Co-Simulation of an Avionics Interface Device

DESIGN DOCUMENT

Team SDDEC21-02
Mathew Weber – Collins Aerospace
Dr. Phillip Jones
Spencer Davis
Matt Dwyer
Braedon Giblin
Cody Tomkins
Prince Tshombe
sddec21-02@iastate.edu
https://sddec21-02.sd.ece.iastate.edu

Created: 03-07-2021 / V1
Revised: 04-05-2021 / V2
Revised: 04-23-2021 / V3
Revised: 04-25-2021 / Final

Executive Summary

Development Standards & Practices Used

In this project, we utilize many practices related to open-source software. Our project hinges on us using open-source repositories, as well as expanding and contributing to these such projects as well. Our team also utilized AGILE-like development, where we utilized a KANBAN style task board to keep track of our ongoing and defined tasks.

Summary of Requirements

- Setup and execute a Cosim model using SystemC TLM backend and Xilinx QEMU processor simulator simultaneously
- Expand the Cosim capabilities by implementing bi-directional memory communication
- Model and test an off the shelf Linux driver for a memory-mapped peripheral

Applicable Courses from Iowa State University Curriculum

- CPR E 381
- CPR E 288
- CPR E 488
- CPR E 308

New Skills/Knowledge acquired that was not taught in courses

Our team acquired new knowledge of hardware simulation platforms, including SystemC TLM and Xilinx QEMU. Our team also explored Linux driver testing and gained insight into how the driver may interface with a memory-mapped peripheral.

Table of Contents

1 Introduction

1.1 Acknowledgement

We want to give our client Matthew Weber a big thanks for providing us with the technologies we need and for his technical help and patience throughout this project. We would also like to thank Dr. Phillip Jones for giving us technical advice and helping us solve problems we have had throughout this project. This project couldn't be done without them.

1.2 Problem and Project Statement

Problem: The Co-Simulation (Co-sim) environment using Xilinx Quick Emulator (QEMU) in conjunction with Xilinx SystemC TLM libraries lacks good technical demonstrations and documentation.

Solution: This project aims to create demos and simulations that can be documented and used as examples for future users of this software. Our team's primary demo will be building will feature an arbitrary Linux driver running in QEMU simulation, with a SystemC backend capable of communicating bidirectionally with the host PC. An application hosted on the host PC can then interact with the SystemC backend, driving the backing registers of the Linux device. We will complete this demo with a comparison between the co-simulation environment and a traditional QEMU simulation, explaining where benefits can be seen with Co-sim instead of current simulation strategies.

1.3 Operational Environment

The project will be created in a Linux environment. We are using an Ubuntu 18.3.4 server. This version of Linux was selected based on prior Co-sim demos, ensuring compatibility with all required tools. Our project will be simulating hardware, so we do not need any special hardware for this design.

Our primary concern with our environment is maintaining correct versioning on all submodules. We can manage this by forking each repository and keeping our own "ground truth" versions that we know will work. Tracking versions that will adequately work in our toolchain is also critical to include in our documentation.

Co-sim is an open-source project, so we will be working with the project community to ask questions and get feedback from the project creators. Working with an open-sourced codebase means we must abide by their standards for coding design and documentation.

1.4 Requirements

1. Identify an off-the-shelf Linux driver for an I²C IMU device
2. Bi-directional communication between the SystemC model and the host PC.
    a. Communication must allow multiple devices to be modified by the front end
    b. Must support read/writes to registers synchronized with QEMU accesses
    c. The interface should be configurable and scalable
3. Front end application to manage the host PCs connection with SystemC backend
4. Documentation and demonstration of design, as well as a robust comparison between Cosim and previous simulation interfaces

1.5 Intended Users and Uses

Our intended users are corporations looking to utilize Co-simulation for testing their products. For instance, the avionics community would be interested in these modeling chains as they can test software drivers before having novel avionics hardware designed and synthesized.

1.6 Assumptions and Limitations

We are assuming that:

- All source code will be published to an open-source repository
- We can freely use all SystemC libraries to model our communication of interfaces

Some of our limitations are:

- Some group members have little experience using a Linux based operating system and need to learn a lot of new material to be able to contribute
- The amount of current documentation of the system process is relatively limited.
- Our contributions and documentation will be constrained by what repository maintainers are interested in having in their projects.
- As of right now, the co-simulation programs do not utilize a convenient user interface, which may make testing our new code difficult.

1.7 Expected End Product and Deliverables

By the end of the first semester:

1. The group should have the foundation of a bi-directional SystemC interface implemented to send and receive data from the SystemC model.
    a. Prototyped implementation - **(IN-PROGRESS: due May 7th, 2021).**
    b. Full implementation with command-line frontend interface - **(PLANNED: due September 2nd, 2021).**
2. A Linux driver for an I²C IMU identified with a basic understanding of the expected backend behavior documented - **(COMPLETED: April 16th, 2021).**
3. Have a line of communication with open-source repository maintainers for demos we are working with - **(COMPLETED: April 15th, 2021).**
4. Documentation on initial Cosim demos generated and submitted to the puiblic Co-sim repository as a pull request **(IN-PROGRESS: due April 30th, 2021).**
5. I²C IMU sensor functional demo
    a. Proto-typed implementation - **(PLANNED: due September 23rd, 2021).**
    b. Graphical representation of IMU data - **(PLANNED: due October 7th, 2021).**

2 Project Plan

2.1 Task Decomposition

This project consists of multiple tasks. Below is a list of those overarching tasks and some of the intricacies involved in each:

1. Initial Cosim demo and environment setup
    a. Setup a shared computing environment for all members to use collaboratively.
    b. Work through the initial demo provided by the client to learn the ropes of the tools as hand
    c. Explore the technologies (SystemC, QEMU, TLM) and how they interact with one another in the simulated environment
2. Modifying the Demo
    a. Understanding how to modify the demo to add additional functionality or alter previous functionality of the timer register counter
    b. Implement the Threading demo provided by our client to augment the initial Co-Simulation demo further
    c. Better understand how all the software interacts and plan on how to add new demo features
    d. Document the process for running the demo for addition to the repository through pull-request submission
3. Reach out to public project maintainers about project direction
    a. What additions would be welcomed by the development teams utilizing the same tools?
    b. What other resources are available to aid in the contribution process?
    c. What areas are most in need of support and extension?
4. Implementing bi-directional communication of the host and the SystemC model
    a. Develop a protocol for modifying any SystemC device data via communications from the host
    b. Understand interactions between simulated hardware in SystemC, the simulated Linux software driver and OS in QEMU, and the input data from the host OS
    c. Implement a front end interface to be run on the host
    d. Demonstrate controllability of the simulation via the host communication interface
5. Identify and obtain a driver for an I^2C IMU and simulate the driver using Co-simulation augmented by host-controlled communications.
    a. Identify an open-source Linux driver for IMUs to use in a demo
    b. Identify a common IMU for simulation
6. Document the additional demonstration in detail
    a. Record all steps to reproduce results from a beginner to intermediate experience level
    b. Receive public feedback from the development community surrounding similar demos and the utilized tools.
    c. Publish a final draft that is accepted for publication
    d. Provide easy handles for other developers to extend the functionality of the demo and understand how to adapt it to their needs.

7. Repeat Process for similar ARINC 717 or UART in Linux Serial System
    a. Follow a similar design pattern utilizing the tools from previous parts to implement more complex standards and devices
    b. Explore additional sensors and testing use cases not thoroughly documented to build out additional testing capabilities
    c. Work off of additional client-side teams working in parallel on similar projects

2.2 Risks And Risk Management/Mitigation

The overall risk for this project is relatively low. The risk is low because it is entirely in software development and utilizing demos already freely provided online. The most significant risk factor foreseen is the poor reception and feedback of our contributions to the public projects. This could occur for several reasons, such as poor maintainer support, unaligned goals for the project's future, or already generated documentation and additions.

To mitigate this risk, we have worked with our client to develop another publication strategy if the primary public repositories do not favor our contributions. This would involve publishing our additional documentation and improvements on our own. Since each of the projects is open-source, meaning free to distribute and alter, there would be no licensing issues. While it would not be a part of the official documentation for the interacting projects, it would likely still contribute to the Co-Simulation development community as a whole.

In addition, when developing the low-level test drivers for the UDP communication protocol and other devices, our client has been generous in providing support from professionals in that area. As such, we will likely struggle at first to generate those low-level drivers, but with the help of the consultant provided, the majority of those risks should be mitigated.

Finally, documentation for the tools we are utilizing and developing are significantly scarcer than other public tools are due to their limited use. As such, part of our project goals is to better the documentation provided for developers wishing to utilize these tools for co-simulation. However, we may run into roadblocks ourselves when trying to use some of these undocumented tools.

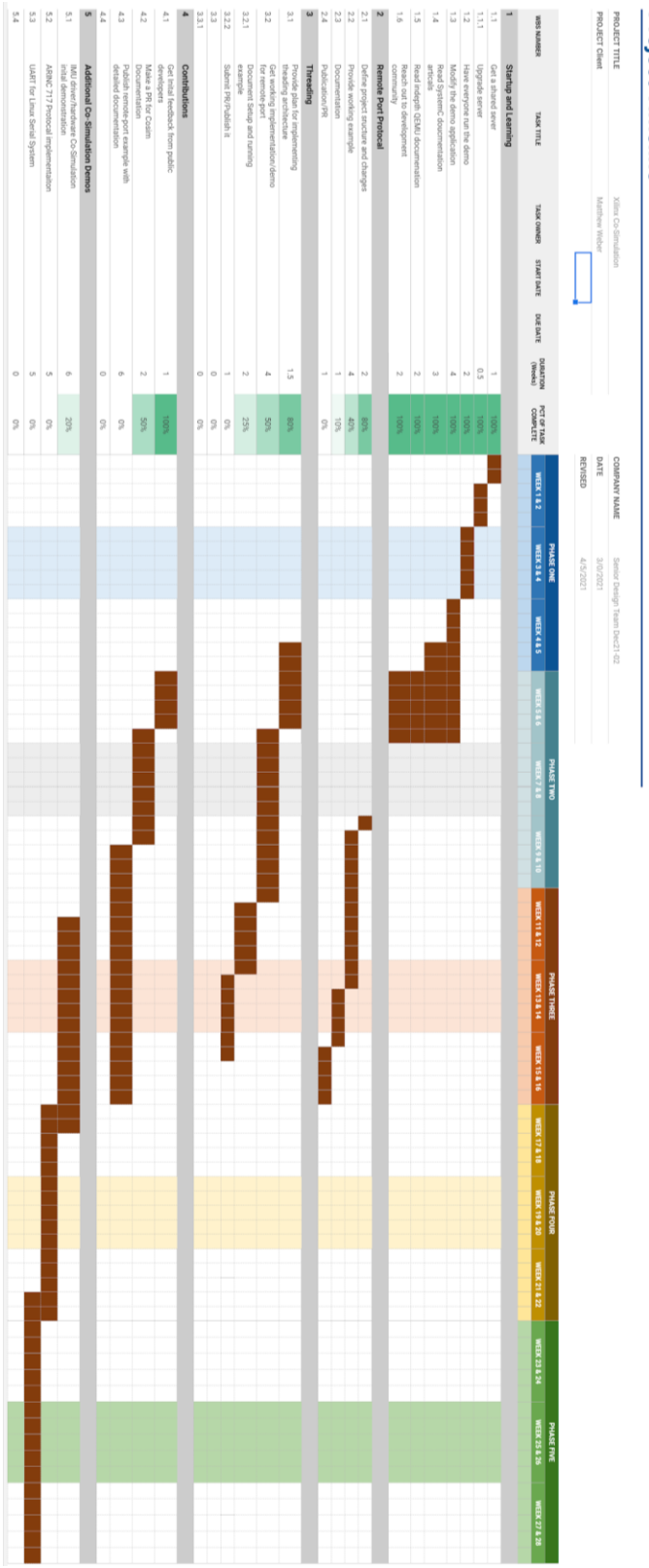
To mitigate the risk of unknown and undocumented tools, our team will be vigilant in documenting all tools and knowledge we gain along the way for our team and others. We will employ a fail-fast methodology of building prototypes and testing often. When we encounter poor results, we will be prepared to search for alternative solutions rather than waste time on a potentially poor solution. We also plan to be aggressive in outreach and support. We will seek guidance from our client and his team of experts, along with the development community of the tools at our disposal. With these resources, we foresee a quick turn-around time if we need to pivot to a new tool.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

1. Initial Cosim demo and environment setup
    a. Have everyone on the team complete the demo application

2. Modifying the Demo
    a. Make a pull request to the *Cosim-Demo* repo and get feedback
    b. Have every member of the team understand and change the memory-mapped register data to the Real-Time Clock (RTC)
    c. Make contact with the *Cosim-demo* repository managers to gain feedback on the additional features and documentation
    d. Publish centralized startup documentation for the *Cosim-demo* repository
    e. Create a list of milestones for implementing the threading framework described by the client.
3. Implement working remote-port communication capabilities for bi-directional communication between TLM and Linux environment
    a. Determine a candidate device for bi-directional memory-mapped simulation from a hardware and firmware side
    b. Create a comprehensive document for adding new simulated devices from a hardware (SystemC side) and software (Linux Buildroot in QEMU).
    c. Develop a test application to aid in understanding both sides of the Linux socket protocol and custom data packets in use
4. Public Contributions
    a. Make 3+ documentation contributions to public Xilinx Co-Simulation repositories
    b. Augment the initial demo application repo (*Cosim-demo*) to include additional bi-directional remote-port capabilities
5. Collaborate with and build off the work of parallel teams work during the summer at Collins
    a. Consult with additional team progress achieved over the summer in the Co-simulation documentation and domain
    b. Understand real internal use cases and need to develop additional demos for advanced communication protocols (ARINC 717, UART, Linux Serial System)

2.4 Project Timeline/Schedule



2.5 Project Tracking Procedures

As our project progresses, we have chosen to use a Trello board to track our progress. This allows us to break each task out and assign it to our team members to complete one at a time. It also allows us to see a better bird's eye view of our project to gauge our progress and current successes and other bottlenecks that might be occurring. We are also utilizing Discord to communicate and share documentation, links, and other comments. This allows us to work collaboratively in that space and provide a single communication channel and resources for the project. We also utilize a shared Gitlab group to house the repositories we need to submit changes to the relevant repositories we plan to contribute to. These would also include our internal development code and documentation that we generate in the process as well. These tools will make up the primary sources of communication and collaboration for our development team during this project. A shared Google Drive folder is used to maintain all administrative documents, diagrams, and presentations among team members. Finally, when contributing to public code repositories hosted on Github, Github pull requests. Our team uses their associated comment systems to solicit feedback and suggestions from the open-source community.

2.6 Personnel Effort Requirements

Task	Effort Estimate (Team Combined Hours)	Description	Reason
Environment setup	20	This involves getting a shared server setup, meetings times established, website updated, communication mediums, and other team dynamics. Along with familiarization with Linux and the tools being used.	Getting a server acquired and team dynamics should be trivial. Learning the tools and getting the basic demo up and running for each team member will take time as these tools are new to all team members.
Modifying The Demo	45	Making a change to the demo to demonstrate understanding and as a starting point to extending the demo with new functionality.	Modifying the demo should be a slightly difficult undertaking. Team members will still be learning the tools and exploring bugs and other errors that are encountered. In addition, once the demo is modified, it needs to be then

			replicated by all team members to ensure that all members understand the process and reasoning behind changes to contribute going forward.
Remote-Port Protocol	60	Implement a bi-directional remote-port protocol so that time-series data devices can be played back. Communication/data can be sent from both the hardware and software virtual interfaces using the TLM port of the QEMU simulation.	This protocol will allow for bidirectional communication between the simulated hardware and the software on the Xilinx ARM processor in QEMU. Ensuring the standard practices of communicating in TLM will be new for this task, and ensuring the communication is working bidirectionally will require significant testing and documentation.
Thread Protocol	20	Better comprehend the threading structure of SystemC for future use in the Co-Simulation server planned.	This technology will likely be used when developing the device communication server implemented in SystemC at the end of the first semester. It is essential to understand this feature and framework initially with the threading demo for custom implementation later on.
Additional Sensor Example	90	Implement an additional sensor as an example to further bolster the offering of the Cosim demo	This involves digging into Linux driver implementation and datasheets for memory-mapped

		repository. This allows a more accessible and broader range of opportunities for others to learn the development tools.	sensors. IT then also involves implementing said sensor in SystemC and the software side of the driver in QEMU and Buildroot for simulation. As no team members have experience in these skills, it will be a significant learning curve.
Public Cosim Demo Documentation	30	Make multiple public improvements to the little documentation provided for the Cosim demo repository and other tooling utilized during the project.	This involves communicating with members of the community, which can be slow at times. It is also essential to compose detailed documentation to help others solve issues and learn the tools. Documentation should not be rushed, although it should be trivial if the tools are known.
Colins Team Collaboration	300+ hours	We intend to meet with additional development teams at Collin's Aerospace to understand different use cases that would benefit from the Co-Simulation framework being developed by our team.	Over the summer, additional development teams at Collins will likely continue to develop other features on our public contributions. As such, during the Fall semester, we plan to meet with those teams to understand their advances and guide our future work around developing additional use cases and protocols that aid their internal needs. It is unclear what exact interfaces we will be asked to implement, although ARINC 717

			and UART for the Linux Serial System have been mentioned as likely goals.

2.7 Other Resource Requirements

For this project, a shared computing environment is needed for our team to develop the additions to the software described effectively. Since we are simulating complex processors in parallel and hardware devices attached to them, this requires a significant amount of computing resources. A powerful Linux server is needed to support these computing needs for our project. This is currently being provided by the Department of Electrical and Computer Engineering and guaranteed until completing our project.

A team communication platform for weekly meetings is also required to communicate with one another. We chose Discord as our preferred platform, as it is free and easy to use. This allows us to work remotely, hold meetings, and share information in real-time with one another when we cannot meet in person. Code repository and hosting services are again provided by the Department of Electrical and Computer Engineering. At the same time, other documentation and demos remain public or provided by our client and his team. Our project aims to utilize the current public resources and guidance to produce additional documentation, tools, and resources for other developers hoping to use the robust Co-Simulation framework provided by Xilinx with QEMU + SystemC-TLM structure. All deliverables aim to be open-sourced and accessible for anyone to use, and readily available to make sense in public development channels.

2.8 Financial Requirements

This project will, in all, require no financial requirements. The shared computing resources have been generously provided by the Department of Electrical and Computer Engineering, and all of the software needed is free and open-source. Any professional consultants provided are done so free of charge through the teams of the client's company (Collin's Aerospace). Any public documentation, tools, feedback, or advice is done so free of charge due to the nature of the open-source software community on the Github platform and the public nature of the Xilinx simulation toolchain.

3 Design

3.1 Previous Work And Literature

Various simulation technologies already exist for simulating both processor behavior and respective simulation environments individually/separately. However, the Co-sim model combines the two. Though this technology exists, there lacks sufficient documentation and demonstrations.

In essence, the Co-sim model as a toolchain is relatively new. Therefore, improving documentation and demos will be a large focus of this project to make the technology more approachable to prospective users.

The majority of project work will be focused on extending the usefulness of an already existing simulation environment. This means that background research is somewhat limited in its scope to learning about the technologies already being used by the system. The project group is currently focused on learning about those technologies.

Background literature for this project includes SystemC tutorials, a Xilinx emulator user guide, co-simulation documents, and any other work found on the open-source forums.

Literature:

Banerjee, Amal, and Balmiki Sur. "SystemC-AMS and SystemC Combinations." *SystemC and SystemC-AMS in Practice*, 2013, pp. 449–455., doi:10.1007/978-3-319-01147-9_17.

Ammari, Ahmed Chiheb, et al. "HW/SW Co-design for Dates Classification on Xilinx Zynq SoC." *2020 26th Conference of Open Innovations Association (FRUCT)*, 2020, doi:10.23919/fruct48808.2020.9087548.

Xilinx. "Xilinx Quick Emulator User Guide." 2019.

3.2 Design Thinking

Co-sim technologies exist yet are not well known within the target community. A relatively new tech, improving documentation and demos will make Cosimulation tech more approachable for the community and hopefully allow for increased usage of these technologies among target constituents.

Our initial design thought to improve Co-sim's documentation and demonstration capabilities was to create a set of highly general examples that would serve as demonstrations. However, we decided we could better serve the community by implementing a more specific, feature-rich example.

With this in mind, we settled on implementing an I^2C device that could be modified via both our QEMU simulated driver and a separate Host controlled remote port stream. This example is beneficial to the community because it represents a real-world use case where a Linux driver may need to be tested against a complex simulated set of hardware.

Finally, we can address much of the requirements of the project by improving documentation. Clear documentation is critical to a new user of the technology understanding how each of the project parts works in tandem.

3.3 Proposed Design

The primary goal of our project is to introduce the Co-simulation technology better and decrease the learning threshold required for end-users to use the technology in their development workflow. Our design begins with documentation. This satisfies the primary non-functional requirements of our project in making a new user understand the value of Co-sim while providing insight on how to initialize a Co-sim environment.

Our documentation will describe, in detail, each component of the simulation interface and how that component interacts with the model as a whole. This crucial layer of visibility will allow someone evaluating the technology insight into how their use case may fit into the model.

Our primary functional requirement of a novel demonstration application that an evaluator can run and experiment with will be implemented via an IMU setup. We selected an IMU because it is a complex subsystem device that may be commonly found in various applications, ranging from automotive, aviation, mobile, and more. We will demonstrate how our simulation model can take an off-the-shelf Linux driver, run it on our system via a host application while having the entire model system be functionally indistinguishable from a real system with an IMU present despite the entirety of the IMU being modeled.

Furthermore, our design will control the backing IMU via a frontend interface that runs on the Host PC. The frontend interface will interact with the SystemC model via a remote port. The Remote Port connection, defined in the Xilinx SoC libraries, will enable our frontend application to directly interact with the SystemC model to modify state machine values, such as sensor readings, on the fly.

Our design will allow an evaluator to quickly view a demonstration of a Linux driver operating seamlessly with a modeled IMU, with the IMU accurately reflecting changing sensor values over time as driven by a host application. This demo will provide a company insight into the value of a robust simulated hardware model and how driver debugging can be done without hardware ever having been developed.

3.4 Technology Considerations

Most technology decisions regarding this project have already been made due to the nature of the proposal by our client. This is because the specific use case and environment have already been described. Our primary goal is to document the toolchain better and develop additional channels between the host and the SystemC model to aid the development Co-Simulation process using existing infrastructure. One strength the co-simulation model offers is increased flexibility compared to a “real world + simulation” model. This removed the need for access to physical hardware devices since all hardware and software are simulated in the software technologies.

For this project, we are using QEMU for the ARM processor simulation. We are using device description files for a Zynq 7000 System. However, we are not constrained by the particular board we are targeting and intend for our work to be general enough so that anyone can replicate it targeting a different board. We are modeling the Programmable Logic (PL) section of our FPGA using SystemC. This design decision was made for us by previous Co-simulation work.

Ubuntu Linux was chosen as our host OS as all of our tools readily support it. Running our software environment on a Ubuntu platform allows us to get support from ETG and easily share our team members' environment.

Finally, we chose Buildroot to generate our embedded Linux image run on our simulated processor. We selected Buildroot because it is straightforward to get an image built and running, and it is widely supported. Other possible options would have been Xilinx PetaLinux and Yocto; however, as the Xilinx repository maintainers offered no objection to us not using Xilinx PetaLinux, we elected to go with the simplest solution.

3.5 Design Analysis

We analyzed our design on two merits: how valuable our contributions to this technology will be for future development teams interested in using Co-simulation and how successful our implementation will be. At the moment, we are highly confident that our design will fulfill its purpose of providing new support materials for evaluators. Our strategy has been evaluated and approved by our client, who shares our optimism about our design.

Our current progress on our design has made our team confident that it is achievable in the specified time frame. We have implemented the basis of a remote port into the SystemC model, and we have begun work on integrating the IMU into our model. Combining each of these components will be the topic of our next semester.

3.6 Development Process

Though this project doesn't fit into any "specific" development process, it most closely resembles the Agile approach because it is being completed through small, iterative progress chunks with frequent feedback and demonstrations. Our group has chosen to use Trello to track progress.

This development process was selected because it allows for high client involvement and is readily applicable to the system when other development processes would make less sense logically.

3.7 Design Plan

The project focus for this semester will mainly center around improving demos and documentation to improve the approachability of Co-sim technology. Next semester, the project focus will shift to adding increased functionality to the system by extending our interface and implementing front-facing interaction methods.

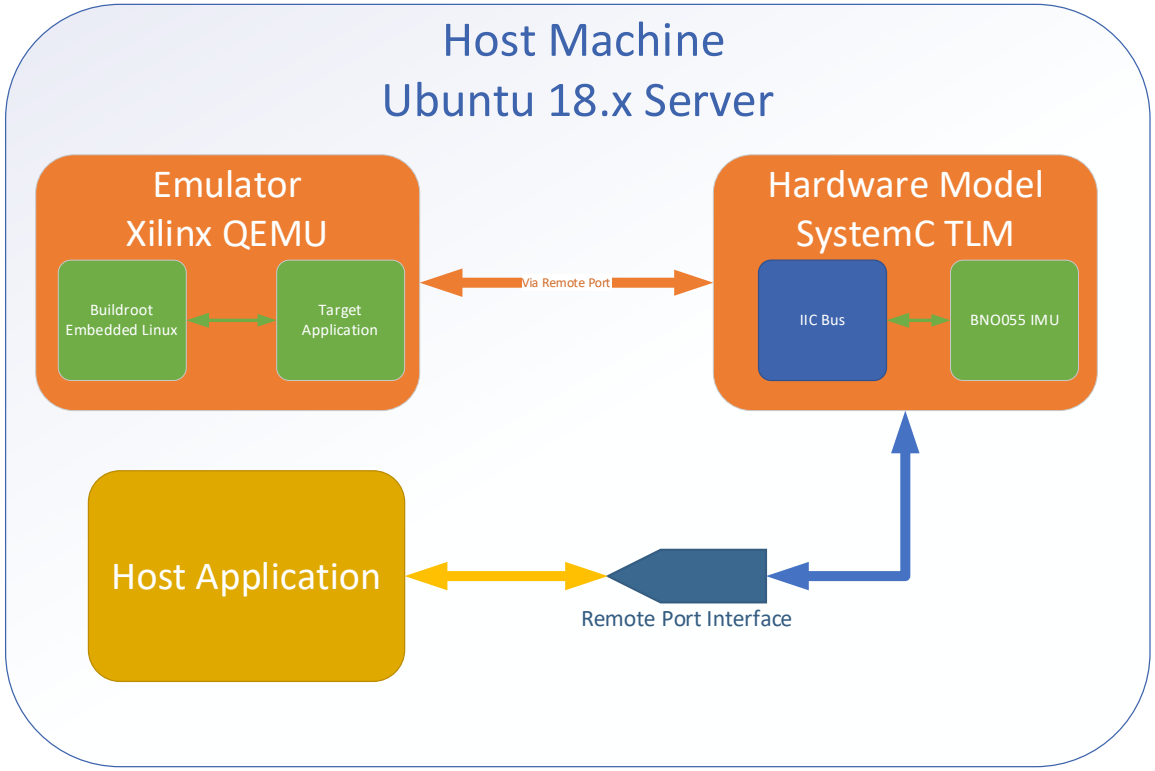


Figure 1: Design Diagram

Figure 1 above shows a high-level implementation of our design. The Host machine runs a QEMU simulation of our ARM processor on the Xilinx Zynq-7000 SoC with a custom compiled version of Buildroot. A Xilinx remote-port connects this instance to the SystemC TLM instance containing our I²C bus to the BNO055 IMU emulated device run atop our SystemC server device. This communicates by means of another Xilinx remote port to the host application used to control our SystemC server from outside our simulated environment.

4 Testing

4.1 Unit Testing

We will test each demo component on its own to the fullest extent possible. The SystemC TLM models for each demo can also be tested individually before attaching the appropriate Linux driver. We can test our hardware implementations via the "devmem" command on Linux. As our project mainly consists of implementing minor contributions to existing infrastructure, most of our work will tightly interface with surrounding components and won't be testable in isolation.

4.2 Interface Testing

The primary interface we have to worry about is the connection between QEMU and our SystemC model. As the nature and specifics of this connection are outside of the scope of our project, we don't need to test it substantially. However, we need to ensure that our SystemC work is accurately being translated into the QEMU environment using this connection. Once we build our UDP

interface, we will need to ensure that data is transferred correctly between the host OS and our SystemC model. Specifically, we need to make sure read and write calls do what they intend to do and that no data is lost when data transfer occurs.

4.3 Acceptance Testing

As part of our acceptance protocols and tests, our primary metrics are feedback from the development community about best practices and community contribution guidelines. As many of the contributions we are making are of documentation in nature, this feedback is vital to ensuring we are adhering to common standards and meeting the basic requirements of the projects to which we contribute. Along with the community input, we go through multiple revisions to our work and continually test each procedure we publish. This is done by a team member following the created documentation step-by-step to replicate the results described and comparing any pain points or needed improvements. Our project will also be contributed to by summer interns at Collins Aerospace under our client in the summer of 2021. As such, they will primarily be using our documentation and work as a starting point for understanding the technology infrastructure and setting up basic development environments. Using our documentation, we will receive additional results and feedback from actual beginner developers to refine the publications further.

We plan to continue gaining feedback from the development community and our client at every step for final project acceptance. We have designed many demonstrations of our project examples, including basic documentation and an initial demo, a proof of concept minimum viable product (MVP) demonstration for our first semester review, a graphical inertial measurement unit (IMU) demonstration with complete Linux kernel integration, and test application, and an ARINC 429 implementation demonstration utilizing our tools. We plan to create full functionality coverage testing cases to demonstrate in the demonstration videos and supplementary documentation in creating these demos. By doing all of this, we hope to cover the majority of use cases that a new or experienced FPGA/embedded developer might encounter or desire to implement.

We also hope that other developers will find our initial offering of tools during the summer and provide additional feedback. By having extra time for our repository to be discovered and implemented, we are only increasing our feedback and improvement opportunities. We expect that we will continue to receive positive feedback on the contributions we have published and plan on publishing as we continue to progress. By maintaining close ties with the developers of these tools and carefully applying their suggestions, we expect to continue to be welcomed in our additions to the development tools of QEMU + SystemC-TLM co-simulation.

4.4 Results

Our team currently has completed several facets of our project. We have obtained a Linux driver for our IMU and have integrated it into our Linux system. We have verified via “lsmod” that our driver is successfully loaded as a Kernel module. Currently, our team is evaluating our remote port implementation. We have begun implementing a basic proof of concept that should provide us a foundation to start testing control of SystemC state machines externally to SystemC.

We have evaluated our documentation by presenting it to our client as well as open-sourced repository maintainers. The maintainers have provided us some advice on modifying our documentation to fit with the repository expectations. We can validate our documentation by receiving approval from repository maintainers to merge our work.

Our following tasks that still require implementation and testing are modeling an I²C Bus in SystemC. This task will require the functionality of our remote port so that we can drive devices on the bus. Once we have this bus implemented, we can begin testing the functionality of our hardware model using the Linux Driver we loaded onto our QEMU system. We will test the I²C bus by executing a sample driver program on our emulator. This program, and the accompanying backend driver, are known as functional on real hardware. If these test programs properly execute on our toolchain, we will know that the entirety of our implementation is functioning.

5 Implementation

Each section of our deliverables is neatly divided into demos. For instance, our client would like us to demo a working out-of-the-box Linux driver running on our Co-sim platform. So, our team will start our implementation with a focus on our Client Demos. Our team will lay out all of the requirements for each demo and then implement them iteratively to add the needed functionality.

To begin implementing each demo, we will start by researching the existing material already in the project. Our demos seek to build upon open-source repositories, and these projects already have significant amounts of implementation and publicly available correspondence relating to potential tasks.

Our team will utilize all of the available resources associated with each project to decide how the feature will fit into the demo. Next, we will reach out to the maintainers to get feedback on our proposed changes. Understanding how the repository maintainers want our changes to fit into the existing demo is key to us properly implementing the change.

From this point, we can go ahead and make our changes. Much of our changes will be in the SystemC demo code that is written to add additional functionality to the QEMU memory interface. We can then test our differences on the demo using basic Linux memory access commands to verify that we are adjusting memory as required.

Finally, these open-sourced projects require a high amount of documentation. We are seeking to expand the documentation already present and generate new documentation so that these repositories can be more accessible. So, our future implementation will need to involve growing the quantity and quality of documentation for each demo and then publishing these documents appropriately so that the community can use them.

6 Closing Material

6.1 Conclusion

So far, our team has demonstrated proficiency in executing a Co-sim demonstration. We have begun learning more about the toolchain and will soon start experimenting with alternate SystemC models driving our QEMU memory model. This will naturally progress us into our first significant deliverable: demonstrating a working Linux Peripheral driver executing Co-sim.

Our next tasks include building our understanding of a SystemC remote port. We have set up a model that utilizes the remote port already. However, we lack sufficient knowledge of the interface to implement our design currently fully. Our team has also made significant headway on documentation, and we plan on adding more and expounding on our current documentation as we progress.

In the future, our team will continue to adopt an iterative approach in adding functionality to the Cosim system. Our team will also be reaching out to open source projects and identifying areas to expand the existing offering. This expansion will be done in the form of new demos, documentation, or additional features.

6.2 References

J. Komlodi and V. Garhwal, "Co-simulation," *Confluence*. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/862421112/Co-simulation. [Accessed: 09-Mar-2021].

Xilinx, "systemctlm-cosim-demo," *GitHub*. [Online]. Available: https://github.com/Xilinx/systemctlm-cosim-demo. [Accessed: 09-Mar-2021].

6.3 Appendices

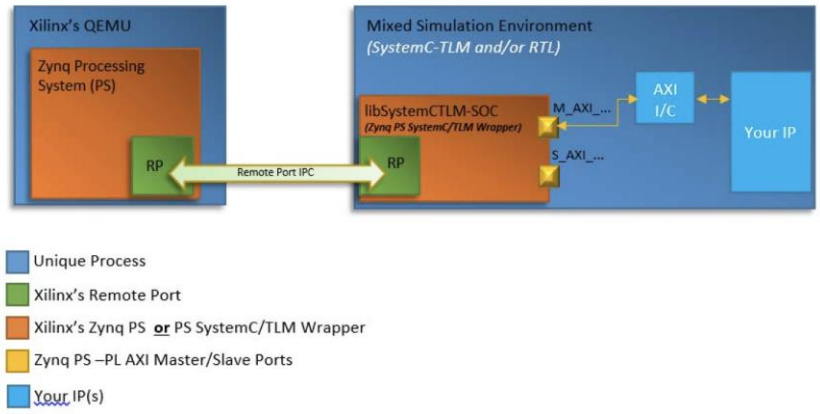


Figure 2 - Xilinx QEMU Mixed Simulation Environment

Xilinx. "Xilinx Quick Emulator User Guide." 2019, p. 37.