Vortex RISC-V GPGPU — RTL Design, FPGA Prototyping & HW/SW Co-Design

Under the supervision of:
Dr. Emad Badry

Vortex RISC-V GPGPU — RTL Design & HW/SW Co-Design

Vortex RISC-V GPU: RTL Design, Verification & FPGA Prototyping

A Hardware–Software Co-Design Approach

Team: 5 Students

Supervisors: University Staff

Duration: Dec 2025 – May 2026

Tools: SystemVerilog, Verilog, UVM (optional), Vivado/Quartus, LLVM, Make, C/C++, RISC-V Toolchain

Motivation & Background

🔥 Why GPU Architecture?

- GPUs dominate AI, ML, and parallel workloads
- Understanding SIMT, warps, thread scheduling is crucial
- Modern GPUs are **complex & closed source**

🚀 Why Vortex GPU?

- Open-source RISC-V GPGPU
- Fully synthesizable RTL
- Portable across ASIC → FPGA
- Designed for educational & research use
- Supports OpenCL (huge advantage)

🎯 Academic Value

- Covers **computer architecture, digital design, parallel programming, compilers, RTL verification, and FPGA prototyping**
- Perfect for final-year graduation level

Project Objectives

Main Goal

Implement, simulate, and prototype the Vortex RISC-V GPGPU on FPGA, while modifying small RTL components and demonstrating HW/SW integration.

Sub-Goals

- ✔ Study GPU architecture: warps, SIMT, pipelines, memory hierarchy
- ✔ Understand Vortex RTL structure
- ✔ Simulate using both RTL & software emulator
- ✔ Run OpenCL workloads
- ✔ Perform minor RTL modifications
- ✔ Synthesize & run on FPGA
- ✔ Build test programs using the Vortex SDK
- ✔ Analyze performance vs CPU

(This is MUCH simpler than designing a 100G PHY.)

System Overview (High-Level)

Diagram :

CPU → Driver → Vortex Runtime → Command Processor → Warp Scheduler → SIMD Cores → Memory Subsystem → Local Memory / Global Memory → FPGA DDR

Core Blocks

- **Command Processor** — receives kernels from host
- **Warp Scheduler** — dispatches threads
- **SIMD Pipeline** — executes instructions in parallel
- **Local Memory** — fast scratchpad
- **Global Memory** — off-chip DDR through AXI
- **CSR Unit** — reports GPU status
- **FPGA Platform** — runs the entire design

What Students Actually Do

✔ Simulation Level

- Run RTL simulation
- Run Vortex emulator & compare results
- Debug kernels (vector add, histogram, matrix multiply)

✔ RTL Understanding & Modification

Examples:

- Modify instruction decoder
- Add a new CSR register
- Add a simple arithmetic instruction
- Modify warp size parameter
- Add debug counters
- Add a pipeline stage or bypass path (optional)

✔ FPGA Prototyping

- Synthesize Vortex core + SoC
- Connect to DDR memory controller
- Use UART/JTAG for control
- Run kernels at hardware speed

✔ Software Stack

- Install Vortex compiler toolchain
- Use OpenCL or RISC-V toolchain
- Write small GPU kernels
- Compare FPGA vs emulator performance

Why This Project Is Feasible for Students

Unlike 100G Ethernet PHY:

❌ No multi-gigahertz SerDes
❌ No FEC
❌ No complex PCS pipeline
❌ No UVM required (but optional)

Instead:

✔ A complete working GPU is already provided
✔ You modify small & understandable parts
✔ The FPGA flow is well documented
✔ The Vortex SDK runs out-of-the-box
✔ You learn architecture + verification + FPGA skills

This project is **much simpler** and **far more achievable**.

Subsystem Breakdown

A. Command Processor (CP)

- Handles kernel launches
- Sends tasks to GPU cores
- Controls memory access
- Students can add new commands or debug instrumentation

B. Warp Scheduler

- Manages warps (groups of threads)
- Selects which warp executes next
- Students may modify scheduling policy

C. SIMD Pipeline

- ALU, integer operations, load/store, branching
- Multistage pipeline
- Students can modify execution units

D. Memory Subsystem

- L1 local memory
- DDR global memory via AXI
- Students can instrument memory latency counters

E. FPGA Integration

- Use Xilinx or Intel FPGA boards
- Map GPU to FPGA fabric
- Connect DDR + UART

HW/SW Co-Design

Students learn:

- OpenCL kernel programming
- GPU runtime API
- Compiler passes
- RTL–software communication
- Performance profiling
- Instruction behavior changes → software performance impact

Examples of kernels to run:

- Vector Add
- Matrix Multiply
- Bitonic Sort
- Mandelbrot
- Convolution

Verification Approach (Simplified)

Unlike the PHY project, full UVM is optional.

Minimum verification:

- Self-checking testbench
- Compare RTL output vs emulator
- Waveform debugging (SimVision/Questa)
- Randomized instruction sequences
- Test memory corner cases

Optional:

- UVM Agent for instruction stream
- Scoreboard comparing against emulator model

Timeline (Dec – May, 6 Months)

Dec

- Study GPU architecture
- Study Vortex RTL structure
- Install toolchain

Jan

- Run emulator tests
- Run RTL simulation
- Understand AXI + memory subsystem

Feb

- Modify small RTL blocks
- Add CSR / counters / lightweight instruction
- Basic verification

Mar

- FPGA synthesis
- Bring-up system on FPGA
- Run first kernel (vector add)

Apr

- HW/SW co-design experiments
- Performance profiling
- Run multiple kernels

May

- Final integration
- Documentation
- Report + presentation

Expected Deliverables

RTL + FPGA

- Synthesizable RTL modifications
- FPGA bitstream & bring-up logs

Verification

- Testbenches
- Simulation outputs
- Emulator comparisons

Software

- GPU kernels
- Performance profiling scripts
- Correctness validation

Documentation

- Project report
- Architecture diagrams
- Timeline and results
- Final PPT

Challenges & Innovations

- ✔ Understanding GPU pipeline & warps
- ✔ Memory subsystem behavior
- ✔ Optimizing for FPGA timing
- ✔ HW/SW interaction & debugging
- ✔ Writing working GPU kernels
- ✔ Comparing emulator vs FPGA performance

Tools & Environment

- Vivado / Quartus
- Verilator / QuestaSim
- RISC-V GCC or LLVM
- Vortex SDK (Driver + Runtime)
- Python/Pandas for performance plots
- GitHub CI

References / Study Materials

✔ Vortex GitHub

🔗 https://github.com/vortexgpgpu/vortex

✔ Vortex Documentation

🔗 https://vortexgpgpu.github.io/

✔ GPU Architecture Basics (NVIDIA — SIMT Explained)

🔗 https://www.youtube.com/watch?v=1J5bK7aOQf4

✔ RISC-V ISA Spec

🔗 https://riscv.org/technical/specifications/

✔ AXI4 Interconnect Overview (ARM Documentation)

🔗 https://developer.arm.com/documentation/ihi0022/latest

Team Members

Mohamed Ahmed Mohamed

Mostafa Mohamed Ahmed

Aya Mohamed abdelmaqid

Hazem Yasser Mahmoud

Mahmoud Elian

Thank You