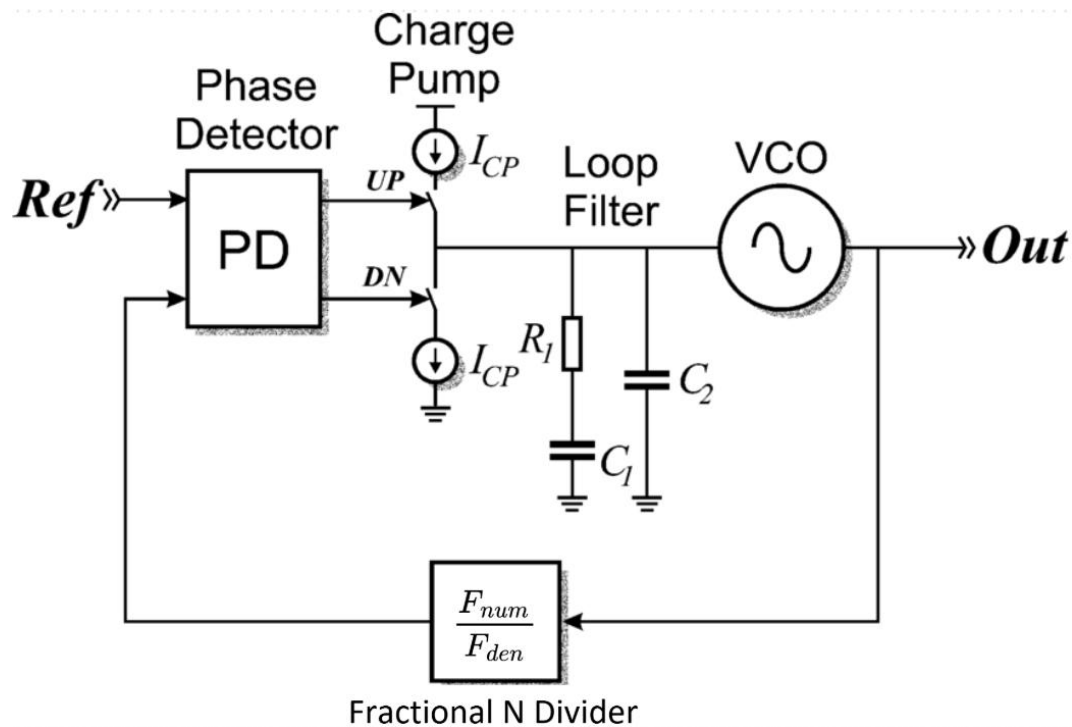


Figure 26: Fractional N Divider

5.6.2.2. PLL Measurements and Characterizations

Low Frequency Oscillator

- Measure frequency output
- Should be a very stable 10MHz

VCO Open Loop Configuration

- DC Sweep the input of the VCO
- Connect the output to a spectrum analyzer
- Verify the VCO gain
- Verify the VCO bandwidth and stable operation at the 900MHz range.
- Measure the noise of the VCO

PLL Closed Loop System

- Verify output frequency matches each channel in the 915MHz Zigbee range
- Measure phase noise
- The phase noise of the PLL must be in the acceptable range for a standard Zigbee receiver.
- During startup measure lock time and overshoot
- Once the frequency is locked to a channel, change the divider control signal and measure the settle time.
- The PLL can lock to each 1MHz channel from 902 to 928MHz

Fractional Divider

- Verify each subcomponent function as expected
- Verify divider system can create a denominator $90.2 \leq N \leq 92.8$
- Frequency sweep prescaler
- Measure maximum operating frequency

5.7. SECURITY TESTING

- Verify that the output of the plaintext is properly encrypted.
- Ensure each round function and round key is properly stored in each register.
- Each substitute-byte transformation, shift row, and mix column operation should be consistent in its output
- Key generated by the expansion algorithm is properly copied and filled.
- Verify the number of cycles each encryption takes matches the expected target.

5.8. RESULTS

5.8.1. Wishbone Crossbar

The Wishbone Crossbar has been tested individually using a module that can generate Wishbone transactions and a simple testbench containing only the Wishbone Crossbar and two dummy slaves, which just cleanly terminate transactions and do not contain any special logic. This testbench confirmed that address mapping, arbitration, and parallel access all work as intended. See the waveforms below for more details.

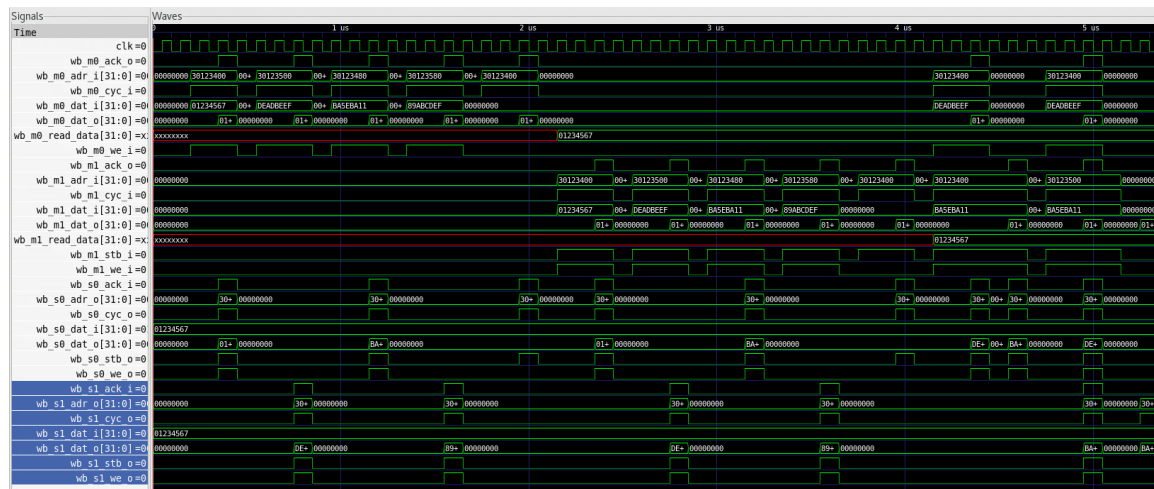


Figure 27: Wishbone Crossbar Testbench Waveform

The first part of the waveform (0 – 2 μ s) demonstrates the address mapping capabilities of the crossbar. Master 0 initiates transactions to both slave 0 (address 0x30123400) and slave 1 (address 0x30123500). The transactions are then routed to the appropriate slave. The second portion of the waveform (2 – 4 μ s) demonstrates the same functionality with master 1 initiating the transactions. The third portion of the waveform (4 – 4.5 μ s) shows the arbitration capability. Master 1 loses arbitration to master 0, and the transaction from master 1 stalls until master 0's transaction has

completed. The final portion of the waveform (4.5 μ s and on) shows parallel access, with both master 0 accessing slave 0 and master 1 accessing slave 1 at the same time. This shows that the Wishbone crossbar meets its requirements and functions as intended.

5.8.2. DFF RAM

The DFF RAM was tested as an individual unit to verify that reads and writes worked as expected. This was done using a simple Verilog testbench that wrote a series of data to each word in the RAM and then read back the data to confirm that it was stored successfully. This also tested the Wishbone wrapper that was put around the RAM macro to adapt it to the bus. The results are shown in the waveform below.

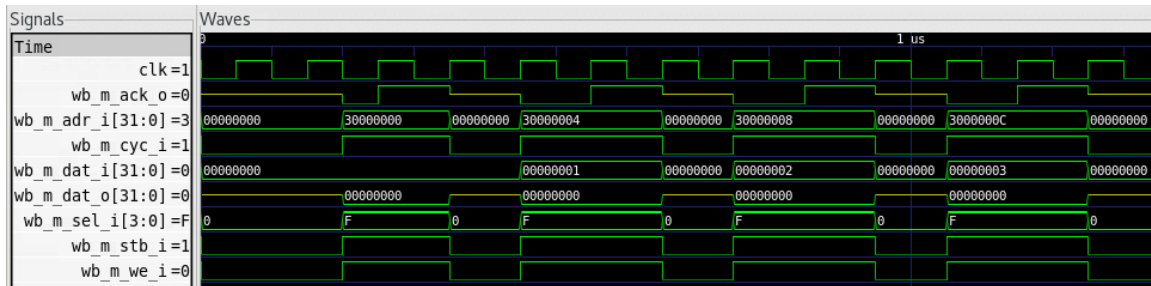


Figure 28: DFF RAM Writes

In the waveform above, four 32 bit words are written to the RAM via a Wishbone interface at the first four word sized addresses. These writes are clean Wishbone transactions, and adhere to the specifications in the Wishbone V4 standard.

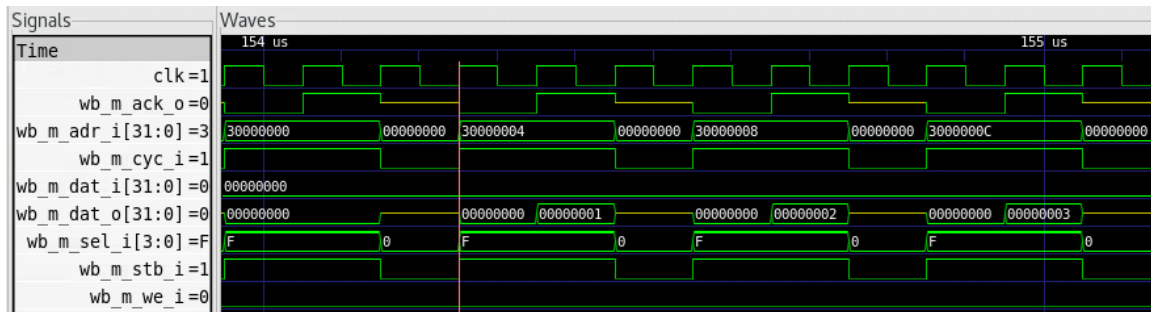


Figure 29: DFF RAM Reads

In this waveform, four 32 bit words are read from the same addresses that were written previously. The data (0, 1, 2, 3) that is read back matches the data that was written. This indicates that the RAM writes and reads function correctly and that RAM can save data to be recalled later. This shows that the RAM meets its requirements.

5.8.3. User Area RISC-V Core

Since the user area RISC-V core relies on RAM for both instructions and data, it was not possible to test it by itself. As a result, this was done using the crossbar, management core, and DFF RAM. The management core loaded a program into the instruction memory for the RISC-V core, and then brought it out of reset. The program executed successfully, writing words to data RAM and

then verifying that they could be read successfully by the management core. A waveform from that test is shown below.

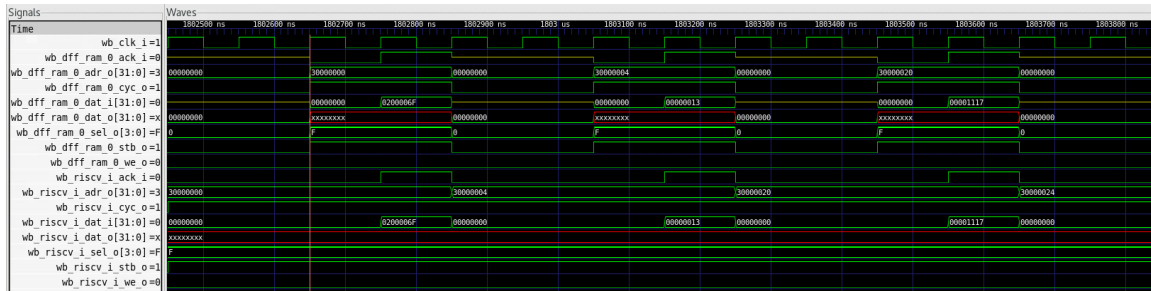


Figure 30: RISC-V Core Instruction Fetching

The above waveform shows the first three instructions that are fetched by the user area RISC-V core from the instruction memory. These match the instructions from the compiled program that was loaded. This shows that the RISC-V core comes out of reset and begins fetching instructions from the correct location, which indicates that the instruction portion of the core is connected and functioning correctly.

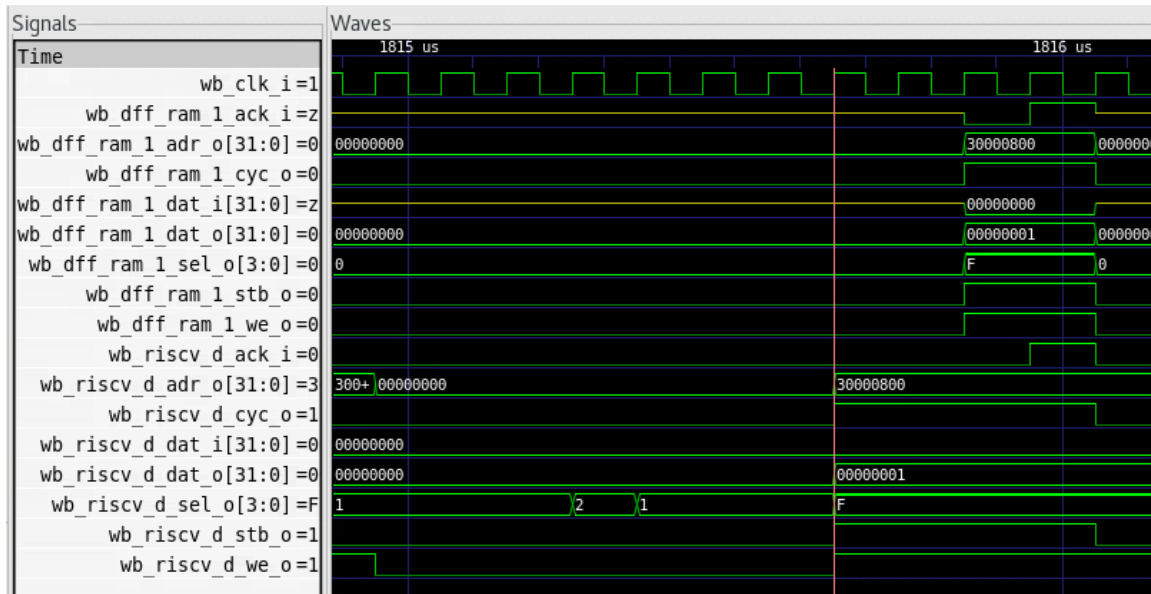


Figure 31: RISC-V Core Data Memory Write

This waveform shows the RISC-V core writing a 32 bit word (1) to data memory. The full test involves writing 8 data words (1-8) to the first 8 words of data memory. These can then be read out by the management core, shown in the next waveform.

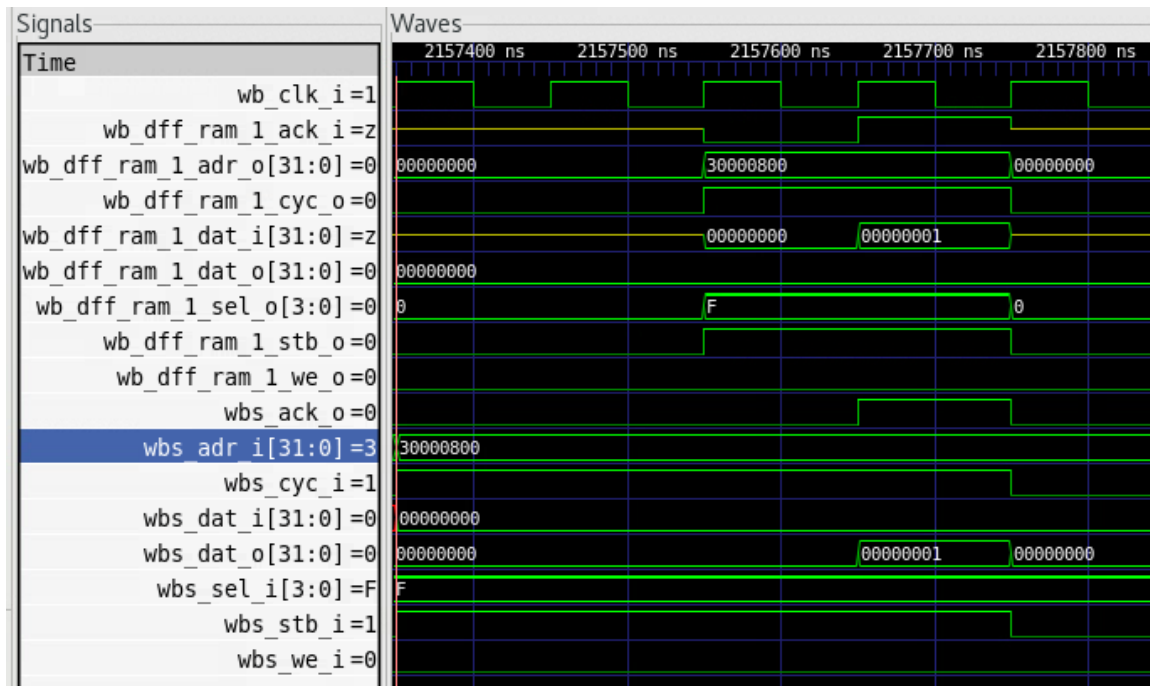


Figure 32: Management Core Data Memory Read

The final waveform shows that the data can be read back from the data memory. Combined, the different parts of this test sequence prove that the user area RISC-V core can be programmed, come out of reset, run a program, interact with the Wishbone bus, and write data that can be read back by the management core. This fulfills the requirements for the RISC-V core to be considered functional.

5.8.4. I2C Peripheral

The I2C peripheral was tested in simulation to make sure that it could generate the transactions necessary to interface with a simple I2C IO expander that was going to be used for a demonstration. This involved multiple byte writes. The results are shown in the waveform below.

The waveform below shows a byte write of AB to I2C address 7B. Then another single byte write to

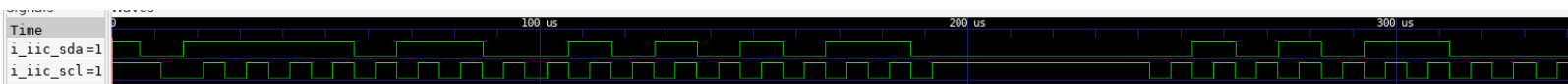


Figure 33: I2C Two Writes

I2C address 7B happens, but the written value is 56 this time. No stop bit is sent after this final write, so the bus is held in an idle state.

Finally, after we verified that our design worked in simulation, we loaded it onto the ARTY A7-100T FPGA and hooked it up to an I/O expander that had some seven segment displays hooked up to it. We wrote a program for the management SOC RISC-V core that configures the I2C wishbone-compatible registers (from the wishbone register file) to perform writes to the I/O expander to count on the seven segment displays. We verified that the writes were successful in hardware by probing the SCL and SDA lines with an oscilloscope and by visually verifying that the seven

segment displays were counting up. A sample waveform and a picture of the seven segment displays are shown below:

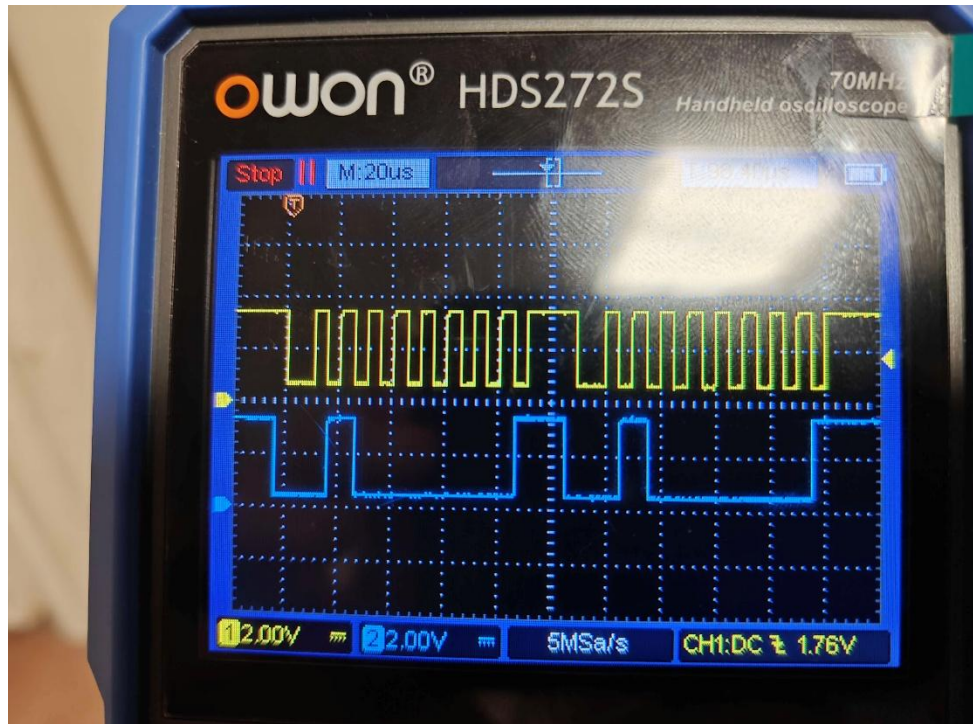


Figure 34: Oscilloscope I2C Readings

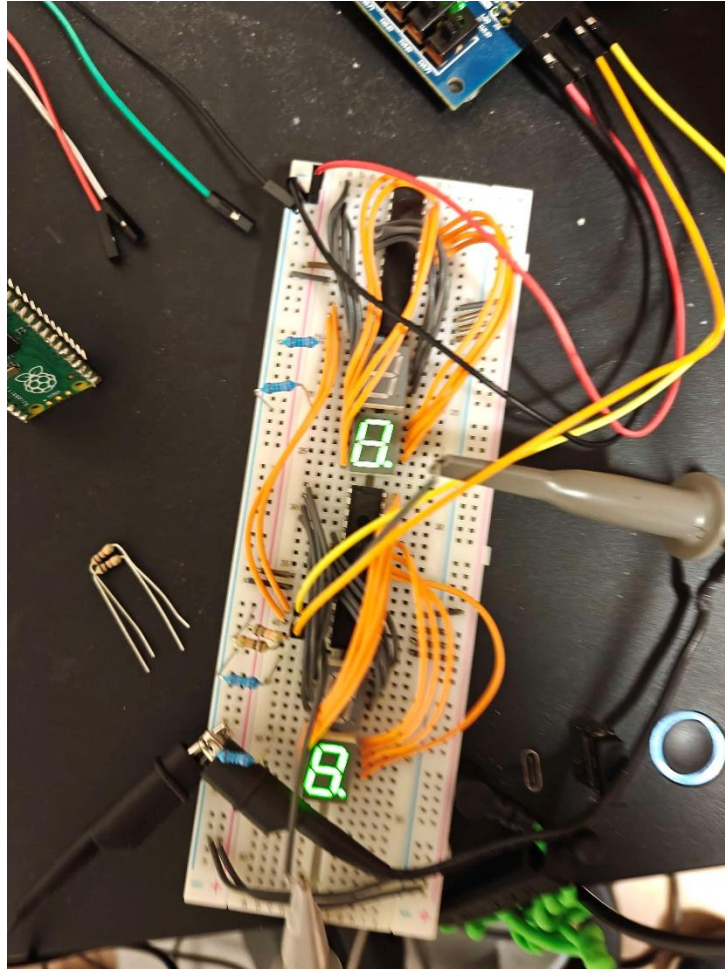


Figure 35: I2C I/O Expander Testbench with Seven Segment Displays

5.8.5. Wishbone Register File

Since the wishbone register file (used by the I2C controller) is generic and can be used by many other modules, we created a separate test for it. The test is a system test where the wishbone register file is instantiated with four registers alongside the Caravel harness description, which contains our full digital design. Wishbone reads and writes are verified in addition to reads and writes from the separate read/write ports. In addition, our system test is self-verifying using signal value checks in Verilog. Below are some of the resulting waveforms showing the correct operation of the wishbone register file.

The waveform below shows a write of A to address 30001000 (register o), which is then read back.

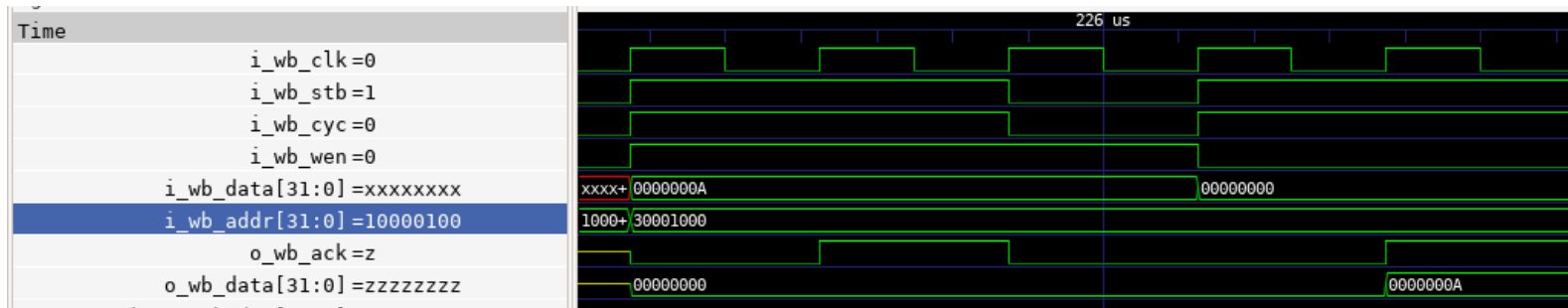


Figure 36: Wishbone Register File Write and Read

The waveform below demonstrates what happens when a wishbone write and a write to the separate write port happen at the same time. *i_non_wb_dat* always reads out the value of register zero, which is the register being written to here. Here, *o_non_wb_dat* is set to AADDE0 and *i_non_wb_wen* is 1, which means that a non-wishbone (separate write port) write of ADDE0 is wanting to happen. But since *i_wb_wen* is also set to 1, the wishbone write takes precedence and goes through. The wishbone write was to address 30001000 and the value in *i_wb_data* (ACCDDEF)

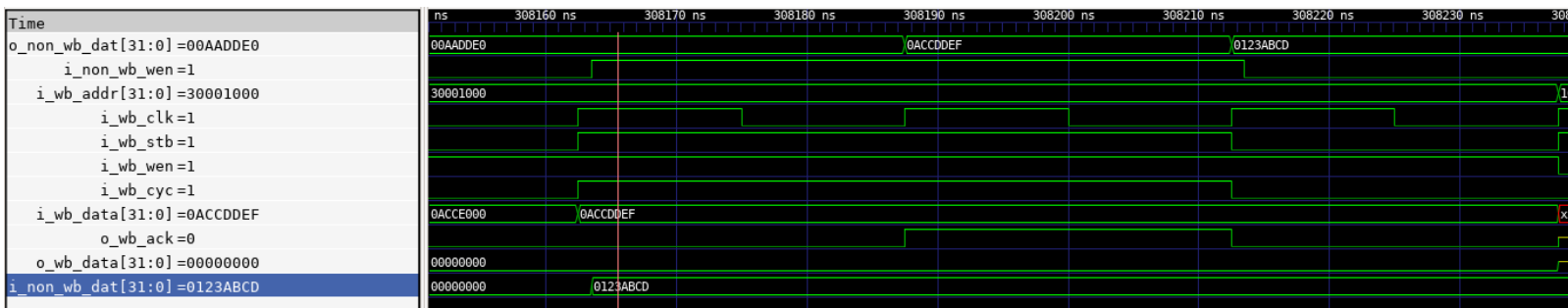


Figure 37: Wishbone Register File Arbitration

is written and can be observed on *o_non_wb_dat*. Immediately after the wishbone write is done, the non-wishbone write executes, which is observed on *o_non_wb_dat* as well. This shows that the module can arbitrate who is writing the register when wishbone and non-wishbone writes are commanded at the same time.

5.8.6. PLL Divider

To test a component in xSchem, the schematic is compiled into a symbol, then connected to simulated input signals. As an example, Figure 38 shows the testbench for the prescaler.

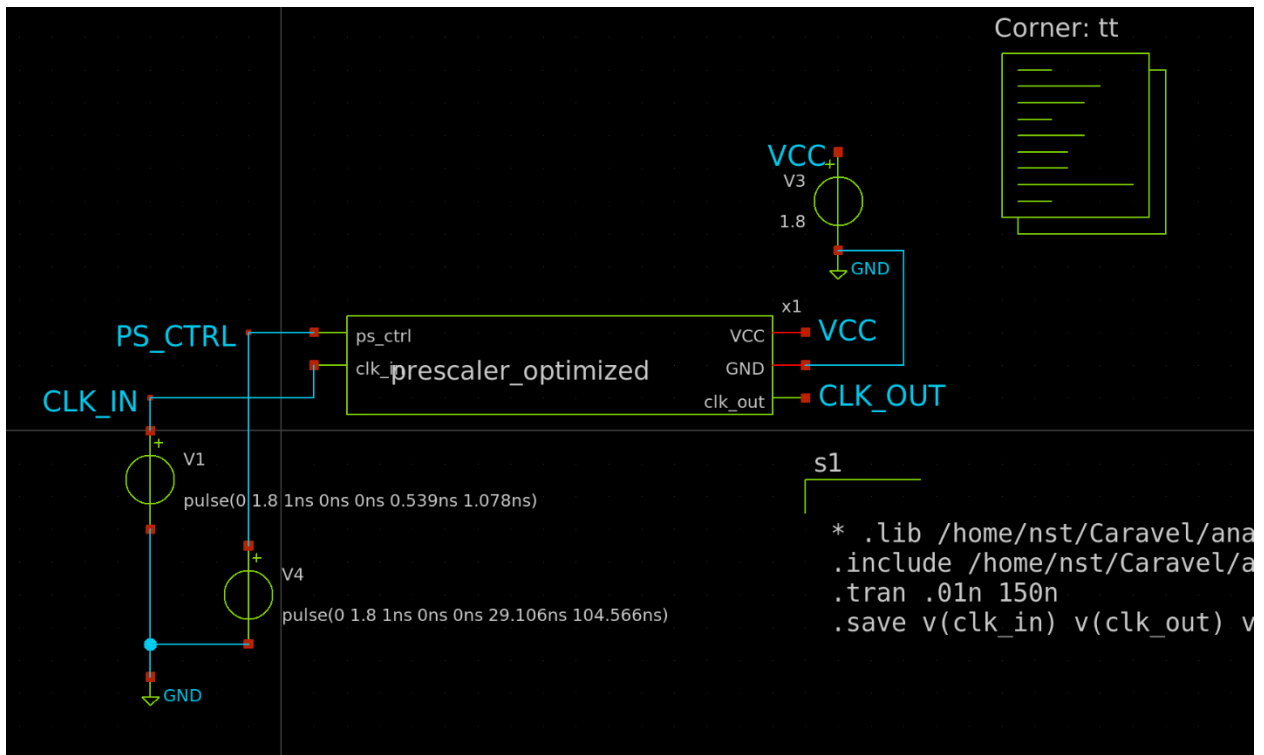


Figure 38: Prescaler Testbench

Clk_in is simulated as a 928MHz square wave and ps_ctrl is set to simulate $N = 3$. The Corner block in the top left sets what corners the component will be simulated with. All prelayout simulations were on typical-typical corners (tt). The code snippet, s1, includes the standard cell libraries, sets the simulation type, step size, and duration, and determines which nets are recorded. The complete test results are on our GitLab page linked at the end of this document.

5.8.6.1. Prelayout Prescaler Results

The prescaler was tested for $N = 0, 1, 2, 3$ at 928MHz. As an example of test result analysis, Figure 39 shows an annotated waveform of the $N = 3$ test results.

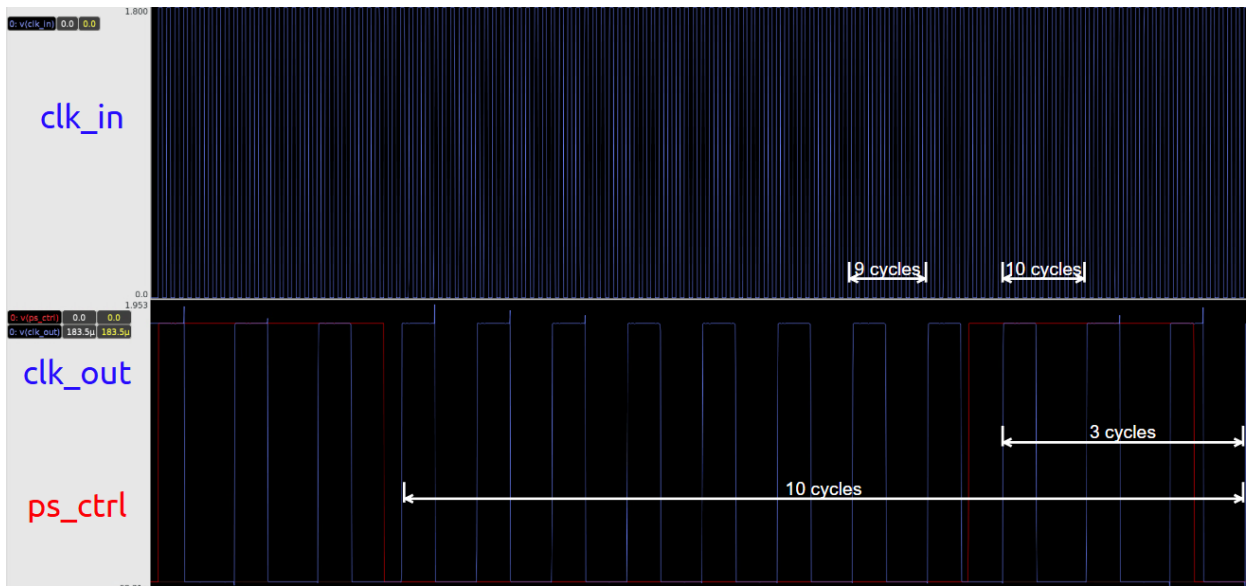


Figure 39: Waveform of N=3 Test Results

Here we see the behavior is as expected when $N = 3$. The control signal is high for 3 cycles and gets reset after 10 cycles. While low, the prescaler is dividing by 9, and while high it is dividing by 10.

5.8.6.2. Prelayout Programmable Counter Results

Using a testbench created as described above, the counter was simulated for $N = 0, 1, 2, 3$ at 103.11MHz, the maximum anticipated frequency of the prescaled clock. As an example of test result analysis, Figure 40 shows an annotated waveform of the $N = 2$ results.

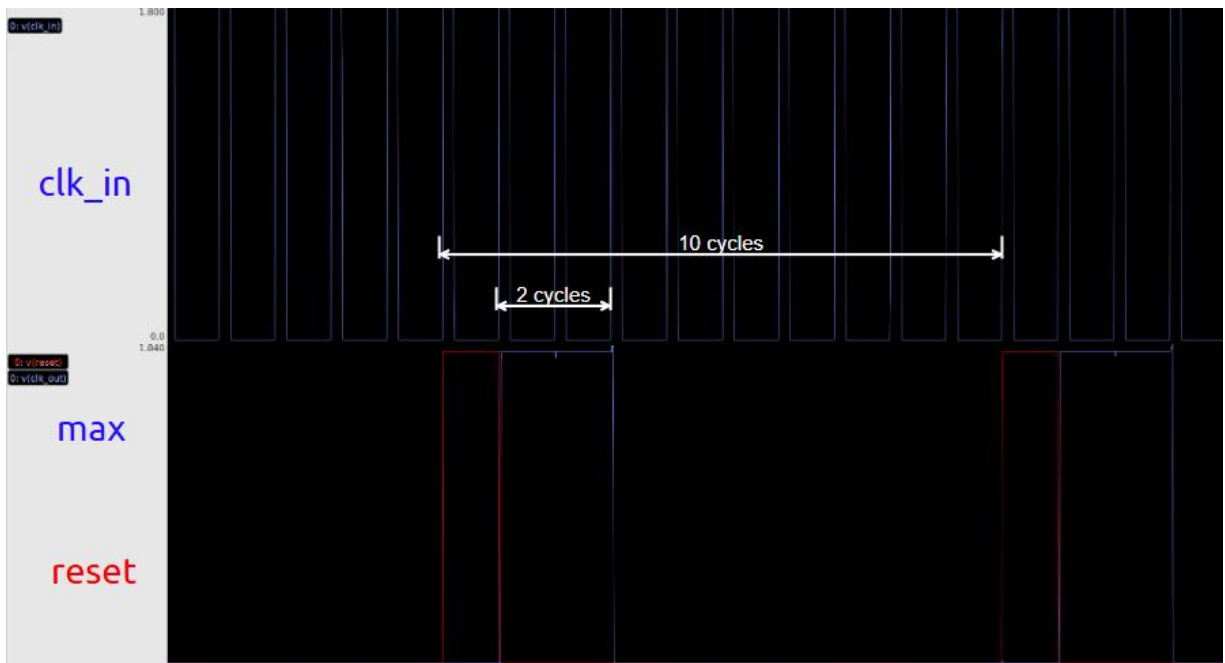


Figure 40: Waveform of N=2 Test Results

Here we see that a reset signal is received every 10 cycles, and max is high for 2 cycles immediately after. The behavior of max is the opposite of expected. This is to compensate for the prescaler inverting the control signal.

5.8.6.3. Prelayout Fixed Divider Results

The fixed divider only takes the prescaled clock as an input and as such only has one test case. The simulation was run at 103.11MHz, the maximum anticipated frequency of the prescaled clock. Figure 41 is an annotated waveform of the results.

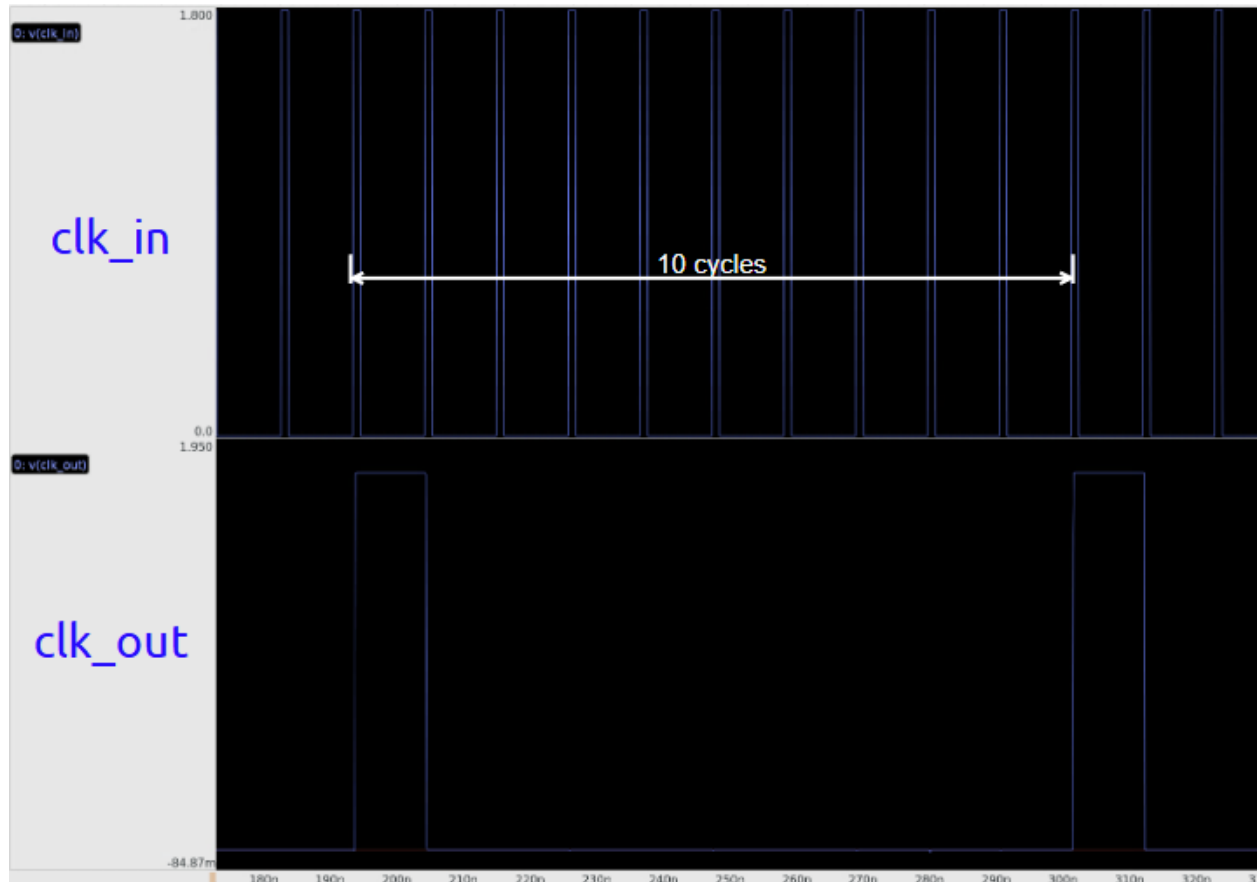


Figure 41: Prescaled Clock Divider Results

Analyzing the results is quite straight forward. We see that that clk_out has one positive edge every 10 clk_in cycles, thus the fixed divider behavior matches expected.

5.8.6.4. Prelayout Accumulator Results

The accumulator has a four-bit input, F, and a two-bit input, N, with a total of 26 input combinations accounted for. The accumulator was simulated with each input at 10MHz. As an example of test result analysis, Figure 42 shows an annotated waveform of the results when $F = 7$ and $N = 1$.

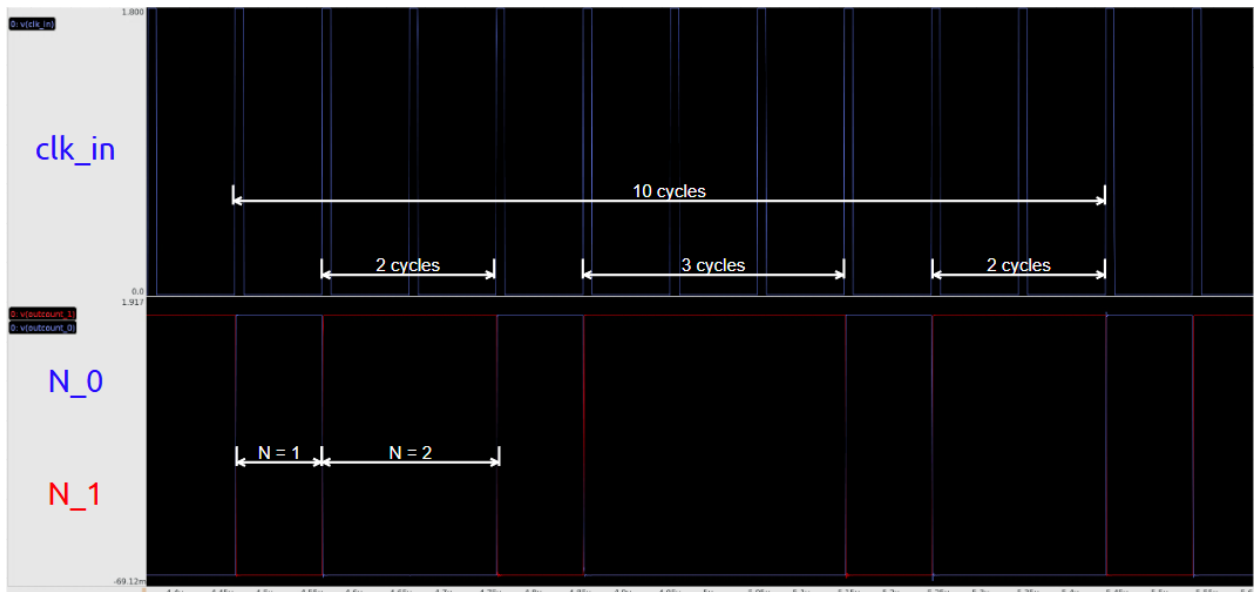


Figure 42: $F=7$ and $N=1$ Accumulator Results

As we can see, for a 10-cycle period, $N = 1$ for 3 cycles and $N = 2$ for 7 cycles resulting in an average N of 1.7. The $N = 2$ cycles are also as evenly distributed as possible across the 10-cycle period. This matches the expected behavior.

5.8.6.5. Prelayout System Results

After confirming the behavior of each component, the divider system was simulated. The test bench is set up by connecting the symbols of each component together, simulating a square wave input to the prescaler, and setting N and F for the accumulator. Figure 43 shows this full system testbench.

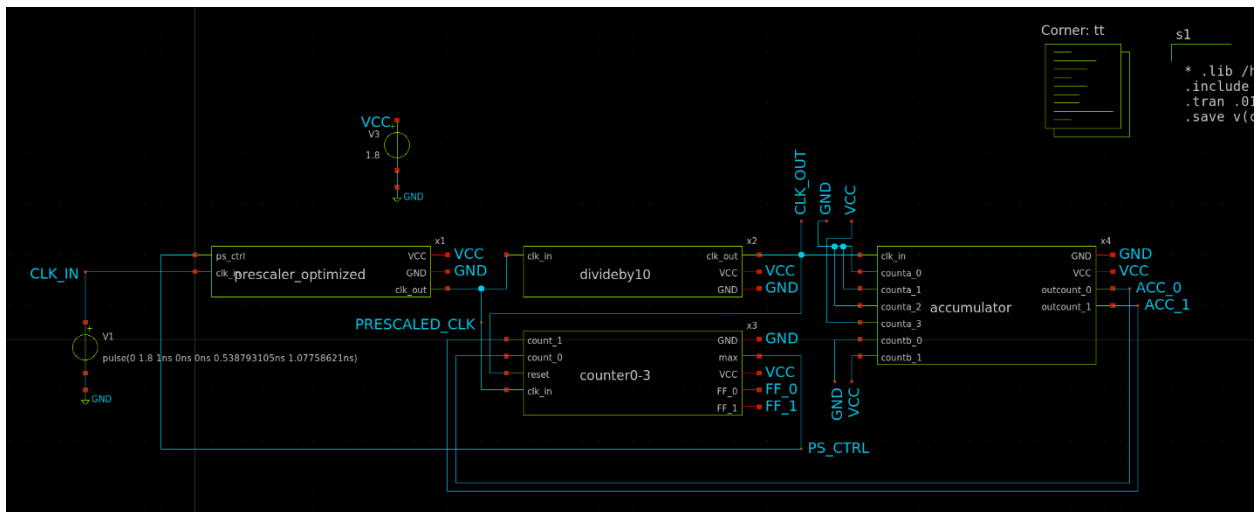


Figure 43: Full Divider Testbench

By setting clk_in , F , N so that $\frac{\text{clk_in}}{N.F} = 10\text{MHz}$, correctness can be verified by checking the average period of clk_out is 100ns. The system was tested for $90.2 \leq N.F \leq 92.8$. As an example of test result analysis, Figure 44 shows an annotated waveform of the results for $\text{clk_in} = 928\text{MHz}$, $N.F = 92.8$.

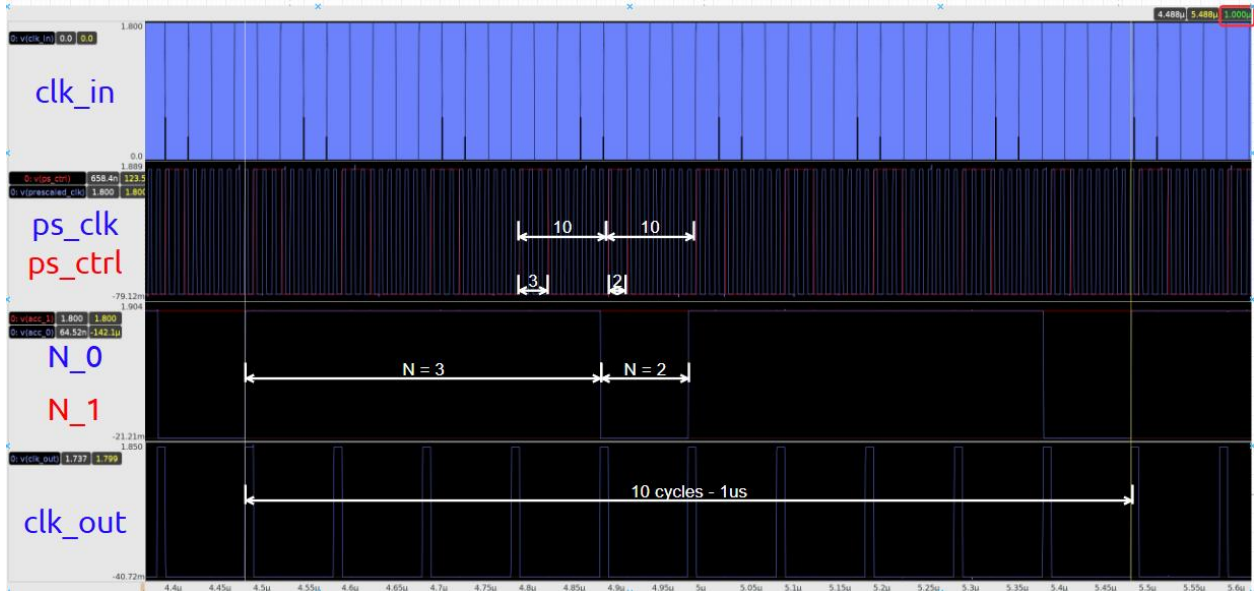


Figure 44: CLK_IN = 928MHz, N.F = 92.8 Accumulator Results

We can see that the accumulator correctly varies between N and $N+1$ at ratio of 2:8. The counter correctly update ps_ctrl based on N . Circled in red in the top right corner, the period of 10 clk_out cycles is 1us therefore the average period of one clk_out cycle is 100ns, matching the expected behavior described above.

5.8.6.6. Poslayout Prescaler Results

After the prescaler is laid out as described in section #, The netlist is extracted from magic and imported into a symbol in xSchem. This symbol can then replace the prelayout component in the testbench. Due to time constraints, only one component for the divider could be laid out. The prescaler was chosen because its high operating frequency is likely to cause the largest difference in behavior after layout, and its maximum operating frequency limit the divider's operating frequency. The laid out prescaler was simulated for $N = 0, 1, 2, 3$. As an example of test result analysis, Figure 45 is an annotated waveform of the results for $N = 3$ at 928MHz with typical-typical corners.

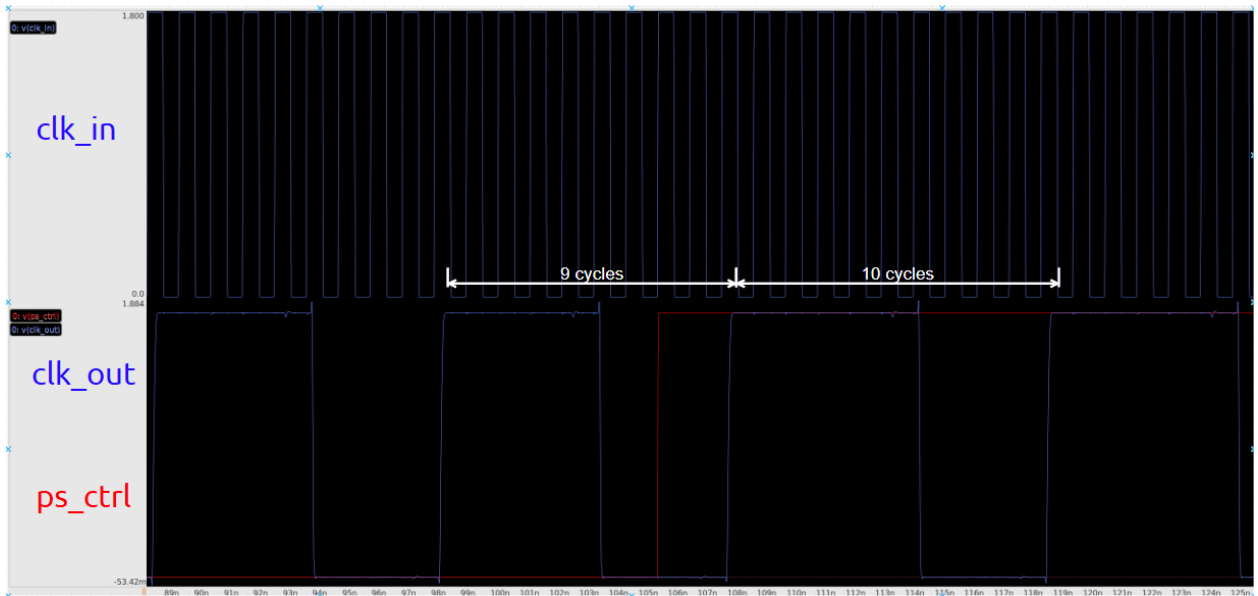


Figure 45: $N = 3$ at 928MHz with *tt* corner

We see that the behavior remains the same after layout, dividing by 9 when **ps_ctrl** is low and dividing by 10 when **ps_ctrl** is high.

Once functionality is verified, we performed a frequency sweep from 100MHz to 10GHz at slow (*ss*), typical (*tt*), and fast (*ff*) corners. By graphing **clk_in** v.s. **clk_out** at ten points per decade, we can clearly see what frequency the divider breaks down at each corner. To do this we modified the `si` code block as shown in Figure 46.

```

si
* .lib /home/nst/Caravel/analog_tutorials/dependencies/pdks/sky130A/libs.tech/ngspice/sky130.lib.spice tt
.include /home/nst/Caravel/analog_tutorials/dependencies/pdks/sky130A/libs.ref/sky130_fd_sc_hd/spice/sky130_fd_sc_hd.spice
.include /home/nst/SDMAY25_47/sdmay25-27/PLL/analog_tutorials/mag/prescaler_optimized_test.spice
.tran .01n 1500n
.save v(clk_out)
.control
set filetype=ascii
shell rm -f sweep_prescaler_results.txt
foreach in_freq M 100 121.825 151.572 192.771 246.686 316.228 400 501.187 630.957 800 1000 1218.254 1515.721 1927.714 2466.866 3162.278 4,000 5011.872 6309.574 8000 10000
    let in_freq = $in_freq Hz
    let in_period = 1/in_freq
    let in_duty = in_period/2
    alter @V1[pulse] [ 0 1.8 1ns 0ns 0ns $in_duty $in_period ]
    run
    meas tran out_period Trig v(CLK_OUT) VAL=0.9 Rise=3 TARG v(CLK_OUT) VAL=0.9 RISE=13
    let out_freq = 1/(out_period/10)

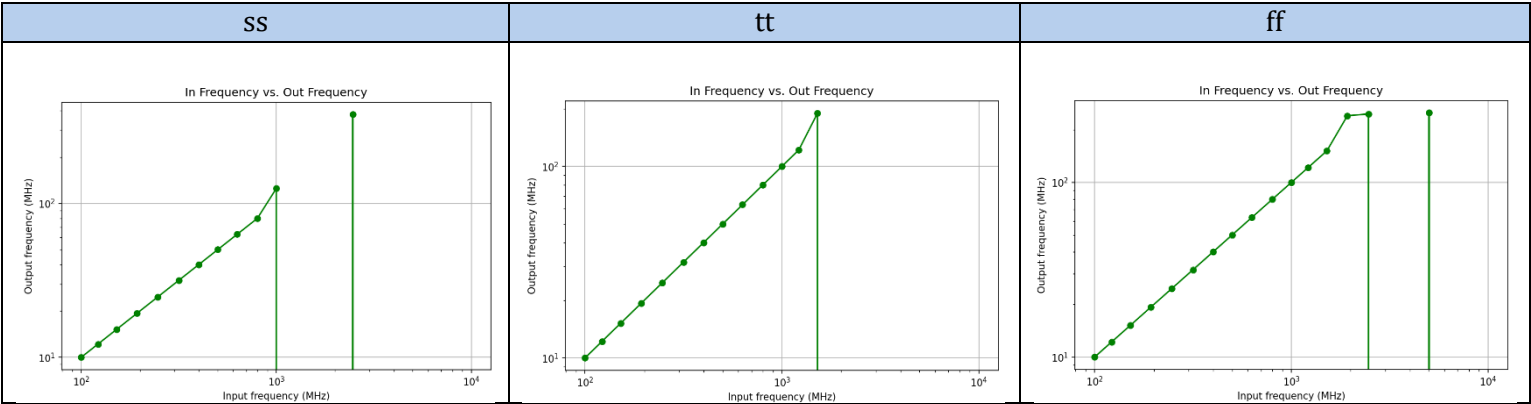
    echo Input Frequency = $in_freq Hz, Output Frequency = $out_freq Hz >> sweep_prescaler_results.txt
end
shell cat sweep_prescaler_results.txt
.endc
end

```

Figure 46: Prescaler Frequency Sweep Code

This sends results to a .txt file, “sweep_prescaler_results”. Using a python script the data was compiled into a graph for each corner, shown in Table 8.

Table 8: Sweep Prescaler Results



We see that while both tt and ff exceed the required 928MHz operating frequency, ss falls short at 800MHz. This suggests that if the divider were fabricated, a portion of the chips would not function properly. This issue was discussed with our client who determined that, as a learning tool instead of a commercial product, he would only need half of fabricated chips to be successful. Because the maximum frequency of the prescaler exceeds the required frequency for both typical and fast corners, the results suggest that more than half of chips would be successfully fabricated.

5.8.7. VCO

5.8.7.1. Low Voltage VCO

The figure shown represents a post-layout simulation at typical-typical corner at 1pF load. The output is linear from as low as 0.6V to around 1V with $K_{vco} = 3.3 \text{ GV/s}$. This value is extremely large compared to industry standards which revolves around tens of MV/s. My design is a factor of a few hundred compared to the norm. However, a lower K_{vco} values will result in a narrower range of operating frequencies to test the process.

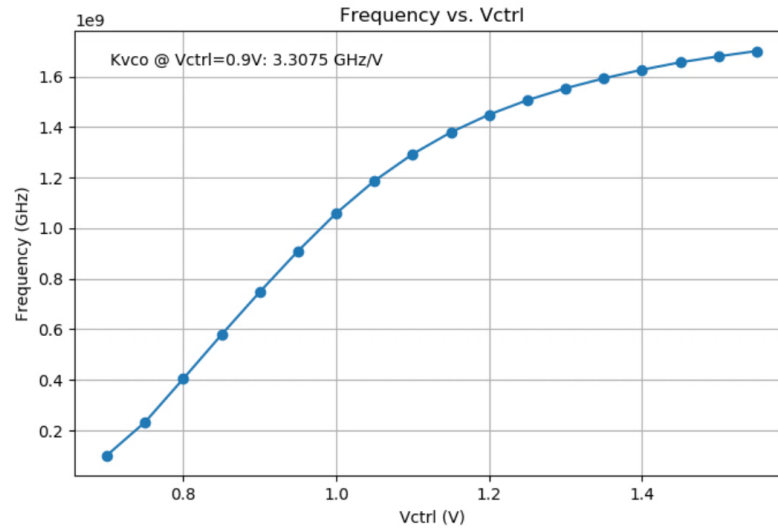


Figure 47: LV VCO Frequency vs Vctrl

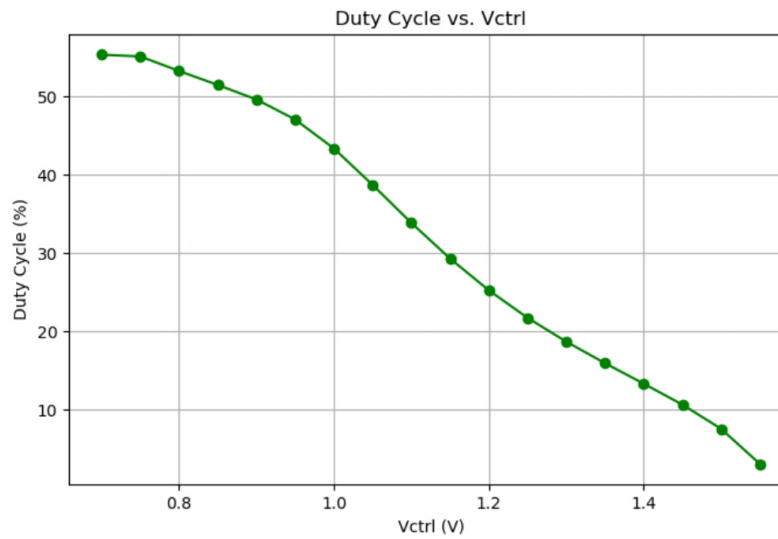


Figure 48: LV VCO Duty Cycle vs Vctrl

Complete Results across all process corners

Table 9: LV VCO Full Results

| Simulations | Corner | C_{load} | V_{ctrl} @900MHz | K_{VCO} in GV/s @900MHz | Full Frequency Range in MHz | %Duty @900MHz |
|-------------|--------|------------|-----------------------|------------------------------|--------------------------------|------------------|
| Pre-layout | SS | 1fF | 1-1.05 | 3.22 | 66 – 1770 | 51 |
| | | 1pF | | | | 42 |
| | TT | 1fF | 0.85-0.9 | 4.89 | 132 – 2230 | 52.5 |
| | | 1pF | | | | 48.9 |

| | | | | | | |
|-------------|----|-----|----------|-------|------------|------|
| Post-layout | FF | 1fF | 0.75-0.8 | 7.25 | 257 – 3900 | 56 |
| | | 1pF | | | | 55 |
| | SS | 1fF | 1.2-1.25 | 2.23 | 120-1200 | 42 |
| | | 1pF | | | | 32 |
| | TT | 1fF | 0.95-1 | 3.3 | 102 – 1790 | 51.7 |
| | | 1pF | | | | 47 |
| | FF | 1fF | 0.8-0.85 | 4.754 | 480-2310 | 55 |
| | | 1pF | | | | 50 |

5.8.7.2. High Voltage VCO

The signal swing of the control voltage of the 3.3V design is $1.15V \leq V_{ctl} \leq 2.3V$ which is significantly better than the initial 1.8V design.

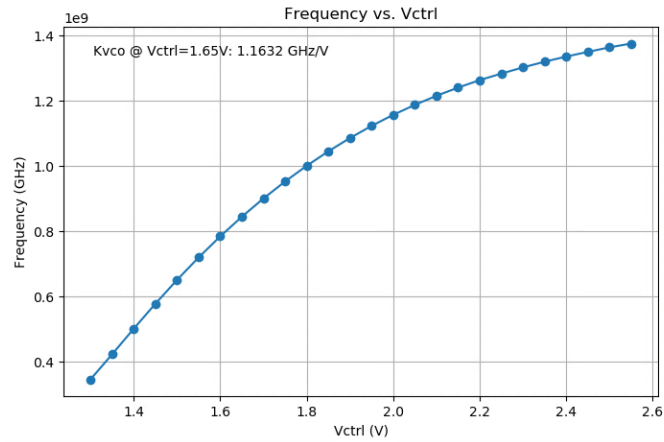


Figure 49: HV VCO Frequency vs Vctrl

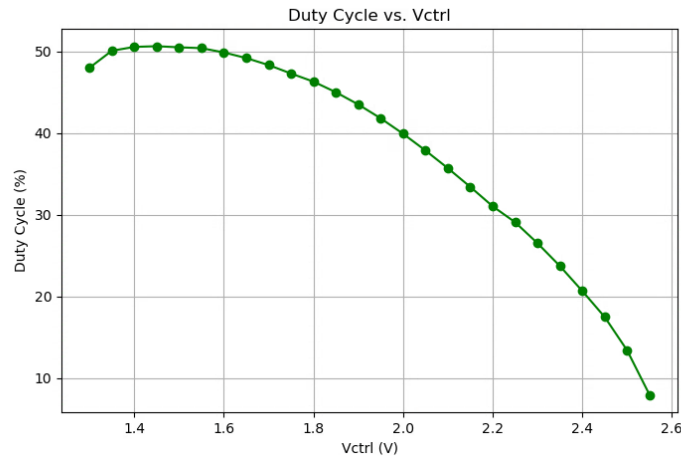


Figure 50: HV VCO Duty Cycle vs Vctrl

Table 10: HV VCO Full Results

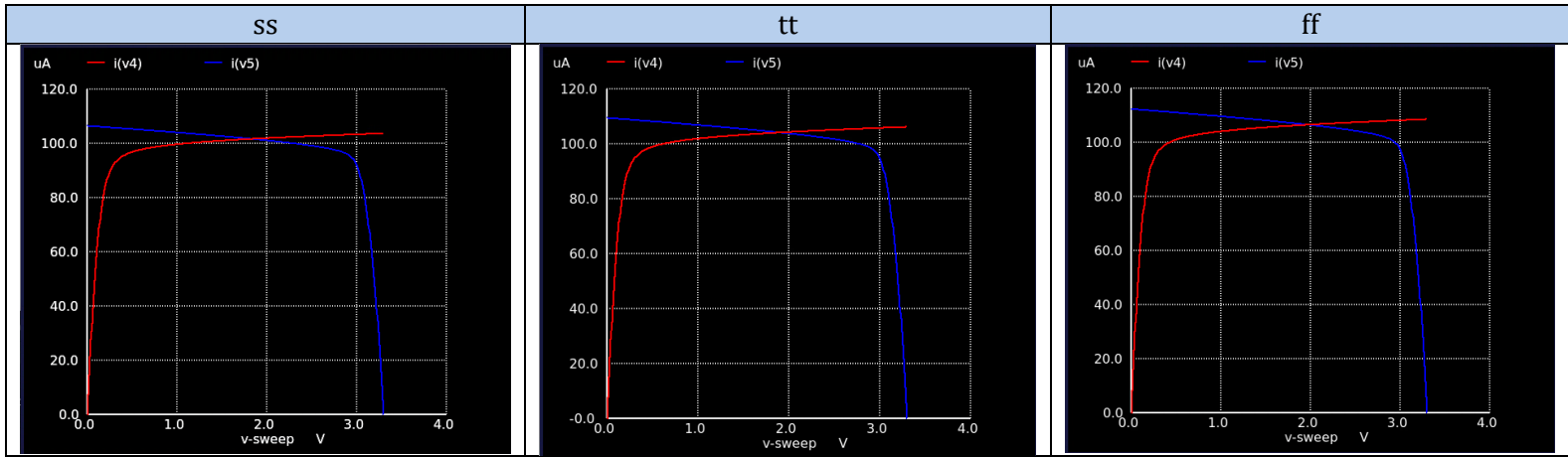
| Simulations | Corner | C_{load} | V_{ctl} @900MHz | K_{VCO} in GV/s @900MHz | Full Frequency Range in GHz | %Duty @900MHz |
|-------------|--------|------------|----------------------|------------------------------|--------------------------------|------------------|
| Pre-layout | SS | 1fF | 2.05-2.15 | 0.8589 | 201-1.03 | 65.2 |
| | | 1pF | | | | 26.4 |
| | TT | 1fF | 1.7-1.75 | 1.163 | 347-1400 | 53 |
| | | 1pF | | | | 47 |
| | FF | 1fF | 1.7-1.75 | 1.1614 | 347-1285 | 53 |
| | | 1pF | | | | 47 |

5.8.8. Charge Pump

The source (blue) /deplete (red) currents as a function of a linear DC sweep of the output voltage (V_{ctl}).

Table 11: Current vs output voltage at process corners

| 1fF Capacitive load | Voltage | Current (uA) |
|---------------------|---------|--------------|
| SS | 1.8 | 101.4 |
| TT | 1.8 | 103.3 |
| FF | 1.85 | 106 |



5.8.9. PFD/Charge Pump

These tests indicate the functionality of the integrated PFD and the charge pump. The figure below shows the current output of the charge pump at 1.8V output voltage. The figures also show an effective reset of the PFD which is observed as the current output decreases to zero. The current spikes are results of the current structure used as the charge pump mosfets try to achieve saturation-level currents.

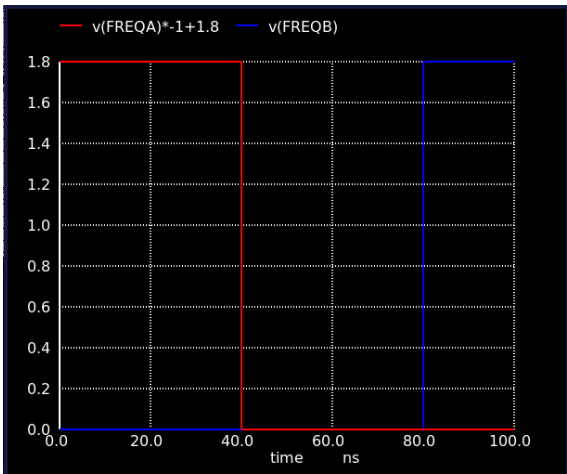
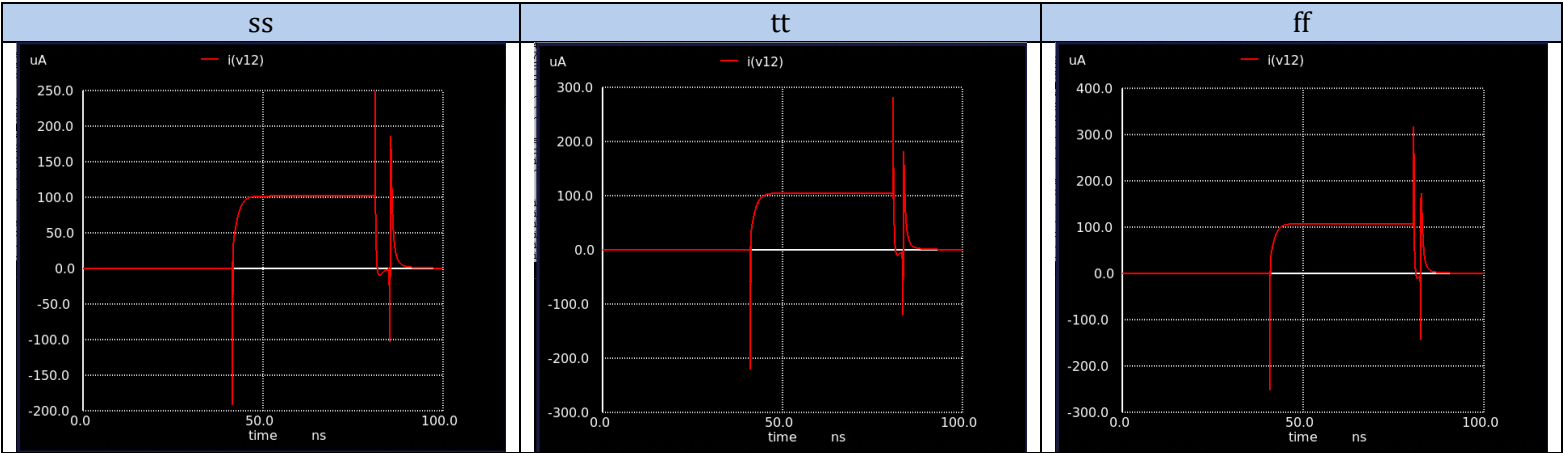


Figure: 51 PFD Input

Table 12 PFD/Charge Pump Full Results



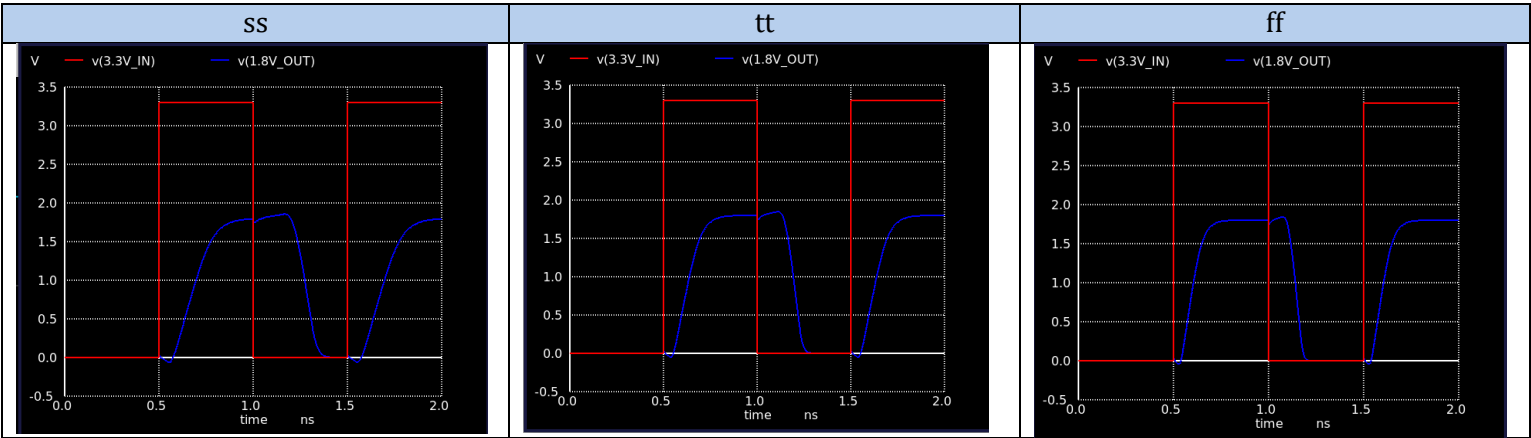
5.8.10. Level Shifters

Unless otherwise specified, a 1fF capacitive load is used.

5.8.10.1. 3.3V to 1.8V Level Shifter

Simulations of the high frequency level shifter at 1GHz across process variations.

Table 13: 3.3V to 1.8V Level Shifter Simulations

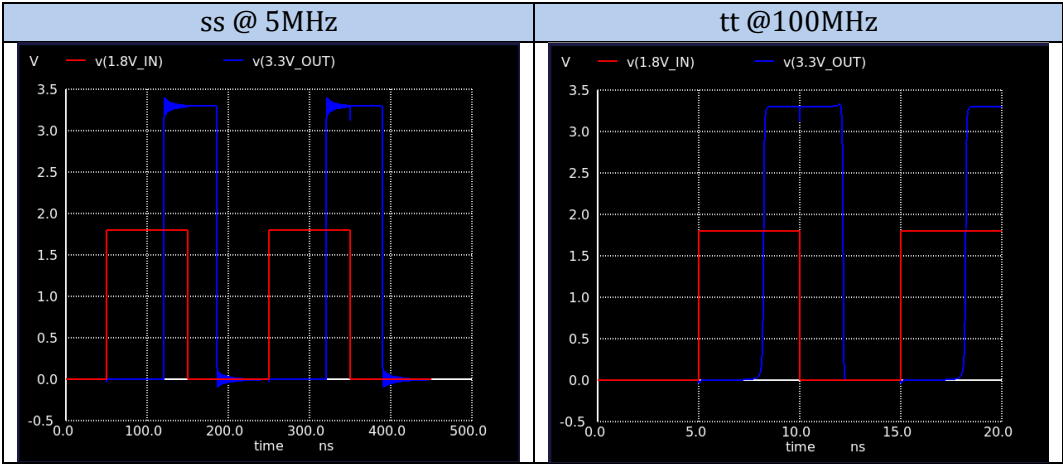


5.8.10.2. 1.8V to 3.3V Level Shifter

5.8.10.2.1. 1.8V to 3.3V Level Shifter Differential

The slow-slow corner did not optimally operate at the 10MHz. The maximum frequency is between 5 and 7 MHz. I changed the topology to a current mirror shifter.

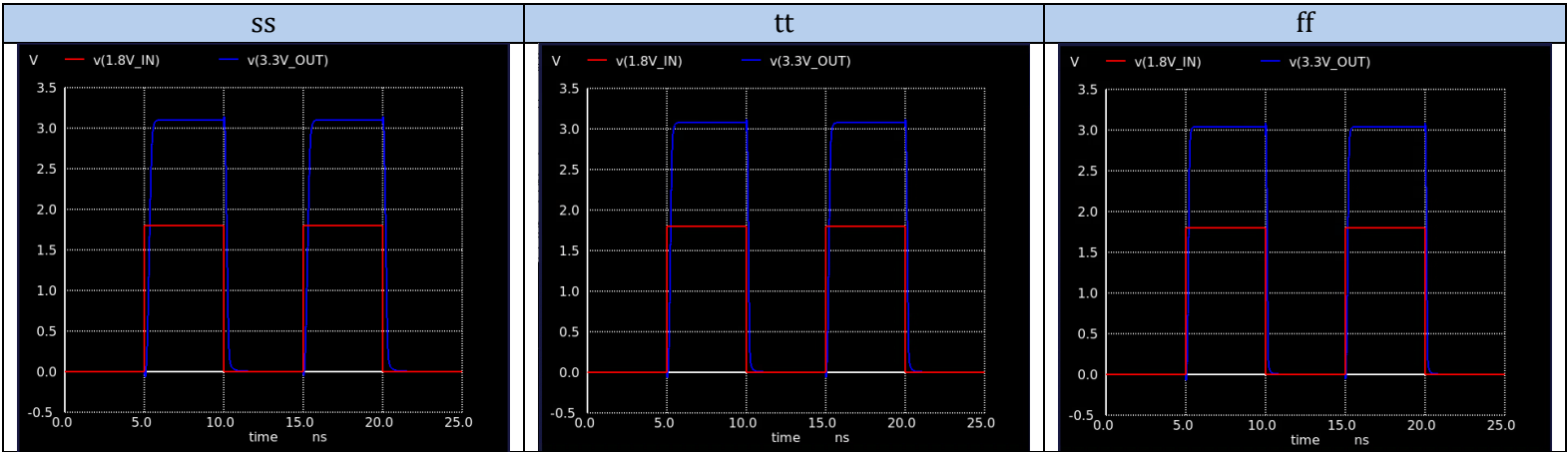
Table 14: 1.8V to 3.3V Level Shifter Differential



5.8.10.2.2. 1.8V to 3.3V Level Shifter Current Mirror

These simulations were conducted at 100MHz which surpasses the required 10MHz PFD frequency.

Table 15: 1.8V to 3.3V Level Shifter Current Mirror Simulations



6. Implementation and Design Analysis

6.1. WISHBONE CROSSBAR

The Wishbone Crossbar is a critical part of the system, since it is responsible for routing bus traffic to and from both VexRISC-V cores, memory, and peripherals. To make sure that the crossbar functioned as intended, it was originally written in a fixed 2x2 (two master, two slave) format to verify functionality. However, this approach was not suitable for the final design, since there would be more than two masters and more than two slaves. To avoid having to do time consuming and error-prone manual rewrites of the crossbar to add or remove masters and slaves, a Python script was created that could procedurally generate the crossbar for any number of masters and slaves. Once the script outputs were confirmed to work, the script was used to create the crossbar instantiated in the top level of the design.

The crossbar follows the rules outlined in the design section to ensure that it has well-defined behavior. This includes having a consistent and well-defined arbitration mechanism, preventing two masters from accessing the same slave simultaneously. Whichever master requests access to a slave first will be granted access provided nothing else is currently using the slave. There is no mechanism for preemption, where one master can interrupt another master trying to access a slave. This is primarily to reduce complexity, and because the Wishbone masters that are used in the project are simple and do not hold onto the bus for more than three clock cycles if the slave functions correctly. If two masters attempt to access the same slave in the same clock cycle, the master connected to the lower numbered Wishbone interface will win arbitration, and the other master(s) will stall until the first master has finished. This helps make sure that behavior is consistent with arbitration, and by providing a priority system, the system can ensure high-priority tasks won't have to wait as long for resources.

In addition to following the rules about well-defined behavior, the crossbar also provides additional functionality to improve performance. If two masters attempt to access two different slaves at the same time, both transactions can be run in parallel. This provides a substantial bandwidth increase over a shared bus architecture, effectively multiplying the bus throughput by the number of masters. In the final design, there are three Wishbone masters, including one to fetch instructions for the second RISC-V core, which needs consistent bus access. By allowing parallel accesses, the instruction fetch doesn't prevent other masters from accessing other parts of memory, which allows for much greater performance while also allowing the instruction memory to be modified externally.

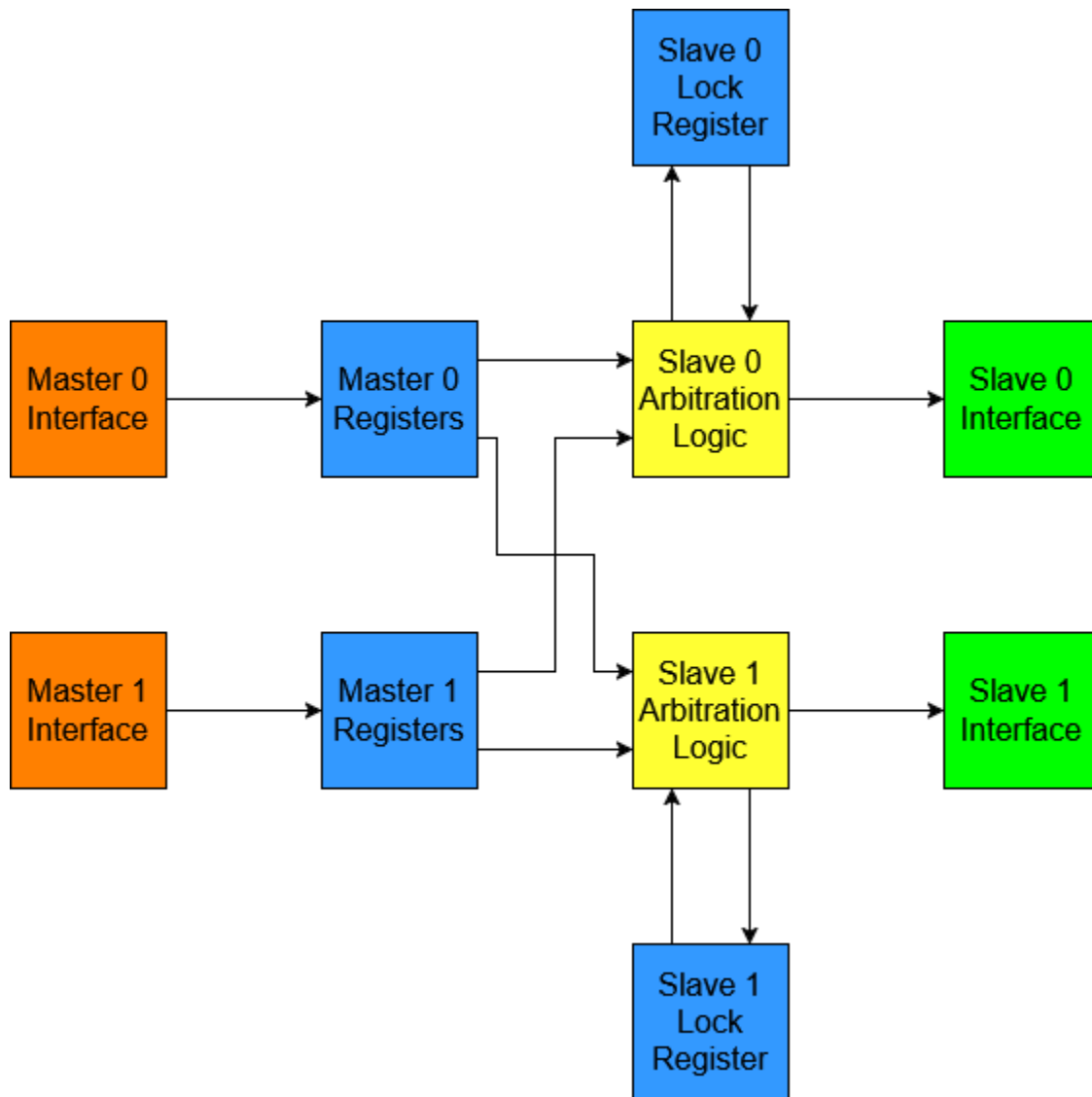


Figure 52: Sample 2x2 Wishbone Crossbar Architecture

6.2. VEXRISC-V CORE

Since the existing RISC-V in the management area of the Caravel chip will not be accessible to the end users, we are implementing our own RISC-V core in the user area using the existing VexRISC-V generator. This core will run the user programs while leaving the management core to be saved for any housekeeping or setup instructions that need to be executed. VexRISC-V makes use of the SpinalHDL project, which is written in the Scala programming language. To aid with generating a custom RISC-V core, many example generation configurations were provided in the VexRISC-V GitHub repository.

To generate the RISC-V core we will be using for the project, we referenced the “wishbone” example and the “smallest processor” example. To keep the die usage small and to keep the overall core implementation simple, we decided to go with most of the settings from the “smallest processor” example, which doesn’t have fancy features such as caches, smart branch prediction, and designated multiplier and division modules. Then, we referenced the “wishbone” example to modify the existing instruction and data bus structures to be wishbone compatible. Once this new configuration was complete, we simply followed the build instructions on the VexRISC-V GitHub repository to generate a single Verilog file that contained the entire RISC-V core. This was then implemented into our top-level design and hooked up to the wishbone bus architecture.

6.3. DFF RAM

Since we are adding a RISC-V core for user programs to run on, it will need an instruction RAM and a data RAM. To implement these RAM blocks, we used an existing DFF RAM macro provided on the Efabless marketplace. This gives us the layout in the 130nm PDK for a tested RAM module that has a simple structure. The largest macro size is 2 KiB and takes up around 1.21 square millimeters of die space. However, the macro is not wishbone compatible and only has a simple single-port interface for reads and writes. So, we had to create a wishbone slave interface to enable the user area RISC-V core as well as the management core to read and write the RAM modules. When memory accesses are made on the cores, the core’s master wishbone signals are populated and the DFF RAM modules need to be able to read those signals to perform the desired function and then let the core know that it is done. After the DFF RAM macros were instantiated and hooked up to the wishbone bus, we were able to connect them to the wishbone crossbar and verified that the management core can read and write to them. See the testing section of this document for more information on the DFF RAM testing.

6.4. I2C CONTROLLER

We implemented the I2C controller as described in our final design. The I2C controller uses the wishbone register file module to enable configuration from the wishbone bus. This means that any device that can write to the wishbone bus can configure the I2C controller. Specifically, both the management SOC RISC-V core and the user area RISC-V core can configure the I2C. The register file was configured to have the five registers described in the design section, which are: control, status, address, write data, and read data. To implement the controller that will assert the SDA and SCL lines, we made use of a large Finite State Machine (FSM) in addition to a clock divider and a bit counter. The FSM keeps track of what part of the I2C frame the controller is on and advances to the next parts of the frame when appropriate. In addition, the FSM provides the clock divider and the bit counter with their new values, which are applied on the next rising edge of the 10 MHz input clock. The clock divider divides the 10MHz to 100KHz to be the I2C clock, SCL. The bit

counter keeps track of which bit of the address and data values that need to be sent out on the SDA line. When all of these components work together, the I2C controller can perform reads and writes on the I2C bus.

6.5. AES SUBSYSTEM

We decided that the best way to implement the AES encryption subsystem was by utilizing a preexisting open-source option. We chose an open-source option over creating our own encryption block because AES is widely used so there are a lot of efficient and reliable open-source options available to us. We considered many open-source encryption options and evaluated the pros of cons of each to make sure we chose the best for our project.

6.5.1. Open-Source AES Options

Table 16: Open-Source AES Encryption Options

| Open-Source Option | Pros | Cons |
|-------------------------------------|--|---|
| E-Fabless AES | <ul style="list-style-type: none"> • Extensive documentation • Can be compacted to reduce space • Multiple cipher modes including CTR • Separate cipher and decipher blocks • Already tested and implemented on multiples ASICs and FPGAs | <ul style="list-style-type: none"> • Larger space required for decryption due to key expansion • The implementation process is iterative making it slightly slower • E-Fabless shutdown making it harder to access |
| OpenCores tiny-AES [10] | <ul style="list-style-type: none"> • Separate cores for 128-, 192-, and 256-bit encryption • Pipelined architecture to increase speed • Multiple cipher modes including CTR • Has been synthesized and tested on FPGAs | <ul style="list-style-type: none"> • May have issues integrating with the WishBone bus • Encryption and decryption are not in separate blocks |
| Various GitHub Options [11][12][13] | <ul style="list-style-type: none"> • Well organized and modular Verilog code that is easy to understand or modify • All GitHub options had extensive testbenches to make sure code was running properly | <ul style="list-style-type: none"> • Many lacked good documentation • Not many cipher modes are offered • Untested on hardware |

E-Fabless AES

This option was available through the E-Fabless marketplace. Unfortunately, during this project, the E-Fabless Corporation has shut down to a lack of funding, but we managed to gain access to the repository with the AES encryption prior to their shutdown.

6.4.2 Selected Option

After analyzing the pros and cons of each of our options we decided to use the E-Fabless AES option due to its versatility, documentation, and extensive previous testing on FPGAs and ASICs. E-Fabless shutting down during this project may complicate things but as of now we still have access to the repository and the documentation and will try to move forward with that.

6.6. PLL

The main components of the PLL are designed in addition to some level shifters because of the VCO and the Charge Pump use 3.3V supply instead of the 1.8V which the rest of the components use. This increase was necessary for optimal operation and saturation of MOS devices.

6.6.1. Fractional Divider

As derived in section 4.3.2.1.5.3 the denominator for the dual modulus subset will be:

$$M \times (A - N) + N \times (M + 1)$$

$$\text{Where } M = 9, A = 10, \text{ and } 0 \leq N \leq 3$$

Therefore, the prescaler values are 9-10, the fixed divider value is 10, and the counter range is 0-3. The accumulator must create a fractional average in the form $\frac{F}{10}$.

6.6.1.1. Prescaler

As derived in section 4.3.2.1.5.4, the prescaler will use four flip flops with Boolean equations:

$$T1! = c! \times Q4$$

$$T2! = Q1! + Q4$$

$$T3! = Q1! + Q2!$$

$$T4 = (c * (Q1! + Q4!) \times (Q1! + Q2! + Q3!) + c! \times (Q4! \times (Q1! + Q2! + Q3!)))!$$

Figure 53 shows the schematic of the circuit Xschem. The flip flops are connected synchronously to avoid a ripple delay.

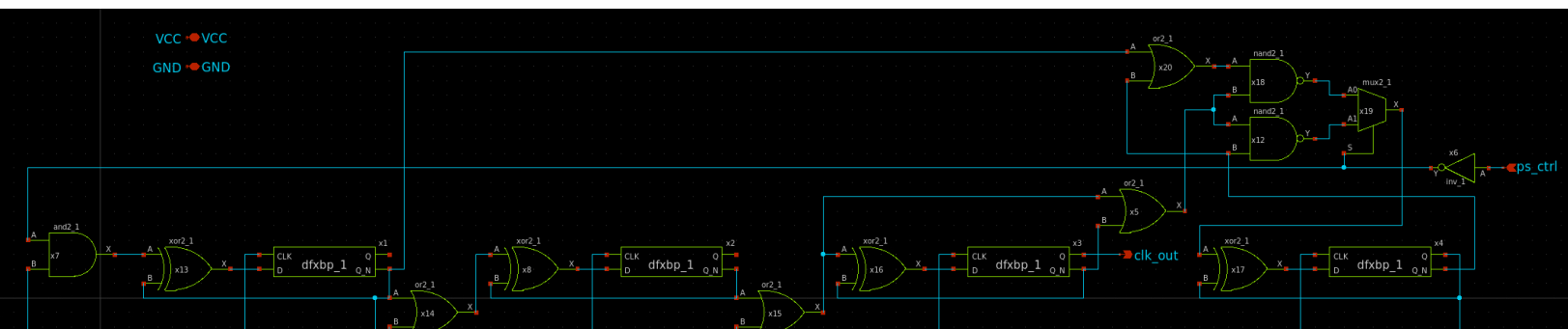


Figure 53: Prescaler Xschem Schematic

All components come from the Sky130A standard cell library. The output is taken from the third flip flop to create a more even duty cycle.

After verifying functionality as described in section 4.3.2.1.5.4, The prescaler was laid out on the Sky 130nm process in Magic. Due to time constraints, this is the only component that was laid out for the divider. It was chosen because its high operating frequency results in the most significant behavior changes during layout, and it determines the maximum operating frequency of the divider. Figure 54 shows the final layout of the prescaler.



Figure 54: Final Prescaler Layout

Clk_in is a delicate signal, being the output of the VCO. Because of this, clk_in is connected to each flip flop using an H-bridge design that keeps the wire 10 microns away from anything running in parallel to prevent signal interference. The H-bridge also limits the difference in distance from the pin to the different flip flops.

6.6.1.2. Programmable Counter

As derived in section 4.3.2.1.5.5, the counter uses two flip flops and the output is:

$$max = R! \times (c0 \times c1 \times Q1! + c0 \times Q1! \times Q2! + c1 * Q2!)$$

$$\text{Where } counting = (c0 \times c1 \times Q1! + c0 \times Q1! \times Q2! + c1 * Q2!)$$

The Boolean equations for the flip flops are:

$$D1 = R! \times Q1 \times counting + (R + counting)! \times Q1$$

$$D2 = R! \times Q1 \times Q2 \times counting + (R + counting)! \times Q2$$

Figure 55 shows the schematic of the circuit in XSchem.

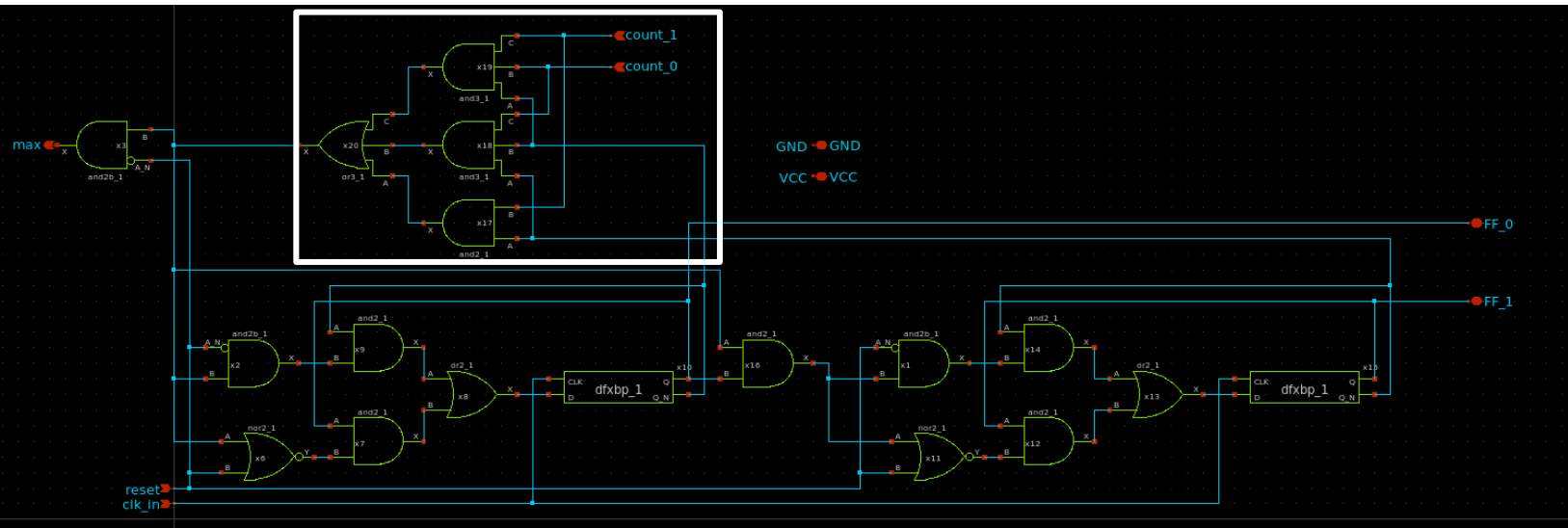


Figure 55: Counter XSchem Schematic

This component consists of several more logic gates than the prescaler, however it operates at a lower frequency. The white box represents the *counting* equation.

6.6.1.3. Fixed Divider

As derived in section 4.3.2.1.5.6, the fixed divider uses four flip flops with Boolean equations:

$$T1 = 1$$

$$T2 = Q1 \times Q4!$$

$$T3 = Q2 \times Q1$$

$$T4 = Q1 \times Q4 + Q1 \times Q2 \times Q3$$

Figure 56 shows the schematic of the circuit in XSchem.

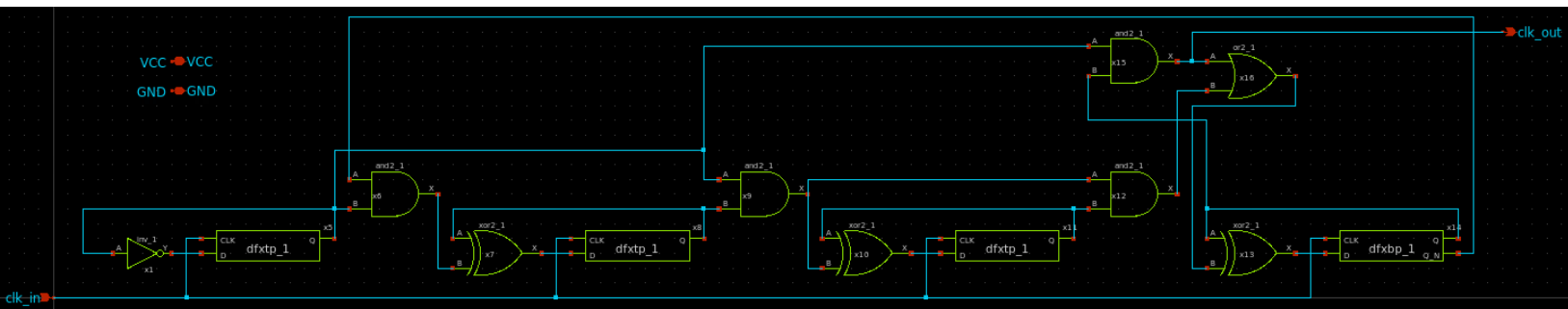


Figure 56: Divider XSchem Schematic

6.6.1.4. Accumulator

As derived in section 4.3.2.1.5.7, the input to the register is:

$$D1 = a0$$

$$D2 = a1! \times c3 + (a3 + c3)! \times a1 + a1! \times a2 \times c3$$

$$D3 = a1! \times c3 + a1 \times a2 + a2 \times a3!$$

$$D4 = (a1 + a2)! \times a3 + c0 \times c3 \times (c1 + c2)!$$

To determine if overflow occurred:

$$Ov = a2 \times a3 + a1 \times a3 + c3$$

To increment N:

$$N0_out = N0_in \oplus Ov$$

$$N1_out = N0_in \times Ov \oplus N1_in$$

Figure 57 shows an annotated schematic of the accumulator in Xschem.

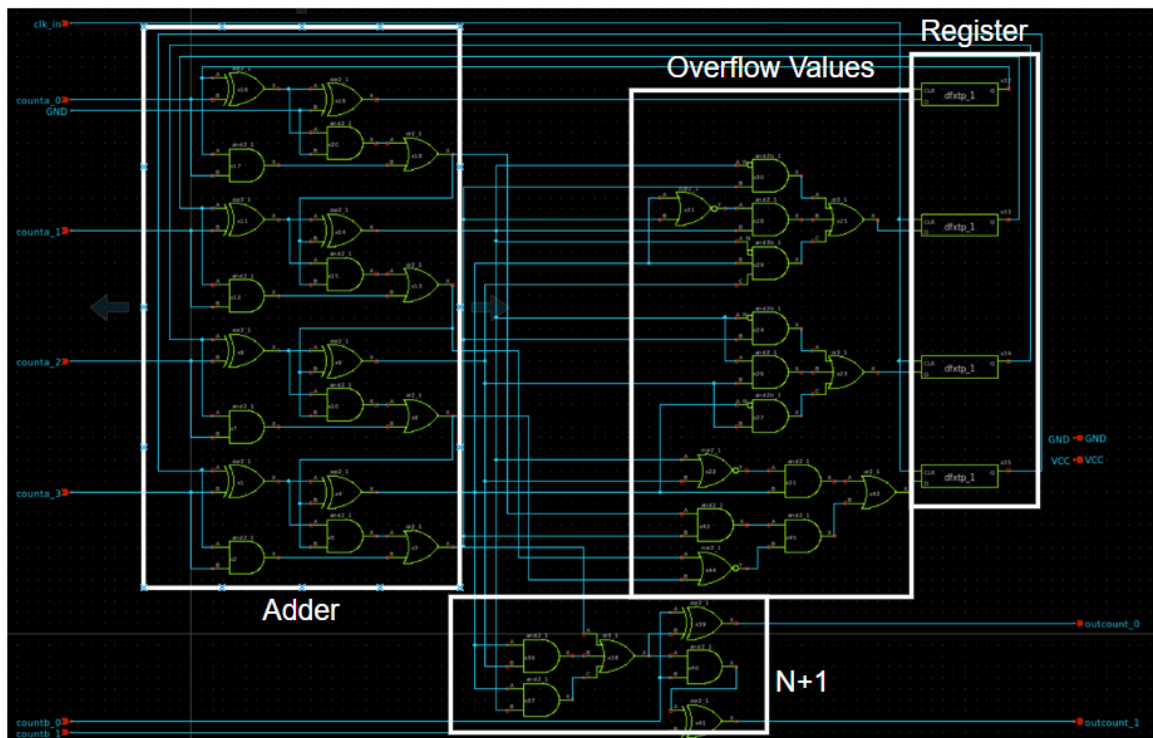


Figure 57: Accumulator XSchem Schematic

With the annotations it is clear how the implemented design matches Figure 58 from section 4.3.2.1.5.7.

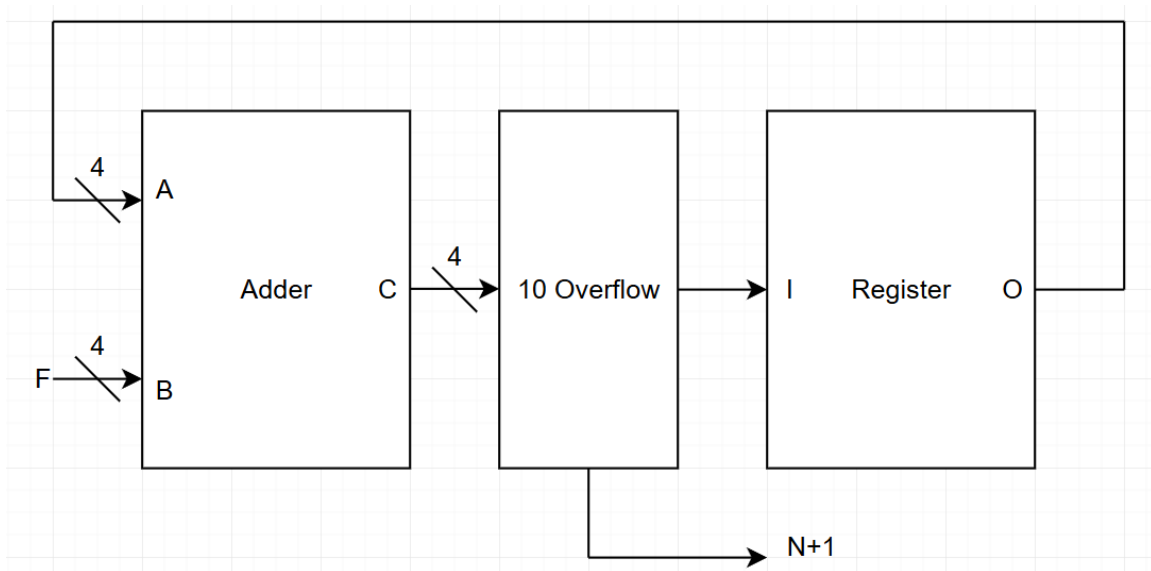


Figure 58: Accumulator Design

6.6.1.5. Divider System

Using the architecture from section 4.3.2.1.5, the complete divider is assembled from the four components covered in this section. Figure 59 shows the circuit in Xschem.

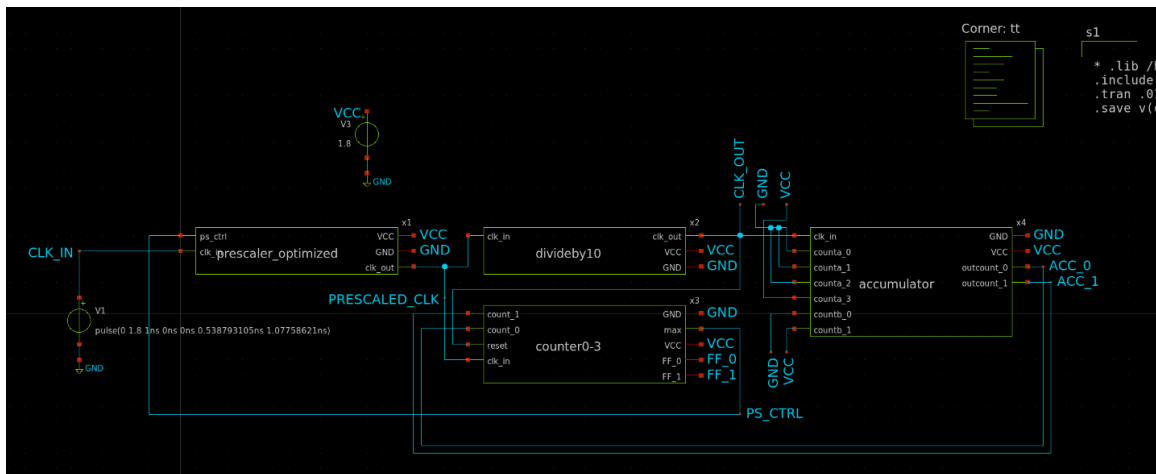


Figure 59: Divider Xschem Schematic

6.6.2. Voltage Controlled Oscillator (VCO)

In my initial design, I utilized the 1.8V supply and the low power oiv8 transistor family. However, due to some signal swing issues with the charge pump, I switched to 3.3V which forced me to use the g5vod1ov5 family.

6.6.2.1. Low Voltage VCO

Initially, I used $L_{min} = 1\mu$ for the PMOS/NMOS current sources but later decreased it due to the 1.03V PMOS threshold drop. I decreased the length to 0.4μ which sacrificed a more linear current

source but decreased the threshold voltage of the PMOS to around 0.9. This resulted in calculated control voltage signal swing of $0.385V \leq V_{ctl} \leq 0.9V$. I also set $W_p = W_n$ to reduce V_{EBN} and resulted in slight enhancement $0.339V \leq V_{ctl} \leq 0.9V$

To get good K_{vco} measurements, I swept V_{ctl} from 0 to VDD and wrote a Ngspice script utilizing TRIG/TARG function to automatically measure and calculate the frequency and duty cycle of the oscillator. To visually plot data, I wrote a python script to read and plot the Ngspice output text file.

Some modifications were needed to the current and inverter channel areas to get the output frequency at 0.8V close to the desired 900MHz range. I also skewed PMOS transistors (M10, M12, M14) to draw more current to get close to 50% duty cycle.

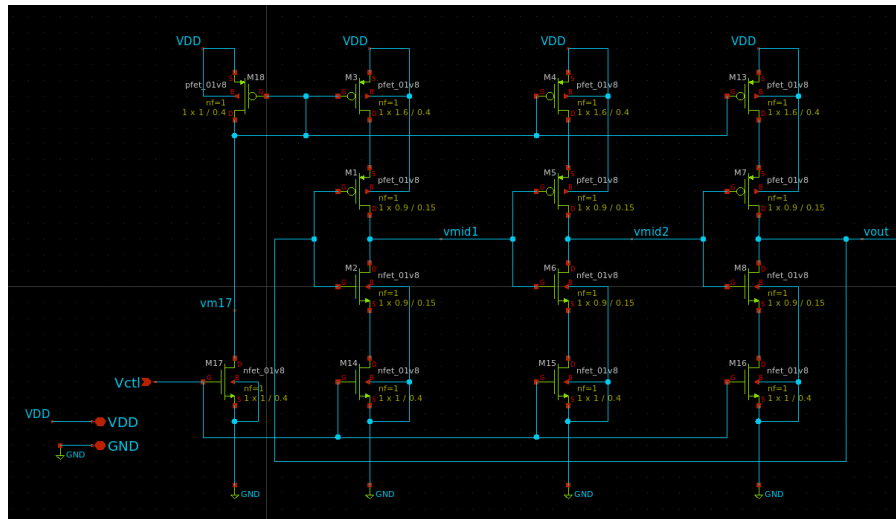


Figure 60: LV VCO Final Schematic

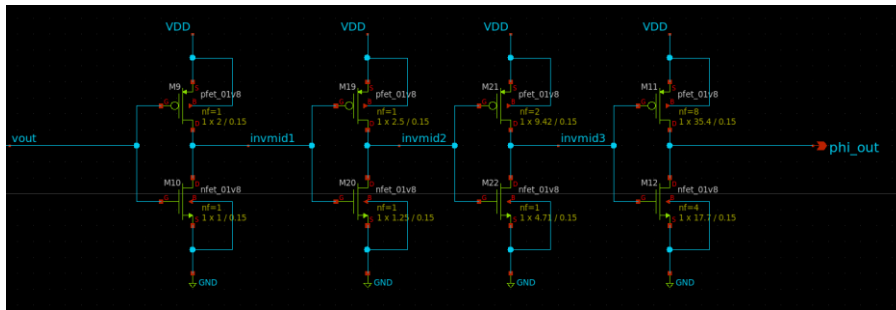


Figure 61: LV VCO Output Buffer

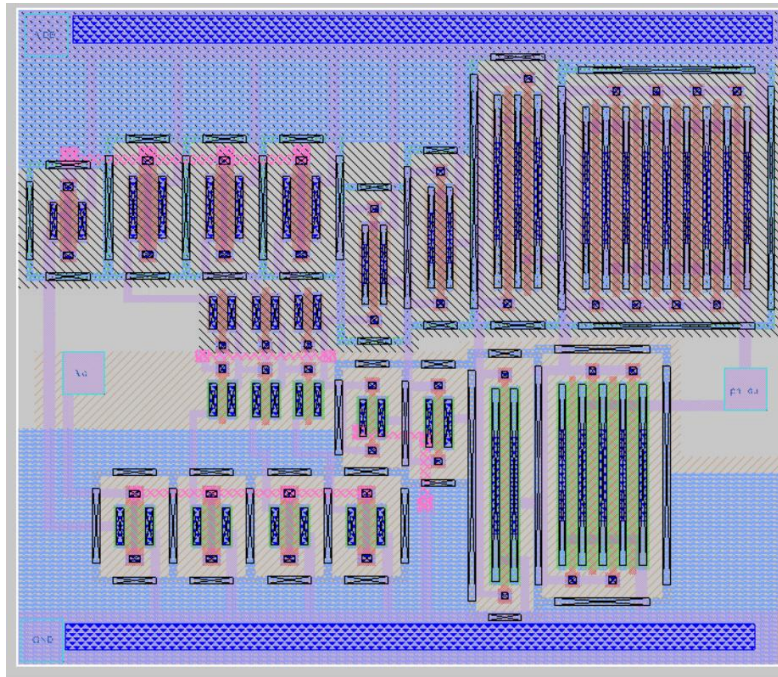


Figure 62: LV VCO Final Layout

6.6.2.2. High Voltage VCO

After laying out and testing the low voltage VCO, I noticed some considerable degradation in the duty cycle. I believe if the PMOS transistors of the inverters were $\frac{\mu_n}{\mu_p}$ instead of 2x larger than the NMOS, the duty cycle would also approach 50%.

Final schematic after re-designing the VCO using the previously used approach and changing the $\frac{w_p}{w_n}$ accordingly:

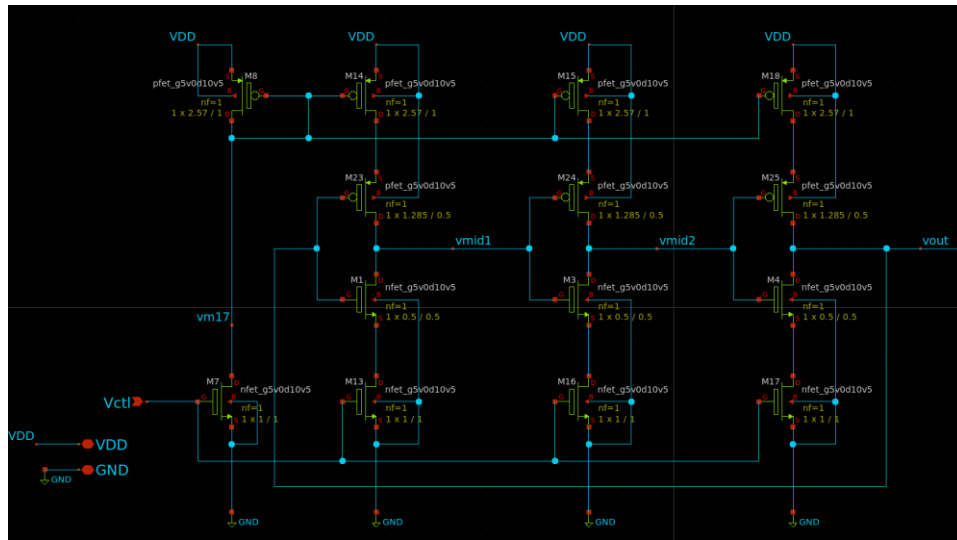


Figure 63: HV VCO Final Schematic

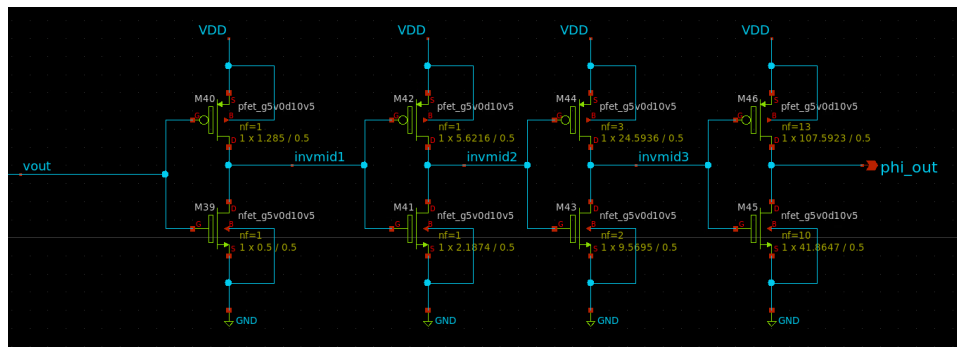


Figure 64: HV VCO Output Buffer

6.6.3. Charge Pump

The Charge Pump circuit is designed using the high voltage device family to effectively supply 100uA to 1mA with a variable off-chip resistor. I decided to use the ESD pad in the Caravel harness which includes a 150ohm series resistor and the ESD diodes. The fixed resistor is accounted for during design and testing. Refer to Appendix 8.3.4 for a generalized circuit analysis.

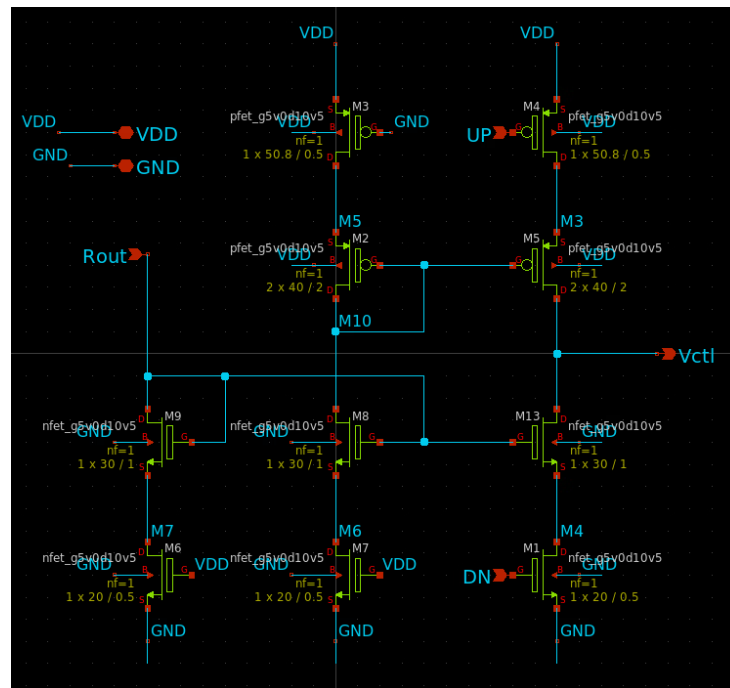


Figure 65: Charge Pump Schematic

6.6.4. Phase Frequency Detector (PFD)

The PFD utilizes DFF, INV, and an AND gates from the 1.8V standard cell library. As discussed previously, an adequate delay block is necessary for reset that is more than twice as much as the accumulated delay between the PFD and the Charge Pump.

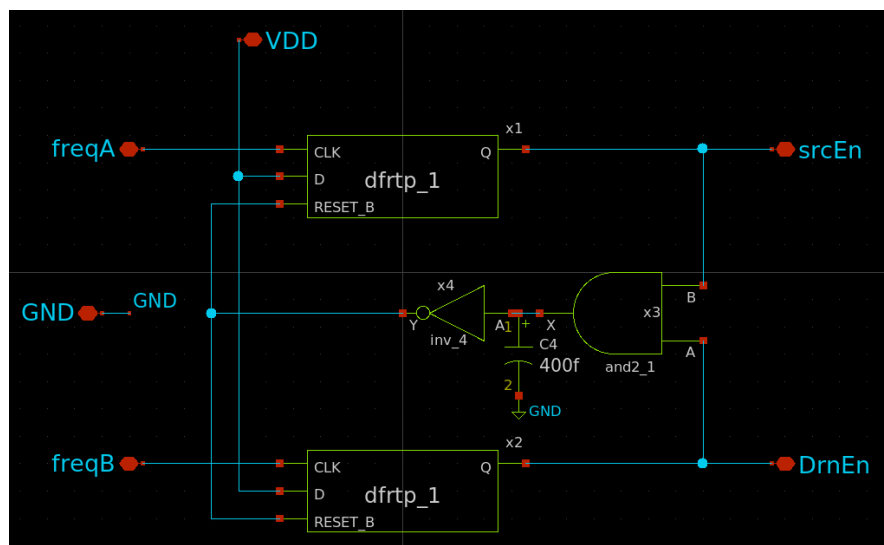


Figure 66: PFD Schematic with Reset Delay

6.6.5. Level Shifters

6.6.5.1. 3.3V to 1.8V Level Shifter

This circuit is used to level down the 3.3V voltage at the output of the VCO to the 1.8V voltage level at the input of the divider. This circuit needs to operate at the VCO high frequency.

This circuit uses a minimum sized inverter that uses the g5vod1ov5 transistor family. This is because the o1v8 family can not handle gate to source voltages greater than the absolute maximum of $|1.95V|$. The second inverter is inserted to invert the signal back and it uses the low voltage transistors.

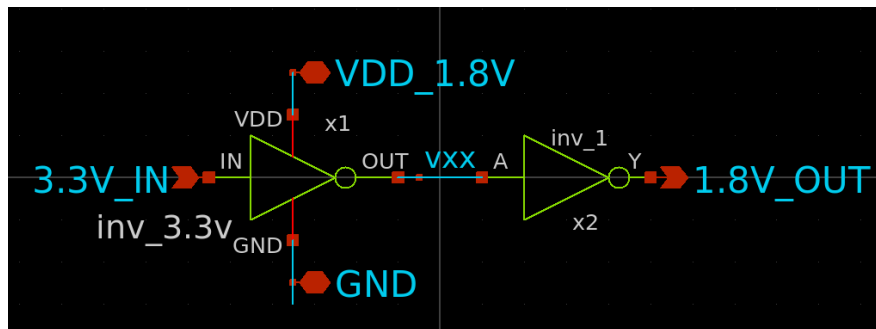


Figure 67: 3.3V to 1.8V Level Shifter Schematic

6.6.5.2. 1.8V to 3.3V Level Shifter

This circuit is used to level up the 1.8V voltage level at the output of the PFD to the 3.3V voltage level needed to drive the Charge Pump drivers. This circuit needs to operate at the 10MHz reference oscillator frequency.

6.6.5.2.1. 1.8V to 3.3V Level Shifter Differential

This circuit uses a basic differential structure to isolate the 1.8V signal from the 3.3V line and to effectively drive the output inverter. This structure does not operate at effectively at the 10MHz frequency in SS process corner. The inverter at the bottom of the figure that inverts the input signal uses low voltage devices. However, the devices use the 3.3V devices to avoid breakdown as voltage nodes approach 3.3V.

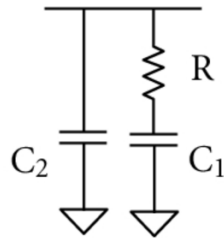


Figure 70: Second Order PLL Loop Filter Schematic

6.6.7. Integrated PLL

This is the integrated PLL design with the PFD, charge pump, buffers, level shifters, 22kOhm off-chip resistor, VCO, and full fractional divider. I included a 150Ohm resistor for the testbench to match the ESD series resistor.

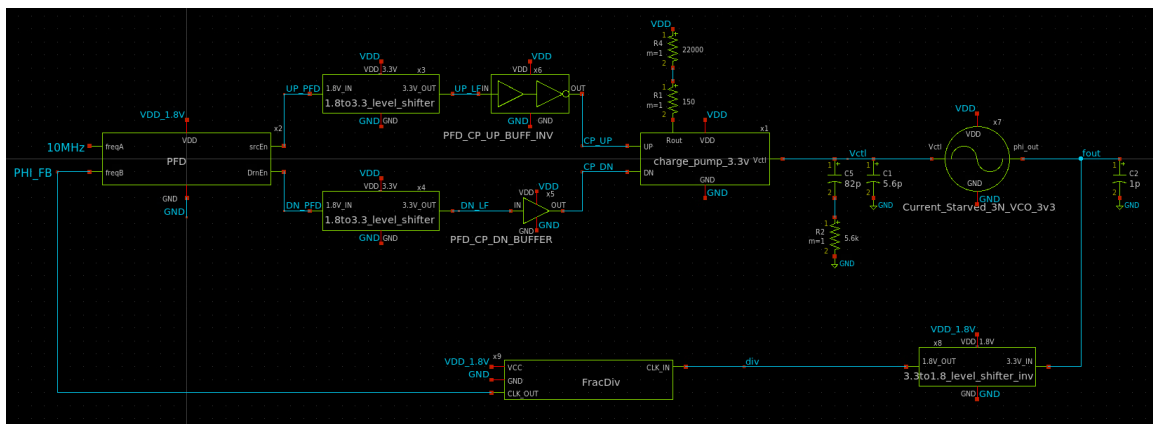


Figure 71 Integrated PLL

7. Ethics and Professional Responsibility

Regardless of the outcome, we wanted to streamline processes for future groups and the ISU Chip Forge co-curricular. We knew that learning the tooling would be difficult, but we could provide solutions and guides to help others avoid our mistakes. In this way, our team practices social responsibility by paving the way for other students. With the closure of Efabless, we can no longer meet our financial responsibility. Our client provided funding for fabrication that can no longer happen. Ultimately, we cannot deliver the project as originally described. Honest communication was important when working with our client to restructure our goals to best meet the remaining requirements.

7.1. AREAS OF PROFESSIONAL RESPONSIBILITY/CODES OF ETHICS

Table 17: Areas of Professional Responsibility and Codes of Ethics

| Area of Responsibility | Definition | Relevant Item from Code of Ethics | Team Interaction |
|-------------------------------|---|--|--|
| Work Competence | Completing assigned tasks and on time. | IEEE Ethics 5, 6 | Team members have put in effort to learn about new tools and processes to ensure that they are competent during design and implementation. |
| Financial Responsibility | Creating a product that meets expectations for its price. | IEEE Ethics 3 | Designed a product that considered the financial amount provided, as well as made it accessible to the public through an open-sourced implementation. |
| Communication Honesty | Being truthful about work that has been completed. | IEEE Ethics 5 | <p>The team has done a good job with timely communication via Discord and reporting hours.</p> <p>The team has made it very clear what work was completed and the state of the project moving forward.</p> |
| Health, Safety, Well-Being | Providing products and services that consider consumer safety, health, and well-being | IEEE Ethics 1 | The team is using industry standards and best practices to make sure our final design is safe and works correctly to provide a positive experience for end users. |
| Property Ownership | Respecting the ideas and products of others. | IEEE Ethics 5 | The team has checked all code used in the project to make sure it is open-source, and we can use it without intellectual |

| | | | |
|-----------------------|---|-------------------|---|
| | | | property infringement. |
| Sustainability | Considering impact on environment and natural resources | IEEE Ethics 1 | Our team is using an older fabrication process that already exists, so fabricating our design does not require many additional resources that could hurt the environment. |
| Social Responsibility | The product or service is beneficial to its users and others. | IEEE Ethics 2, 10 | Our team has designed a product that enables users to learn about all the inter-workings of it. This product can be used in many educational contexts and promotes critical thinking and thinking like an engineer. |

Our team has performed well with work competence, as each member has contributed significantly to the research and development of our project. Each member has been open-minded in creating a positive environment where each member can freely express their creative ideas and obtain feedback from one another. This signifies strong performance because each member actively contributes to design decisions, research, and testing, and has completed their tasks on time. One area that our team needs to improve is sustainability, as while our project does use existing fabrication processes that wouldn't require many resources, our design does not aim to benefit the sustainability of the environment in many ways future to improve, there could be more considerations on power consumption for environmental concerns.

7.2. FOUR PRINCIPLES

Table 18: Four Ethical Principles

| | Beneficence | Nonmaleficence | Respect for Autonomy | Justice |
|------------------------------------|---|---|-------------------------------------|---|
| Public Health, Safety, and Welfare | Our design will implement encryption to protect user data | Our design is designed with best practices to avoid potential safety concerns | Design allows complete user control | Our design is tailored towards students and general education about design process. |
| Global, Cultural, and Social | Allows people without many | Our design will be accessible to | The open nature of the design | The design's documentation |

| | | | | |
|---------------|--|--|---|---|
| | resources to learn about RF MCUs | people with different experiences and backgrounds in engineering | allows different users to use it for their unique purposes | and openness combine to allow users from marginalized communities to use it |
| Environmental | Efables is utilizing previous processes to reduce waste | Our design is using best practices to reduce excess energy | Users will have some control over power consumption | The Efables organization and fabrication facilities already exist, so the fairness of environmental impact is determined by Efables and not us. |
| Economic | Allows for students and educational institutions to fabricate at a much lower cost than alternatives | The Efables organization and fabrication facilities already exist, so implementing this project does not have a large economic impact. | Users can decide to fabricate this design if they want to or just view the artifacts associated with the design for free. | This project is entirely open-source and any existing artifacts we pull in are open-source. This means that no one has to pay to see our artifacts. |

One point that is important to the project is the intersection of Global, Cultural, and Social and Respect for Autonomy. The open-source nature of our design will allow for users to use the design as they please, without the limitations of licensing that would come with closed-source designs. One of the largest constraints on the project is that it be open-source, so this will be implemented by default, but we will also help support this with good documentation, which makes it easier for users to do whatever it is they want with the product.

One point that we have not emphasized on the whole is environmental impact in any category. However, our design is functioning as a proof of concept and a prototype, and thus will not have a significant environmental impact, since only a small number will be produced. Additionally, we have limited ability to change the environmental impact, especially around the manufacturing process, since this is a constraint from our client. That being said, the process is based on older technology, and as such is reducing waste by reusing said technology. This results in a lesser environmental impact than trying to keep up with the most modern processes.

7.3. VIRTUES

7.3.1. Responsibility

The team defines responsibility as fulfilling the tasks assigned and expected of each other, in a timely manner. Being responsible means also being held accountable of what you oversee and making ethical choices. The team has met and will continue to meet the virtue of responsibility by holding each other accountable for the tasks that are assigned and provide updates on the progress of tasks. Team members will notify the team of any pending issues that may arise.

7.3.2. Respect

The team defines respect as treating each other equally and showing regard for each other's abilities and contributions to the team. Being respectful to one another also entails listening to each other's ideas and valuing each other's views. The team has and will continue to show respect to each other by listening to each other's ideas and showing consideration for one another even in disagreements.

7.3.3. Flexibility

The team defines flexibility as a willingness to compromise with one another and embracing new challenges and ideas. The team has already shown flexibility in establishing meeting times outside of class and allowed each other flexibility in work during high stress environments. The team will continue to show flexibility by being considerate of each other's time and commitments and be open to a workflow that best fits the schedules of each other.

7.3.4. Individual Virtue Assessment

7.3.4.1. *Ibram Shenouda*

The virtue that I demonstrated throughout the semester is responsibility. As an electrical engineer who is pursuing a career in VLSI design, this project hugely contributes to my overall goal of becoming a design engineer. I believe I was responsible in learning new engineering concepts about ASIC design and Phase Locked Loops. Because no undergraduate class covers PLL concepts and design implementation, I took on the responsibility of educating myself these concepts to ensure proper working conditions of the PLL.

The virtue that I believe that I have not demonstrated so far is flexibility. Due to my tedious schedule and our large group contributors, it was time consuming to pick meeting times in the beginning of the semester. After a few weeks, we decided to meet on Saturdays which mitigated most of our time conflicts.

7.3.4.2. *Noah Thompson*

The virtue that I demonstrated the most throughout the first semester of this project is flexibility. Our project requires a significant amount of analog design and our team only has one electrical engineering student, Ibram. To implement our desired analog designs, I volunteered to learn analog and mixed signal design to assist in the development of the analog components. I recognized that, while it is not my expertise, we needed a larger analog team to implement the PLL. My goal is to contribute as best I can, where I can to ensure we have a successful project.

The virtue I have not demonstrated as well is responsibility. Due to my lack of knowledge in the mixed signal domain. I spent the start of the semester researching and learning to be able to assist to the analog team. I did contribute to team assignments as much as I should have initially. This improved as I became more comfortable with analog design and felt I could dedicate more time to team assignments.

7.3.4.3. *Nathan Stark*

The virtue I have demonstrated most throughout this semester has been responsibility. I have communicated clearly what my status on issues is, attended meetings to the best my schedule allows, and taken on extra tasks to make sure we got all assignments completed on time. Responsibility is important to me because I know how frustrating it can be to have to work with people who don't take responsibility, and I make it my goal to make sure that I am not contributing to a negative experience for my teammates.

The virtue I have not demonstrated as well is flexibility. I had many other commitments this semester and while I allocated a sufficient number of hours to the project, sometimes they didn't line up well with other team members. This sometimes resulted in miscommunications, which sometimes resulted in inefficiencies where team members ended up waiting for things or I ended up waiting for other team members.

7.3.4.4. *Nolan Eastburn*

So far, I have demonstrated the virtue of responsibility the best in this first semester of senior design. I have clearly communicated what I will work on and have kept to my commitments. At times, I have pushed some commitments a few days ahead, but I have clearly communicated this to make sure my teammates were okay with it. In addition, I committed to getting the seven-segment project into the fabrication tapeout, which took a lot of time out of my week. I made sure that the seven-segment project was implemented by the fabrication tapeout deadline to show that our team knew how to take a concept and fabricate it.

I have identified that one of my weaker virtues is flexibility. I enjoy getting immersed into things I enjoy such as embedded programming and digital design. Because of this, I at times did not communicate with the analog team to understand their design. I was so sucked into my own work that I did not take time to learn about what others were doing, which hurt my overall understanding of the project. In the future, I plan to be more flexible by taking time to learn about what my other team members are doing instead of getting sucked into my own work.

7.3.4.5. *Ethan Kono*

A virtue I have demonstrated throughout this project is respect for the rest of my team members by being an active listener and open to new ideas. Respect is important to me because it enables a positive environment where everyone feels they can contribute ideas freely and equally and provides everyone with the opportunity to express their thoughts in a constructive manner.

A virtue I have not demonstrated throughout this project as well has been flexibility. It is important to me because the team has been great at being flexible with meeting times and actively contributing even when they are busy, but I have not contributed nearly as much when I have been busy with other commitments. Going forward, I can be more flexible with my time by making this project more of a priority and commit more of my time towards the project.

7.3.4.6. *William Custis*

The virtue I have demonstrated during this project is respect for my team members by trusting in the quality of their work, especially when they were working on components that were out of my area of expertise. Respect is an important virtue to me as without it a team will fall apart. A lack of respect tears down trust and teamwork. By respecting my teammates, we are able to work together better and be more efficient, which has been critical to our project.

A virtue that I have been lacking in during the course of this project has been flexibility. While I came to as many of our out of class meetings as I could, I missed a few due to having conflicts at that time. Flexibility is an important virtue in a group project, especially a group project of this size. It will always be difficult to find a time for all six of us to meet as seniors in engineering. I could demonstrate this virtue better by trying to make more time to attend our meetings or offer more times that would work better for me.

8. Closing Material

8.1. SUMMARY OF PROGRESS

As the semester comes to an end, so does the first iteration of this project. It is useful to take a step back and summarize all that our team has accomplished. After all our hard work, we were able to complete the designs, execute tests for the modules listed below. However, some of the analog components do not have their layouts finished yet and are marked as such on the list below.

- Wishbone crossbar
- User RISC-V core
 - User RISC-V core reset module
 - Linker script for user programs
 - User program loader software (management SOC RISC-V loads user programs)
- Instruction and data DFF RAMs
- I2C controller
- AES 128-bit encryption/decryption module
- PLL Components
 - PFD – pre layout
 - Low voltage VCO – laid out
 - High voltage VCO – pre layout
 - Divider
 - Prescaler – laid out
 - Programmable Counter – pre layout
 - Fixed Divider – pre layout
 - Accumulator – pre layout
 - Loop filter – off chip
 - Charge pump – pre layout
 - Reference Oscillator – off chip

8.2. VALUE PROVIDED

This project addresses the described users' needs by providing an open-sourced radio microcontroller that allows users to learn about the underlying components. The documentation provided also allows for ISU chip Forge members to learn about advanced analog and digital components that go into the fabrication of a radio microcontroller. This fits into the broader context of producing material that can be used in learning environments regardless of experience so that anyone interested in the production and development of radio microcontrollers can utilize this project and its documentation. Examples of this value being provided is that the documentation this project provides and the overall project and source code will be made available to the ISU Chip Forge GitLab where members will have access to the work detailed in this document.

8.3. NEXT STEPS

There is currently no team set to take over this project and no organization to fabricate through. The project will continue to be hosted on the ISU Chip Forge GitLab page for club members and senior design to reference and learn from. We hope that the project will resume in the future when a new organization to fabricate through is found. Prior to fabrication, the remaining analog components need to be laid out, and the digital designs need to pass gate-level simulation. Then the two subsystems need to be integrated. Testing the complete system is likely impossible due to computing requirements. Fabricating our design will allow users to experiment with a physical chip providing a unique and valuable learning opportunity.

9. References

- [1] "CC1352P," *CC1352P data sheet, product information and support* | TI.com. [Online]. Available: <https://www.ti.com/product/CC1352P>. [Accessed: 07-Dec-2024]
- [2] *ESP32*. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/425/MFG_ESP32-DEVKITC-VE.jpg
- [3] "At just \$6, raspberry pi pico W brings Wi-Fi to IOT designs - news," *All About Circuits*. [Online]. Available: <https://www.allaboutcircuits.com/news/at-just-six-dollars-raspberry-pi-pico-w-brings-wi-fi-to-iot-designs/>. [Accessed: 07-Dec-2024]
- [4] *STM32*. ST Microelectronics [Online]. Available: <https://estore.st.com/media/catalog/product/s/t/stm32wb09kev6tr.jpg?quality=80&bg-color=255,255,255&fit=bounds&height=&width=>. [Accessed: 2024]
- [5] "TI CC2540F256RHAR," *CC2540* | Buy TI Parts | TI.com, https://www.ti.com/product/CC2540/part-details/CC2540F256RHAR?utm_source=google&utm_medium=cpc&utm_campaign=ocb-tistore-promo-epd_opn_en-cpc-storeic-google-ww&utm_content=Device&ds_k=CC2540F256RHAR&DCM=yes&gad_source=1&gclid=CjoKCQiAgdC6BhCgARIsAPWNWHo4CRbtU3ZF1Rh5Fdk2EehsXaQrK8UR7MPKl9aB3ZX4Ntm79qtXZgoaAj5YEALw_wcB&gclsrc=aw.ds (accessed Dec. 7, 2024).

- [6] “Dual-modulus prescaler,” *Wikipedia*, 03-Dec-2024. [Online]. Available: https://en.wikipedia.org/wiki/Dual-modulus_prescaler. [Accessed: 07-Dec-2024]
- [7] “Digital PLL, All digital PLL, Analog PLL,” *Movellus*, 02-Apr-2023. [Online]. Available: <https://www.movellus.com/all-digital-pll-phase-locked-loop/>. [Accessed: 07-Dec-2024]
- [8] S. Palermo, 2024 [Online]. Available: <https://people.engr.tamu.edu/spalermo/ecen620.html>. [Accessed: 07-Dec-2024]
- [9] Frequency spectrum of sigma-delta modulator output bits. | download scientific diagram, https://www.researchgate.net/figure/Frequency-spectrum-of-sigma-delta-modulator-output-bits_fig10_2977787 (accessed Dec. 8, 2024).
- [10] H. Xing, “AES,” OpenCores, https://opencores.org/projects/tiny_aes (accessed May 4, 2025).
- [11] Gouravo486, “Gouravo486/AES-core-engine-,” GitHub, https://github.com/Gouravo486/AES-Core-engine-?source=post_page-----2bef178db736----- (accessed May 4, 2025).
- [12] Ayusdixit, “Ayusdixit/AES: 128 bit AES implementation,” GitHub, <https://github.com/ayusdixit/AES> (accessed May 4, 2025).
- [13] Michaellehab, “Michaellehab/AES-Verilog: Advanced Encryption Standard (AES128, AES192, AES256) encryption and decryption implementation in Verilog HDL,” GitHub, <https://github.com/michaellehab/AES-Verilog> (accessed May 4, 2025).
- [14] “A study on hardware attacks against microcontrollers,” Federal Office for Information Security, <https://www.bsi.bund.de/EN/Service-Navi/Publikationen/Studien/Hardware-Angriffe/Hardware-Angriffe.html> (accessed Dec. 7, 2024).
- [15] “Attacks against industrial machines via Vulnerable Radio Remote Controllers: Security Analysis and Recommendations,” Trend Micro (US), <https://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/attacks-against-industrial-machines-via-vulnerable-radio-remote-controllers-security-analysis-and-recommendations> (accessed Dec. 7, 2024).

10. Appendices

10.1. OPERATION MANUAL

This section describes the basics of using various parts of the design. It assumes that you have the repository checked out on a computer that has access to both the ChipForge toolchain and the ChipForge command line tool. If you do not have these, please see the ChipForge documentation

(<https://git-pages.ece.iastate.edu/isu-chip-fab/documentation/#/intro/index>) for installation instructions. The pages in the ChipForge documentation will also serve as a good introduction to the basics of the toolflow. Please have a basic understanding of these tools before continuing this guide.

You will also need access to an FPGA if you want to run the design on hardware. Currently the only officially supported device by the ChipForge toolflow is the Digilent Arty A7 100, although the design has unofficially been ported to the Digilent Basys 3, which also uses an Artix 7 FPGA. The general process of porting to other Xilinx FPGAs is described in this manual, but other vendors have not been tested.

10.1.1. Creating, Compiling, and Executing Programs for the User Area RISC-V Core

Inside of your local checkout of the git repository, enter the 'programs' folder and create a new directory with the name of your program. Inside this folder, create a C file that matches the name of the folder. This will be the primary file for your program. Next, copy the example Makefile from the example folder inside of 'programs' and modify it to compile your C program. You can now write your C program. Keep in mind that all the addresses contained in the management area are not accessible to the user area core. A list of accessible addresses and their functions are shown below.

Table 19: Address Map for Wishbone Crossbar

| Address | Function |
|------------|--|
| 0x30000000 | Instruction RAM for RISC-V core. WARNING – Do not modify unless you know what you are doing, this can cause unpredictable behavior. |
| 0x30000800 | Data RAM for RISC-V core. Also accessible to the management core. By default, the stack pointer for the user area RISC-V core is set to the end of this memory. |
| 0x30001000 | Processor reset block. Writing a '1' to this address will reset the processor, and it will remain in reset until the register is cleared to '0'. This cannot be done by your program, since the processor will not be active if reset. Use with caution. |
| 0x30001800 | I2C module, more details on interfacing with this module are provided in section 10.1.2. |

After writing the program, change back into the top level of the repository and run 'make program-<name>', where <name> is the name of the program you created. This will compile your C program into a .hex file, which a Python script will then convert into a simple array. This array will be placed in a C header file, which can be included in management area code with '#include <generated/second_core.h>'.

In a management area program, the array 'PROGRAM' will contain a series of 32 bit words, each one representing an instruction for the user area RISC-V core. Write each of these words to the

instruction RAM starting at 0x30000000. Then, write a '0' to the address 0x30001000 to take the second core out of reset and allow it to begin running.

10.1.2. Using the I2C Module

The digital design contains an I2C module that can communicate with I2C slaves at the base I2C frequency of 100 kHz. It does not support multi-master arbitration but does support clock stretching. The interface is controlled by using the five registers described below.

Table 20: Register Summary for I2C Module

| Register | Offset | Field | Field Name | Function |
|------------|--------|-------|------------|---|
| Control | 0x00 | | | Control signals for module. |
| | | [0] | BEGIN | Writing this to a '1' causes the I2C module to begin a transaction. |
| | | [1] | RW | Writing this to a '1' causes the transaction to be a read, '0' is a write. |
| | | [2] | START | If set to '1', the module will issue a start to the bus and resend the address and R/W bit. If the master already has control of the bus, this will be a repeated start. Note that if the master does not have control of the bus, a start will be sent regardless of this bit state, since this is required by the protocol. |
| | | [3] | STOP | If set to '1', the module will send a stop condition and release the bus after the transaction, otherwise, the master will retain the bus. |
| Status | 0x04 | | | |
| | | [0] | BUSY | If this bit is a '1', the module is currently executing a transaction, if it is a '0', the module is available. |
| Address | 0x08 | | | |
| | | [6:0] | ADDR | This 7 bit field is the address of the I2C slave to be written to/read from. 10 bit addressing is not supported. |
| Write Data | 0x0C | | | |
| | | [7:0] | WDATA | This 8 bit field is the data to be written to the slave if the transaction issued is a write. |
| Read Data | 0x10 | | | |
| | | [7:0] | RDATA | This 8 bit field will contain the data read from the slave on a read transaction after the transaction is complete. |

To begin a single-byte write transaction, place the address and data in the appropriate registers, write the R/W bit in the control register to 0, and set the BEGIN, START, and STOP bits to 1. If you wish to do a multi-byte write, the process is identical, except that the START bit should only be set

on the first byte and the STOP bit should only be set on the last byte. Reads function the same way. For repeated starts, simply set the START bit when you need the repeated start to occur. Make sure to set the R/W bit to the appropriate value, since it may be different.

10.2. USER JOURNEY MAP

USER JOURNEY MAP / EE201 Student RF Research


| <div>  <div> USER INFO <i>"Here's a quote I said!"</i> </div> </div> <div> SCENARIO An undergraduate student is researching radio design with experience from EE201 </div> <div> EXPECTATIONS <ul style="list-style-type: none"> The student is working with the Efabless Process The student is learning from our project </div> | | | | | |
|---|--|---|--|---|--|
| STAGES | CONNECTING TO CARAVEL ▶ | LEARNING DIGITAL DESIGN ▶ | LEARNING ANALOG DESIGN ▶ | RF MODULE ARCHITECTURE ▶ | RF MODULE DESIGN |
| GOALS | To connect to the Caravel Board and load designs onto it. | Create functional digital circuits on the caravel board. | Create functional analog circuits on the caravel board. | Interpret and understand the current design of the RF Module and general RF architecture. | Modify the RF module with your own designs. |
| ACTIONS | <ol style="list-style-type: none"> 1. Connect to the board 2. Flash a tutorial design to the board 3. Experiment with the tutorial design | <ol style="list-style-type: none"> 1. Learn digital design principles 2. Create digital circuits | <ol style="list-style-type: none"> 1. Learn analog signal design 2. Design analog circuits | <ol style="list-style-type: none"> 1. Learn about radio frequencies 2. Learn RF module architecture 3. Learn RF protocol | <ol style="list-style-type: none"> 1. Understand RF module component design 2. Create new RF module components using digital and analog design |
| TIME | Little, medium if spending more time experimenting | Medium, assuming not prior background in digital design | Medium to long. The user already has some experience from EE201, but analog design is more difficult in Caravel | Long, to understand RF at the necessary level of the next stage. | Long, the process for testing and tape-out can take half a year |
| PAIN POINTS | <ol style="list-style-type: none"> 1. Learning how to connect with SSH | <ol style="list-style-type: none"> 1. Debugging will be different from prior experience | <ol style="list-style-type: none"> 1. Analog design is more difficult in Caravel compared to LTSpice 2. Less documentation | <ol style="list-style-type: none"> 1. Misunderstanding components 2. Misreading documentation | <ol style="list-style-type: none"> 1. Testing both analog and digital and locating issues 2. Tape out behaving differently than caravel tests |
| PAIN POINT RESOLUTIONS | <ol style="list-style-type: none"> 1. Clear tutorials 2. Chip Forge as a resource to ask questions | <ol style="list-style-type: none"> 1. Digital circuit examples 2. Resources to learn digital design | <ol style="list-style-type: none"> 1. Analog circuit examples 2. Resources to learn analog design | <ol style="list-style-type: none"> 1. RF module example design 2. Clear documentation and design decision explanations 3. Resources to learn RF architecture | <ol style="list-style-type: none"> 1. Design examples 2. Testing plans 3. Tape-out tutorials |

Figure 72: User Journey Map

10.3. SECURITY ANALYSIS

The security analysis appendix is an overview of the potential attack vectors and vulnerabilities on microcontrollers and radio frequency modules. This appendix splits the analysis into two major sections, 8.3.1.1 which covers the vulnerabilities in microcontrollers, and section 8.3.1.3, which covers the vulnerabilities in radio frequency modules. Each section covers how the different attack vectors work in concept, how they might affect the system, and proper countermeasures and solutions for each attack vector or vulnerability. The conclusion is meant to serve as a way to analyze which attack vectors are of the highest priority and concern within the context of this project.

10.3.1. Attacks & Vulnerabilities in Microcontrollers

Control Flow Manipulation Attacks:

Control flow determines the order in which instructions are executed. Attacks on control flow aim to modify the flow of execution on a program by executing malicious code or prevent code execution to bypass password checks or encryption. Some of the most common types of physical attacks on embedded devices and microcontrollers according to the Federal Office for Information Security [10] are on voltage and clock glitching, electromagnetic fault injection (EMFI) and laser fault injection (LFI).

Fault Attacks:

Fault attacks are a subunit of control flow manipulation attacks that specifically used for causing “erroneous behavior” such as bypassing execution of specific instruction(s). Corrupting data on memory, corrupting data during a bus transfer, modifying the instructions in the program, or changing the program counter to execute instructions in different orders [14].

Side-Channel Attacks:

- Physical properties of the microcontroller can be observed to obtain a cryptographic secret.
- Power consumption and electromagnetic emissions are the most exploitable in microcontrollers

Encryption & Key Distribution:

Encryption is the bare minimum of security in wireless systems. When communicating wirelessly with radio frequencies it is very easy for messages to be intercepted. While most information being transmitted using this microcontroller will not be of critical importance, having the choice to encrypt a message or not could provide a good learning opportunity for students or club members to learn about the importance of encryption.

AES is the encryption recommended by ZigBee documentation. Additionally, it is one of the most popular encryption methods for wireless communications and is incredibly secure.

AES is a symmetrical cipher so the receiving device will need to have the key used to encrypt the message to be able to decrypt it. Data is encrypted using the network key on a ZigBee network and every device on the network has the same password key. There are a couple options the ZigBee protocol supports for distributing the network key to new devices, but we will focus trust center networks for this project as we will always have our one device, we trust being our own microcontroller. When a new device wants to join the network, it will contact the trust center which can then send a copy of the network key to the new device, directly disallowing it from joining, or just ignore it preventing it from joining the network. For the purposes of our project, we will use a simplified version of this where we either allow or disallow the device from joining as we will likely only be ever communicating between our device and one other.

10.3.2. Countermeasures and Solutions for Vulnerabilities in Microcontrollers

Countermeasures to control flow manipulation attacks can be implemented in both hardware and software, and which one is implemented depends on the scenario. Listed below are the pros and cons of both hardware and software implementations.

- Hardware Pros:
 - o Less impact on system performance as it doesn't use execution time from the CPU
- Hardware Cons:
 - o Can be expensive to implement
 - o Cannot be retrofitted after development
 - o Require modification of the system hardware and potential additional peripherals.
- Software Pros:
 - o No special hardware is required
 - o Retrofitting of additional security measures can be done via software updates
- Software Cons:
 - o Compiler optimizations may have to be disabled
 - o Increases code base and complexity for developers

The actual solutions for control flow attacks include the following list below. Each of these potential solutions can be implemented in hardware or software, depending on where or how it is implemented, but are all potential options to consider when mitigating control flow manipulation attacks.

Classic Loop Hardening:

- Involves duplicating loop counters & exit conditions so that each condition is checked twice before exiting the loop.
- Mainly utilized to detect fault injection attempts.

Instruction Redundancy:

- Execute critical instructions at least two times, to ensure the same result is returned, meaning even if instructions are skipped, the critical instruction will still execute, even if it is skipped.

Function Duplication:

- Involves duplicating a function, and having both functions take the same input, but store the results in different variables and compare both after execution to detect fault attempts.

Countermeasures to side-channel attacks:

Masking:

- Involves applying random masks to secret values in order to create secret shares that are used for cryptographic computations [15].

Blinding:

- Like masking but utilizes an algorithm to mask sensitive values [15].

Shuffling:

- Hides the order that values are processed, and schedules operations at random, which thereby hiding the side-channel measurements with secret information [15].

10.3.3. Attacks & Vulnerabilities in Radio Frequency Modules

Replay Attacks:

“Replay attacks record the RF packets and replay them to obtain basic control of the machine” [15]. This involves the attacker intercepting a valid message and then either delays it or resends it or misdirects it. In extreme cases where sensitive information such as passwords are leaked, an attacker may retransmit the sensitive information after a delay, leaving little indication that the data was even stolen or compromised.

Command Injection:

Command injection attacks in a radio frequency context often involve the attacker recording commands via a data transmission of radio frequency module, capturing data and utilizing reverse engineering to derive other commands and then retransmitting known commands to attack a system. “Knowing the RF protocol, the attacker can arbitrarily and selectively modify RF packets to completely control the machine” [15].

E-Stop Abuse:

“Attack can replay e-stop (emergency stop) commands indefinitely to engage a persistent denial-of-service (DoS) condition” [15]. This type of attack looks to take advantage of specific instructions sets or features on radio frequency modules, ones that specifically send signals to the receiver unit in an emergency that cuts power off. Attackers who intercept these commands can create denial of service situations by preventing the radio frequency module from transmitting any data by consistently sending an e-stop command to prevent it from running.

Malicious Repairing:

“The attacker can clone a remote controller or its functionality to hijack a legitimate one” [15]. This attack mostly applies to radio frequency modules that allow for a “cloning” feature that allows creation of copies of transmitting units. A clear example would be having multiple transmitting units but only one receiver, which would allow multiple operators to control a single receiver.

Malicious Reprogramming and Remote Attack Vectors:

“The attacker ‘trojanizes’ the firmware running on a remote controller to obtain persistent, full remote control” [15]. This attack utilizes scenarios where IT endpoints are not secured, allowing any firmware to be flashed to the microcontroller. If the microcontrollers are not code-protected and allow for flashing of the microcontroller’s memory to reassign and reconfigure a device before it is installed or sold, allowing for potential backdoors or harmful code to be installed on the device. In cases like this project that are open-sourced, it is important to consider.

10.3.4. Countermeasures and Solutions for Vulnerabilities in Microcontrollers

Countermeasures for replay attacks:

- Encryption
- Authentication from network protocol

Countermeasures for command injection attacks:

- Utilize open-standard RF protocols
- Encrypted communication protocols
- Authentication

Countermeasures for e-stop abuse attacks:

- Encrypted data transmission
- Limited access control to sensitive commands

Countermeasures for malicious re-pairing attacks:

- Code obfuscation makes it difficult to reverse engineer.
- Secure boot mechanisms, ensuring integrity of firmware

Countermeasures for malicious re-programming and remote attack vectors:

- Firmware rollback capabilities to restore to previous versions in case of compromise.

10.3.5. Conclusion

Due to the scope of this project, not every security concern can be implemented. The vulnerabilities are listed below based on the level of priority, being either high priority, medium priority, or low priority, with each choice being made regarding how likely the attack is to take place, its difficulty to pull off, and how reasonable it is given our projects environment.

High-Priority:

- Encryption

Medium-Priority:

- Replay Attacks
- Control flow manipulation attacks
- Fault attacks

Low-Priority:

- Side-channel attacks
- Malicious repairing attacks
- Malicious reprogramming and remote attack vectors

Encryption makes the most sense to implement, as it prevents attacks and vulnerabilities on both microcontrollers and radio frequency modules and is a minimum level implementation of security that meets the project environment. A form of encryption covers against a lot of different possible attack vectors, providing a strong basis that would deter most beginner level attackers. Due to this project likely being used in a class and project environment, attempting to provide too much

security takes away other features this project aims to implement, especially if the countermeasures to some of the attack vectors were implemented in hardware with the limited die-space that the Efabless process provides. Even software implementations would somewhat hinder the user from using this project to its fullest extent. Since this project likely won't be sold commercially as well and serves as a learning platform, noting the security concerns while providing users options in securing the device itself allows the user to make educated decisions on what level of security best fits what the user wants to experiment with when using this device.

10.4. PLL CURRENT STARVED VCO CIRCUIT ANALYSIS

Current Starved Oscillator:

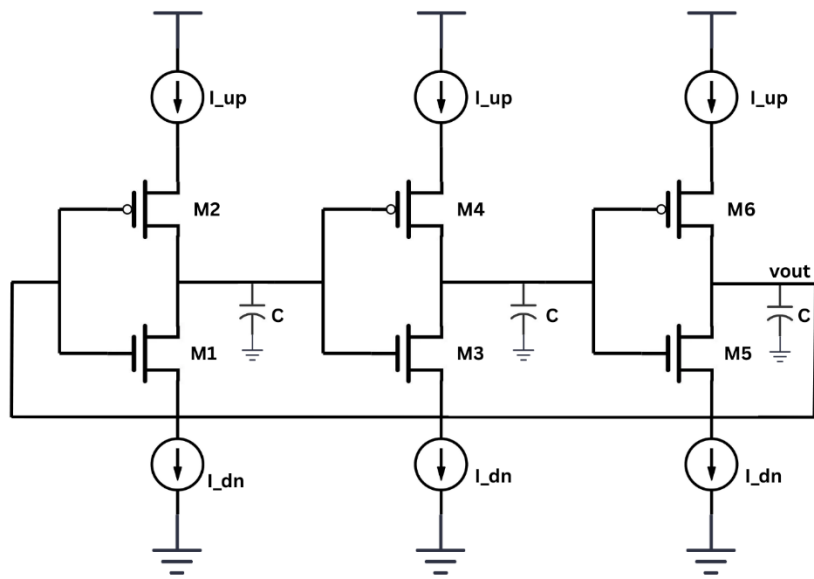


Figure 73: Current Starved Oscillator

The oscillator mainly consists of an odd number of inverters in series. The frequency of the oscillator is equivalent to the inverse of the accumulated delay of the inverters. The delay of each inverter is equal to:

$$T = t_{lh} + t_{hl}$$

For equal rise/fall time, the source and sink currents must match. This will affect the duty cycle of the oscillator. We believe a 40% - 60% duty cycle across corners is acceptable.

$$I_d = I_{up} = I_{dn}$$

$$t_{lh} = t_{hl} = C \frac{V_{DD}}{2I_d}$$

$$T_{total} = NT$$

$$F_{out} = \frac{I_d}{NV_{DD}C}$$

The total capacitance at the output of each inverter stage is proportional to C_{ox} and the channel area of the inverter mosfets. W_p, W_n, L_p, L_n are inversely proportional to the F_{out} while I_d is directly proportional to the frequency.

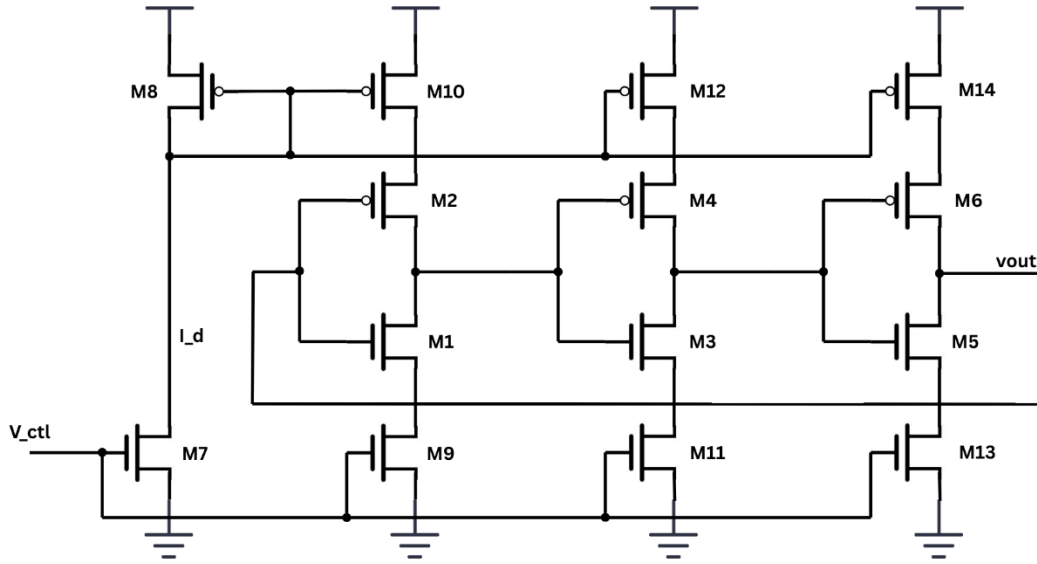


Figure 74: Unbuffered Voltage Controlled Oscillator Schematic

For the initial design, I used minimum sized inverters and mapped those currents to an NMOS-input current source with a diode connected PMOS.

$$I_{M7} = \frac{1}{2} \mu_n C_{ox} \frac{W_n}{L_n} (V_{ctl} - V_{thn})^2$$

$$I_{M8} = \frac{1}{2} \mu_p C_{ox} \frac{W_p}{L_p} (V_{DD} - (V_{ctl} - V_{thn}) - |V_{thp}|)^2$$

$$I_{m8} = I_{m7}$$

$$V_{EBN} = V_{ctl} - V_{thn}$$

$$\frac{W_p}{L_p} (V_{DD} - V_{EBN} - |V_{thp}|)^2 = \frac{\mu_n}{\mu_p} \frac{W_n}{L_n} (V_{EBN})^2$$

$$\text{for simplicity, } L_{MIN} = L_p = L_n$$

$$V_{EBN} = \frac{V_{DD} - |V_{thp}|}{1 + \sqrt{\frac{\mu_n}{\mu_p} \frac{W_n}{W_p}}}$$

To keep the transistors in saturation,

$$V_{EBN} \leq V_{ctl} \leq (V_{DD} - |V_{thp}|)$$

Buffered VCO:

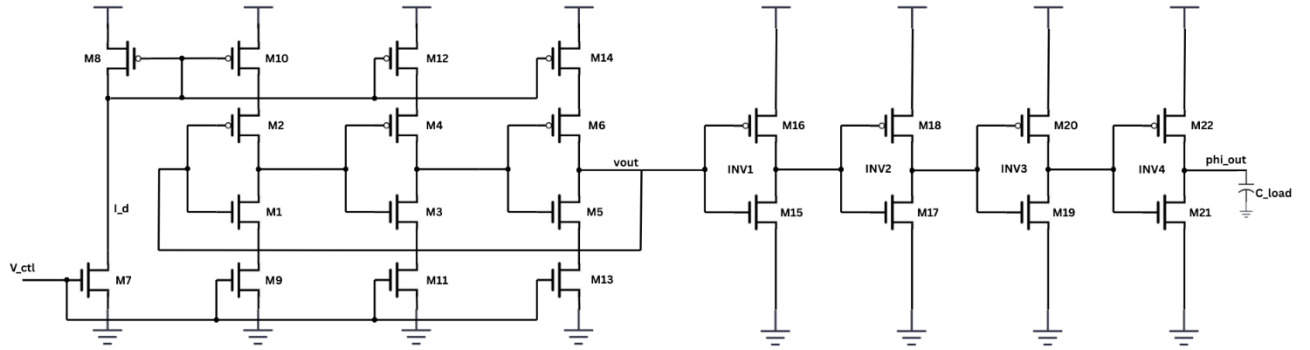


Figure 75: Buffered Current Starved Voltage Controlled Oscillator Schematic

The VCO output needs to be buffered because any loading on the VCO output node will increase the delay thus reducing the frequency.

10.5. PLL BUFFER INSERTION THEORY

In digital circuits, driving large loads necessitates buffer insertion to minimize delay and optimize performance.

The best number of inverter stages to minimize delay is calculated by the following equation.

$$\frac{\log(H)}{\log(3)} \leq N \leq \frac{\log(H)}{\log(4)}$$

The minimum theoretical delay is

$$D_{MIN} = N(GBH)^{\frac{1}{N}} + P$$

The logical effort G_i of an inverter is 1 and the logical effort H is equal to the ratio of the load capacitance over the total capacitance at the input node, “vout” for the VCO. This circuit only contains one branch, therefore $B = 1$. The parasitic delay P_i of the inverter is 1.

$$G = \prod G_i$$

$$H = \frac{C_{total}}{C_{vout}}$$

$$P = \sum P_i$$

Practically The unbuffered VCO output would be driving the first inverter in the output buffer and the inverter in the feedback configuration which would make $B > 1$ depending on the actual sizing. However, this ultimately increases the size of the first inverter in the output buffer, which is not ideal since it increases the loading effect, thus decreasing the operating frequency.

To effectively size the transistor to match the minimum delay, I used these equations,

$$\hat{f} = (GBH)^{\frac{1}{N}}$$

At last inverter INV_N , $C_{out(N)} = C_{load}$

$$C_{in(N)} = \frac{G_N C_{out(N)}}{\hat{f}}$$

For equal rise/fall time:

$$\frac{W_p}{W_n} = \frac{\mu_n}{\mu_p}$$

Set the desired channel length and use the two equations below to calculate the width of the mosfets at INV_N

$$C_{in(N)} = \frac{\mu_n}{\mu_p} \frac{W_n}{L_n} + \frac{W_n}{L_n}$$

$$C_{in(N)} = \frac{W_n}{L_n} \left(\frac{\mu_n}{\mu_p} + 1 \right)$$

To size INV_{N-1} , set $C_{out(N-1)} = C_{in(N)}$ and repeat until the first inverter INV_1

10.6. PLL CHARGE PUMP CIRCUIT ANALYSIS

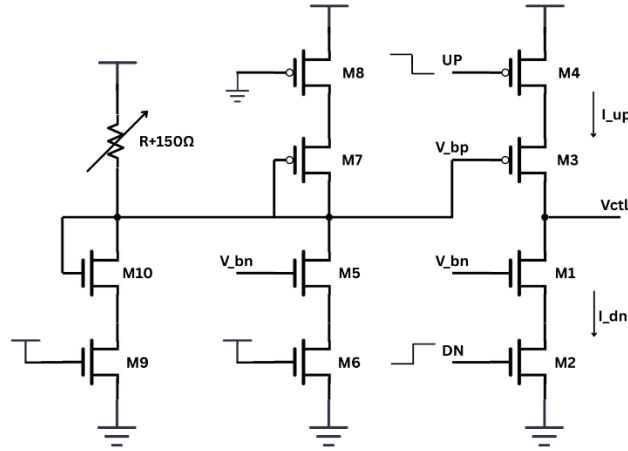


Figure 76: Conventional Charge Pump Schematic

Analysis at the output stage:

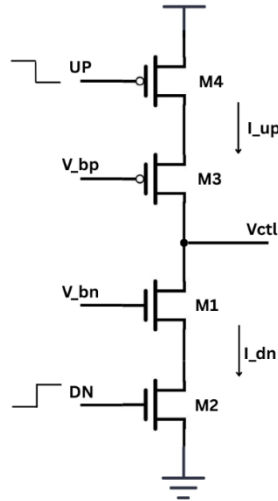


Figure 77: Charge Pump Output Stage

$$V_{ctl} \geq IR_{DN} + V_{EBN}$$

$$RI_{DN} + V_{EBN} \leq V_{ctl} \leq V_{DD} - IR_{UP} - V_{EBP}$$

$$I_{M1} = \frac{1}{2} \mu_n Cox \frac{Wn}{Ln} (V_{bn} - IR_{DN} - V_{thn})^2$$

$$I_{M8} = \frac{1}{2} \mu_p Cox \frac{Wp}{Lp} (V_{DD} - IR_{UP} - V_{bp} - |V_{thp}|)^2$$

$$I_{M8} = I_{M1}$$

$$\mu_n \frac{Wn}{L_n} (V_{bn} - IR_{DN} - V_{thn})^2 = \mu_p \frac{Wp}{L_p} (V_{DD} - IR_{UP} - V_{bp} - |V_{thp}|)^2$$

$$\frac{\mu_n}{\mu_p} \frac{Wn}{L_n} \frac{L_p}{W_p} (V_{bn} - IR_{DN} - V_{thn})^2 = (V_{DD} - IR_{UP} - V_{bp} - |V_{thp}|)^2$$

$$\sqrt{\frac{\mu_n}{\mu_p} \frac{Wn}{L_n} \frac{L_p}{W_p}} (V_{bn} - IR_{DN} - V_{thn}) = (V_{DD} - IR_{UP} - V_{bp} - |V_{thp}|)$$

$$V_{bp} = V_{DD} - \sqrt{\frac{\mu_n}{\mu_p} \frac{Wn}{L_n} \frac{L_p}{W_p}} (V_{bn} - IR_{DN} - V_{thn}) - IR_{UP} - |V_{thp}|$$

Analysis at the middle stage:

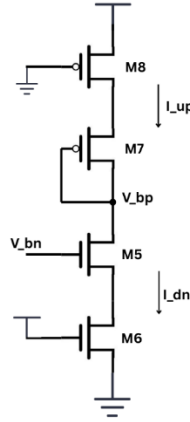


Figure 78: Charge Pump Middle Stage

$$I_{M5} = \frac{1}{2} \mu_n Cox \frac{Wn}{L_n} (V_{bn} - IR_{DN} - V_{thn})^2$$

$$V_{bn} = \sqrt{\frac{2I}{\mu_n Cox \frac{Wn}{L_n}}} + IR_{DN} + V_{thn}$$

Analysis of the current reference (First Stage)

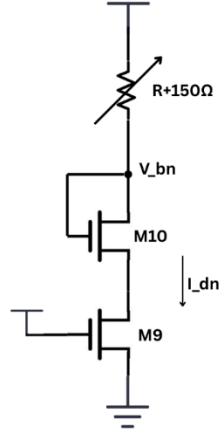


Figure 79: Charge Pump Current Reference (First Stage)

$$I = \frac{V_{DD} - V_{bn}}{R_x + 150}$$

$$V_{bn} = V_{DD} - I(R_x + 150)$$

$$I_{M10} = \frac{1}{2} \mu_n C_{ox} \frac{W_n}{L_n} (V_{bn} - I R_{DN} - V_{bn})^2$$

$$I = \frac{1}{2} \mu_n C_{ox} \frac{W_n}{L_n} (V_{DD} - I(R_x + 150) - I R_{DN} - V_{bn})^2$$

$$I(R_x + 150) = V_{DD} - \sqrt{\frac{2I}{\mu_n C_{ox} \frac{W_n}{L_n}}} - I R_{DN} - V_{bn}$$

$$R_x = \frac{1}{I} \left[V_{DD} - \sqrt{\frac{2I}{\mu_n C_{ox} \frac{W_n}{L_n}}} - I R_{DN} - V_{bn} \right] - 150$$

Mapping the Resistor values to the maximum/minimum desired currents and the desired signal swing that is required for the VCO control voltage.

$$R_{x(max)} = \frac{1}{I_{min}} \left[V_{DD} - \sqrt{\frac{2I_{min}}{\mu_n C_{ox} \frac{W_n}{L_n}}} - I_{min} R_{DN} - V_{bn(min)} \right] - 150$$

$$R_{x(min)} = \frac{1}{I_{max}} \left[V_{DD} - \sqrt{\frac{2I_{max}}{\mu_n C_{ox} \frac{W_n}{L_n}}} - I_{max} R_{DN} - V_{bn(max)} \right] - 150$$

10.7. CODE

Here is the link to our Git repository: <https://git.ece.iastate.edu/sd/sdmay25-27#>

10.8. TEAM ORGANIZATION

Complete each section as completely and concisely as possible. We strongly recommend using tables or bulleted lists when applicable.

10.8.1. Team Members

1. Nolan Eastburn
2. Noah Thompson
3. Nathan Stark
4. Ibram Shenouda
5. Ethan Kono
6. Will Custis

10.8.2. Required Skill Sets for Your Project

- Designing digital components with HDL
 - o Verilog will be used, but VHDL experience useful as well
- Testing digital components with testbench simulations
- Writing programs using C for embedded systems
 - o Memory mapping will be used for most peripherals
- Using Git for version control
- Using NGSpice for analog simulation

10.8.3. Skill Sets covered by the Team

- All team members have experience in some HDL (VHDL or Verilog) and C for embedded systems from previous ISU coursework
- Nathan and Nolan have experience in unit testing from previous work experience
- All team members have experience in Git from prior courses
- Ibram and Noah have experience in NGSpice

10.8.4. Project Management Style Adopted by the team

Our team is going with a Waterfall approach for project management. This was chosen because our project is strongly interconnected and we have a hard, inflexible deadline at the end of the project. Because of this, Waterfall seemed to be the best choice since it has a strong structure for design, implementation and testing.

10.8.5. Initial Project Management Roles

Team Organization - Noah

Project Management – Will

Analog Design Lead - Ibram

Digital Architecture Developer - Nathan

Digital Architecture Developer - Nolan

Security Architecture Developer – Ethan

10.8.6. Team Contract

Team Members:

- | | |
|--------------------------|--------------------------|
| 1) <u>Nolan Eastburn</u> | 2) <u>Noah Thompson</u> |
| 3) <u>Nathan Stark</u> | 4) <u>Ibram Shenouda</u> |
| 5) <u>Ethan Kono</u> | 6) <u>Will Custis</u> |

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

Weekly Advisor/Client Meeting:

Monday 2:00pm Durham 353

Weekly General Team Meeting:

Saturday 10:00 AM TLA

Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

Team Discord

Microsoft Teams (Advisor/Client)

2. Decision-making policy (e.g., consensus, majority vote):

Majority rules (4 members needed to approve)

At least 4 team members need to be present to make a decision

3. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

Minutes will be kept in OneDrive folder, which is shared amongst the team

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

Members are expected to attend 80% or more of all team meetings.

If unable to attend, notify as far in advance as possible.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

Complete all tasks by deadline.

Each member is responsible for their assignment tasks.

Communicate about roadblocks and give advanced notice if deadline is likely to slip.

3. Expected level of communication with other team members:

Communicate regularly at meetings and on Discord.

Bring up all issues and roadblocks encountered in a timely manner.

If you know the answer to a question that is posted, promptly answer it.

4. Expected level of commitment to team decisions and tasks:

All team members are expected to follow through on obligations, even if they did not vote in favor of it.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
 - a. Team Organization - Noah
 - b. Project Management - Will
 - c. Analog Design Lead - Ibram
 - d. Digital Peripheral Lead - Nathan
 - e. CPU/Memory Architecture Lead - Nolan
 - f. Software Lead - Ethan

2. Strategies for supporting and guiding the work of all team members:

Post solutions to common problems in Discord or OneDrive so everyone can see it.

Time at meetings will be devoted to discussing issues.

3. Strategies for recognizing the contributions of all team members:
 - a. Discussions of what was done the week prior
 - b. Presentations to our faculty advisor

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
 - a. Noah:
 - a. Embedded C code
 - b. VHDL and Verilog
 - c. Micro Processor integration
 - b. Nolan:
 - a. Knowledge of C programming for MCUs
 - b. Decent experience in VHDL, basic experience in Verilog
 - c. Intermediate Git knowledge
 - d. Have experience with system and unit testing
 - e. Can adapt to many different codebases quickly
 - c. Nathan:
 - a. HDL experience in VHDL
 - b. Some verification experience for VHDL
 - c. C/C++ programming for MCUs

- d. Ibram:
 - a. Analog Design
 - b. PCB design and testing
 - c. Embedded C
 - e. Ethan:
 - a. Security compliance and national standards (NIST)
 - b. Networking & Protocol Security
 - c. Wireless Security
 - f. Will:
 - a. Experience Programming in C
 - b. Some experience with encryption methods
 - c. Network and Wireless Security
2. Strategies for encouraging and supporting contributions and ideas from all team members:
 - Everyone is encouraged to contribute any ideas and will always be heard by the rest of the team.
 - Ideas will be taken into consideration by the team and will be discussed thoroughly to promote the inclusion of new concepts.
 3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
 - Individuals who feel that there are inclusion issues should bring up any problems with the team first.
 - When a problem is brought to the team, everyone is required to review any concerns together as a group.
 - If the problem consists of the individual should go directly to the faculty advisor for advice or potential guidance for themselves and the team as a whole.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
 - a. Finish the documentation report
 - b. Design the high-level hierarchy of the project
 - c. Implement some of the components that will go into the project
 - d. Show proof-of-concept of the project as a whole
 - e. If all goes well, develop a basic prototype of the project (this may be unrealistic depending on the complexity)
2. Strategies for planning and assigning individual and team work:
 - a. Task breakdown
 - b. Assign tasks with members with the most applicable skillsets
 - c. Flexible to reevaluate and reassign tasks based on needs.
3. Strategies for keeping on task:
 - a. Consistent weekly meetings with the team and professor
 - b. Acknowledging individual contributions

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

Issues will first be discussed during the team meetings or via Discord. All team members are expected to discuss issues in good faith.

2. What will your team do if the infractions continue?

If the issue cannot be resolved amongst the team, an instructor or faculty advisor will be informed to try and mediate the dispute.

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*

b) *I understand that I am obligated to abide by these terms and conditions.*

c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

- 1) Nolan Eastburn DATE: 9/19/2024
- 2) Noah Thompson DATE: 9/19/2024
- 3) Nathan Stark DATE: 9/19/2024
- 4) Ibram Shenouda DATE: 9/19/2024
- 5) Ethan Kono DATE: 9/19/2024
- 6) Will Custis DATE: 9/19/2024