

Co-Simulation of an Avionics Device

SDDEC21-02: Matt Dwyer, Braedon Giblin, Cody Tomkins, Spencer Davis, & Prince Tshombe

Faculty Advisor: Dr. Phillip Jones
Client: Matthew Weber (Collin's Aerospace)
Website: <https://sddec21-02.sd.ece.iasfate.edu/>

Background: QEMU



- Processor emulator that allows execution of programs for specified board/processor
- Allows us to simulate a and ARM Cortex A9 for a Zynq-7000 FPGA SoC platform
- Can run embedded Linux or bare metal programs
 - We are using Buildroot to compile and assemble an Embedded Linux Boot Image and File System

```
[test@donizetti ~]$ qemu-arm ./ls --color /
bin  etc  lib64  mnt  root  srv          system-upgrade-root  var
boot home lost+found opt  run  sys          tmp
dev  lib  media  proc sbin system-upgrade  usr
[test@donizetti ~]$ uname -a
Linux donizetti 4.6.7-300.fc24.x86_64 #1 SMP Wed Aug 17 18:48:43 UTC 2016 x86_64
x86_64 x86_64 GNU/Linux
[test@donizetti ~]$ file ./ls
./ls: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked
, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.0.0, stripped
[test@donizetti ~]$ █
```

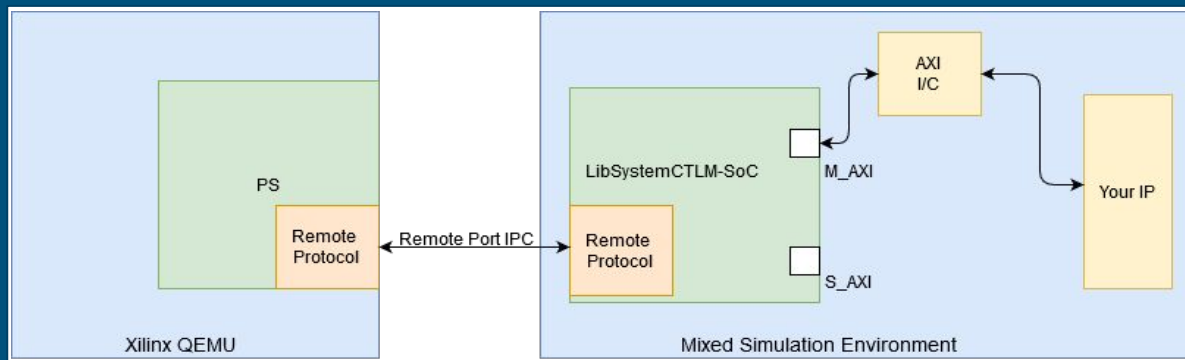
Background: SystemC-TLM

- Hardware modeling language that uses plain C/C++ syntax to model hardware systems
- Allows complete simulation of Memory Mapped hardware, AXI systems, and more
- *Libsystemctlm-soc*: Xilinx SystemC library that provides an interface between a SystemC modeling environment and a Zynq-7000, Zynq Ultrascale+, and Versal ACAP computation platforms



Hardware/Software Co-Simulation

- Simulate the processor your code is running on (Embedded ARM Cortex-A9)
 - Processor emulator (QEMU)
 - Buildroot Linux
- Simulate the hardware interactions and mock all calls made
 - Hardware implementation (SystemC)
 - Hardware transaction modeling (TLM)
- Connect the two simulated environments (FPGA PS-PL connection)
 - Xilinx Remote Port



Problem Statement

- Steep learning curve for beginners
 - Few documented example projects
 - Lacking basic documentation
- Desire for additional flexibility
 - Once the simulation has been setup, difficult to manipulate data “Generated” by simulated hardware
 - Desire to “feed” data into the system from an external source
 - Processing System (QEMU) being none the wiser, assumes it is a real device

How to set up and run the Co-Simulation Demo

This demonstration shows how to compile and run the Co-Simulation demo of Buildroot in QEMU with a simulated device in SystemC. This configuration is tested working for Ubuntu 18.0.4 and assumes that a `cosim` directory is created in your home directory. This walkthrough also assumes that the device being emulated by QEMU is the Xilinx Zynq-7000 SoC. This SoC seemed like a good candidate but the concept can apply to any QEMU machine which plugs in a compatible remoteport bus interface.

Dependencies

Below are the dependencies needed to compile all the libraries in this demo:

```
sudo apt update
sudo apt install cmake gmake gcc qemu-kvm qemu-system qemu-user-static verilator
```

Setup and Compilation

Run these commands to clone and build the necessary repos (`~/cosim` assumed as the base directory).

Create the base directory

```
mkdir ~/cosim
```

SystemC Setup

```
cd ~/cosim
SYSC_VERSION=systemc-2.3.2
wget https://www.accelera.org/images/downloads/standards/systemc/systemc-2.3.2.tar.gz
tar xf ${SYSC_VERSION}.tar.gz && cd ${SYSC_VERSION}/
```

Intended Users

- Corporations who simultaneously develop hardware/software solutions
 - Aerospace, Defense, Industrial Automation, Automotive
- Users looking to extensively test hardware and software independently of one another
- People interested in applying Co-Simulation to their own project who are stymied by the barrier of entry

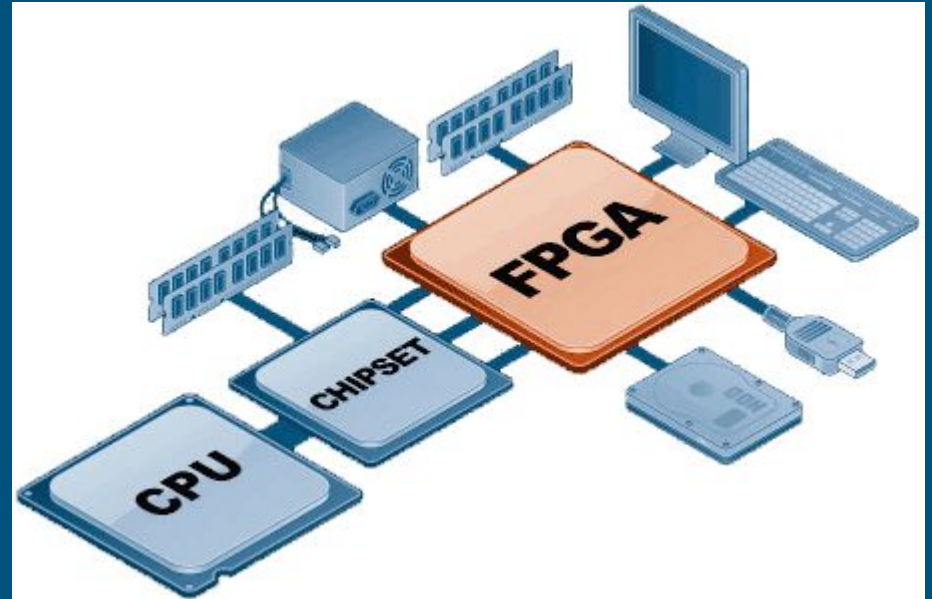


JOHN DEERE



Use Case Example

- You are a software engineer developing a hardware driver to control a new temperature sensor
 - Interfacing with it over an I²C interface
 - You have the technical documentation
 - You have v0.1 of the driver written
 - Now What?
- How can you test your driver code in a simulated environment before a engineering sample of the sensor has been produced?



Functional Requirements/Deliverables

- Documentation
 - Document an initial environment setup walkthrough → PR has been submitted, being revised
 - Create an additional demo to for a more complex system → PPM demo working, publishing soon
- External Data Source/Modeling Tool
 - Model an I²C Bus in SystemC and corresponding test application → Initial research completed
 - Drive a simulated IMU device over I²C with static data → Initial research completed
 - Develop Remote port custom communication tunnel for external data source tool → Test application with decoded packets working
 - Demonstrate an off-the-shelf Linux IMU driver running on QEMU, working with modeled hardware → IMU and driver selected, test application in development

Non-Functional Requirements

- Author supplemental documentation for Xilinx technologies utilized
- Contribute all to centralized (Xilinx Confluence or otherwise) documentation body
- Documentation describing in full our I²C/Remote Port implementation resources utilized
- Implement additional examples with other protocols in our custom SystemC device server
- Support multiple device simulation simultaneously in the SystemC device server

How to set up and run the Co-Simulation Demo

This demonstration shows how to compile and run the Co-Simulation demo of Buildroot in QEMU with a simulated device in SystemC. This configuration is tested working for Ubuntu 18.0.4 and assumes that a `cosim` directory is created in your home directory. This walkthrough also assumes that the device being emulated by QEMU is the Xilinx Zynq-7000 SoC. This SoC seemed like a good candidate but the concept can apply to any QEMU machine which plugs in a compatible remoteport bus interface.

Dependencies

Below are the dependencies needed to compile all the libraries in this demo:

```
sudo apt update
sudo apt install cmake gmake gcc qemu-kvm qemu-system qemu-user-static verilator
```

Setup and Compilation

Run these commands to clone and build the necessary repos (`~/cosim` assumed as the base directory).

Create the base directory

```
mkdir ~/cosim
```

SystemC Setup

Hardware/Software/Technology Platforms used

QEMU: Machine emulator. In this project, simulates a Zynq-7000 processor.

SystemC: C++ classes used for event driven simulation, used in tandem with QEMU to provide co-simulation.

LibRemotePort: Socket-work in C++ that allows systems to connect over Unit sockets, used for Host system → SystemC communication

Server: Ubuntu 18.0.4 Server

Project Management: Github - Version Control, Trello - Task management, and Discord - Collaboration

Co-Sim System Clock Demo

```
bgiblin@sddec21-02:~/cosim/systemctlm-cosim-demo$ LD_LIBRARY_PATH=~/cosim/systemc-2.3.2/src/.libs/ ./zynq_demo unix:${HOME}/Cosim/buildroot/handles/qemu-rport-_cosim@0 1000000
```

```
SystemC 2.3.2-Accellera --- Mar 11 2021 21:24:50  
Copyright (c) 1996-2017 by all Contributors,  
ALL RIGHTS RESERVED
```

```
open socket  
connect to /home/bgiblin/cosim/buildroot/handles/qemu-rport-_cosim@0
```

```
Info: (I702) default timescale unit used for tracing: 1 ps (trace.vcd)
```

```
0 bash
```

```
1 bash
```

PPM State Machine Demo

```
void PPM_Receiver::gen_ppm()
{
    const int FRAME_TIME = 20000; // 20000 us
    const int CH_NUM = 6; // just use one channel, 0 to 5
    const int frame[] = {1500, 800, 1000, 1250, 2000, 1750};
    const sc_time rx_period(FRAME_TIME, SC_US);

    SC_REPORT_INFO("PPM_IN", "Beginning State machine\n");

    while (reg_ctrl == 0x0); // wait for enable signal

    // endless loop
    while (true) {

        // capture simulation time at the start of the transmission period loop
        sc_time rx_period_start = sc_time_stamp();

        for (int i=0; i < CH_NUM; i++) {
            reg_ch_n[i] = frame[i];
            wait(sc_time(frame[i], SC_US));
        }
        reg_count++;

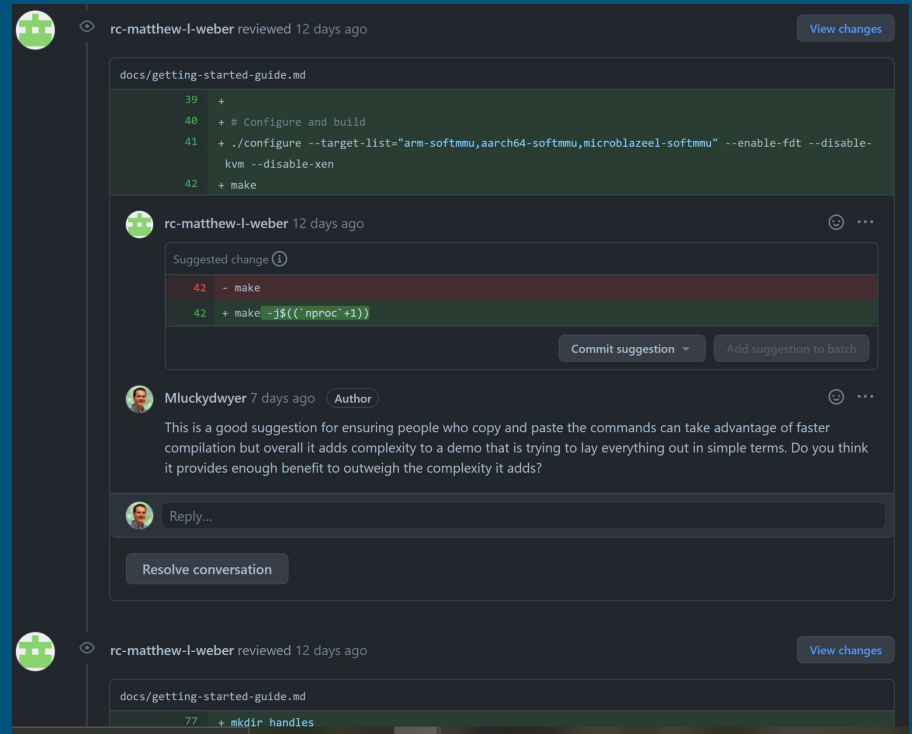
        sc_time rx_period_consumed = sc_time_stamp() - rx_period_start;

        if (rx_period_consumed > rx_period)
            SC_REPORT_WARNING("PPM", "Channels to Long!\n");
        // otherwise, wait out the remaining part of the period
        else
            wait(rx_period - rx_period_consumed);
    }
}
```

```
en=4
Channel 0: 0x000005DC
Channel 1: 0x00000320
Channel 2: 0x000003E8
Channel 3: 0x000004E2
Channel 4: 0x000007D0
Channel 5: 0x000006D6
Info: PPM_IN: Reading Count
Channel 0: 0x000005DC
Channel 1: 0x00000320
Channel 2: 0x000003E8
Channel 3: 0x000004E2
Channel 4: 0x000007D0
Channel 5: 0x000006D6
Info: PPM_IN: Reading Count
Channel 0: 0x000005DC
Channel 1: 0x00000320
Channel 2: 0x000003E8
Channel 3: 0x000004E2
Channel 4: 0x000007D0
Channel 5: 0x000006D6
Info: PPM_IN: Reading Count
```

Detailed Design - Documentation

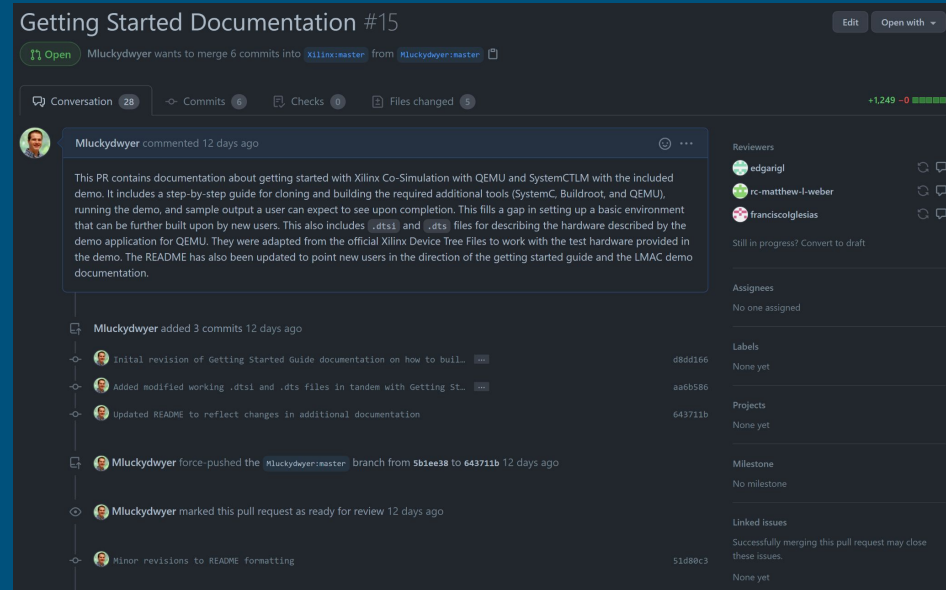
- Generate Documentation that describes how to setup and build existing Demos
- Explain how to modify existing Demos to extend the hardware/software capability
- In depth documentation on new functionality that describes use cases, setup, and modification
- Receive feedback from development community and iterate



The screenshot displays a GitHub pull request interface. At the top, a review by 'rc-matthew-l-weber' is shown for the file 'docs/getting-started-guide.md'. The code diff shows lines 39-42 with additions: line 39 has a plus sign, line 40 has '# Configure and build', line 41 has './configure --target-list="arm-softmmu,aarch64-softmmu,microblazeel-softmmu" --enable-fdt --disable-kvm --disable-xen', and line 42 has '+ make'. Below the diff, a 'Suggested change' by 'rc-matthew-l-weber' is shown, where line 42 is changed from '- make' to '+ make -j\${('nproc'+1)}'. A comment by 'Mluckydwyer' (Author) follows, discussing the suggestion: 'This is a good suggestion for ensuring people who copy and paste the commands can take advantage of faster compilation but overall it adds complexity to a demo that is trying to lay everything out in simple terms. Do you think it provides enough benefit to outweigh the complexity it adds?'. A 'Reply...' input field and a 'Resolve conversation' button are visible. At the bottom, another review by 'rc-matthew-l-weber' is shown for the same file, with line 77 having '+ mkdir handles' added.

Documentation Status

- We have reached out to Xilinx Co-Sim Repository Maintainers and have established communication
- Submitted Pull Request for a running a “getting started demo”
- Have been iterating over the pull request to complete suggestions made my maintainers
- Contributes towards our fulfillment of improved end-user documentation deliverable



The screenshot shows a GitHub Pull Request (PR) titled "Getting Started Documentation #15". The PR is open and ready for review. The author is Mluckydwyer, who wants to merge 6 commits into the xilinx:master branch from the Mluckydwyer:master branch. The PR description states: "This PR contains documentation about getting started with Xilinx Co-Simulation with QEMU and SystemCTLM with the included demo. It includes a step-by-step guide for cloning and building the required additional tools (SystemC, Bulldroot, and QEMU), running the demo, and sample output a user can expect to see upon completion. This fills a gap in setting up a basic environment that can be further built upon by new users. This also includes .dtsi and .dts files for describing the hardware described by the demo application for QEMU. They were adapted from the official Xilinx Device Tree Files to work with the test hardware provided in the demo. The README has also been updated to point new users in the direction of the getting started guide and the LMAC demo documentation."

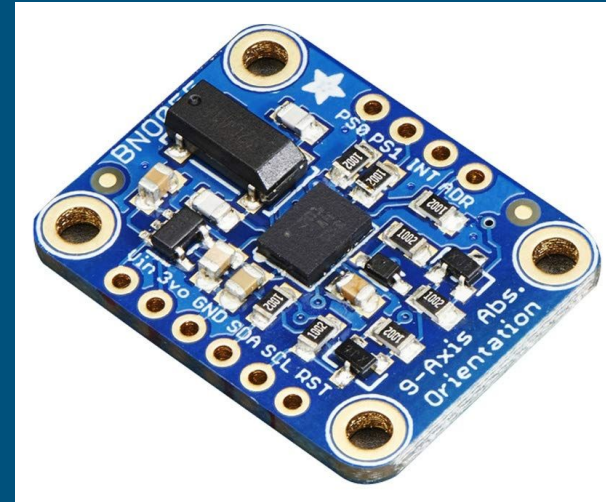
The PR history shows the following commits:

- Mluckydwyer added 3 commits 12 days ago
 - Initial revision of Getting Started Guide documentation on how to buil... (d86d166)
 - Added modified working .dtsi and .dts files in tandem with Getting St... (aa6b586)
 - Updated README to reflect changes in additional documentation (643711b)
- Mluckydwyer force-pushed the Mluckydwyer:master branch from 5b1ee38 to 643711b 12 days ago
- Mluckydwyer marked this pull request as ready for review 12 days ago
- Minor revisions to README formatting (51288c3)

The PR is currently in the "Ready for review" state. The reviewers listed are edgarigl, rc-matthew-l-weber, and franciscoglesias. There are no assignees, labels, projects, or milestones assigned to this PR. A note indicates that successfully merging this PR may close some linked issues.

Detailed Design - I²C IMU Implementation

- Write a state machine in SystemC to model an I²C bus
- Identify a real IMU with a Linux driver that we will use as our test device
- Model backend value registers of IMU using SystemC + remote interconnect to host to allow host to control IMU data values
- Demonstrate and document a working mixed simulation consisting of
 - QEMU, running Buildroot Embedded Linux with IMU Kernel Module and front end test application
 - SystemC modeling an I²C bus with one or more devices attached
 - Host system modifying SystemC parameters on the fly to vary IMU data



I²C IMU Implementation Status: SystemC

- Built and tested state machined consisting of a Pulse Position Modulation encoder / decoder
- Executed a bare metal driver & runner program developed independently as a course project, demonstrating Mixed Simulation functioning with no software changes needed
- Remote Port Protocol Identification and Research
 - Identified Xilinx SystemC SoC Remote Port library as a candidate for SystemC to Host communications
 - Have established a one directional handshake between SystemC and host program, verifying read/write connectivity between the two

Test Plan

- Documentation will be tested by ensuring it conforms to all standards set forth by repository maintainers
- Documentation will also be tested by Collins Intern teams that attempt to follow our demos and work on tangential projects
- All commits will pass code review by repository maintainers, ensuring that we meet the standards outlined by each repository

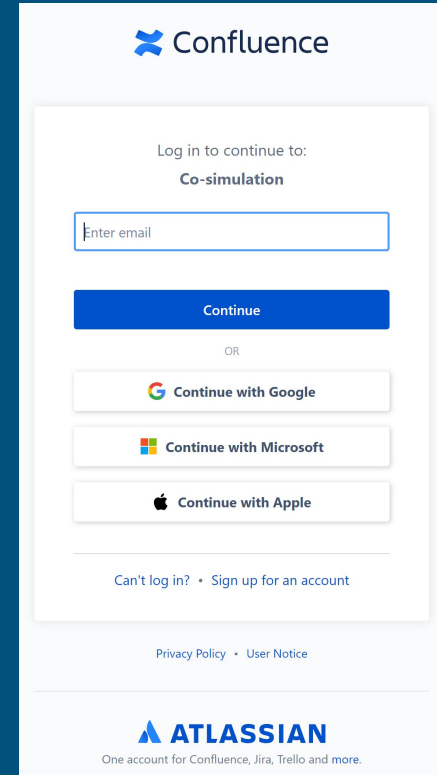
Test Plan

- Each module will be unit tested to the fullest extent possible
- Integration testing will be conducted in stages with known working system states
- Each of the modules, along with the full demonstration of our working system, will be documented to ensure quality of testing.



Constraints & Considerations

- Open-source community sentiment to our proposed additions
 - Welcoming and positive from our numerous interactions
 - Generous suggestions and critiques
- Unfamiliarity with simulating technologies
 - Internal team documentation for basic demo progressions
 - Team requirement to complete basic demos
- Minimal documentation for Xilinx implementations
 - Reaching out to experts in development community
 - Reverse engineering implementation source
 - Evaluating responses from compliant implementations



The image shows a screenshot of the Confluence login interface. At the top, the Confluence logo is displayed. Below it, the text 'Log in to continue to:' is followed by 'Co-simulation'. There is an input field for 'Enter email' and a blue 'Continue' button. Below the button, the word 'OR' is centered. There are three social login options: 'Continue with Google', 'Continue with Microsoft', and 'Continue with Apple'. At the bottom of the login area, there are links for 'Can't log in?' and 'Sign up for an account'. Below the login area, there are links for 'Privacy Policy' and 'User Notice'. At the very bottom, the Atlassian logo is shown with the text 'One account for Confluence, Jira, Trello and more.'

Future Milestones

- Further document internal tooling and technologies used
- Develop external Python/C++ tool for external data sourcing
 - Document multiple use cases
 - Document implementation for future development
- Implement additional sensors/protocols in SystemC device server
- IMU graphical interface demonstration

Thank You For Listening!



Questions?

Current Project Status

Completed:

- Outreach to open-source community was successful
- Initial Documentation for a co-simulation demo in a pull request to main repository
- Application involving LibRemotePort configuration is working (?)

Our work may be given to interns at Collin's Aerospace to use/work on over the summer.

Prototype Implementations

- Implemented basic remote port application
 - Still reverse engineering protocol
 - Documentation and consistent operation still in progress
- PPM demo working as intended
 - Documentation needed
 - Cleanup and publication needed
- Documentation
 - Basic walkthrough 3rd revision completed
 - Needs to be pushed for additional feedback
- IMU Driver
 - Still being compiled into the Buildroot kernel

Problem Statement

The existing co-simulation environment provided by Xilinx, which utilizes a SystemC TLM and QEMU, lacks sufficient documentation for a newer user to learn and use it. It also has opportunities for an expanded interface to allow more robust testing

Task Contributions of Each Member (Each person fill out)

Matt: Open-Source Community contact, demo documentation, remote-port/communication protocol architecture, administrative documentation

Braedon: PPM SystemC state machine / demo, IMU kernel module compilation, Threading demo build, version control configuration

Cody: Initial demo set-up, documentation for original demo, technology research, IMU research/set-up

Spencer: Website updating, IMU research