Premise:

- To truly understand a system, it is necessary to implement the system.
- Mastery of computer engineering is demonstrated by being able to design and implement a processor and the kernel that can launch a shell.

Goals:

- Synthesis 15-410 and 18-447 in order to develop an understanding of the interface between hardware and software.
- Experience the thrills of debugging a complex system with neither a ground truth software implementation nor a ground truth hardware implementation
- Expand on the 18-447 experience by delivering a fully functional processor instead of a toy implementation
- Develop a unique speciality spanning kernels and processors
    - Only one company (Apple) is seriously pursuing both the design of custom Silicon and the development of custom kernel/OS; however, even at Apple, the only person spanning both sides is Tim Cook

- Proposal 1: Pebbles kernel on RISC-V processor running on FPGA
    - Learning Objectives:
        - Explore virtual memory through implementation (RISC-V standard = x86)
            - Explore TLB translation through implementation
        - Explore caching through implementation
        - Explore memory through implementing a memory subsystem
        -
        - Explore privileged instructions through implementation
        - Explore atomic instructions through implementation
        - Explore interrupts through implementation (both timer and keyboard)
        - Explore traps/system calls through implementation (for a real kernel)
    - Unknown unknowns:
        - How to implement console display on FPGA
        - How to implement keyboard input on FPGA
        - How to setup the test and build infrastructure to compile and simulate both a kernel and a processor
    - Additional Requirements
        - Implement a RISC-V processor with RV32I+MA extensions
        - Implement a Pebbles kernel for a RV32I+MA architecture
    - Demo:
        - Milestone 1: Demonstrate a Kernel game running on an FPGA
            - Console Output
            - Keyboard Input
            - Subset of privileged instructions
            - Interrupts
            - Game Implementation

        - Cache
        - Boot loader
    - Milestone 2: Demonstrate a uKernel (~410-p4-fondle) with VM running on an FPGA
        - Console Output
        - Keyboard Input
        - Subset of privileged instructions
        - Virtual Memory system
        - Interrupts
        - uKernel with VM
        - Cache
        - Boot loader
    - Milestone 3: Demonstrate a fully functioning kernel running on FPGA (boot to a shell)
        - Console Output
        - Keyboard Input
        - Virtual memory system
        - All privileged instructions
        - Traps
        - Interrupts
        - Full kernel
        - Cache
        - Virtual Memory system
        - Boot loader
    - Milestone 4: Demonstrate a fully functioning kernel with concurrency running on FPGA (boot to a shell with threads)
        - Console Output
        - Keyboard input
        - Virtual memory system
        - All privileged instructions
        - Traps
        - Interrupts
        - Full kernel
        - Cache
        - Virtual Memory system
        - Boot loader
        - Synchronization primitives/pthread library for RV32I+MA
    - Milestone 5: Demonstrate a fully functioning hypervisor running multiple full kernels running on FPGA (boot multiple kernels)
        - Console Output
        - Keyboard Input
        - Virtual memory system
        - All privileged instructions

            - Traps
            - Interrupts
            - Full kernel
            - Cache
            - Virtual Memory System
            - Full hypervisor
            - Hardware support for virtualization (ring -1)
            - Boot loader
- Extension 1.1: Pebbles kernel on RISC-V processor running on FPGA with exotic memory system
    - Learning Objectives
        - Explore more exotic virtual memory systems through implementation
            - Software TLB fill (MIPS?)
            - Hashed page table (PowerPC)
        - Develop mastery of virtual memory by demonstrating the ability to seamlessly switch between virtual memory subsystems without data loss
    - Unknown unknowns:
        - How to switch between virtual memory systems without losing state
        - How to implement console display on FPGA
        - How to implement keyboard input on FPGA
        - How to setup the test and build infrastructure to compile and simulate both a kernel and a processor
    - Additional Requirements
        - Implement a RISC-V processor with RV32I+MA extensions
            - Implement privileged instructions
        - Implement interrupts (timer + keyboard)
        - Implement traps/system calls
        - Implement caching
        - Implement console output
        - Implement keyboard input
        - Implement boot loader
    - Demo:
        - Milestone 1: Demonstrate performance counter metrics on different virtual memory implementations (~410-p4-fondle)
            - Multiple virtual memory systems (possibly in different bitstreams)
        - Milestone 2: Expose virtual memory subsystem via system call; demonstrate different performance for user level code based on virtual memory system selected
            - Multiple virtual memory systems in single bit stream
            - Ability to flush and switch to a new virtual memory system
- Extension 1.2: Pebbles kernel on RISC-V processor running on FPGA with accelerated functional units via a custom RISC-V extension
    - Learning Objectives

        - Explore hardware acceleration for static functions in the kernel
            - Hardware accelerated fork
            - Hardware accelerated timer context switch
            - Hardware accelerated memset
        - Explore RISC-V architecture extensions
    - Unknown unknowns
        - How to implement console display on FPGA
        - How to implement keyboard input on FPGA
        - How to setup the test and build infrastructure to compile and simulate both a kernel and a processor
    - Additional requirements
        - Implement a RISC-V processor with RV32I+MA extensions
            - Implement privileged instructions
        - Implement a Pebbles kernel for RV32I+MA target
        - Implement interrupts (timer + keyboard)
        - Implement traps/system calls
        - Implement caching
        - Implement console output
        - Implement keyboard input
        - Implement boot loader
        - Implement virtual memory
    - Demo
        - Milestone 1: Demonstrate performance metrics between a software implementation and accelerated implementation in basic uKernel
            - Hardware acceleration
            - Basic kernel
        - Milestone 2: Demonstrate performance metrics between a software implementation and accelerated implementation for fully functional kernel
            - Hardware acceleration
            - Fully functional kernel
- Extension 1.2.1 Pebbles kernel on RISC-V processor running on FPGA with user mode hardware extensions via custom RISC-V extension
    - Learning Goals
        - Explore RISC-V architecture extensions
        - Explore hardware acceleration for user mode functionality
            - Profile sample binaries to determine the critical path
    - Unknown unknowns
        - How to implement console display on FPGA
        - How to implement keyboard input on FPGA
        - How to setup the test and build infrastructure to compile and simulate both a kernel and a processor
        - Will it be necessary to rewrite/modify the compiler to leverage extensions?

- Additional Requirements
    - Implement a RISC-V processor with RV32I+MA extensions
        - Implement privileged instructions
    - Implement a Pebbles kernel for RV32I+MA target
    - Implement interrupts (timer + keyboard)
    - Implement traps/system calls
    - Implement caching
    - Implement console output
    - Implement keyboard input
    - Implement boot loader
    - Implement virtual memory
- Demo
    - Milestone 1: Demonstrate performance metrics between software implementation and hardware implementation of user mode functions
        - Fully functional kernel
        - User mode program with hardware acceleration extensions in binary

- Extension 1.3 Pebbles kernel on RISC-V processor running on FPGA with user mode FPGA acceleration (via system call)
    - Learning Goals
        - Explore RISC-V architecture extensions
        - Explore FPGA acceleration for user mode programs
    - Unknown unknowns:
        - How to program an FPGA dynamically
            - Option 1: Partial Reconfiguration
            - Option 2: Networked FPGA/Dedicated FPGA
        - How to implement console display on FPGA
        - How to implement keyboard input on FPGA
        - How to setup the test and build infrastructure to compile and simulate both a kernel and a processor
        - How to compile a user mode software program with a bitstream that can be deployed to FPGA fabric
    - Additional Requirements
        - Implement a RISC-V processor with RV32I+MA extensions
            - Implement privileged instructions
        - Implement a Pebbles kernel for RV32I+MA target
        - Implement interrupts (timer + keyboard)
        - Implement traps/system calls
        - Implement caching
        - Implement console output
        - Implement keyboard input
        - Implement boot loader
        - Implement virtual memory

    - Demo
        - Milestone 1: Demonstrate performance metrics between software implementation and hardware implementation of user mode program
            - Fully functional kernel
            - User mode program with acceleration system calls
            - User mode acceleration bitstream
- Extension 1.4.1 Pebbles kernel on RISC-V processor running on FPGA with statistical predictions in cache eviction
    - Learning Goals
        - Explore architectural design space by implementing statistical predictions for cache eviction
    - Unknown unknowns
        - What complexity is practical in hardware for statistical predictions?
        - What statistical models provide benefit for hardware cache eviction?
        - How to implement console display on FPGA
        - How to implement keyboard input on FPGA
        - How to setup the test and build infrastructure to compile and simulate both a kernel and a processor
    - Additional Requirements
        - Implement a RISC-V processor with RV32I+MA extensions
            - Implement privileged instructions
        - Implement interrupts (timer + keyboard)
        - Implement traps/system calls
        - Implement console output
        - Implement keyboard input
        - Implement boot loader
        - Implement virtual memory
    - Demo
        - Milestone 1: Demonstrate a Kernel game running on an FPGA
            - Console Output
            - Keyboard Input
            - Subset of privileged instructions
            - Interrupts
            - Game Implementation
            - Cache
            - Boot loader
        - Milestone 2: Demonstrate a uKernel (~410-p4-fondle) with VM running on an FPGA
            - Console Output
            - Keyboard Input
            - Subset of privileged instructions
            - Virtual Memory system
            - Interrupts

        - uKernel with VM
        - Cache
        - Boot loader
    - Milestone 3: Demonstrate a fully functioning kernel running on FPGA (boot to a shell)
        - Console Output
        - Keyboard Input
        - Virtual memory system
        - All privileged instructions
        - Traps
        - Interrupts
        - Full kernel
        - Cache
        - Virtual Memory system
        - Boot loader
    - Milestone 4: Demonstrate a fully functioning kernel with concurrency running on FPGA (boot to a shell with threads)
        - Console Output
        - Keyboard input
        - Virtual memory system
        - All privileged instructions
        - Traps
        - Interrupts
        - Full kernel
        - Cache
        - Virtual Memory system
        - Boot loader
        - Synchronization primitives/pthread library for RV32I+MA
    - Milestone 5: Demonstrate a fully functioning hypervisor running multiple full kernels running on FPGA (boot multiple kernels)
        - Console Output
        - Keyboard Input
        - Virtual memory system
        - All privileged instructions
        - Traps
        - Interrupts
        - Full kernel
        - Cache
        - Virtual Memory System
        - Full hypervisor
        - Hardware support for virtualization (ring -1)
        - Boot loader

- Extension 1.4.2 Pebbles kernel on RISC-V processor running on FPGA with statistical predictions in scheduler
- Extension 1.5 Pebbles inspired RTOS on RISC-V processor running on FPGA with real time timing guarantees