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.iastate.edu/

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

Linux: Linux IIO subsystem, Linux Kernel Modules, Linux IO remapping

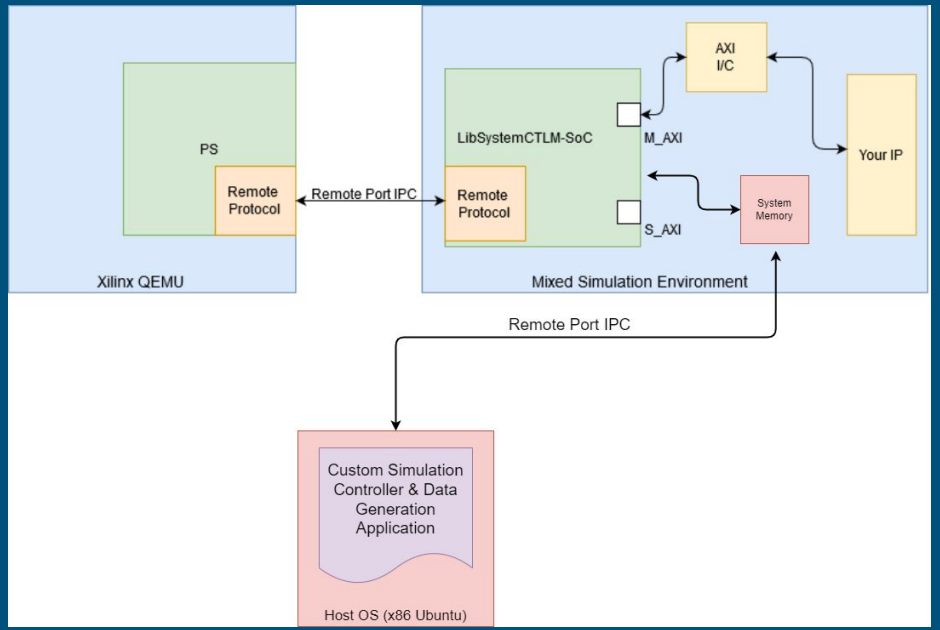
Server: Ubuntu 18.0.4 Server

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

Industrial I/O: description

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



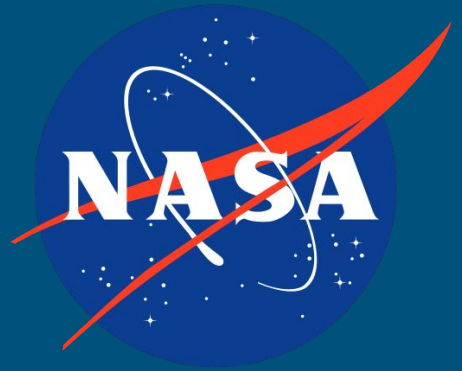
Problem Statement

- 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

```
RUN
-------------------------------------
When running you will need to make sure the program can link to your
SystemC/TLM libraries. You will also need to give arguments to the application.
The first argument points to the QEMU machine-path to use. The second argument
is the icount value to use. The arguments should line up with the QEMU command
line arguments.

A Versal example:
In one terminal, in the demo directory
LD_LIBRARY_PATH=/usr/local/systemc-2.3.2/lib-linux64/ ./versal_demo \
    unix:/tmp/qemu/qemu-rport-_amba@0_cosim@0 10000

A ZynqMP example:
In one terminal, in the demo directory
LD_LIBRARY_PATH=/usr/local/systemc-2.3.2/lib-linux64/ ./zynqmp_demo \
    unix:./qemu-tmp/qemu-rport-_amba@0_cosim@0 10000

A Zynq-7000 example:
LD_LIBRARY_PATH=/usr/local/systemc-2.3.2/lib-linux64/ ./zynq_demo \
    unix:./qemu-tmp/qemu-rport-_cosim@0 1000000
```

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

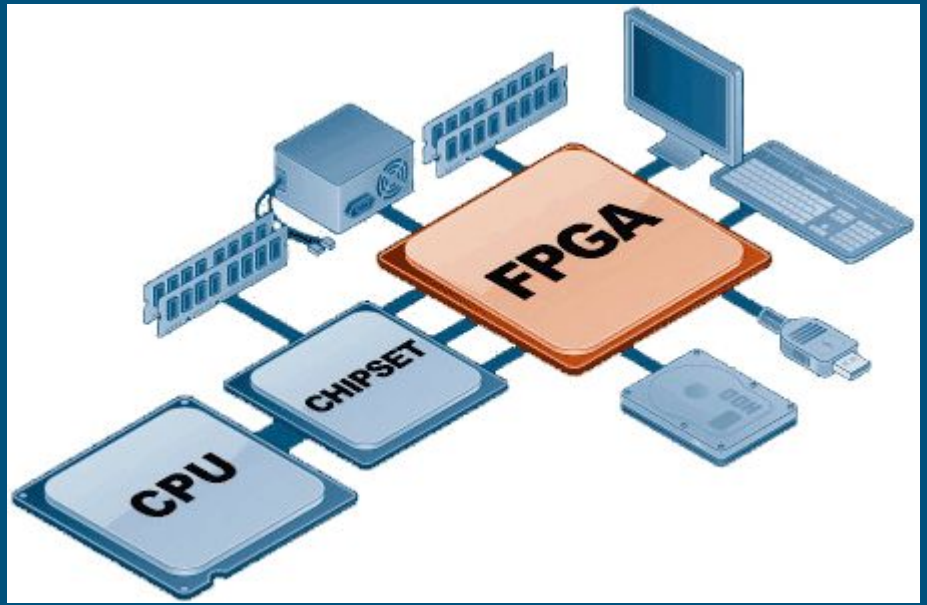


JOHN DEERE

XILINX®

Use Case Example

- You are a software engineer developing a hardware driver to control a new temperature sensor
  - Interfacing with it over an I^2C 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
  - Create an additional demo to for a more complex system
- External Data Source/Modeling Tool
  - IIO device created in software, configured to mock an accelerometer
  - Drive a simulated IIO device via a SystemC model with static data
  - Develop Remote port custom communication tunnel for external data source tool

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 I^2C/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

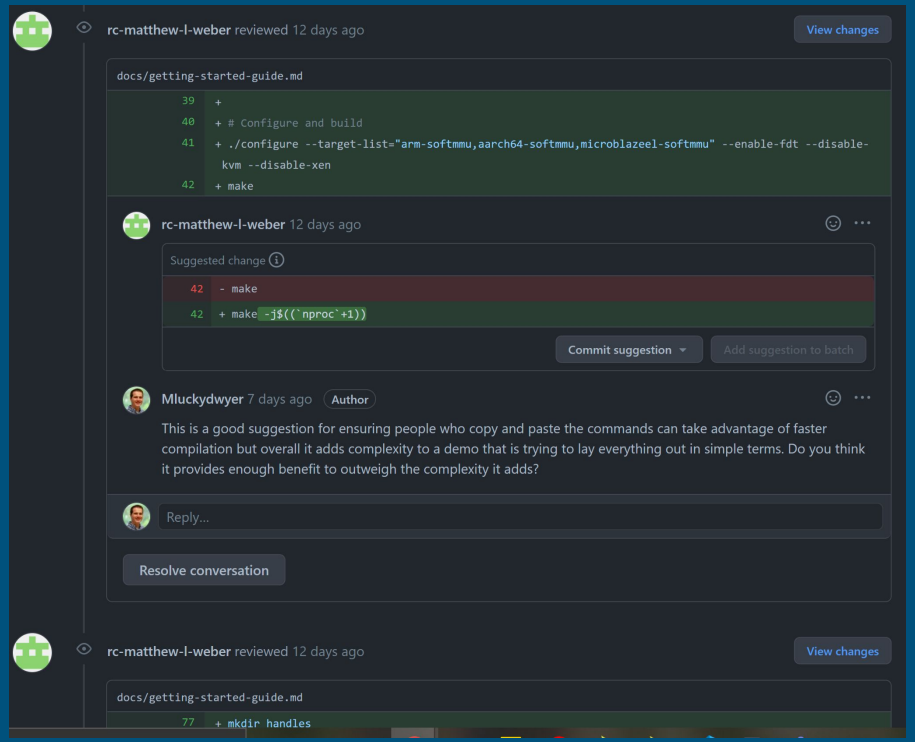
```
mkdir ~/cosim
```

SystemC Setup

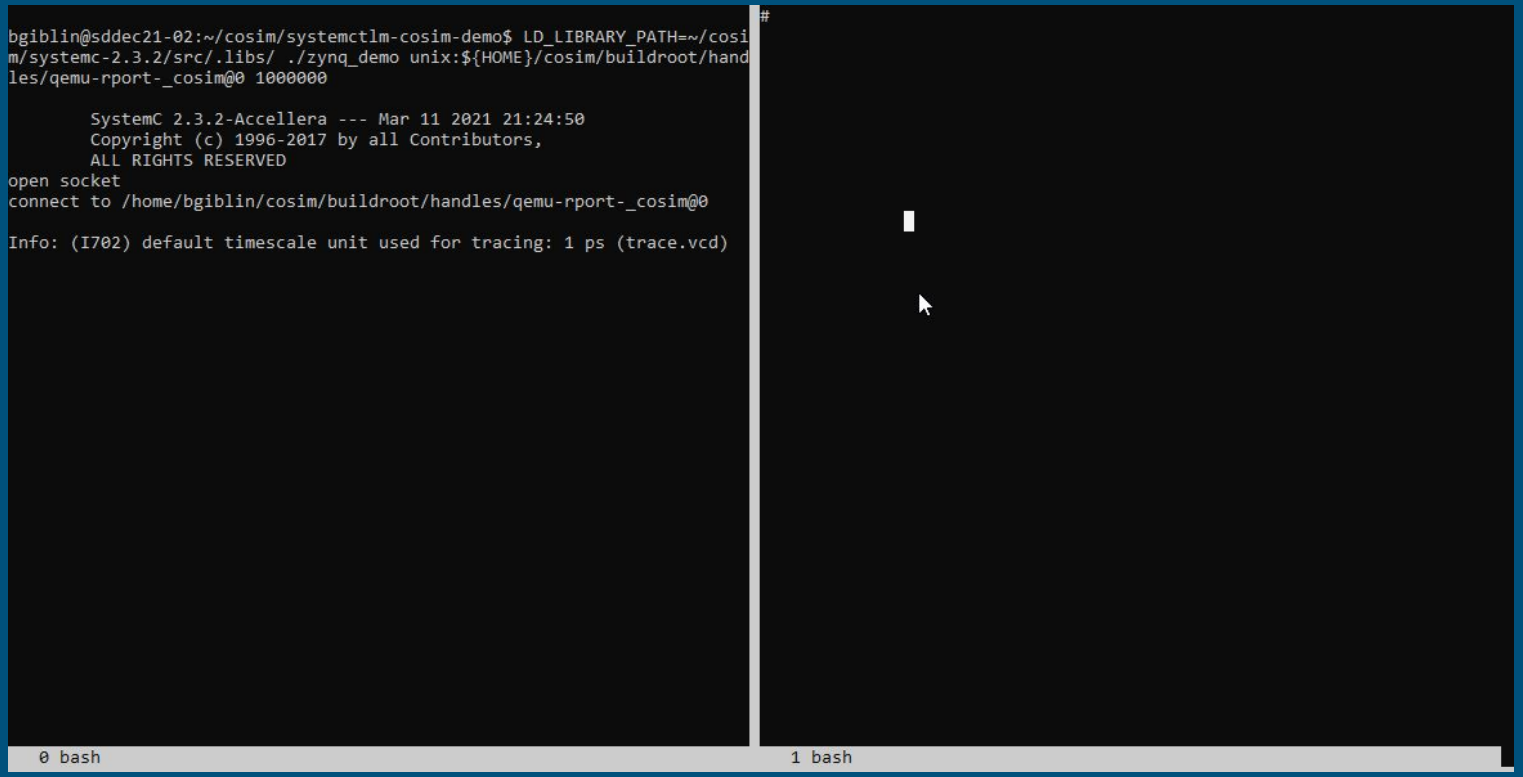
```
cd ~/cosim
SYSC_VERSION=systemc-2.3.2
wget https://www.accellera.org/images/downloads/standards/systemc/systemc-2.3.2.tar.gz
tar xf ${SYSC_VERSION}.tar.gz && cd ${SYSC_VERSION}/
```

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



Co-Sim System Clock Demo



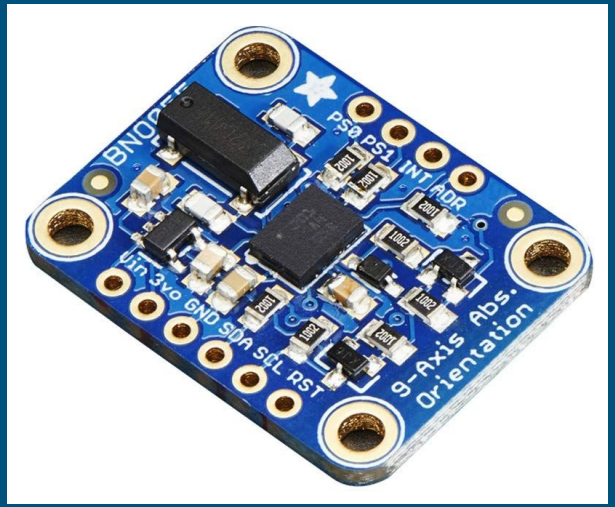
PPM State Machine Demo

```
void PPM_Receiver::gen_ppm()
{

  const int FRAME_TIME = 20000; // 20000 uS
  const int CH_NUM = 6; // just use one channel, 0 to 15

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

Detailed Design - Novel Linux Device Implementation

- Write a state machine in SystemC to model bus transactions
- Pipe the SystemC memory mapped registers into the IIO subsystem
- 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 I^2C bus with one or more devices attached
  - Host system modifying SystemC parameters on the fly to vary IMU data



IIO Demo

Buildroot

SystemC

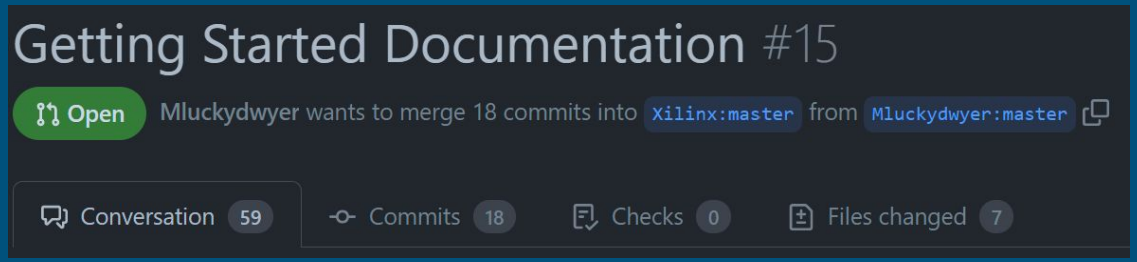
Host

- Full out of tree buildroot setup
- 2 static channels, one random channel, 1 sine wave channel
- IIO driver has been configured to read from IO memory
- End-to-end demonstration of cosim->driver->back to host interaction

Remote-Port Demo

- QEMU instance running Buildroot linux with Zynq-7000 device tree
- Host application creates Unix socket for SystemC
- SystemC connects to both systems and maintains writable memory space
- Read and write from either device

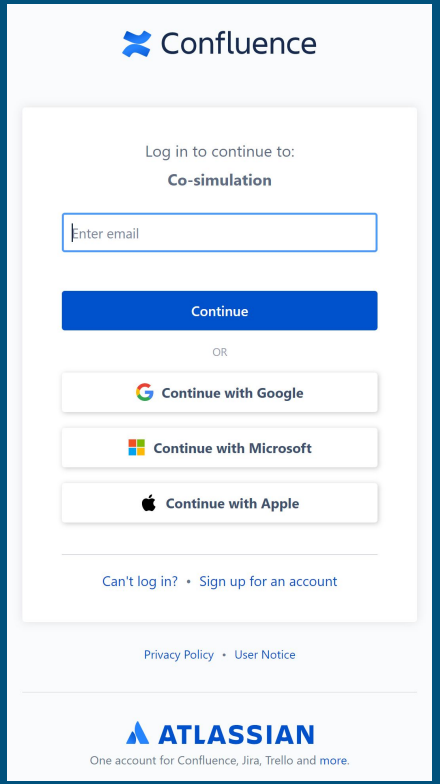
Testing



- Individual model/demo (Remote-Port and IIO)
  - Manual verification using raw memory accesses (devmem)
  - Automated tests with custom simulation testing applications
- Demo Documentation
  - Clean code examples
  - Rich documentation of setup and running demos
  - All documentation reviewed by team, client, and open-source community
  - Committed documentation tested and reviewed by protocol authors
- Process
  - Tests conducted in controlled Co-Simulation environments
  - All done in latest SystemC release
  - Started with basic static value returns to frontend from backend SystemC
  - Working system “bootstrapped” to DUT frontend/backend

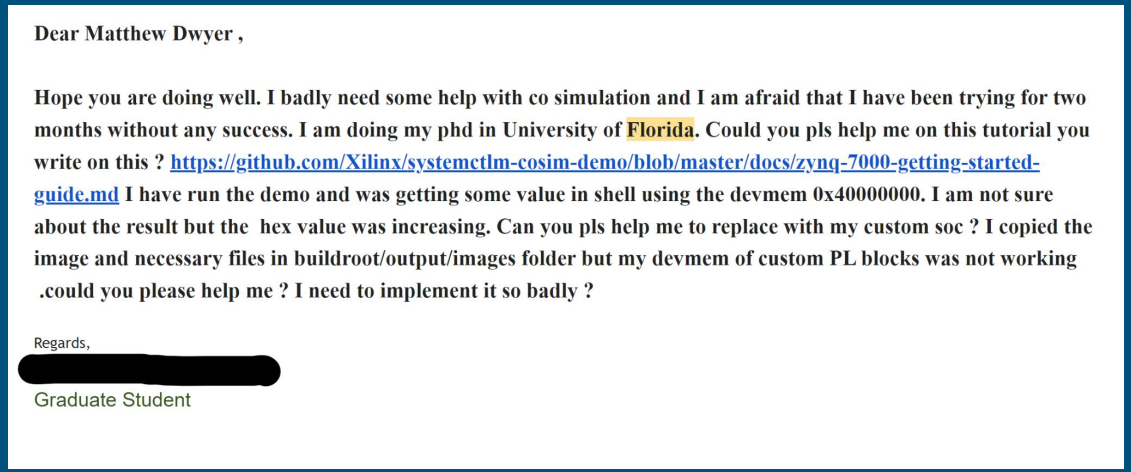
Constraints & Considerations

- 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



Community Impact

- Documentation put to work quickly!
- Proved just how important effective documentation is
- We were able to identify shortcoming in our existing documentation and fill those gaps
- Within 2 weeks of publication



Dear Matthew Dwyer ,

Hope you are doing well. I badly need some help with co simulation and I am afraid that I have been trying for two months without any success. I am doing my phd in University of Florida. Could you pls help me on this tutorial you write on this ? https://github.com/Xilinx/systemctlm-cosim-demo/blob/master/docs/zynq-7000-getting-started-guide.md I have run the demo and was getting some value in shell using the devmem 0x40000000. I am not sure about the result but the hex value was increasing. Can you pls help me to replace with my custom soc ? I copied the image and necessary files in buildroot/output/images folder but my devmem of custom PL blocks was not working .could you please help me ? I need to implement it so badly ?

Regards,

[redacted]

Graduate Student

Thank You For Listening!

Questions?

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

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: Also made contact with open-source community, initial demo set-up, documentation for original demo, technology research, IMU research/set-up, note-taker

Spencer: Website updating, IMU research