Nova1 Project Update

An Open-source Embedded AI Platform

The Nova Open-Source Platform

Open source RISCV-based embedded AI reference platform

Objective: to provide students with hands-on development experience with latest technologies

Staffed by student team with industry mentors

Focus on latest open-source tools and low-cost development methodologies

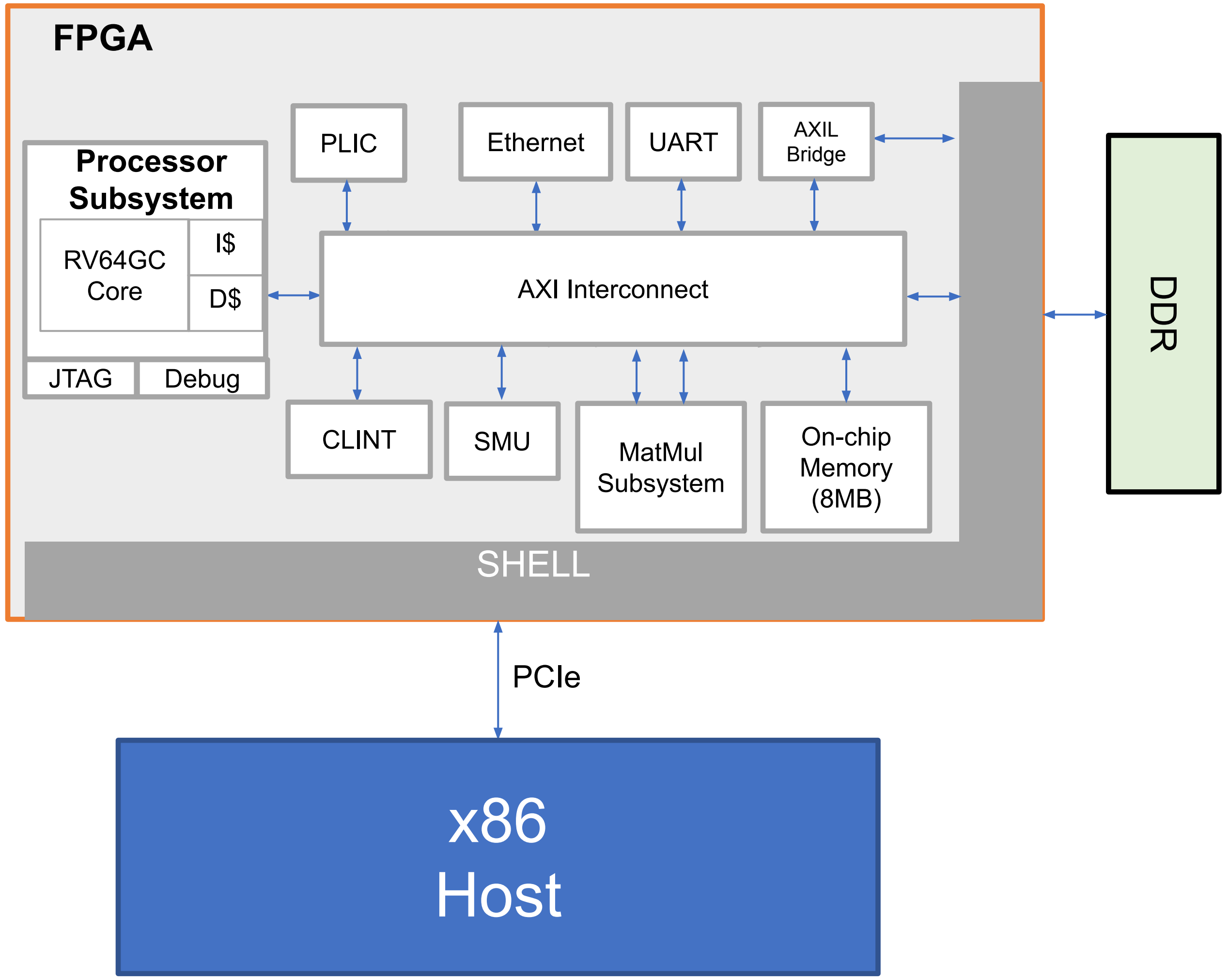
- Linux, RISCV, SystemVerilog, Verilator
- AWS Cloud FPGA

Intended to be used for teaching and research

Disciplines Covered

- Hardware design, verification, physical design
- Firmware and embedded software
- AI/ML applications
- Cloud development, FPGAs, etc.

Nova1: UCI + UITU



Emulation Team(MERL-UITU)

Objective

- AWS Cloud FPGA emulation system
  - System management and related functions
  - Virtual JTAG for debug
  - UART emulation stub
  - Host drivers and emulation software
- Processor Subsystem Design
  - Based on CVA6 project from Chips Alliance
- Integration of MatMul accelerator
  - Tests for confirming MatMul access to system memory
  - Test software for communicating with host (calls gemmlowp)

Milestones

1. Rerun tests with baseline release
2. Bring up baseline release on FPGA
3. FS1 development: Integrate MatMul accelerator initial release
4. Bring up FS1 in FPGA
5. FS2 development: Integrate MatMul final release
6. Bring up FS2 in FPGA

Student Team:
Nameer Iqbal, Syeda Rafia, Khadija, & Abdul Muheet (VI th and VIIIth Semester students)
Academic-Student Advisors:
Sajjad Ahmed, Zeeshan Rafique, Zain Rizwan, Dr. Ali Ahmed
Industry Mentor:
Hamza Khan

Platform Software Team(MERL-UITU)

- **Objective**
  - Boot Linux on Nova Platform
  - Linux kernel
  - Root File System (memory based)
  - Device tree for Linux kernel

- **Milestones**
  1. Modify Linux kernel to work on emulation platform
  2. Boot CVA6 in emulation
     - Depends on FS1
  3. Boot Linux in emulation
     - Depends on FS1

Student Team:
Shahzaib Kashif and Muhammad Shahzaib
Academic- Student Advisor:
Sajjad Ahmed, Zeeshan Rafiq, Zain Rizwan, Dr. Ali Ahmed
Industry Mentor:
Farhat Abbas

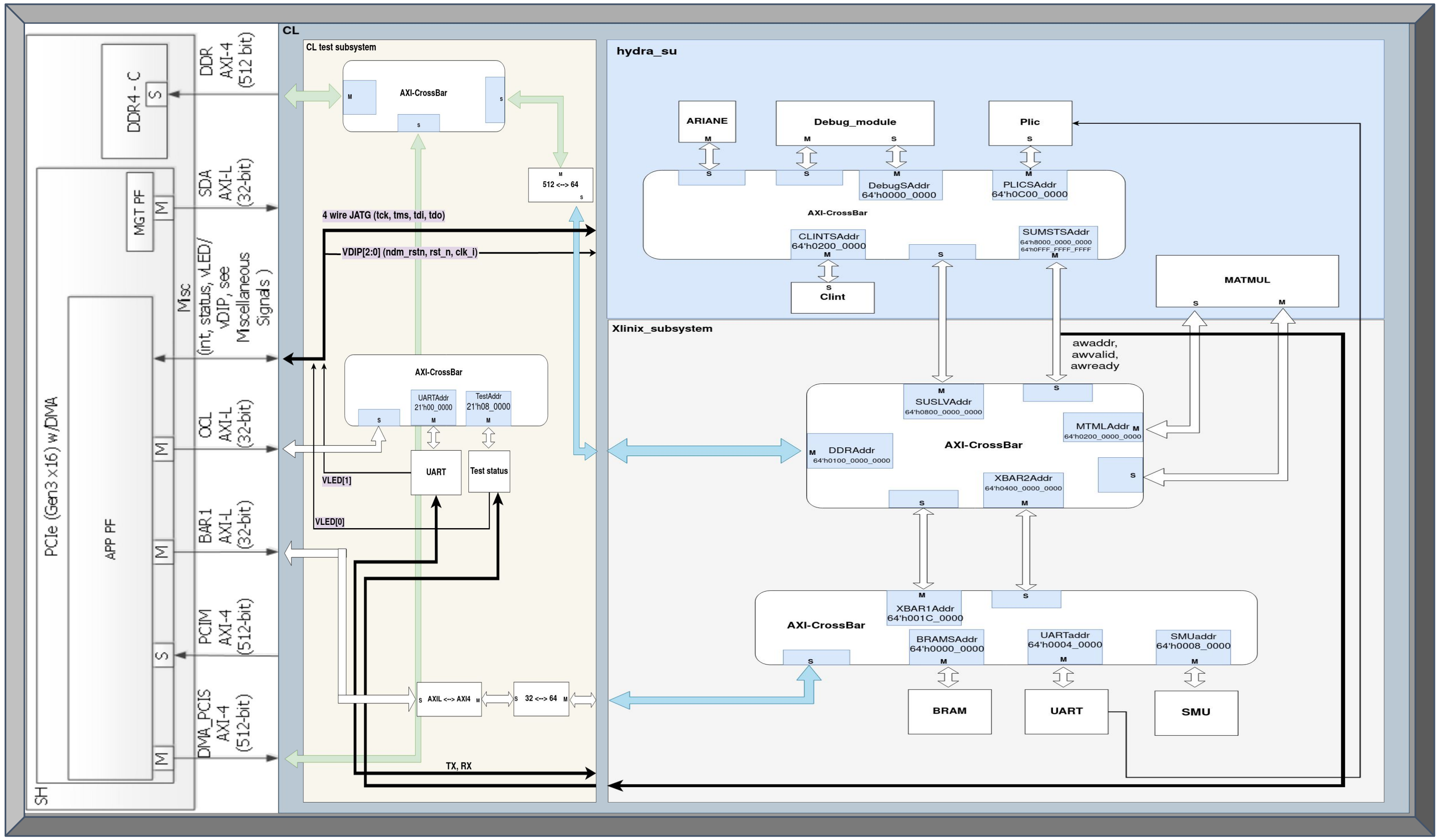
Prerequisite For Emulation Team

- Basic Verilog/System Verilog Design Knowledge.
- Basic FPGA emulation experience, preferable Xilinx FPGA.
- Verilator, Git, RISC-V Assembly & C- language.
- Computer Architecture and Organization Undergrad Course (Preferably RISC-V ISA)
- Able to run assembly tests on RISC-V (RV32I/64I) processor in verilator and on emulation platform.
- Basic knowledge of bus architecture (AXI), peripherals (UART) and interrupts. [can be ramp-up during project]
- Understanding of Cloud FPGA (AWS-F1)workflow [can be ramp-up during project].

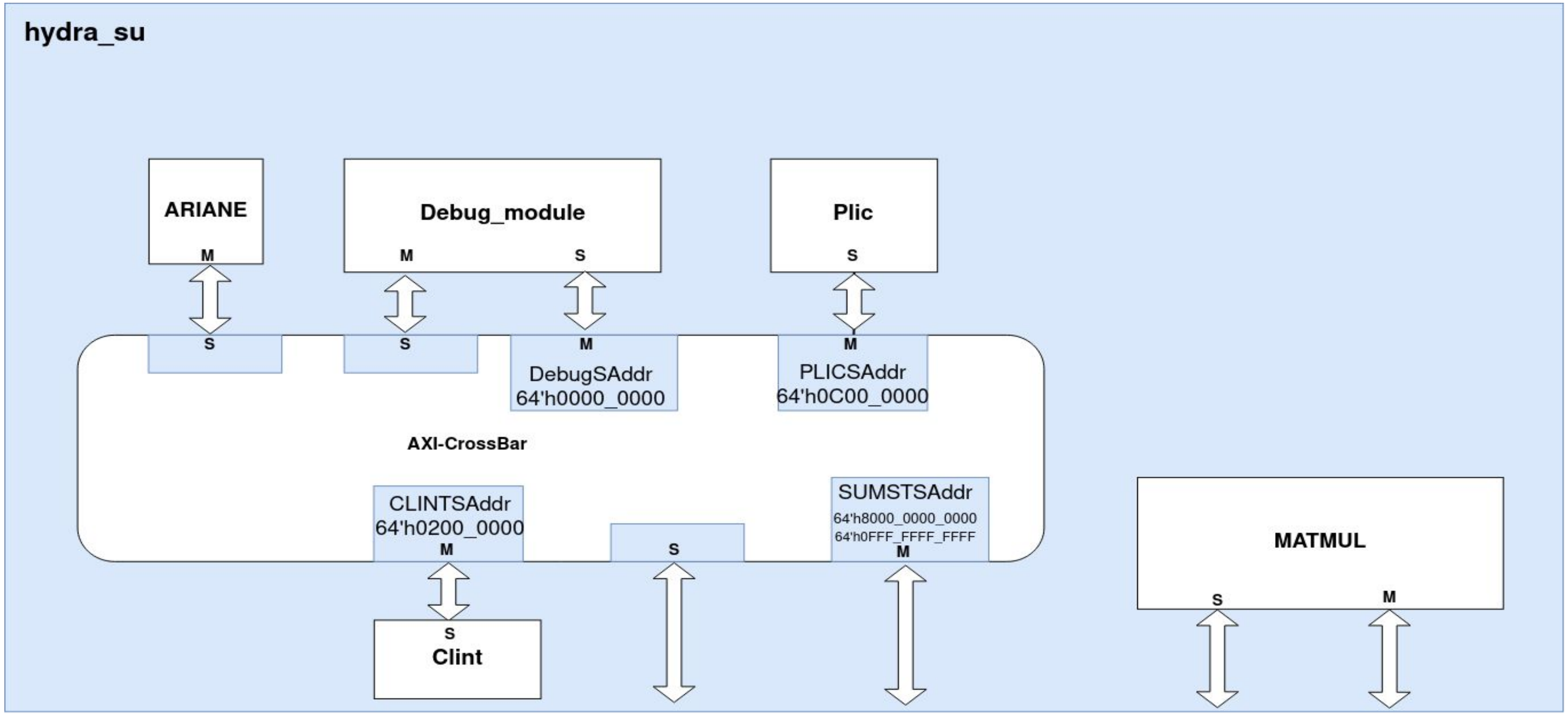
Nova Goals

- To provide undergrad students, hands on experience of creating application class embedded SoC using open source, commercial and custom designed IPs [Project -based Learning].
- Booting RTOS (Zephyr) or Linux Kernel on Hydra-SU with minimum peripheral interface of (UART) and MATMUL accelerator on AWS-FPGA
- Integration and testing of MATMUL (for embedded matrix multiplication)

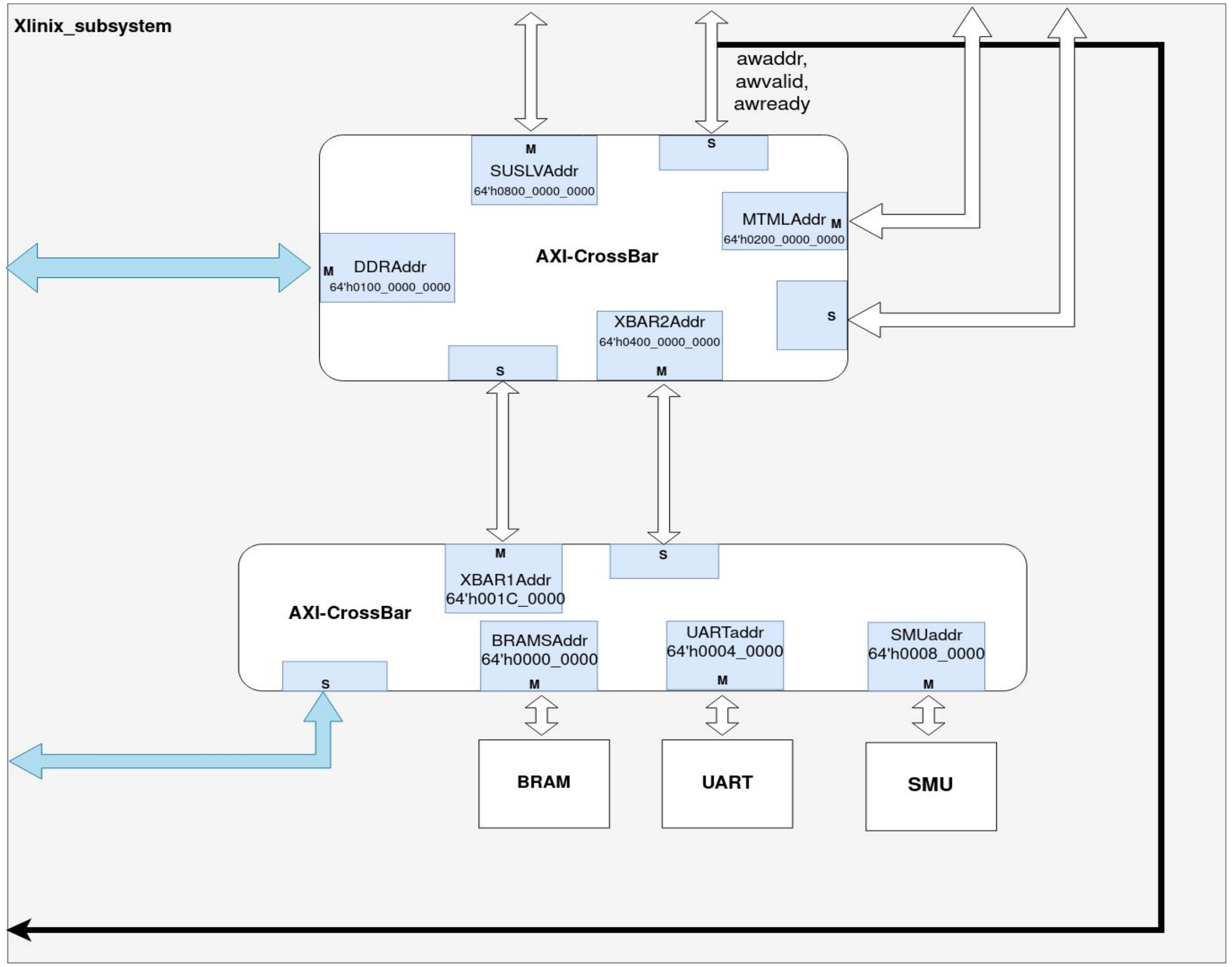
Nova System Diagram For Aws-Fpga



Hydra-Su

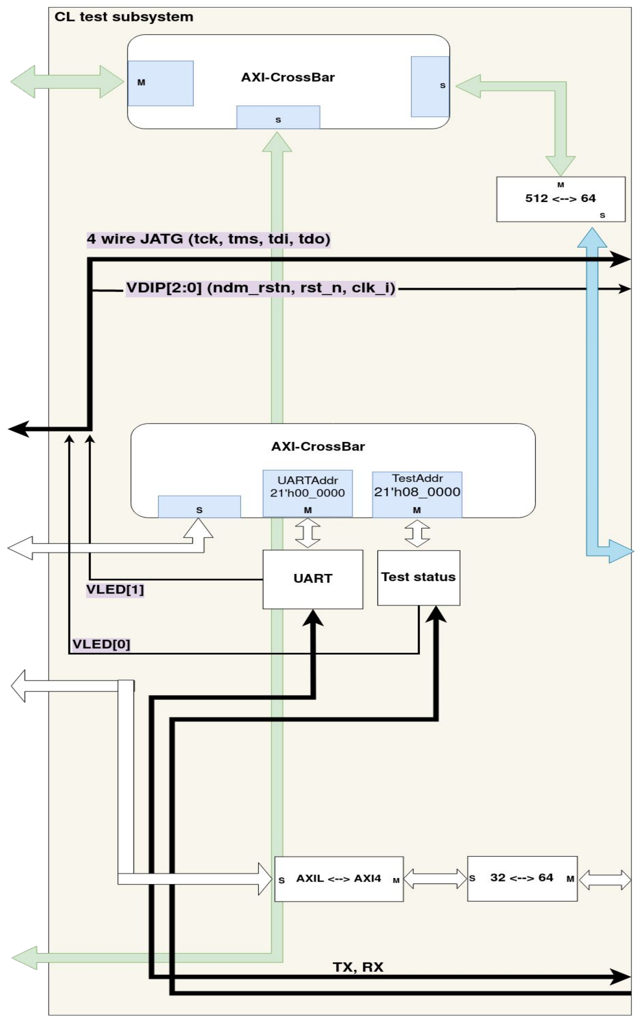

hydra_su
ARIANE
M
Debug_module
M
S
Plic
S
S
S
M
DebugSAddr
64'h0000_0000
M
PLICSAddr
64'h0C00_0000
AXI-CrossBar
CLINTSAddr
64'h0200_0000
M
S
SUMSTSAddr
64'h8000_0000_0000
64'h0FFF_FFFF_FFFF
M
S
Clint
MATMUL
S
M

Xilinx Subsystem



Test Subsystem

- The only access point on aws-fpga is the PCIe port. Therefore, test-subsystem is implemented for making compatibility between NOVA subsystem and aws-fpga shell.

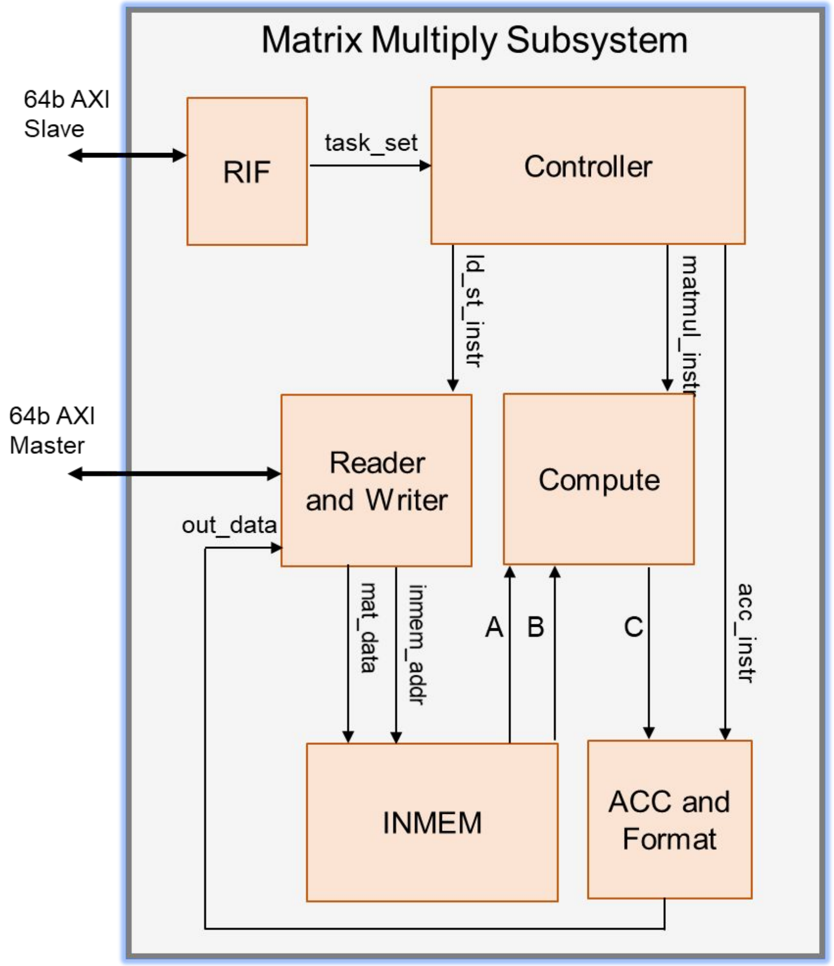


Matrix Multiply Subsystem

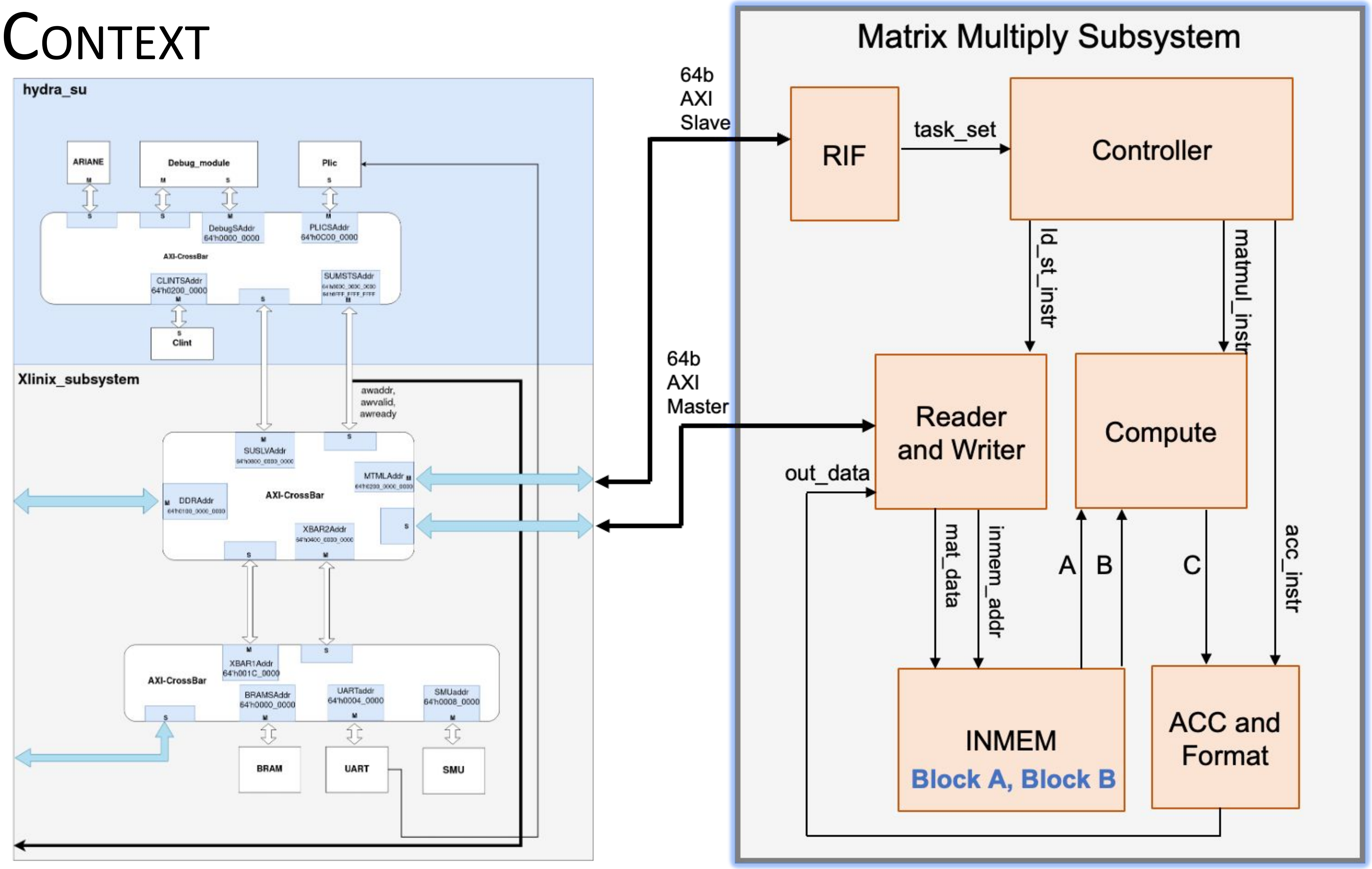
- Designed to offload Matrix Multiply operations from system host (gemm kernel)
  C = A x B
- Support for 8b,16b, or32b signed or unsigned integer elements.
- Support matrices of arbitrary dimensions using block matrix multiply with 128x128 (8b) blocks
- Matrix dimensions limited by system memory
- Full-precision intermediate results.
- Throughput rate matched to 64b AXI subsystem

Team Members:
Asma Khan, UC Irvine (Hardware Design/Verification)
Calvin Huynh, UC Irvine (Software Design)
Zainab Ashai, UC Irvine (Hardware Design)

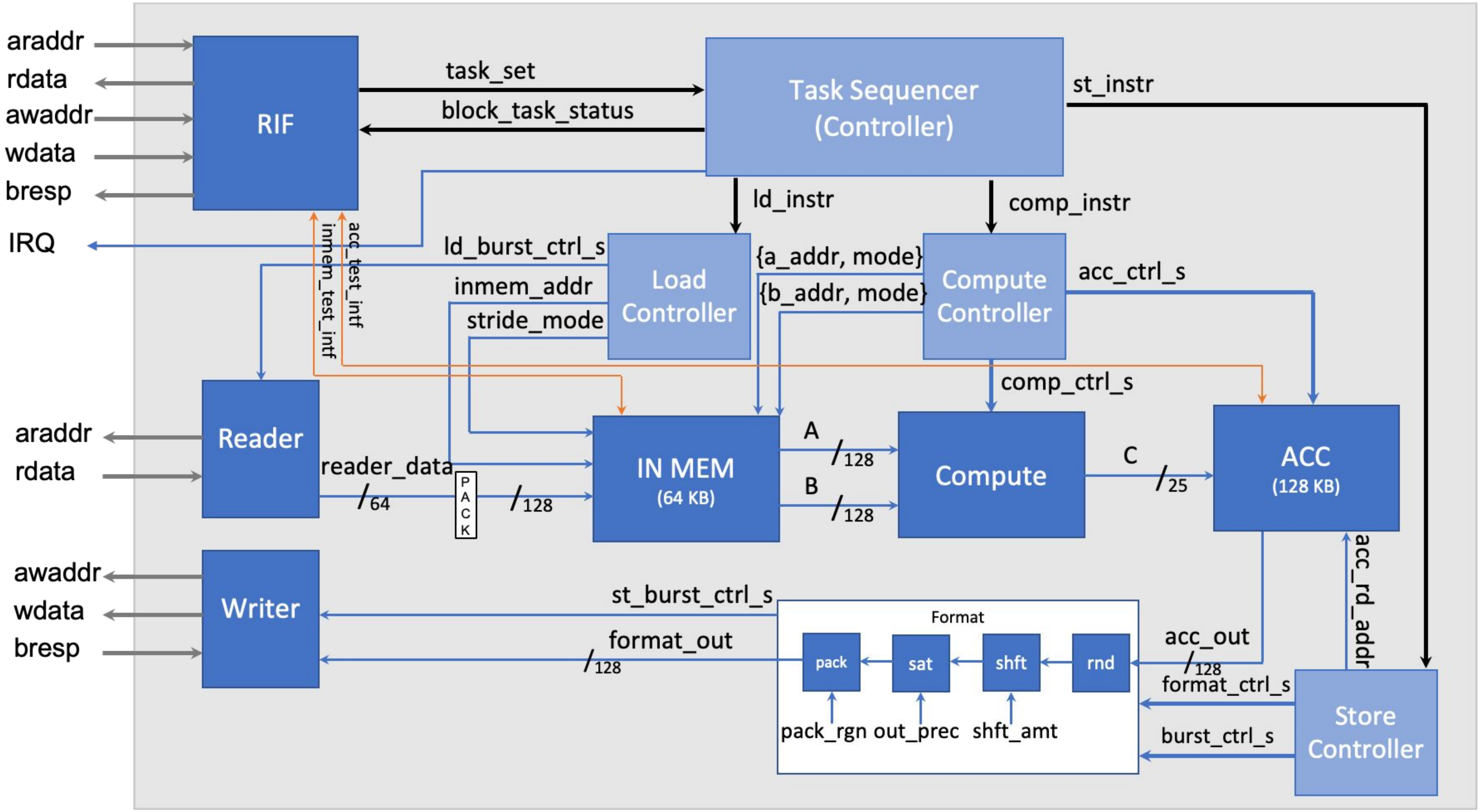
Industry Mentors:
Zainab Khan, Xcelerium
Hamza Khan, Xcelerium



System Context



Matrix Multiply Subsystem Architecture



Aws-Fpga Simulation Flow

- Aws provides a complete simulation setup with simulation models of DDR, PCIe BAR interfaces, virtual dip witches, virtual LEDs and PCIe DMA interfaces.
- For accessing PCIe BAR interfaces aws provides systemverilog dpi tasks and C language APIs.
- For projects like NOVA the Booting process has two parts.
    - Loading program into an internal memory (DDR/BRAM) through one of the PCIe BAR interfaces while keeping the execution unit at reset.
    - De-asserting the reset of execution unit so it can start boot up.
    - The current test, tests the software interrupt by first configuring it using clint and once the interrupt is raised it jumps to trap handler executes the routine, return back to main code and write the status at tohost memory location.

Aws-Fpga AFI Generation Flow

AFI (Amazon FPGA Image) is a bitstream file use to program FPGA.

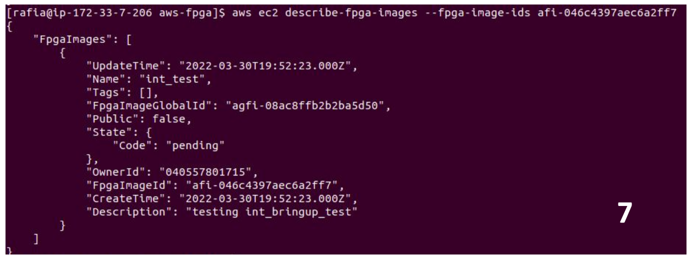
Following steps are needed to be run to generate AFI.

1. Synthesis and implementation.
   - By running “aws_build_dcp_from_cl.sh” bash script.
   - It will create a DCP (Design Checkpoint).
2. Create S3 bucket (folder) on aws server.
   - *aws s3 mb s3://<bucket-name> --region <region-name> # Create an S3 bucket. Choose a unique bucket name (e.g., aws s3 mb s3://my_awsfpga --region us-east-1)*
   - *aws s3 mb s3://<bucket-name>/<dcp-folder-name> # Create a folder for your tarball files (e.g., aws s3 mb s3://my_awsfpga/dcp)*
3. Copy DCP to S3 bucket.
   - *aws s3 cp $CL_DIR/build/checkpoints/to_aws/*.Developer_CL.tar s3://<bucket-name>/<dcp-folder-name>/*

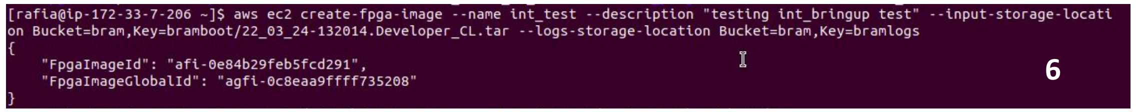
Aws-Fpga AFI Generation Flow (cont..)

4. Create a folder to save your logs.
   - *aws s3 mb s3://<bucket-name>/<logs-folder-name> # Create a folder to keep your logs*
   - *touch LOGS_FILES_GO_HERE.txt # Create a temp file*
   - *aws s3 cp LOGS_FILES_GO_HERE.txt s3://<bucket-name>/<logs-folder-name>/*
5. Check your policy (do you have correct rights to create AFI) *[optional]*
   - *check_s3_bucket_policy.py --dcp-bucket <bucket-name> --dcp-key <dcp-folder-name>/<tar-file-name> --logs-bucket <bucket-name> --logs-key <logs-folder-name>*
6. Once your policy passes the checks, create the Amazon FPGA image (AFI).
   - *aws ec2 create-fpga-image --name <afi-name> --description <afi-description> --input-storage-location Bucket=<dcp-bucket-name>,Key=<path-to-tarball> --logs-storage-location Bucket=<logs-bucket-name>,Key=<path-to-logs>*

     eg. aws ec2 create-fpga-image --name first_afi --description "This is my first AFI" --input-storage-location Bucket=my_awsfpga,Key=dcp/tarbar_name.tar --logs-storage-location Bucket=my_awsfpga,Key=logs
7. Check if the AFI generation is done. You must provide the FPGA Image Identifier returned by create-fpga-image:
   - *aws ec2 describe-fpga-images --fpga-image-ids <AFI ID>*



```
[rafia@ip-172-33-7-206 aws-fpga]$ aws ec2 describe-fpga-images --fpga-image-ids afi-046c4397aec6a2ff7
{
    "FpgaImages": [
        {
            "UpdateTime": "2022-03-30T19:52:23.000Z",
            "Name": "int_test",
            "Tags": [],
            "FpgaImageGlobalId": "agfi-08ac8ffb2b2ba5d50",
            "Public": false,
            "State": {
                "Code": "pending"
            },
            "OwnerId": "040557801715",
            "FpgaImageId": "afi-046c4397aec6a2ff7",
            "CreateTime": "2022-03-30T19:52:23.000Z",
            "Description": "testing int_bringup_test"
        }
    ]
}
```



```
[rafia@ip-172-33-7-206 ~]$ aws ec2 create-fpga-image --name int_test --description "testing int_bringup test" --input-storage-locati
on Bucket=bram,Key=bramboot/22_03_24-132014.Developer_CL.tar --logs-storage-location Bucket=bram,Key=bramlogs
{
    "FpgaImageId": "afi-0e84b29feb5fcd291",
    "FpgaImageGlobalId": "agfi-0c8eaa9ffff735208"
}
```

Progress Update(Matmul)

Hardware

- Tested compute unit and accumulator
- Implemented RIF, INMEM, task sequencer
- Set up AXI4-compliant testbench for subsystem
- MMSS can load and store data to and from system memory

Software

- Implemented General Matrix Multiply Library (gemmlowp)
- Modeled Matrix Multiply Sub-System with RIF Interface
- Validated the model and gemmlowp for 8b unsigned/signed multiplication for variable matrix sizes

Next Steps

- Verifying functionality of controllers
- Testing overall system
- Implementing format unit

Next Steps

- Validating 16b/32b unsigned/signed multiplication
- Porting TensorFlow Lite Application to RISC-V
- Integrating our gemmlowp into TensorFlowLite

Progress Update(Emulation)

Hardware

- Tested Hydra-su with BRAM, DDR and UART in end to end simulation.
- Integrated MATMUL initial release.

Next Steps

- Complete system bringup on FPGA.

Software

- Designed a runtime C driver for loading program image through PCIe.
- Tested Driver in real-time on FPGA.

Next Steps

- Writing driver for uart.
- Booting linux kernel.

Useful Learning Resources

- https://www.legupcomputing.com/blog/index.php/tag/aws-f1-tutorial/
- https://github.com/aws/aws-fpga
- https://github.com/aws/aws-fpga/blob/master/hdk/docs/RTL_Simulating_CL_Designs.md
- https://caslab.csl.yale.edu/courses/EENG428/19-20a/
- Vivado IP Integrator Flow: https://www.youtube.com/watch?v=IABjJx-vnyI
- Students Tutorial 1:

  https://caslab.csl.yale.edu/courses/EENG428/19-20a/tutorials/axi4lite_interface_development.pdf
- Students Tutorial 2: https://caslab.csl.yale.edu/courses/EENG428/19-20a/tutorials/cl_axi_adder_development.pdf

THANK YOU!