

Multicore Operational Analysis Tooling (MOAT)

SENIOR DESIGN DEC'24 TEAM 09

Anthony Manschula, Alexander Bashara, Joseph Dicklin, Hankel Haldin

Client: The Boeing Company

Advisors: Steve VanderLeest (Boeing), Joe Zambreno (ISU), Phillip Jones (ISU)

IOWA STATE UNIVERSITY

Problem Statement

- Motivation: Increasing computational demand of avionics programs necessitates higher performance multicore systems
- Problem: Such systems can exhibit **multicore interference** between shared resources
 - Can cause system to stall for access and introduce unpredictable delay
 - Practical example: Radio control program interferes with flight control program
- **Airworthiness certification: Must quantify the worst-case execution time in the context of high-interference scenarios**
- This project aims to create a flexible tool for analyzing the performance of programs on multicore embedded systems in high interference situations

Market Survey & Research

	Multicore Operational Analysis Tool (MOAT)	RapiDaemon	MASTECs	OTAWA (Open Tool for Adaptive WCET Analysis)	Multicore Test Harness
Pros	Open source, built with modern hardware in mind	Designed by a team of professional engineers, specifically, for compliance testing	Offers timing analysis software and multicore performance consulting	Open source, supports several ISAs (e.g., ARM, RISC-V, etc.)	Open source & has good instructions
Cons	Produced under tight time deadlines	Closed source & expensive	Potential portability issues to North America	No recent builds, not well-known	Relatively old; development stopped five years ago

Figure 1: Market research summary

Conceptual Sketch

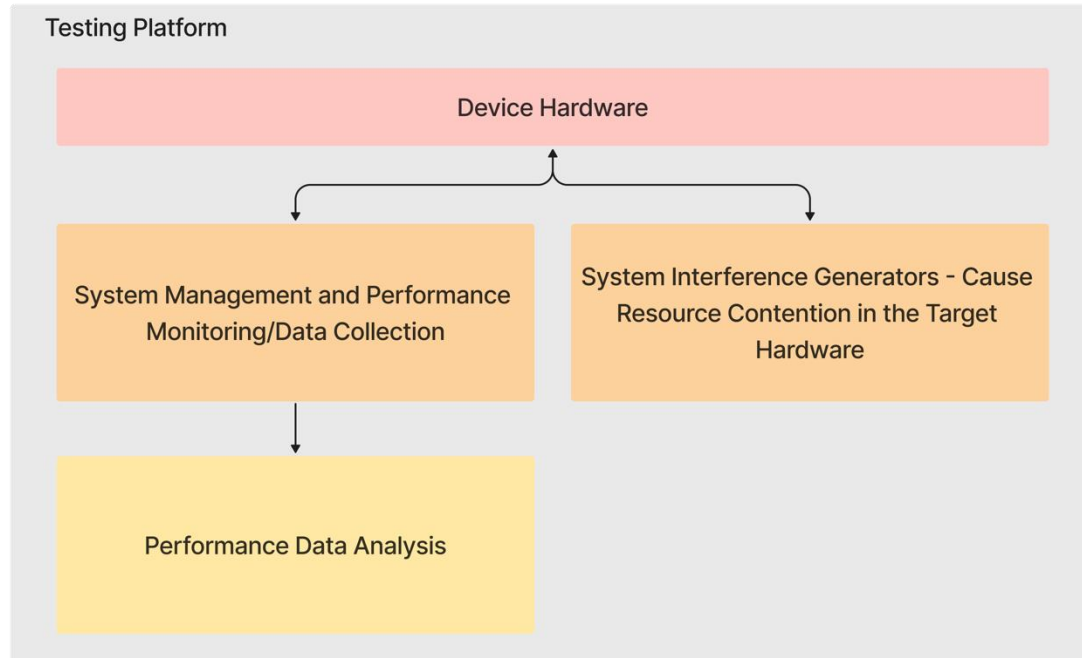


Figure 2: High-level view of the subsystems that comprise our design

Functional Requirements

- Toolset must thoroughly and methodically stress the system in a reproducible way
- Toolset must focus on major points of resource contention (Shared data caches, memory bandwidth, I/O bus usage, etc.)
- Accurately simulate potential worst-case scenarios (i.e., simulate a rogue process using too much CPU time/memory/I/O bandwidth)
- Toolset must collect and analyze performance data to facilitate analysis of system performance and factors contributing to program worst-case execution time

Non-Functional Requirements

- **Architecture** - System must implement a processor based on the ARMv8 instruction set and have two or more cores
- **Form-factor** - Single-board computer (Raspberry Pi, Pine64 family, etc.), or FPGA board with Xilinx UltraScale+ MPSoC (Xilinx ZCU family or similar)
- **Hypervisor** – Our design must use a type 1 hypervisor (Xen) to partition underlying system resources

Other Constraints and Considerations

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

User Interface Requirements:

- Command-line utility for automated testing
- Ability to run single tests manually or from a predefined sequence

System Design

Component Decomposition

- **Hardware** – Processor cores, memory, I/O, etc. - Xilinx ZCU106 FPGA Board
- **Xen Hypervisor** – Manages hardware resource allocation to domains (guests) running on the system
- **Domain 0 (Dom0)** - Linux environment manages configuration and operation of guest domains (DomU's), runs user interface utilities, and performs system performance monitoring
- **Guest Domains (DomU)** – Hosts interference generators
- **Interference Generators** – Generate resource contention in areas as requested by the test, such as level 2 processor cache or main memory
- **User Interface** – Manage test flow, analyze test execution results, generate visuals

Block-Level System Diagram

The system is broken up into SoC hardware, Xen and its associated DomU's, the Linux kernel, and the external host machine used for data analysis

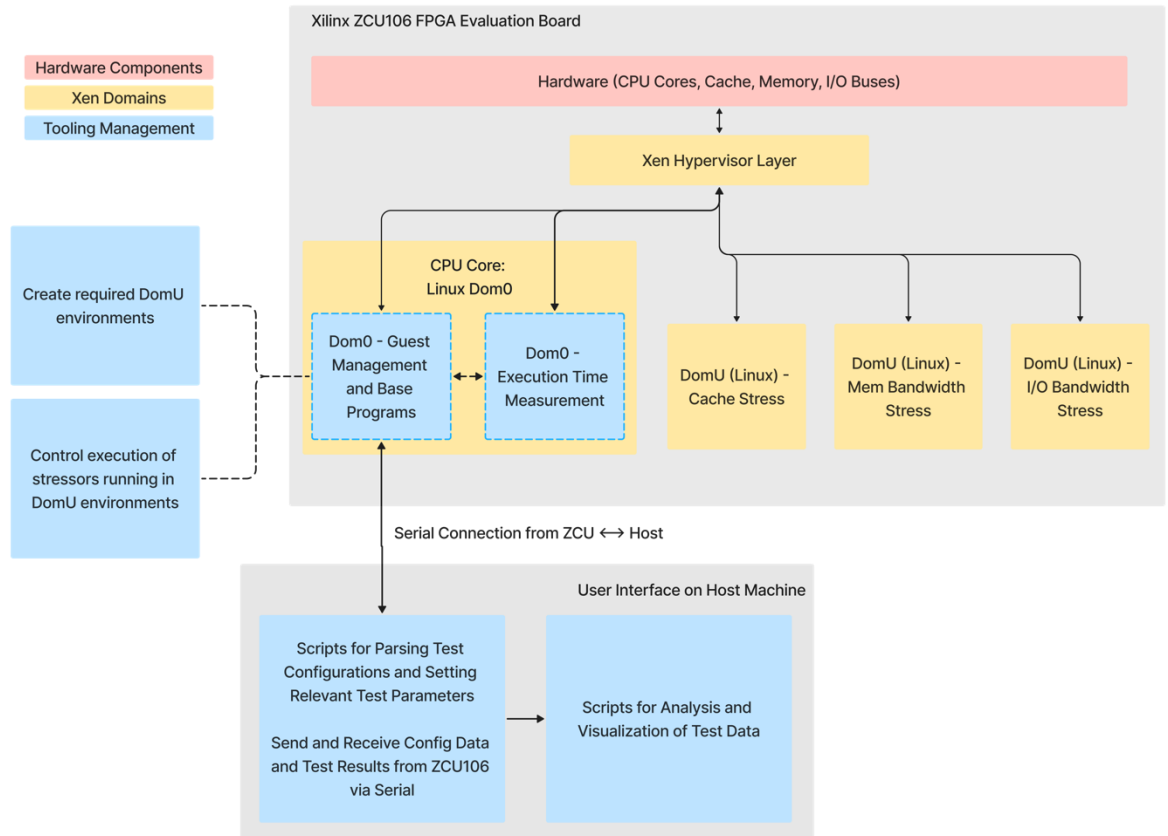


Figure 3: Block-Level System Diagram

Embedded Software

- The Xen Image was built using Yocto and PetaLinux tools
 - Frameworks for Embedded Linux on ARM
 - **Not trivial:** Including Xen in these builds requires hardware-specific changes
 - Utilized Ubuntu Base as the guest OS for DomU's
 - Lightweight, minimal overhead (important)
 - Modified to include software relevant to stress testing
 - Thinking forward: Documented build processes for potential future groups

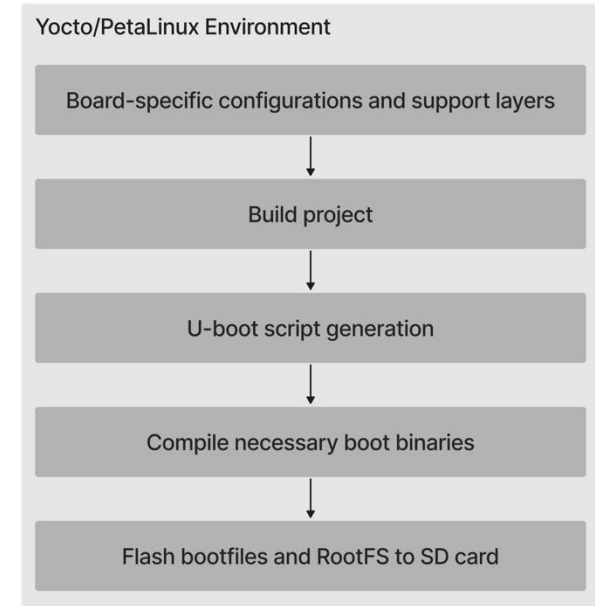


Figure 4: Image configuration and build process

How does our Design Generate Interference?

- Caused by simultaneous utilization of shared resources both on and off the SoC by different processes (“architectural interference”)
 - e.g., Cache, Memory, I/O
- Interference – Stress-NG
 - Open-source performance testing library
- UI parses test definition for interference type on a per-core basis (0 to 3 cores running)
 - DomU assigned to that given core is started
 - Parameters are passed to the DomU to initiate contention generation

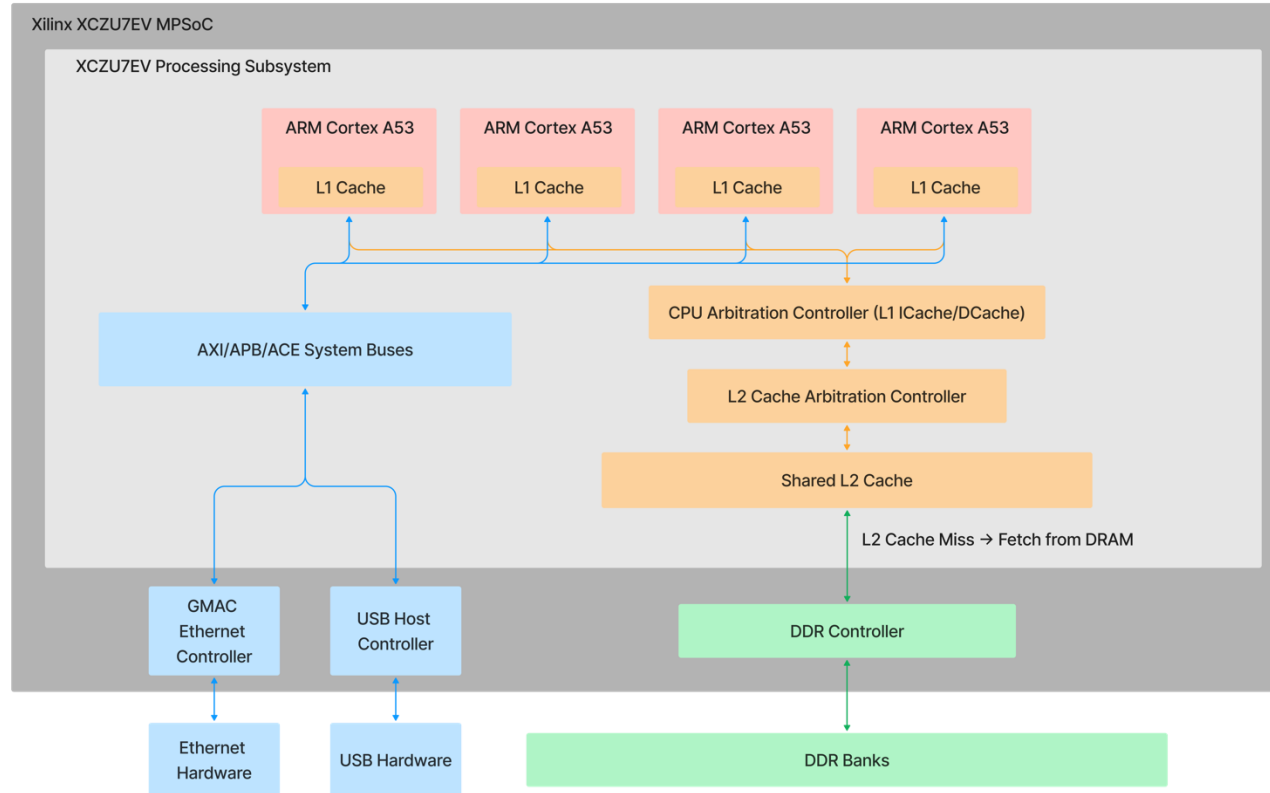
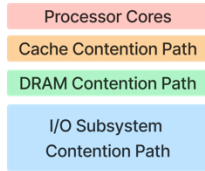
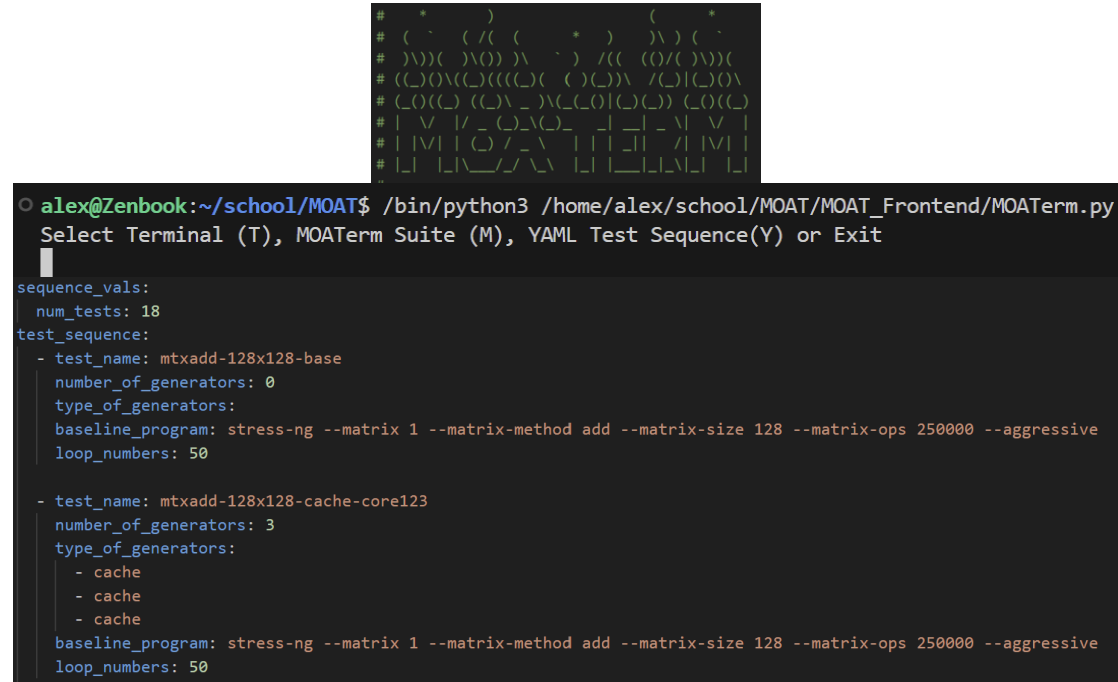


Figure 5: XCZU7EV MPSoC Shared Resource Diagram

How does the User Interact with the Design?

- “Host” device communicates with the hardware via serial
- Test management is done via MOATerm - Multicore Operational Analysis Terminal
 - Manual input of one-off input of parameters
 - Batch testing via YAML test definition



```
# * ) ( *
# ( ^ ( / ( * ) \ ) ( ^
# \ ) ( \ ) \ ^ ) / ( ( / ) \ ) (
# ( ) \ ( ) ( ( ( ) ( ) \ ) / ( ) ( ) \
# ( ) ( ) ( ) \ _ ) \ ( ) ( ) ( ) ( ) ( )
# | \ / _ ( ) \ ( ) _ _ | _ | _ \ / |
# | \ / | ( ) / _ \ _ | | | | / | \ / |
# | _ | \ _ / / \ _ | | _ | \ _ | |
"
```

```
alex@Zenbook:~/school/MOAT$ /bin/python3 /home/alex/school/MOAT/MOAT_Frontend/MOATerm.py
Select Terminal (T), MOATerm Suite (M), YAML Test Sequence(Y) or Exit
sequence_vals:
  num_tests: 18
test_sequence:
- test_name: mtxadd-128x128-base
  number_of_generators: 0
  type_of_generators:
    baseline_program: stress-ng --matrix 1 --matrix-method add --matrix-size 128 --matrix-ops 250000 --aggressive
    loop_numbers: 50
- test_name: mtxadd-128x128-cache-core123
  number_of_generators: 3
  type_of_generators:
    - cache
    - cache
    - cache
  baseline_program: stress-ng --matrix 1 --matrix-method add --matrix-size 128 --matrix-ops 250000 --aggressive
  loop_numbers: 50
```

Figure 6: Automated YAML file parsing

Testing and Results

Data Analysis

- Victim execution data is stored in YAML format in Dom0
 - Contains performance metrics from each test loop (wall clock time, number of computations, etc.)
- Execution data is parsed to reveal averages, standard deviations, and worst-case execution times for a given run with varying levels of resource contention

```
metrics:  
  - stressor: stream  
    bogo-ops: 500  
    bogo-ops-per-second-usr-sys-time: 40.309607  
    bogo-ops-per-second-real-time: 40.043140  
    wall-clock-time: 12.486533  
    user-time: 12.372094  
    system-time: 0.031897  
    cpu-usage-per-instance: 99.338949  
    max-rss: 14328  
    mb-per-sec-memory-read-rate: 966.005623  
    mb-per-sec-memory-write-rate: 644.003749  
    mflop-per-sec-double-precision-compute-: 84.410859
```

Figure 7: Raw Test Output

```
WCET: [41.303706, 47.132261, 50.971415, 54.823704]  
Average: [40.871041, 46.141347, 49.662922, 53.80408]  
Standard Deviation: [0.05469, 0.196313, 0.389714, 0.269007]
```

Figure 8: Statistics Parser Output

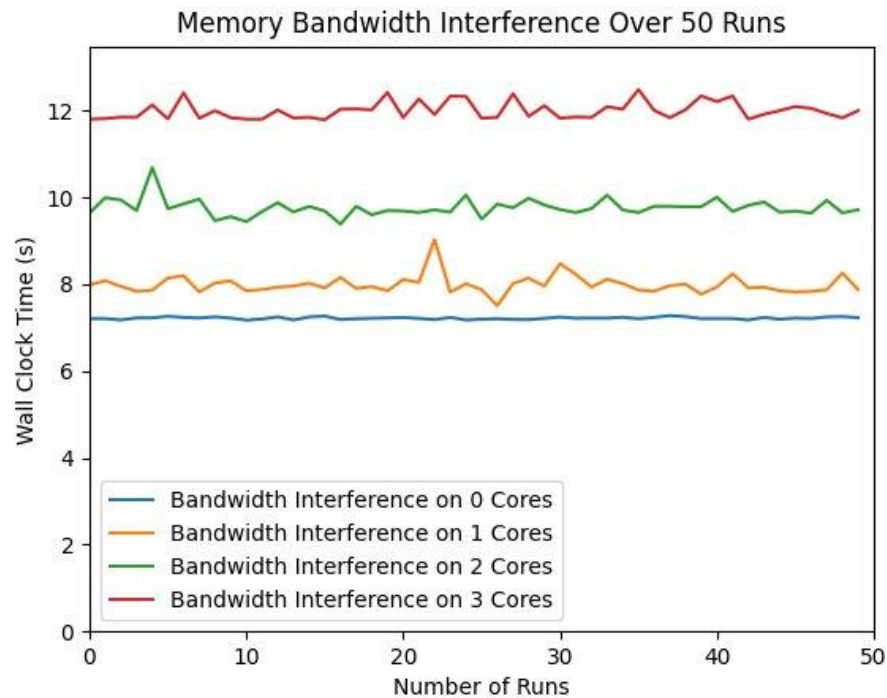


Figure 9: Parser graph output for a bandwidth-oriented victim when subjected to varying levels of memory bandwidth contention

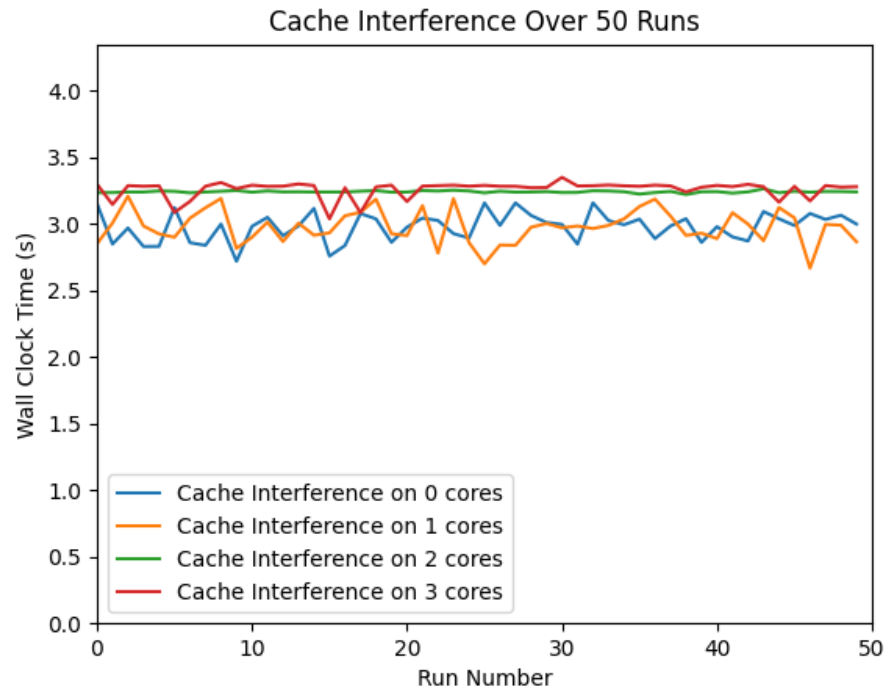


Figure 10: Parser graph output for a matrix add victim when subjected to varying levels of L2 cache contention

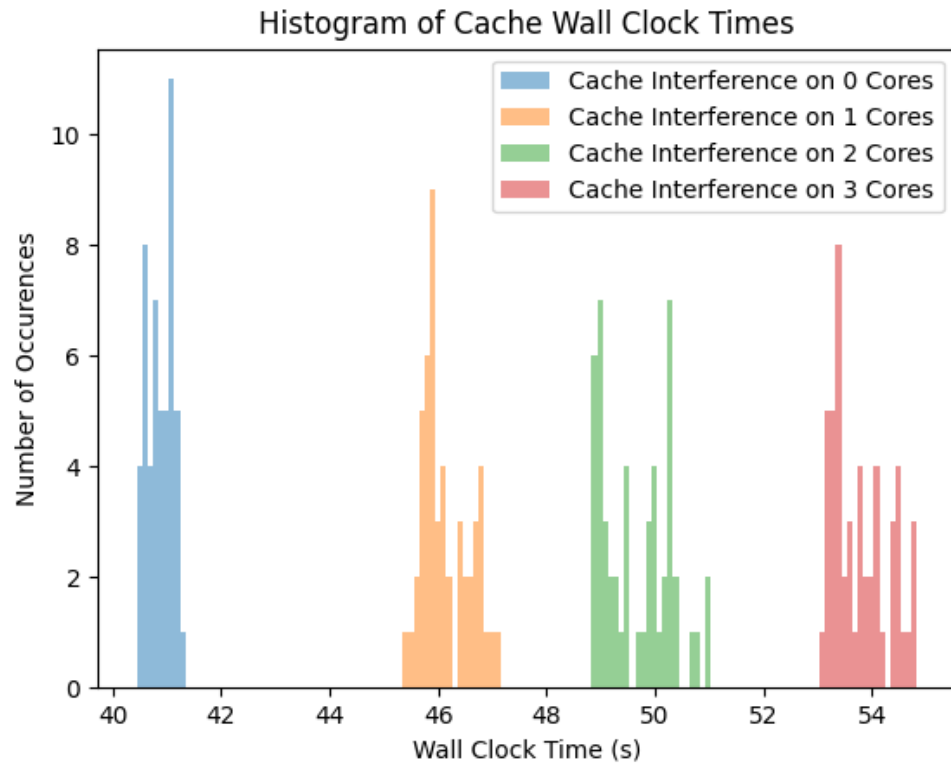


Figure 11: Histogram of the Cache Interference over runs with multiple cores

Conclusion

Current Project Status

- Able to generate system interference on one, two, or three cores
 - Uses Stress-NG for both interference generation and base test cases
- Provides a CLI (Command Line Interface) for direct serial terminal access on the board or access to the testing suite
 - Can run tests manually or automatically via parsing sequences from a YAML file
- Currently open source on GitHub per request of client
 - <https://github.com/2Manchu/MOAT>
 - Provides source code and tool documentation

Opportunities for Future Work

- Evaluate techniques to mitigate interference
- Create a graphical user interface
- Explore other forms of interference
 - Cache Coherency
 - Other forms of I/O

Questions?

Supplemental Material



Hardware Selection Matrix

- Selection process involved evaluating several different platforms for cost, features, and support for the tools necessary for our project

Raspberry Pi	<ul style="list-style-type: none"> Large amount of community support High volume of previous Xen on Pi repositories Easily accessible support/resource documents Popular Meets ARM requirements 	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> Long delivery time after purchase
Alinet ZUB-1CG	<ul style="list-style-type: none"> Cheaper option compared to other possible FPGA (Field Programmable Gate Array) boards Xilinx Ultrascale+ MPSoC has direct contributions to the Xen Hypervisor project Large volume of resources/support documents 	<ul style="list-style-type: none"> ZUB-1CG is not officially supported
Hikey 960	<ul style="list-style-type: none"> Meets ARM requirements 	<ul style="list-style-type: none"> Old hardware Unable to source from a reputable seller
Xilinx ZCU105	<ul style="list-style-type: none"> Large volume of documentation Direct Xen on Xilinx build information Meets Xen/ARM requirements 	<ul style="list-style-type: none"> Price (\$1700)

Users

- Avionics Engineers
 - Responsible for developing and validating avionics systems
 - Need a stress testing tool for their ARM-based hardware development platform
 - Allows for effective validation of their work as engineers
- Avionics Engineering Managers
 - Manage a team of Avionics Engineers
 - Provide evidence that the projects they are managing can be certified under military and civilian authority

Important Engineering Standards and Advisories

- FAA AC20-193
 - This advisory is concerned with the use and compliance of multi-core processors in avionics systems.
- CAST-32A
 - Position paper arguing on safety, performance, and integrity of airborne software operating on multicore systems.
- ARINC 653
 - Defines acceptable methods of resource partitioning on hardware running avionics programs

Task Responsibility and Contributions

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

Users

- Avionics Engineers
 - Responsible for developing and validating avionics systems
 - Need a stress testing tool for their ARM-based hardware development platform
 - Allows for effective validation of their work as engineers
- Avionics Engineering Managers
 - Manage a team of Avionics Engineers
 - Provide evidence that the projects they are managing can be certified under military and civilian authority