Multicore Operational Analysis Tooling

DESIGN DOCUMENT

Senior Design December 2024 – Team 9

Client: The Boeing Company

Advisors: Steve VanderLeest (Boeing) & Joseph Zambreno (ISU)

Team Members	***Roles***
Alexander Bashara	Embedded & Cache Engineer
Joseph Dicklin	I/O Engineer
Hankel Haldin	Platform Bringup Engineer
Anthony Manschula	Project Coordinator & Memory Engineer

Team Email:

sddec24-09@iastate.edu

Team Website:

https://sddec24-09.sd.ece.iastate.edu

Executive Summary

Development Standards, Guidelines & Practices Used

- FAA: AC 20-193
- IEEE Code of Ethics
- CAST-32A
- SAE Aerospace Standards
- RCTA/DO-178C
- ASTM (American Society for Testing and Standards)
- POSIX (Portable Operating System Interface)
- ARINC 653
- FACE (Future Airborne Capability Environment)
- Waterfall & Agile Development Workflow

Summary of Requirements

Physical Requirements

- Architecture: System must implement a processor based on the ARMv8 instruction set
- Form-factor: Single-board computer (Raspberry Pi, Pine64 family, etc.), or FPGA board with Xilinx UltraScale+ MPSoC (Xilinx ZCU family or similar)

User Knowledge Requirements

- Working understanding of Linux environments and how they are structured in the context of embedded systems
- Worst-case execution time and its influencing factors
- Familiarity with multi-core computer architectures, caching, memory, and I/O
- Provide documentation with a sufficient level of detail to allow the user to learn any of the above at a high level

Functional and Technical Requirements

- Toolset must thoroughly and methodically stress the system in a reproducible way
- Toolset must focus on major points of resource contention (processor time, memory usage, IO bus usage, etc.)
- Accurately produce potential worst-case scenarios (rogue process uses too much CPU time/memory/IO bandwidth)
- Toolset must collect and analyze performance data to demonstrate an upper bound on worst-case execution time for our platform

User Interface and Experience Requirements

- Command-line utilities are documented with a level of detail sufficient to allow users with less technical knowledge to use the tool effectively
- Provide a user-friendly GUI for managing and interpreting test results

Applicable Courses from Iowa State University Curriculum

Computer Engineering

- CPRE 1850 – Introduction to Computer Engineering and Problem Solving in C
- CPRE 2880 – Embedded Systems I: Introduction
- CPRE 3080 – Operating Systems: Principles and Practice
- CPRE 3810 – Computer Organization and Assembly-Level Programming
- CPRE 4580 – Real Time Systems
- CPRE 4880 – Embedded Systems Design
- CPRE 5810 – Computer Systems Architecture

Computer Science

- COM S 3110 – Design and Analysis of Algorithms
- COM S 4150 – Safety-Critical and High-Dependability Software Systems

New Skills and Knowledge Acquired Through Research

Computer Architecture Knowledge

- ARM Cortex A72/A53 Cache Organization
- Cache Coherency
- Cache Coloring
- DMA Controllers
- Cacheable/Non-Cacheable Memory Access
- ARM64 Boot Flow

Hypervisor Knowledge

- Xen Hypervisor

Linux Skills

- Building/Deploying PetaLinux with Xen Support for Xilinx UltraScale+ MPSoC
- Building/Deploying Embedded Linux with Xen Support for ARM64 using Yocto Project
- Configuring U-Boot Bootloader for ARM64 and Xen

Performance Profiling Skills

- Configuring/Reading ARMv8 Hardware Performance Counters
- Linux PERF Framework
- Valgrind Application Profiling Tools

Table of Contents

List of Figures, Tables, and Definitions

Figures and Tables

Important Definitions and Terms

I/O - Input/Output, referring to methods of transferring data in and out of a system.

System-on-Chip (SoC) - Processor, memory, I/O, and graphics processing (sometimes) hardware all housed on a single piece of silicon.

Worst-Case Execution Time (WCET) - Maximum amount of time a program can be expected to take to execute under a set of system conditions designed to produce a worst-case load scenario.

ARMv8 – Version 8 of the ARM Instruction Set, which defines important characteristics of processors that implement it, such as supported data types, register configurations, and memory management.

Main Memory/DRAM - Large pool of reasonably fast volatile (meaning values are erased after power loss) storage that stores instructions and data of programs currently running on the system.

Cache – Small pool of very fast volatile storage that the processor can use to store frequently used data items and program instructions instead of searching for them in main memory. There may be multiple levels of cache in a processor, with some being shared between all cores, and others being private to each core.

Datapath – A path taken by a request from a particular component before reaching its destination component (for instance, the CPU to DRAM data path might be used by a request made by the processor to retrieve a value from main memory).

Virtual Machine – A software emulation of a computer's physical hardware. High speed VMs (such as the one the team is working with) may require support for this process in hardware through "virtualization hooks".

Type 1 Hypervisor – A hypervisor manages the allocation of physical hardware to all virtual machines running on a system. Type 1 hypervisors have lower resource overhead and require that they are the highest-privileged level of software running on the system (i.e. there is no other software, such as an operating system, operating between it and the physical hardware).

Xen – Open-source type 1 hypervisor with ARMv8-based system support.

FPGA (Field Programmable Gate Array) – a special integrated circuit that contains reprogrammable logic.

1. Introduction

1.1. Problem Statement

Avionics systems are responsible for controlling many safety-critical aspects of modern airplanes, such as engine management modules and flight envelope protections. As these systems become increasingly computerized, the amount of input data they consume and act upon becomes progressively larger. Our project client Boeing has provided us with a need to stress test multicore systems for interference. Within multicore systems, it's important to test the I/O (input/output) bandwidth, cache interference, CPU (central processing unit) performance and main memory performance under a worst-case set of system load conditions to determine how program execution time is affected. Such a tool is essential to safe operation of multicore systems in safety-critical environments.

This tool allows users to quantitatively identify worst-case execution times (WCET) in which resource contention is extreme. For those who are unfamiliar, worst-case execution time refers to the maximum amount of time a given process can be expected to take under worst-case system conditions. This is essential for avionics certification, as it provides a definitive way to prove that performance degradation is within an acceptable bound, or that mitigations can be applied to move the execution time inside of that bound. Overall, the desired outcome of this project is a stress-testing platform that will allow us to pinpoint areas of resource contention, apply multicore stress via those points, and apply relevant performance mitigations to the system to arrive at a WCET for our reference hardware platform.

1.2. Intended Users

When designing our empathy maps, we were able to identify several key stakeholders/users directly affected by our design project. This includes the Boeing avionics development team working hands on with the multicore stress testing platform, the Boeing team managers in charge of a technical avionics testing team and ensuring regulatory compliance, as well as Boeing customers purchasing a designed/tested product using this multicore stress test tool.

Looking at the user personas for each group, we can see that the avionics development team can be described as a group that manages Linux avionics development and validation for Boeing aircraft. They would need a system stress testing tool developed on an ARM-based platform with resource testing deliverables like execution time, system temperatures, etc. easily accessible. As this user group is directly involved in the testing of in-development avionics, a stress test tool is of the utmost value to the team.

Looking at another user group, the Boeing managers, although not needing a stress testing tool for the success of their role as a manager, they would need a stress tool easily accessible to their project engineers such that they can ensure their systems can be flight-certified under civilian or military authority. Again, as stated by the development team, this tool ensures the success of their in-development product, making the value extremely high for the company. When considering this user group from an economic standpoint, the Boeing manager role consists of client-to-team communication (e.g. economic considerations). Although less important than the safety considerations a product such as this brings up, the economic factors surrounding a stress tool are still important.

Looking at the last user group described, the product customer, although less involved with the testing suite than the avionics development team, wants to purchase the safest product available. This tool ensures independence from core-to-core interference when running multi-core applications, making the product safer as a result. A customer in this context can be closely described as an airline company wishing to purchase a new Boeing airliner to utilize on commercial flights. Their operation relies heavily on the FAA

(Federal Aviation Administration) approval of the aircraft. Therefore, this analysis tool is crucial to the success of their purchase.

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

Physical/Resource Requirements

- ARMv8 processor subsystem
    - Choosing a hardware platform using a processor implementing the ARMv8 instruction set is critical due to its widespread usage across applications.
- Form-factor: Single-board computer (Raspberry Pi, Pine64 family, etc.) or FPGA board implementing a Xilinx MPSoC (Xilinx ZCU family)
    - The single-board computer form factor is useful in that it keeps costs down and hardware is easy to find. However, certain FPGA boards are also acceptable if they implement an ARMv8 processor subsystem.

User Experiential Requirements

- Linux environments
    - Both the developers and the users must be comfortable working in Linux environments, as this is where tool development and usage will take place.
- Worst-case execution time (WCET) and its influencing factors
    - Developers and users must be familiar with this concept, as it is the core metric that this project aims to determine. They must understand how both program and hardware architecture may influence it, and how to utilize those factors to produce a thorough measurement.
- Familiarity with multi-core computer architectures
    - To create tools and take measurements to determine WCET for a given multicore system, the developers must understand the data paths between various components in the system, how they interact, and how to abuse them. Additionally, to truly understand the meaning of the results produced by the tools, users must possess some amount of knowledge of the underlying hardware.
- Documentation providing a sufficient level of detail to allow the user to learn any of the above at a high level
    - Since our tool will be used by both technical and nontechnical user groups, a comprehensive set of documentation is essential. This will consist of documentation regarding how to build a system image, use the tools, and view/analyze results. Additionally, as we will be handing off the project to another group, clear communication of our progress to them is critical for a smooth transition.

Functional/Technical Requirements

- Properly and methodically stress the system
    - Since the toolset applies to verifying hardware for a safety-critical setting, it is important that stress applied to the system is as intense as possible. If it is not, the measurement of WCET may be based on data that was collected with a flawed methodology.
- Identify major points of resource contention (processor time, memory usage, IO bus usage, etc.)
    - The team must consider all possible avenues that resource contention could arise from. Showing evidence of thorough testing is important in being issued an FAA certification.
- Demonstrate an upper bound on worst-case execution time for our platform
    - The tool set must provide an effective way of measuring and analyzing performance metrics of various runs, both with and without the system under stress.

UI Requirements

- Develop a well-documented command line tool to interface with our design
    - The command line tool can be used to interact with and initiate runs with the toolset. Due to inclusion of several base programs and stress options, the team needs clear

documentation for acceptable values and the corresponding behavior that they are linked to.

- Provide a user-friendly GUI for managing and interpreting test results
    - A GUI is much more friendly to look at than text output from a command line. The GUI should clearly present the results of a set of tests in a straightforward manner.

2.2. Engineering Standards

FAA: AC 20-193

- This standard is defined by the U.S. Department of Transportation. It is concerned with the use of multi-core processors in avionics systems. Our design is directly applicable to this area, hence its inclusion.

IEEE Code of Ethics

- While this standard applies to any engineering effort, our design must ensure the public's safety. Our design provides critical information to systems whose failure could lead to severe injury or death.

CAST-32A

- This document outlines the aspects of multi-core systems of concern to the safety and performance of avionics systems and will help guide the aspects of testing that our toolset needs to achieve.

SAE Aerospace Standards

- These standards define the safety and reliability of various aspects of avionics systems. Our design will stress test multi-core systems that will support avionics systems like controls and communications.

RCTA/DO-178C

- This standard is concerned with the quality of software used in avionics systems. It defines a safety assessment process that categorizes software into five tiers of criticality. Our design must adequately characterize a hardware platform to assess the criticality of a software fault or failure.

ASTM (American Society for Testing and Standards)

- ASTM publishes technical standards that are directly applicable to many engineering efforts, including avionics.

POSIX (Portable Operating System Interface)

- POSIX defines a set of standards that ensure compatibility between operating systems. Our design will be a part of a larger set of software tools and systems. It should therefore be able to interface with these tools in a standardized way.

ARINC 653

- This standard is concerned with the space and time partitioning of safety critical avionics systems. Our design must be able to isolate and test distinct aspects of a multi-core system, like CPU usage, memory, and bus traffic to inform the decisions made by this standard.

FACE (Future Airborne Capability Environment)

- This standard defines an avionics environment for military airborne platforms. It is concerned with making real-time safety-critical computing applications more robust, portable, and secure. This is the end goal of our design.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our project uses a hybrid management style, utilizing both waterfall and agile development strategies. This has allowed us to plan out the entire project broadly, then use agile development strategies to focus on week-to-week goals to meet broad deadlines. To achieve these goals, we are using Git to manage our code and tracking issues using Gitlab Issues. We are also meeting weekly with our client, Boeing, to make sure that we are on the right track and that they are happy with our progress. These strategies allow us to make consistent progress and track where we are relative to our goals.

3.2 Task Decomposition

We divided our project into major parts, then subdivided those parts into smaller parts that can be taken care of by one or two people. This strategy integrates well with our management and tracking procedures. Figure 1 below details all the tasks that compose our project.

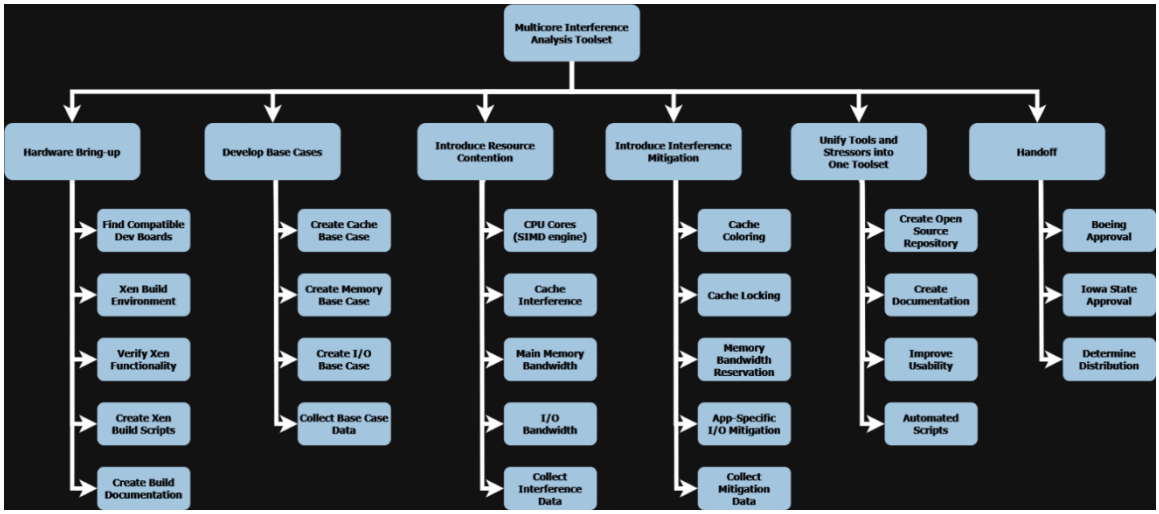


Figure 1: Task Decomposition Chart. Boxes in the second row correspond to a major milestone in our project, with the boxes under corresponding to sub-tasks needed to achieve these milestones. Each sub-task is dependent on the task above it

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Metrics/Evaluation Criteria:

- Technical metrics:
  - Tool suite generates interference on all aspects of the hardware platform
    - Cache, memory, I/O buses, SIMD engines, etc.
    - Options should be configurable to service multiple different platforms
  - Worst-Case Execution Time (WCET) Criteria
    - Perform several experiments to generate interference
    - Use statistical analysis of execution time to determine an upper bound on WCET
  - System resource usage
    - Characterize the underlying interference channel causing the WCET
  - Same functionality in command-line version of tool suite as the GUI version
- Usability metrics:
  - Users rate appearance and usability of the tool suite > 7 on a scale of [1 – 10]
  - Command-line version of the tool suite is as user friendly as GUI frontend

Milestones:

- Xen hypervisor is functional on our target development platform
- Identify resource contention points on our target platform
- Tools induce *some* amount of stress on the identified contention points
- Tools thoroughly induce stress on the identified contention points
- Quantitatively prove mitigation tools improve performance of the system while contention is underway
- Obtain a worst-case execution time for our system
- Integration of testing and tools into one unified suite.

3.4 PROJECT TIMELINE/SCHEDULE

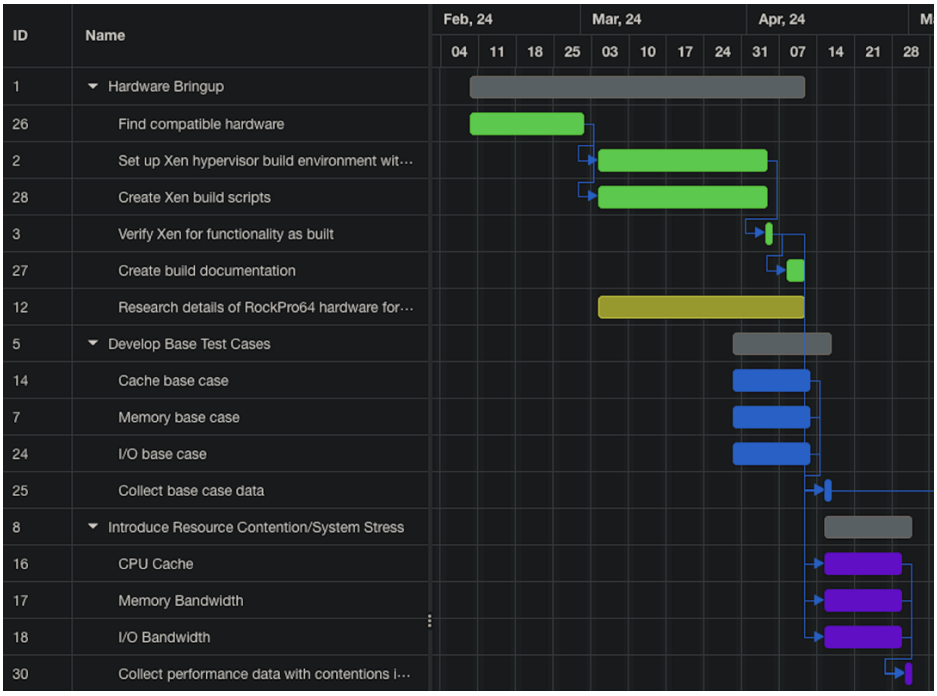


Figure 2: Semester 1 Project Timeline.

For semester 1, we have several deliverables we are planning on targeting. The first corresponds to "Verify Xen for functionality as built" in the chart and is scheduled for April 5th. We will deliver a report to our client that thoroughly confirms the proper functionality of the system as well as scripts and documentation regarding the bring-up process. The second, identify resource contention points (yellow), comes shortly after as we do research on the board while getting it set up and is targeted for April 11. Again, the team will be delivering that report to Boeing during our weekly scheduled meeting that outlines our research and conclusions. The third deliverable, inducing stress on the identified contention points, should be finished by April 29th. The team will deliver a presentation with data measured from test utilities and performance monitors showing the effect of the multicore stress on system programs, as well as a library of programs that were developed to generate stress.

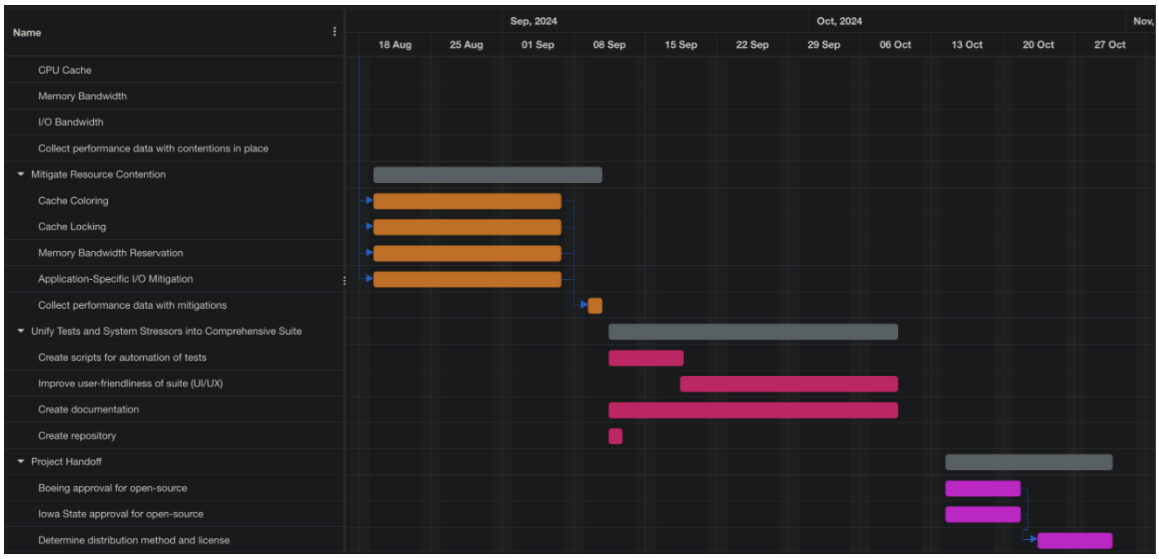


Figure 3: Semester 2 Project Timeline.

For semester two, we have several more deliverables. The first is quantitatively proving that our mitigation techniques improve the execution time of our base case programs, which will happen when we collect the performance data on September 10th. For this deliverable, the team will compile a comparative analysis report with performance data from the system without performance mitigations and from one with performance mitigations. From there, we can determine a bound on worst-case execution time for our system. This will be easiest to do after we have improved the suite's usability (this milestone is targeted for October 9th), so we estimate that it should be completed by October 31st. The team will be delivering a set of scripts that perform statistical analysis on the data gathered in the previous step along with a report of our conclusions.

3.5 Risks and Risk Management/Mitigation

Hardware selection:

- *Find compatible dev board*
  - Risk: we struggle to find a board that meets our project's needs
  - Probability: .80
  - Mitigation: acquire multiple boards (within budget) to ideally find one that works
  - Mitigation: we can emulate a hardware environment in Linux
- *Xen build environment*
  - Risk: we face challenges building Xen & its toolchain for our platform
  - Probability: .75
  - Mitigation: team leverages industry experts at Boeing
- *Verify Xen functionality*
  - Risk: Xen & and its toolchain do not work after installation
  - Probability: > .90
  - Mitigation: communicate our difficulties to our client to get unstuck early when we encounter issues
- *Create Xen build scripts*
  - Risk: our set up is not easily replicable / portable to a script
  - Probability: < .10
- *Risk: the selected hardware platform is incompatible with our client's and project's needs*
  - Probability: > .80

    - Mitigation: Have multiple hardware options (e.g., RockPro64, RaspberryPi 4, ZCU106)

- *Risk: we encounter trouble installing Xen on our hardware platform*
    - Probability: > .80
    - Mitigation: communicate our work and where we are stuck to the Boeing team to get unstuck

Develop Base Cases:

- *Create Cache Base Case:*
    - Risk: we cannot find all the relevant information for the cache for our given platform
    - Probability: .20 (we specifically chose platforms for which we would have this information)

- *Create Memory Base Case:*
    - Risk: we cannot find all the relevant information for the memory configuration for our given platform
    - Probability: .20 (we specifically chose platforms for which we would have this information)

- *Create I/O Base Case:*
    - Risk: we cannot find all relevant information for the I/O characteristics for our given platform
    - Probability: .20 (we specifically chose platforms for which we would have this information)

- *Collect Base Case Data:*
    - Risk: we have no way to collect relevant metrics on the researched interference channels
    - Probability: .30

Introduce Resource Contention:

- *CPU Cores (SIMD Engine)*
    - Risk: properly implementing the interference generator is more time consuming than originally planned
    - Probability: .60
    - Mitigation: communicate our work and where we are block to the Boeing team to get unstuck

- *Cache Interference*
    - Risk: properly implementing the interference generator is more time consuming than originally planned
    - Probability: .60
    - Mitigation: communicate our work and where we are block to the Boeing team to get unstuck

- *Main Memory Bandwidth*
    - Risk: properly implementing the interference generator is more time consuming than originally planned
    - Probability: .60
    - Mitigation: communicate our work and where we are block to the Boeing team to get unstuck

- *I/O bandwidth*
    - Risk: properly implementing the interference generator is more time consuming than originally planned
    - Probability: .60
    - Mitigation: communicate our work and where we are block to the Boeing team to get unstuck

- *Collect Interference Data*

    - Risk: our test base cases do not adequately stress the system (i.e., demonstrate WCET)
    - Probability: .60
    - Mitigation: we can use our client's expertise in the given domain to increase the likelihood that our test cases demonstrate the WCET for our hardware platform

Introduce Interference Mitigation:

- *Cache Coloring*
    - Risk: none of the existing cache coloring tools work for our project
    - Probability: .20 (cf, Stress-NG)
- *Cache Locking*
    - Risk: we have no method of enforcing cache locks in software / hardware
    - Probability: .10
- *Memory Bandwidth Reservation*
    - Risk: none of the existing memory reservation tools work for our use case
    - Probability: .05 (cf, MemGuard)
- *App-Specific I/O Mitigation*
    - Risk: the I/O behavior of applications varies widely, and it will be hard to measure
    - Probability: .60
    - Mitigation: Get input from our client on what applications we should use to adequately mitigate this form of interference
- *Collect Mitigation Data*
    - Risk: we are not able to accurately measure how our mitigation methods reduce interference
    - Probability: .40 (simply compare metrics with interference on / off)

Unify Tools and Stressors into One Toolset:

- *Create Open-Source Repository*
    - Risk: we are not able to create a public repository due to NDA
    - Probability: > .50 (?)
    - Mitigation: we need to communicate with both Boeing and Iowa State University early on to determine which parts of the project can and cannot be open-sourced
- *Create Documentation*
    - Risk: development of the project was not continuously documented, and knowledge is lost
    - Probability: .60
    - Mitigation: maintain light documentation of work throughout the project so it can be expanded on during this stage
- *Improve Usability*
    - Risk: our tool is not intuitive to use for knowledgeable users
    - Probability: .70
    - Mitigation: perform a usability study with our Boeing clients to improve the usability of our project
- *Automated Scripts*
    - Risk: the scripts we produce are not able to be reused by users of the project
    - Probability: .20

Handoff:

- *Boeing Approval*
    - Risk: Boeing does not approve the handoff of our project due to NDA
    - Probability: .30

- *Iowa State Approval*
    - Risk: Iowa State University does not approve the open sourcing of the project
    - Probability: .50
    - Mitigation: Communicate with both Boeing and Iowa State University early on to determine what parts of the project can be open sourced
- *Determine Distribution*
    - Risk: there is no platform we can publish our project on or no license that applies to its distribution
    - Probability: .10

3.6 Personnel Effort Requirements

Using the task decomposition table from 3.2, we separated the major tasks into the rows of the table below with the smaller sub-categories/tasks placed along the columns. Using our best judgement, we assigned rough time estimates for each sub task using the assumption that a single team member or two would be assigned to each task. As the project is still underway, the times are subject to change.

Hardware Bring-up	**Develop Base Cases**	**Introduce Resource Contention**	**Introduce Interference Mitigation**	**Unify Tools and Stressors to One Toolset**	**Handoff**
Compatible Dev Boards (3hrs)	Cache Base Case (5hrs)	CPU Cores (12hrs)	Cache Coloring (10hrs)	Create Open-Source Repository (2hrs)	Boeing Approval (1hr)
Xen Build (35hrs)	Main Memory Base Case (5hrs)	Cache Interference (16hrs)	Cache Locking (16hrs)	Create Documentation(4hrs)	Iowa State Approval (1hr)
Verify Xen (2hrs)	I/O Base Case (5hrs)	Main Memory Bandwidth (12hrs)	Memory bandwidth Reservation (14hrs)	Improve Usability (8hrs)	Determine Distribution (1hr)
Xen Scripts (2hrs)	Collect Base Case Data (8hrs)	I/O Bandwidth (12hrs)	I/O Mitigation (14hrs)	Automated Scripts (8hrs)	
Build Documents (4hrs)		Collect Data (8hrs)	Collect Data (8hrs)		

Figure 4: Personnel effort table along with hour estimations

Total: 216 hours

3.7 Other Resource Requirements

The main requirement for this project is an ARM development board that supports Xen Hypervisor; for our project that is a RockPro64 accompanied with a MicroSD card for booting and a Xilinx ZCU106 development board. In the case of the RockPro64, a USB to TTL converter is also required to communicate with the board over a serial connection. This was primarily purchased to aid with client-to-team debugging. Along with the board, SD card and USB to TTL converter, a computer with a USB type A is also required to interface with the development board.

4 Design

4.1 Design Context

4.1.1 Broader Context

Our design is situated entirely within the avionics community. Although a multicore stress test tool is usable for other aeras of engineering design, our hardware specific requirements and relevant engineering standards narrow down the scope of potential users of our tool. The tool is currently being designed for Boeing, but realistically in the future once our project becomes an open-source asset, 3rd party companies could use the tool to design safer avionics systems. With the growing complexity of avionics platforms, it's imperative that multicore systems become reliable enough to bring down what is needed in terms of hardware footprint.

Area	Description	Examples
Public health, safety, and welfare	Our project affects several stakeholder groups, including engineers and other employees of Boeing, as well as the general public that could use products incorporating components that were tested with our solution.	Ideally, our testing system contributes to safer software and more advanced performance mitigation techniques in the avionics space, along with a better understanding of execution times in the context of a multicore system experiencing a high load. If such testing is not thorough enough, loss of property and life, and reputation could occur for our stakeholders.
Global, cultural, and social	Our project focuses on the verification of multicore systems in safety-critical applications like avionics. A project's effectiveness and safety are only as good as the culture of the workplace it is developed in.	Usage of the MOAT solution could contribute to a positive change in safety culture in the workplace where it is being applied by providing a more accessible toolset for safety-critical verification.
Environmental	Our project's primary environmental impacts include energy consumption and resource usage (specifically rare earth metals). Because our tool is merely a software device, the main aera of concern is wasted energy usage.	Realistically, the energy impact is rather low when comparing our tool to the broader avionics system. Nonetheless, the testing software will require some level of energy use to run, in turn affecting energy production factors surrounding the environment.
Economic	To get our testing tool up and running, certain hardware must be purchased. This is due to the platform type (ARM) which the software was designed for. In addition to the specific board for testing, a serial to display adapter is required to properly run/debug the system. Upfront costs are trumped by the reduction in economic factors, such as	Although not expensive, buying the hardware to run Xen can be considered an economic consideration. The byproduct, however, is the better development of products from a safety standpoint, which allows for the reduction of backup systems onboard.

	safety hardware, systems re-design factors, etc.	

Figure 5: Areas of ethical concern

4.1.2 Prior Work/Solutions

Our work developing this project is motivated by our client's research in multicore integrated modular avionics [2]. The overarching use case for our design is a tool suite that supports the work of engineers to ensure safety critical systems react to inputs within a deadline. The final deliverable of our project is a tool-suite that provides an engineering team with relevant data regarding a multicore system's WCET [3]. Our design also bolsters the efficiency of the verification process in that it allows a system designer to partition a multicore system into discrete subcomponents and characterize the Worst-Case Execution Time (WCET) on a case-by-case basis. This reflects the verification process in our client's industry [1]. Ultimately, our efforts are directed towards implementing a framework as outlined by our client's research [4].

There are several products on the market that are similar to our design. They range from professionally developed, closed source to open-source academic projects. These tools provide us with a frame of reference in the market and how our tool suite can address specific needs. Each one of the designs, detailed along with their pros and cons in figure 6 below.

	Multicore Operational Analysis Tool (MOAT)	RapiDaemon	Multicore Test Harness	OTAWA (Open Tool for Adaptive WCET Analysis)	MASTECS
Pros	Open source, build with modern hardware in mind	Designed by a team of professional engineers, specifically for compliance testing	Open source & has good instructions	Open source, supports several ISAs (e.g., ARM, RISC-V, etc.)	Joint project offering timing analysis software, consulting, and documentation
Cons	Less polished than competitors due to time restrictions	Closed source & expensive	Very old, not built for Xen, development stopped four years ago	No recent builds, not well-known	Received funding from EU, so potential issues with portability to North America

Figure 6: Analysis of market competition

4.1.3 Technical Complexity

Our project approach leverages and aggregates several core computer engineering and embedded systems concepts. One of our project's challenges is choosing a hardware platform that accommodates the tools we require and meets our client's architecture requirements. This requires familiarity, if not relatively advanced knowledge, in several domains to adequately implement our design.

Our project requires the interoperability of the following tools, components and subsystems:

- How a computer's memory hierarchy works

    - The design requires that we understand a modern computer's memory hierarchy to generate as much traffic to the next level of storage as possible
- Mechanics of multi-level caching in an ARM processor
    - Our team must gather detailed information the cache structure of our chosen architecture
    - The research we do must allow us to maximize the number of cache misses
- Understanding of our platform's bus and I/O layout
    - A common source of interference in multicore systems comes from contention on shared communications pathways between cores
    - Our design must be aware of these pathways and exploit them to uncover interference channels
- Knowledge of multicore processor systems
    - Our design must be able to demonstrate that an application running on one core can interfere with the proper execution of another application on a different core
    - This requires a deep understanding of how cores that make up a multicore system arbitrate resource sharing
- Software development knowledge
    - Our final deliverable is a tool suite that has both a UNIX-like command line interface and GUI (Graphical User Interface), which requires an adequate knowledge of software development tools and practices (e.g., version management, software libraries, etc.)
    - This requires knowledge of writing POSIX (Portable Operating System Interface) compliant software at the OS level to be distributed to a wider audience
    - The team must also develop an intuitive and usable GUI on top of the command line interface
- FPGA (Field Programmable Gate Array) Development Boards
    - Interfacing with embedded systems
- SBCs (Single Board Computers)
    - Requires knowledge of how to use resource-constrained computing environments
    - Evaluate the costs and benefits of several different boards
    - Requires knowledge of efficiently reading data sheets to find relevant information
- How to configure and use a type-1 hypervisor
    - Using the hypervisor correctly depends on the team's understanding of isolating computer system resources
    - Requires knowledge regarding the virtualization of computer hardware and subsystems
    - A type one hypervisor runs directly on computer hardware, so configuration knowledge is necessary
- Profiling and analyzing application performance
    - Our team must know how to leverage and use existing performance tools
    - Our design also necessitates that we know how to install these tools on our platform

In addition to the preceding technical aspects our design is also constrained by our client's requirements ISA (Instruction Set Architecture) requirements. The project description stipulates that the final version of our design must run an ARM multicore processor. Ideally, our client would like us to choose a multicore processor that has around six cores.

Due to its scope this project, once implemented, will exceed current industry solutions in that it provides an open-source alternative to existing Worst-Case Execution Timing software. The team must synthesize multiple domains of computer engineering, computer science, and engineering economics to deliver the final form of our client's request.

4.2 Design Exploration

4.2.1 Design Decisions

Per our client Boeing's guidance, we have been able to condense our project into 3 broad categories: Hardware Platform, Resource Contention Channels as well as Hypervisor. These three categories are elaborated upon below, allowing us to describe the decision-making process in detail, providing insight into our project's engineering plan.

4.2.1.1 Hardware Platform

The first major decision that was necessary for our team was the choice of hardware platform that our framework would run on. The three stipulations given to the team by Boeing were that our hardware should utilize an SoC (system-on-chip) that contains at least two processor cores, that those cores implement the ARMv8-A instruction set, and that the hardware should be in the format of a single-board computer (SBC). From there, the team identified several other constraints to guide our decision, including hardware availability, age, and feature set. To ensure that we could begin work on the project in an expedient manner, the platform we selected needed to be in stock with an estimated shipping time of no more than one to two weeks.

Hardware age was another critical consideration that the team had to make, as the ARMv8-A instruction set was publicly released in 2011. As many improvements have been made to SoCs utilizing ARMv8-A technology since 2011, it was critical that the team found a platform released within the last 5-6 years, ensuring that our testing was relevant by considering hardware of a more modern design. The feature set of our hardware was also something that had to be accounted for. This includes things such as peripheral connectivity (USB, Ethernet, PCIe (peripheral component interconnect express)) and memory technology and capacity. These features would be critical for allowing us the most flexibility in exploiting resource contention channels, as described in the following section.

4.2.1.2 Interference Channels

The primary motivation of our work concerns developing a tool set that will help our client in verifying multicore avionics systems for compliance with airworthiness regulations, as outlined in FAA Advisory Circular 20-193. Part of this verification process involves identifying and characterizing performance detriments resulting from simultaneous access of shared device resources. For example, memory, Level 2 processor cache and I/O (input/output) subsystems (via USB Ethernet, PCIe, or USB). Using our platform feature set as a guide, as stated in section 4.2.1.1, the team had to make several decisions on which resources would be targeted and how they would be exploited (simulating a potential bad actor in real avionics system) to show we are achieving a true worst-case scenario.

The first area of contention the team identified was the processor cache. The processor cache is responsible for storing data that has been accessed recently or frequently (depending on the data replacement algorithm used by the designer), and considerably boosts system performance by reducing the frequency of main memory accesses (which are much slower to perform than cache accesses). Among some other methods, the team chose an attack vector known as "cache thrashing", which is a programming technique designed to maximize cache misses, meaning that data stored in the cache will frequently have to be ejected, and new data read from main memory. This creates a considerable amount of stress on the cache subsystem and could lead to performance degradation in other programs trying to utilize the cache.

The second area of contention that the team identified was the memory subsystem. Any application running on the target platform will need to utilize memory to store information about the work that it is performing,

and as such, is a prime area for resource conflict. While modern systems have sufficient protections in place to prevent programs from using too much memory capacity, far fewer protections are in place to prevent programs from using too much memory bandwidth. Bandwidth refers to the rate that information can flow between two given subsystems on a device. In this case, focus is on the CPU-to-DRAM data path, as the CPU often performs processing tasks on program data that is stored in main memory. If a program utilizes an excessive amount of memory bandwidth, the performance of other applications running on the system could experience unpredictable latency when accessing data. For this reason, it is critical that the effects of memory bandwidth contention are analyzed.

The third area of contention that the team identified was the I/O (input/output) subsystem, which handles access to peripheral devices over a variety of protocols. Given that modern avionics applications require a vast amount of throughput, it's imperative that our testing suite analyses the operation of I/O where interference arises. Such interference concerns include bandwidth limitations, DMA (Direct Memory Access) stressing, resource contention overlapping, end-to-end latency, and delay jitter. If a program overloads the available bandwidth, the performance of other important subsystems may take a hit, leading to eventual failure.

4.2.1.3 Hypervisor

When it comes to the choice for hypervisors, the team had only one choice: Xen. Our reason for using Xen is primarily informed by two main reasons: Xen is the only major Hypervisor to support the ARM Architecture and Boeing specified that our team use Xen in our design. While alternatives like KVM (Kernel-based Virtual Machine) exist, Xen is better suited for use in the avionics industry. Within the Xen hypervisor, the team has decided to use bare-metal DomU programs to not deal with multiple OSes taking up compute power on the chip. This will allow us to thoroughly test the worst-case execution time while running the interference generators.

4.2.2 Ideation

Per Boeing's request, we are using an ARM SoC and Xen Hypervisor. Boeing's suggestion stems from this platform's use in safety-critical avionics systems. This enables our design to closely mirror what is used in industry. There are many different options when it comes to ARM SoCs, so our team had to select one that meets the requirements of Xen and has multiple cores.

4.2.2.1 Raspberry Pi

The Raspberry Pi is a very popular ARM Multicore SBC with lots of community support. This led our team to consider this board first. We very quickly were able to find others who had gotten Xen running on Raspberry Pis, but the documents and code repositories were a few years old. Our team had a Raspberry Pi already, so we decided to start with this board. Early in development we found that the bootloader used by the Raspberry Pi board is a very proprietary and closed source. This was a major roadblock for our system as it prevented us from getting Xen working. This led us to move away from the Raspberry Pi and pursue other options.

4.2.2.2 RockPro64

The RockPro64 is a popular alternative to the Raspberry Pi made by Pine64. This board meets the ARM requirements of our design and is readily available. In searching for Xen documentation, we found a few references to Xen support in documentation and others getting Xen working on the platform. This platform meets the ARM SBC requirements of our project and has easily accessible and open-source documentation so that we can thoroughly test the platform. For these reasons, this has been the main platform the team has been developing.

4.2.2.3 Avnet ZUB-1CG

The Avnet ZUB-1CG is an affordable Xilinx Ultrascale+ MPSoC based development board. Our team started to look at this board as a hardware platform as Xilinx is a major contributor to the Xen Hypervisor project and therefore many of their devices support Xen. Our team found lots of documentation about Xen on Ultrascale+ MPSoCs, including how to build Xen on Petalinux, Xilinx's embedded Linux platform, and how to configure Xen DomU's. While working on getting Petalinux built with Xen for this board, our team discovered that the ZUB-1CG is not an officially supported version of the Ultrascale+ MPSoC for Xen. This led us to move away from this specific version of the Ultrascale+ MPSoC platform for something with a bit more support and power.

4.2.2.4 Hikey 960

When looking for ARM SBCs that supported Xen, our team found the Hikey 960 which Xen had listed as a supported platform. As we started looking into this platform further it appeared that it met our requirements. The issue was that this board was very old and no longer being produced so the team was unable to source one. This caused the team to quickly move away from this platform.

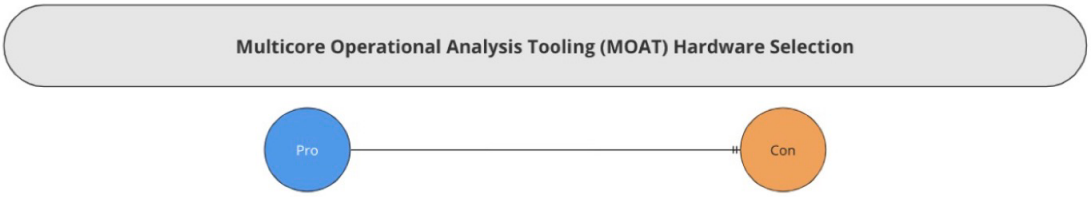
4.2.2.5 Xilinx ZCU106

The ZCU106 Evaluation Kit is a development board based around the XCZU7EV Ultrascale+ MPSoC. This board, as with the Avnet ZUB-1CG, has lots of documentation from Xilinx about how to build Xen for the Ultrascale+ MPSoC platform. This board is also referenced in the Petalinux build software that supports Xen. This information has led the team to consider this board as a potential hardware platform that would work well with Xen and multicore testing.

4.2.3 Decision-Making and Trade-Off

When determining the hardware platform our team was going to use, we looked across the internet on sites such as the Xen wiki to consolidate a list of possible solutions. This netted us the pros & cons list included below in Figure 1. To help gauge what our client desired, we presented our findings to Boeing during our weekly team-client meetings. Over the course of the weeks following, along with testing and hardware bring up, we were able to check boards off the list until we ended with our current board of choice, the RockPro64. As most of the boards on the list complete our task, there are small differences such as the delivery time, document availability and most importantly the repo access that guided our final decision.

As noted below, when working with the Pi, we quickly found that the bootloader was closed source and complex to work with. Although the large amount of community support is a good component when considering the longevity of a product, the amount of work required early on deterred us from this option. After moving away from the Pi, the team discussed with ETG the possibility of purchasing a board from a 3rd party non-name brand store. Due to policies in place, the team was forced to scrap this board due to limitations in availability from reputable locations. This led the team to proceed with purchasing the RockPro64. Although not our first choice, tests and client guidance directed us in this direction. As for the remaining FPGA board, the Xilinx, per Boeing's request, we are also proceeding with this hardware as a backup in case of unique quirks or failures of our current primary system.



	Pro	Con
Raspberry Pi	• Large amount of community support • High volume of previous Xen on Pi repositories • Easily accessible support/resource documents • Popular • Meets ARM requirements	• Repositories we found were outdated (2-5 Years old) • Closed source & proprietary bootloader • Requires a large amount of debugging work to get old code to run
RockPro64	• Direct Xen on ARM support (located on the Xen Wiki) • Readily available hardware - can be acquired quickly unlike other boards • Previous Xen on RockPro project resources available • Easily accessible open source documents	• Long delivery time after purchase
Avnet ZUB-1CG	• Cheaper option compared to other possible FPGA (Field Programable Gate Array) boards • Xilinx Ultrascale+ MPSoC has direct contributions to the Xen Hypervisor project • Large volume of resources/support documents	• ZUB-1CG is not officially supported
Hikey 960	• Meets ARM requirements	• Old hardware • Unable to source from a reputable seller
Xilinx ZCU106	• Large volume of documentation • Direct Xen on Xilinx build information • Meets Xen/ARM requirements	• Price ($1700)

Figure 7: Decision matrix for hardware selection.

4.3 Proposed Design

4.3.1 Overview

Our current design utilizes a single-board computer (a Raspberry Pi would be a familiar example), that contains various pieces of hardware and is a fully functioning system while maintaining a small size. A critical part of our design is a software component known as a hypervisor, which allows for fine-grained control over the hardware resources that applications can use on the system. Coupled with a base test, and programs designed to stress (in other words, use computing resources) the system in very specific ways, this fine-grained control is essential in ensuring precision in our comparative performance measurements. The purpose of taking comparative performance measurements is to show that a given type of stress cannot reduce the system's performance beyond a certain point. Sections 4.3.2 and 4.3.3 describe the components and how they operate in a more technical capacity.

4.3.2 Detailed Design and Visual(s)

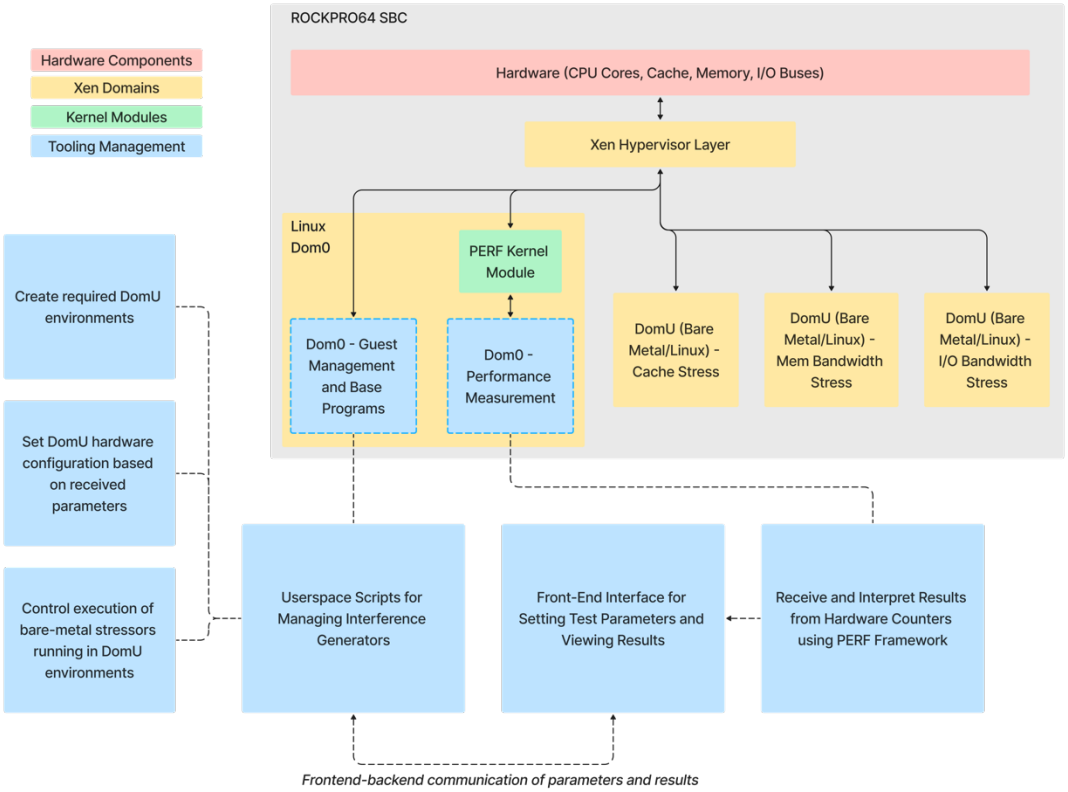


Figure 8: Block diagram of the system, including hardware and software components and how they are connected.

Figure 1 presents the schematic of our system. It consists of several components, including the hardware contained in the SoC, the Linux kernel, which hosts various kernel modules as well as Xen and its associated domains, and the various actions that will be performed by the programs running under Xen.

We will begin with Xen - Xen is an open-source hypervisor that creates and manages VMs that run a variety of guest OSs. Xen guests may also be of the bare-metal variety, in which a program is compiled into

assembly language and run directly on hardware without an operating system. As defined in our list of terms, Xen runs directly on hardware, making it a type-1 hypervisor. This is beneficial to us because it not only reduces resource overhead, but also affords us increased granularity when it comes to allocating resources to the various domains we will create and manage under it. Xen has two types of domains (virtual machines) that the user can create: Dom0 and DomU. Dom0 is the manager domain in which configuration of the main hardware and various DomU environments (guest domains/virtual machines) takes place, and this configuration is performed via hypercalls, which go up through the kernel to the hypervisor running on hardware. Once a guest VM (DomU) is configured, it can be set up to run another operating system (such as Linux), or to run a bare-metal program. A bare-metal program is a program that has been compiled down from high-level source code (like C) to the assembly code corresponding to the platform it will be running on. The advantage of this method is that no resource overhead will be consumed by an operating system running on that guest domain, as the program is being executed directly on hardware.

Our Xen Dom0 is responsible for several essential tasks. The first is managing guest domains and base programs that our team will be using to generate performance data. “Dom0 - Detailed View 1” in figure 1 above illustrates the actions that Dom0 will need to implement. The first is presenting the user with an interface capable of accepting test configuration parameters and displaying test results. This interface will be command-line based and will store test parameters, such as base test, selected points of resource contention (blue boxes in Figure 1), and any guest hardware properties (assigned CPU cores, amount of RAM, etc.) in a configuration file. The configuration file will then be read by the backend scripts, which are written in BASH, that will execute commands using the Xen management toolchain (“xl”) to instantiate one or more Xen DomU environments running bare-metal programs to initiate resource contention. The second task that Dom0 is responsible for performing is receiving and interpreting results from the PERF kernel module. The PERF module is a part of the Linux kernel which communicates with performance counters embedded in the hardware of the SoC and then makes the data available to user-space programs. Performance counters provide useful metrics such as processor cycles taken to execute a given program, and the number of cache misses. The ability to analyze this information for our base program is essential to determine that one, our interference generators are generating resource interference, and two, how much of a detriment to performance is observed to the base test when resource contention is enabled.

Xen DomU environments are less complex than Dom0 since they are purely responsible for executing a program (or programs) designed to stress a certain part of the system. In our case, we are interested in targeting the shared (L2) CPU cache subsystem, the data path between the CPU cores and main memory, and the data path from the CPU cores to the I/O subsystem on the Rockchip RK3399 SoC. These subsystems were chosen because they are shared between processor cores, meaning that programs on independent cores could still affect each other by misusing (stressing) the various subsystems. An illustration showing the channels that these methods take through the SoC are provided below.

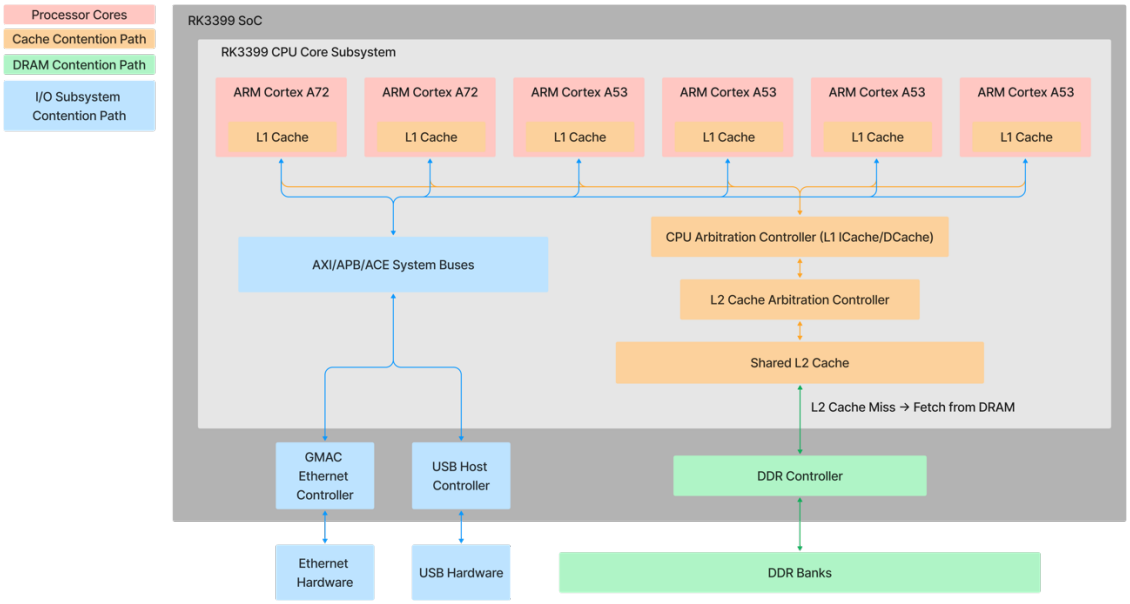


Figure 9: Components and interference channels on RK3399 SoC and associated hardware

The process of stressing the cache occurs through the CPU to cache data path, which can be seen in orange above. As can be seen, each processor core incorporates its own level 1 cache for instructions and data, however, all six cores share a cache arbitration controller and a single level 2 cache. Since caching data significantly increases application performance, the DomU environment running this stressor will be executing a program that is very cache-unoptimized, meaning that L2 cache misses are frequently occurring. This would replace the data that is present in cache for other programs running on other cores, and in theory, this should affect their performance as the data they had cached is no longer available.

The memory stress environment will follow a similar approach to cache stress. Since cache misses necessitate access to main memory, abusing this property can generate a very large amount of traffic to main memory banks, thereby stressing the available memory bandwidth for all programs on the system. Lastly, the I/O subsystem stress occurs via a system bus that is present on the CPU subsystem and facilitates communication with other controller modules present on the SoC. Multiple programs may be communicating information over a local or wide area network or via other USB peripherals, so generating a large amount of I/O traffic from a DomU on one core will stress the Quality-of-Service behavior (the manner in which I/O traffic is prioritized for transmission) of the shared Ethernet and USB controllers that manage accesses from all components on the SoC.

4.3.3 Functionality

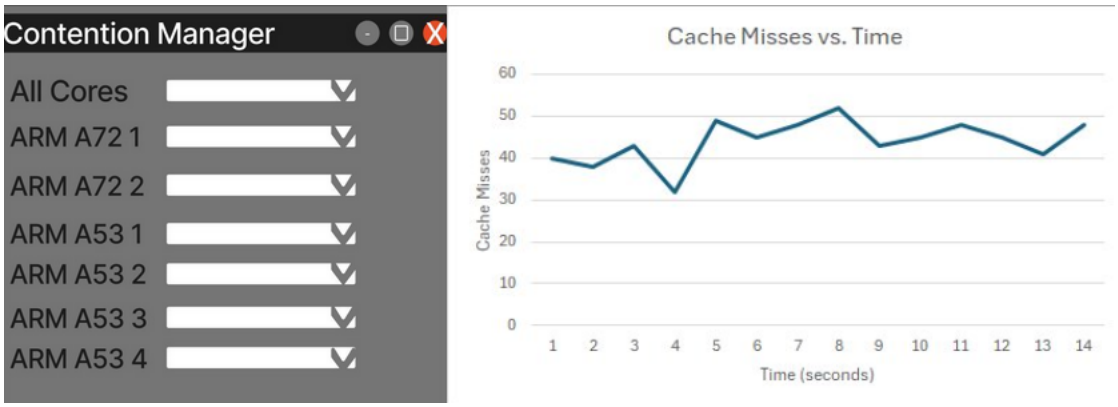


Figure 10: Wireframe diagram of a mock GUI user interface

Our end users have two options for interfacing with our design. The user will interact with our design primarily via a command line interface in which they type predefined commands that carry out various operations. Upon entering a command specifying several parameters, the application will run, and the tool will produce a file that details the performance output of the test. The command line tool allows more technically advanced users to interact with our tool in a programmatic fashion; that is, a user would be able to write scripts for the command line tool to run a variable number of tests and store the output somewhere. A final version of our design will also provide a visual GUI interface, as shown in figure 2, that outputs visual feedback on how the stressors are affecting the system's performance. The GUI allows the tool to be used in a more intuitive manner for less technically inclined users. Furthermore, the graphs the GUI produces are also of value to technical users when communicating results to non-technical users.

4.3.4 Areas of Concern and Development

At this stage of our project, our design is a rough outline of the major components of our client's requirements. There are many components to our design that require us to iteratively refine details as we work closely with our client week-to-week. Our design satisfies the hardware requirements in that we are working with an ARM based SoC running a Xen hypervisor. Currently, the team's effort is focused on identifying interference channels for our hardware platform. This is the primary interest of our client and deliverable of our design. Our client has also requested a command line-like utility that has all its functionality mirrored in a GUI interface. The final aspect of our design, the user interface, has been prototyped but not implemented.

Our team has several concerns for delivering a finished product to our client. The primary concern of our design is centered around time. To deliver a final product that satisfies all user requirements requires our team to closely adhere to an established project schedule to allow for adequate time in each stage of our design. Various technical aspects of our design may require debugging and research to be implemented correctly. Subsequent stages of the project may be hindered by getting stuck in the early stages.

Another concern is focused on how our design will capture the metrics relevant to our client's requirements. Ultimately, our design seeks to prove the Worst-Case Execution Time (WCET) for a given hardware platform, but we have not clearly defined how this can be achieved. If this aspect is missing from our design, the user will not be able to use the tool as intended and our client will not receive the product they requested.

Our weekly meetings with our client are likely the best method to get unstuck, clearly define the expectations of the final product, and properly address these concerns. To resolve our concerns of being

blocked on technical issues, we can schedule time with the engineers our client includes in our weekly meetings to get unstuck. Regarding gathering performance metrics, we can articulate our concerns to our client and ask for guidance on how this information can be gathered from our current design.

4.4 Technology Considerations

The following section details the technologies that our design uses. Each technology is concerned from the standpoint of its strengths and weaknesses, as well as viable alternatives when applicable.

4.4.1 RockPro64

The RockPro64 is a Single Board Computer (SBC) that closely aligns with what our client requested in terms of a hardware platform. It uses the ARM instruction set and has a multi-core heterogeneous processor. This architecture allows our team to partition the system resources such that a victim application running on one core can have its performance degraded by a stressor running on another core. Furthermore, we were able to find abundant hardware documentation for the platform online between the official manufacturer and an online wiki. This aids our overall design in that the more knowledge we have of the underlying hardware, the better able we can write stressors for our platform.

Relative to other SBCs that we considered, like the Raspberry Pi 4 or HiKey 960, the RockPro64 was not only the easiest to work with but also the most well-documented. One weakness is that Xen may be easier or better suited to running on an FPGA like the Xilinx ZCU board. We speculate many embedded applications that use Xen run on an FPGA board rather than an SBC. Ultimately, its weaknesses are negligible when one considers that it is the only board that meets our design requirements, is easy to work with, and relatively cheap compared to other options.

4.4.2 Xen Hypervisor

Xen is an open-source type I hypervisor with abundant documentation. Additionally, our client has several engineers with experience using Xen for different applications. This allows us to create virtual machines on our hardware platform to properly isolate the system's hardware resources to properly characterize interference channels.

The challenge of using this technology lies in its learning curve. Xen is widely used in industry, but it is commonly running on hardware different than what our team is using. This has meant that we have spent more time than we had originally expected troubleshooting Xen issues when it was originally meant to simply be a supporting technology to our overall design. The alternative to using Xen is that our project can in some capacity be emulated in a Linux environment.

4.4.3 Stress-NG

Stress-ng (i.e., Stress-NextGeneration) is a software library used to stress the various subsystems of a computer. The primary advantage that this technology lends to our project is that it implements many stress base cases. This is useful for our project in that it allows us to uncover previously unidentified interference channels on our hardware platform, which is the fundamental goal of our design. Stress-ng is ultimately an accelerator for our work; it allows us to get to the most important parts of our design sooner. Without it, we would need to write custom stressors for each base case.

Stress-ng's weaknesses lie in that it is a general library intended for use in various systems. If there is a specific aspect of our platform we wish to exploit, we may need to write custom software to accomplish the behavior we want. This can prove to be a time-consuming process given how closely our software is

integrated with the hardware. There are not many alternatives to Stress-ng, so writing our own stressors in C and assembly are likely not a viable option.

4.4.4 Perf

Perf is a performance tool built into the Linux kernel that will allow us to gather performance metrics. This is extremely useful to our design when we are trying to characterize how our stressors are affecting victim applications. Its primary strengths are that it is built into the Linux kernel meaning we already have access to it on our target platform. Furthermore, perf also has extensive documentation which we can reference to better understand how to use it for different aspects of our design. The primary weakness in the context of our design is that it requires us to learn how to use the tool and integrate it into our larger design.

One alternative to perf is gprof, which is another performance analysis tool for Unix like applications. The drawback of using this approach is that our client has experience with using perf, so we would likely need to learn how to use a new tool to accomplish something that we already know how to use.

4.5 Design Analysis

At this point in the project, the team is focused on hardware bring-up and researching interference channels. For hardware bring up, the team is working with the RockPro64 to get a working version of Xen Hypervisor on it and make sure that it can be configured with the tools that we need. The team is also working on getting a Petalinux image built so that we can pursue an FPGA based backup platform for further development of the tools. For the interference channel research, the team has divided the interference channels into Cache, Memory Bandwidth, and I/O. The team has been spending time looking at the architecture of these systems and seeing how they can be exploited to interfere with other cores.

Using what the team has learned about the ARM architecture and Xen Hypervisor, the design has been tweaked and improved to attempt to minimize issues. The biggest issues appear during the hardware bring-up which is nearing completion. From there, time will be spent writing software to exploit the shared resources on the hardware. The team should be able to proceed with the planned design with little interference from scheduling or other unforeseen issues. The project is overall feasible, and the team is confident in our ability to complete it.

5 Testing

Our testing philosophy is centered around isolating systems and performing unit tests to ensure that they fit into our larger design. Due to the multiple layers of hardware and software to successfully implement our project, most of our tests are focused on ensuring that the system has the same functionality after a unit is integrated as it did before. This allows us to build on a solid base.

Naturally, most of our testing is regression testing. The scripts we have stored in our GitLab repository allow us to iterate quickly on our design and recover to a working state if a regression test fails. This aids not only the reproducibility of our work but also its dissemination to a wider audience.

The primary challenge of testing our project also stems from the scope of testing that must be performed. Our design requires us to perform testing at the hardware, software, and user levels.

5.1 Unit Testing

In the current stage of our project the primary units under test in our design are our hardware platform's cache, memory, and I/O. We perform these tests by first partitioning the underlying hardware platform's resources using the Xen hypervisor. We then start an interference generator (i.e., a program that exploits the hardware's vulnerability to shared resource contention) provided by a software library like stress-NG or custom written by one of the engineering team. Using profiling tools like perf we can collect metrics and statistics relevant to the Worst-Case Execution Time (WCET) of the system.

Our units under test, as an itemized list:

- Shared (Level 2) Processor Cache
    - The programs we have written aim to eject as many cache lines as possible
    - Our programs are also written in such a way to minimize cache consistency
- Main Memory/DRAM
    - The current memory base cases seek to maximize the amount of traffic from the device's main memory to secondary storage (i.e. disk)
    - Our testing of this component also includes ensuring that memory locality (i.e. commonly used data) is minimized
    - Cacheable and non-cacheable access types are considered
        - Cache size and replacement algorithms can be abused to generate large amounts of memory traffic
- Input/Output Buses
    - The current base cases seek to generate as much traffic on the system's shared I/O buses to create resource contention
    - We are considering a test program that creates contention for the platform's Ethernet controllers, which introduces an interference channel for multiple applications utilizing it but running on different cores

Unit testing tools and libraries:

- Perf (Linux performance analysis tool) – this tool allows us to analyze the execution time and values accumulated in hardware performance counters while running our test programs and compare them against the base case values
- Stress-NG – software library for stressing computer hardware
- MemGuard – software library that allows a user to reserve memory for an application
- Xen hypervisor – tool that allows us to partition system resources for application profiling
- Valgrind Application Profiling Tools – this software library allows our tests to maximize the number of cache misses for our platform's cache hierarchy
- ARMv8 Hardware Performance Counters – these registers contain information related to the number of cache hits/misses that provide our tests with concrete metrics

5.2 Interface Testing

The interfaces in our design are mostly hardware interfaces. For a detailed graphical representation of these interfaces, see figure 9 of section 4.3.2 in this document. Examples include:

- Data path from the CPU cores to L2 cache
- Data path from the CPU cores to main memory
- Data path from CPU cores to I/O (ethernet/USB)
- Our programs also interface with the kernel of the guest operating system to measure performance metrics

Many of the tools we are using to perform interface testing have overlap with the tools we are using to perform unit testing:

- Perf (Linux performance analysis tool) – this tool allows us to analyze the execution time of our test programs and compare them against normal execution times
- Xen hypervisor – tool that allows us to partition system resources for application profiling
- Valgrind Application Profiling Tools – this software library allows our tests to maximize the number of cache misses for our platform's cache hierarchy
- ARMv8 Hardware Performance Counters – these registers contain information related to the number of cache hits/misses that provide our tests with concrete metrics

Interaction between these components is inherent to a multicore system. For this reason, we must carefully partition system resources to ensure that the test we are running is affecting the interface we are targeting. The partitioning abilities of the Xen hypervisor allow us to test these interfaces in isolation or in aggregation.

5.3 Integration Testing

Our project has a relatively wide scope that requires the integration of multiple subsystems for its proper realization. Due to is complexity, there are several critical integration pathways that require testing:

Integration Path 1: Xen on ARM

This step involves installing the Xen hypervisor on our chosen hardware platform. In practice, this has proven to be more time-consuming than initially planned, but it is a critical milestone in our design as well as our client's expectations. This has largely been tested via regression testing. In other words, we have attempted to install Xen by making iterative changes to the build and configuration process and rolled back to a known working state if an implementation failed.

Integration Path 2: Installing supporting libraries on the hardware platform

Libraries that support our overall design, like perf, Valgrind, and MemGuard must not only be supported on the platform but also co-exist with every other library. Our primary method of addressing this integration path is regression testing and rolling back to a known working state if the implementation fails.

Integration Path 3: Integrating all tools into one tool suite

As our design approaches its final form, we will need to integrate all hardware and software tools we have created into one comprehensive software package. This includes the integration of our custom written interference generators with existing open-source libraries found in stress-NG. Finally, this step will also include the integration of a UNIX-like command line tool with a Graphical User Interface. Usability studies with our client will expose the aspects that need most improvement at this stage of the project. Integrating all components should also pass all regression tests as well.

Tools:

- Git & GitLab – distributed version control allows each engineer to work independently or collaboratively on aspects of the project. Should any new feature not pass regression tests, we can simply revert the Git history to a previous known working state
- Perf (Linux performance analysis tool) – this tool allows us to analyze the execution time of our test programs and compare them against normal execution times. This will allow us to determine how different aspects of our design change as new components are integrated

5.4 System Testing

To ensure that our design functions we have written a set of scripts that run our interference generating programs. These scripts essentially allow us to automatically re-verify our unit tests and ensure that every component of our design is in a working state after a change is made.

Due to the nature of our design, this can be time-consuming. Each interference generator must run its set of applications with its specific resource partitioning schemes to accurately characterize the underlying hardware. This is to say that each of the unit tests must be run iteratively. This testing's comprehensive nature also ensures that each interface of our design is verified.

In future development, we will need to implement a user GUI (Graphical User Interface). To ensure a consistent experience for our users, we also plan on introducing CI/CD (Continuous Integration / Continuous Deployment) pipeline to our GitLab repository. This will let us run a set of test cases against each commit to ensure that the system meets functional requirements. The latest version of the project will then be deployed to the user, resulting in the user being provided with the best version of our work that also meets non-functional requirements in the shortest timeframe.

Tools:

- Git & GitLab – allows each engineer to independently make progress on the project. The version history Git maintains also ensures that the project can be reverted to a known working state if a new addition fails regression testing.
- CI/CD - allows our team to run automated testing for each change made to our repository of work. Additionally, this gets functioning versions of our project to users faster.
- Bash scripting – allows us to automate the unit tests we want to run on each iteration of the design

5.5 Regression Testing

Our primary method of implementing regression testing comes from analyzing the output of new versions of interference generators. If we find that the results of an interference generator change after a change in the code, we can simply roll the changes back locally or in our GitLab repository.

In the later portion of our project, we will be developing a GUI for our tool suite. This section of our project can rely heavily on our GitLab repository's CI/CD pipeline. If any change does not pass all the pre-defined tests the change will fail and will not make it into the primary version released to our users.

Tools:

- Git & GitLab – allows each engineer to independently make progress on the project. The version history Git maintains also ensures that the project can be reverted to a known working state if a new addition fails regression testing.
- CI/CD - allows our team to run automated testing for each change made to our repository of work. Additionally, this gets functioning versions of our project to users faster.

- Bash scripting – allows us to automate the unit tests we want to run on each iteration of the design
- Perf (Linux performance analysis tool) – this tool allows us to analyze the execution time of our test programs and compare them against normal execution times. This will allow us to determine how different aspects of our design change as new components are integrated

5.6 Acceptance Testing

Our team works very closely with our client. This means that we have meetings with our client in which we present our progress on open action items and the result of our efforts. Our client can then guide us where they want to see the project go. In practice, we are performing acceptance testing with our client at the end of every sprint. This ensures that both the functional and non-functional requirements are being met and are up to our client's expectations. As our design takes a more definite shape, we also plan on performing a usability study in which an engineer from outside of our team can use our tool. This will provide us with more data on how the functional and non-functional requirements of our design have evolved or could otherwise be improved.

5.7 Results

Our testing so far has revealed that we are succeeding in generating stress on the underlying hardware. Figure 11 below demonstrates the output of one of our unit tests applied to the system's memory bandwidth. The unit test's output shows the metrics we use to determine how well our tool suite generates interference on the underlying hardware.

As our project grows in complexity, we will add more interference generators which will need to be tested iteratively on the hardware. We will be developing a set of scripts in parallel that ensure that all unit tests work as we previously expected them to after changes are introduced to the project. This form of system testing ensures that each unit and interface is tested at each step of the project.

These unit tests ensure that the core functional requirements of our design are being met and further developed. As mentioned, we intend to introduce a CI/CD pipeline that enforces a standard for the quality of work that we are releasing to our end users.

Lastly, we perform extensive acceptance testing with our client. We present our results as a team with our client who gives us feedback and guides the direction of our project. This allows us to ultimately produce a design that is very closely aligned with what our client wants. It also forces our team to keep non-functional requirements in mind.

It's also important to note that these results are still preliminary. As mentioned, challenges in getting Xen functional on our hardware prevented us from performing as much testing as was desired. Therefore, as we move into the fall and begin integrating tests, we will have many more results to evaluate.

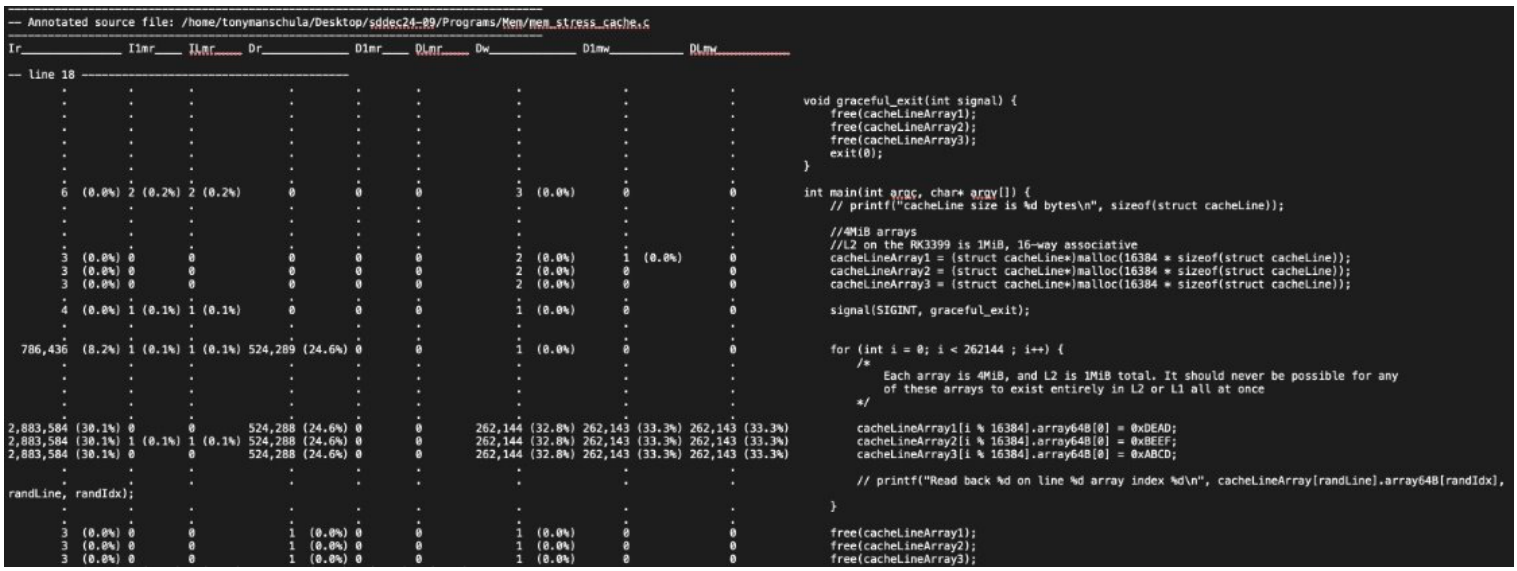


Figure 11: A theoretical cache utilization profile generated over one of our stress tests using the CacheGrind library

6 Implementation

For our project, the development and implementation of the backend (tooling development, stress generation, and system bring up) are tightly coupled. Details and rationale behind this process are discussed in section 4.3.2. However, our requirements stipulate that the toolset should have a user-friendly GUI for controlling test parameters, initiating tests, and analyzing test results. This process of designing the GUI is highly decoupled from the implementation. For a short discussion on the design of the GUI, see section 4.3.3. Currently, the team is planning on implementing the GUI using Python for its ease of use. This will happen once the backend portion of the project is fully complete.

7 Professional Responsibility

7.1 Areas of Responsibility

Area of Responsibility	IEEE Code of Ethics	NSPE Code of Ethics
Work Competence	Ensure that the team has the ability and knowledge to complete every project completely and safely.	Engineers shall undertake assignments only when qualified.
Financial Responsibility	Provide clients and users with accurate and realistic estimates for the costs associated with the projects.	Act as good agents on a client's behalf.
Communication Honesty	Provide honesty in all aspects of feedback, criticism, and design abilities.	Avoid deceptive acts & make statements in a truthful manner.
Health, Safety, Well-Being	The health, safety, and well-being of the users and public is the number one priority.	Hold paramount the safety, health, and welfare of the public.
Property Ownership	Treat others and their property fairly and with respect.	Act as good agents on a client's behalf.
Sustainability	Strive to implement positive sustainability practices to better the lives of users and surrounding communities.	Adhere to the principles of sustainable & development in order to protect the environment for future generations.
Social Responsibility	Act as responsible members of society and strive to better one's surroundings.	Honorably conduct themselves as to enhance the reputation of the profession.

Comparing IEEE's code of ethics to NSPE (National Society of Professional Engineers) we can see a lot of the same values; prioritize safety and health while avoiding un-honorable acts, etc. At face value IEEE can be described as a worldwide association of electronic and electrical engineers, which differs from NSPE which encompasses primarily U.S. based engineers.

7.2 Project Specific Professional Responsibility Areas

- Work Competence – This Area of Responsibility is extremely important for our project as we are dealing with safety-critical systems. If our system performs incorrect analysis on multicore embedded systems, it could lead to failures in airplanes mid-flight and potential loss of life and property. – ***Performance (High)***
- Financial Responsibility – Our project has a low cost as it is primarily software based and is portable across the ARM architecture. – ***Performance (Low)***
- Communication Honesty – This area applies to our project as we are completing work for our client, Boeing. It is very important that we keep communication open and keep our client informed about our progress, any challenges encountered, and any issues with our work. In a broader context, the results of our work could inform the safety decisions across the avionics industry. This means that our team has a strong professional and ethical responsibility to communicate clearly, unambiguously, and with absolute transparency. – ***Performance (High)***
- Health, Safety, Well-Being – The project does not directly affect people or have many health risks. As stated earlier, however, if our project is not built well, it could lead to failing avionics mid-flight. – ***Performance (N/A)***

- Property Ownership – Our project is Open Source and will ultimately be available for anyone to use. This leads to Property Ownership not being something that affects our project. **– *Performance (N/A)***
- Sustainability – This Area of Responsibility is also a low priority for our project as it has very little direct environmental impact. Our software also runs on hardware that consumes very little power. – ***Performance (N/A)***
- Social Responsibility – The Social Responsibility of our project is to improve the efficiency of Avionics systems while maintaining the highest levels of safety. – ***Performance (Medium)***

7.3 Most Applicable Professional Responsibility Area

For our Multicore Analysis Tooling project, Work Competence is the most important Area of Responsibility. If the team does not complete the project competently, there could be serious consequences, such as airplanes crashing and avionics systems failing. For this reason, the team has spent lots of time researching methods to validate WCET and use industry contacts to verify the correctness of our system.

8 Closing Material

8.1 Discussion

Given the current state of our project, there are some key insights we would like to discuss. Firstly, the process of getting Xen installed on any of the hardware platforms that our team worked with proved to be significantly more difficult than anticipated. This process required use of many embedded systems development frameworks like Yocto and PetaLinux, all of which have varying levels of documentation (and varying age of documentation). While it took longer than the group would have preferred, we have fulfilled the requirements concerning processor architecture and device form factor with Xen fully functional on both the RockPro64 and Xilinx ZCU106 FPGA development board.

While the Xen bring up process was time consuming, the team was able to engage in extensive research regarding the technical details of our hardware in preparation to write interference generators. Therefore, we believe that the approach we describe in section 4.3.2 sufficiently fulfills technical requirement two. In conjunction, we also believe that, over the course our research on our hardware and activities configuring embedded Linux environments to bring up Xen, we have fulfilled the developer-focused aspects of the four User Experiential Requirements.

8.2 Future work

Due to unforeseen project delays, the current future work plan consists of generating interference, developing mitigation methods, as well as creating a frontend user interface for the testing tool. In addition to exploring and mitigating these interference forms, if time permits, other forms of interference will be explored (Cache Coherency). Finally, once the preliminary steps have been concluded (eg. Interreference and GUI), the team will identify existing gaps in preparation for project hand-off.

8.3 Conclusion

The team currently has two systems with functioning Xen installations, as well as several tests that we have developed. In order to achieve our goal of determining worst-case execution time on either of these hardware platforms and creating a polished final product, the following intermediate steps must be accomplished:

- Finish testing, development, and performance evaluation of base tests

- Develop and implement programs that introduce shared resource contention
    - Configure Xen to run them alongside the base test
- Extract and analyze hardware performance counter data using perf framework
- Develop automated scripts and a GUI to improve test suite usability

Retrospectively, the team should have focused initial efforts on the Xilinx FPGA board implementation of Xen, as that was where the most documentation was available. While we are in a place where both platforms are functional, the amount of time it took to get to a working state on the Xilinx board was minute (a matter of days) in comparison to the amount of time invested in troubleshooting the RockPro64 (a matter of weeks).

8.4 References

[1] S. H. VanderLeest and C. Evripidou, "An Approach to Verification of Interference Concerns for Multicore Systems (CAST-32A)," *SAE International Journal of Advances and Current Practices in Mobility*, vol. 2, no. 3, pp. 1174–1181, Mar. 2020, doi: https://doi.org/10.4271/2020-01-0016.

[2] S. H. VanderLeest and D. Matthews, "Incremental Assurance of Multicore Integrated Modular Avionics (IMA)," in *IEEE Xplore*, ieeexplore.ieee.org: IEEE Xplore, Nov. 2021. Accessed: Jan. 16, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/9594404

[3] S. H. VanderLeest and S. R. Thompson, "Measuring the Impact of Interference Channels on Multicore Avionics," *arXiv.org*, Jan. 06, 2021. https://arxiv.org/abs/2101.02204 (accessed Apr. 16, 2024).

[4] S. H. VanderLeest, J. Millwood, and C. Guikema, "A Framework for Analyzing Shared Resource Interference in a Multicore System," in *IEEE Xplore*, IEEE Xplore: IEEE Xplore, Dec. 2018. Accessed: Jan. 16, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/8569651

9 Team

9.1 Team Members

- Alex Bashara – Embedded and Cache Engineer
- Joseph Dicklin – I/O Engineer
- Hankel Haldin – Platform Bring up Engineer
- Anthony Manschula – Project Coordinator and Memory Engineer

9.2 Required Skill Sets for Your Project

This project requires familiarity with computer architecture and an understanding of how code runs at a low level and knowledge of real-time embedded systems.

9.3 Skill Sets covered by the Team

- Alex Bashara – Real Time Embedded Systems
- Joseph Dicklin – Low Level Electrical Systems
- Hankel Haldin – Low Level System Programming
- Anthony Manschula – Computer Architecture Knowledge

9.4 Project Management Style Adopted by the team

The team adopted an agile project management style, focusing on weekly meetings with the team and advisors to checkpoint progress and set goals for the following week. Issues have been tracked in Gitlab with weekly updates during our advisor meetings.

9.5 Initial Project Management Roles

- Alex Bashara – Cache Interference Lead
- Joseph Dicklin – I/O Interference Lead
- Hankel Haldin – Hypervisor Lead
- Anthony Manschula – Memory Interference Lead

9.6 Team Contract

Team Name: *MOAT*

Team Members:

1) Alexander Bashara

2) Anthony Manschula

3) Hankel Haldin

4) Joseph Dicklin

Team Procedures

1. ***Day, time, and location (face-to-face or virtual) for regular team meetings:*** Sundays at 12pm in person (Subject to Change).

2. ***Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):*** Discord, GitLab Boards (issue tracking, management), Trello

3. ***Decision-making policy (e.g., consensus, majority vote):*** Consensus.

4. ***Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):*** Will take meeting notes and minutes in the shared OneNote notebook.

Participation Expectations

1. ***Expected individual attendance, punctuality, and participation at all team meetings:*** Attendance to all class sessions, timely attendance to out of class meetings (what the team deems needed per week), participation in Boeing meetings (ideas, questions, etc...), as well as other/misc. requirements discussed within the team.

2. ***Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:*** Be responsible and proactive about meeting deadlines and be sure to communicate in a timely manner if you do not believe you will be able to meet a deadline. Team members should be able to problem solve on their own but not be afraid to ask a question if they cannot find an answer.

3. ***Expected level of communication with other team members:***

Respond to Discord messages within a day's time.

Group members should provide notice if they are not able to attend a meeting. If a group member is stuck or otherwise having difficulty completing work, they should inform the group so continued progress can be made on the project.

4. ***Expected level of commitment to team decisions and tasks:*** If a group member takes responsibility for a task, he should be committed to completing it by assigning a deadline to it. If a group member is blocked on the completion of a task, it is that person's responsibility to ask for help.

Leadership

1. ***Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):*** Team organization & client interaction – Anthony, Hardware research – Joe, Embedded engineering and testing lead – Alex, Hypervisor & platform bring up – Hankel

2. ***Strategies for supporting and guiding the work of all team members:*** Be transparent with your skills and deficiencies, allowing your fellow team members to aid in the completion of your individual role.

3. ***Strategies for recognizing the contributions of all team members:*** Talk about what each team member accomplished over the week and what their goals are for the next week.

Collaboration and Inclusion

1. ***Describe the skills, expertise, and unique perspectives each team member brings to the team:*** Each member comes from a similar knowledge base in terms of computer and electrical

engineering knowledge. However, each member has a different focus in terms of specialized expertise (digital logic, operating systems, etc.).

2. ***Strategies for encouraging and supporting contributions and ideas from all team members:*** Everyone on the team has a unique skill set and chance to contribute in a meaningful way, so clearly communicate what your strengths are. Help others when you have the chance to and take the opportunity to move the project forward when they are presented.

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?):*** Members should be open about their progress and any environment issues in meetings so that they can be addressed quickly.

Goal setting, Planning, and Execution

1. ***Team goals for this semester:*** TBD. As stated above, we must first meet with our Boeing contact to get an in-depth overview of the complete project. Once we know that, we will discuss within the team what is reasonably achievable given the semester period.

2. ***Strategies for planning and assigning individual and teamwork:*** Plan the major deliverables after our first meeting with Boeing rep. From there, fill out sub-tasks necessary to achieve these deliverables on time. Tasks will be assigned based on knowledge level and confidence in the subject. At the weekly member meetings on Sunday, tasks will be reviewed in terms of status and time frame, and other tasks will be created or removed as necessary to keep the project organized.

3. ***Strategies for keeping on task:*** Set meaningful and achievable goals each week so that we can continuously make progress on the project.

Consequences for Not Adhering to Team Contract

1. ***How will you handle infractions of any of the obligations of this team contract?*** Have a civil discussion as a team to see how we can fix the issue.

2. ***What will your team do if the infractions continue?*** If the initial inter-team discussions prove unproductive, discuss recommendations with the project advisor.

****

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) Joseph Dicklin DATE: 1/30/24

2) Hankel Haldin DATE: 1/30/24

3) Alexander Bashara DATE: 1/30/24

4) Anthony Manschula DATE: 1/30/24