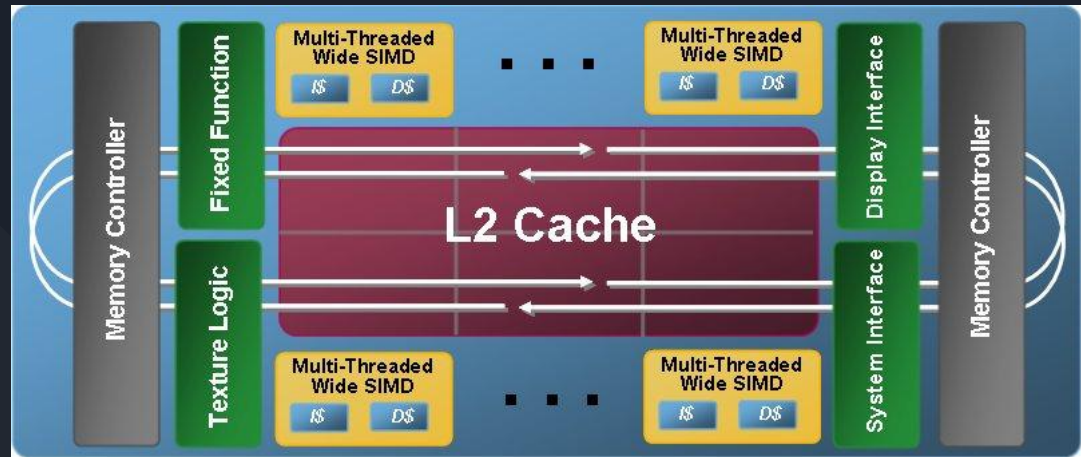


Wannabee Larabee

Alexander Gotsis, Cyril Agbi, and David
Gronlund - Team A3





Use Case

- Provide the floating point resources of an FPGA...
 - ...while also allowing easier programming for a typical set of floating point operations than Verilog
- Intel Larabee
 - many CPU cores as a GPU, instead of using a specialized graphics processing core
- Solve the same use case...
 - general purpose, highly parallelized computing, but instead of using an x86 architecture was implementing RISC-V on our cores



Requirements

- Need general purpose, lightweight implementation of RV-32I to serve as a vector core
- Each small vector core needs to also have a floating point register file, which supports both single precision floating point, as well as the RISC-V vector extension
- Either Linux will be running on the ARM cores integrated into the FPGA, or on another RISC-V core implementing the supervisor extension
- *At least* four separate vector cores are memory mapped into the main controller core




RISC-V

- Each Vector core will be small, the RV-32I part will only take up 1000 registers and about 2000 LUTs
 - Preferably hit 200 MHz, and perform about 0.5 MIPS/MHz
- Each vector core will have an FPU supporting addition, subtraction, multiplication, division, and square root
 - Match the clock performance of the core, and be able to perform floating point operations at about 25% of the maximum floating point throughput of that FPGA



Solution Approach

- Our initial supervisor target is the ARM cores on the FPGA we are using, but we would like to move to another RISC-V CPU running Linux on the FPGA if we have time and room
- We are targeting an Ultra-96 board initially, but we hope to expand to another larger FPGA board if possible



Testing, Verification, and Metrics

- Vector Cores
 - running the RISC-V verification suite of compiled code
- ...then
 - vector extension enabled assembler along with GCC
 - developed by a graduate student at CMU to test out our vector implementation



Testing, Verification, and Metrics

- Floating Throughput
 - matrix multiplication traditionally used in graphics
 - compare to hypothetical max
- Write a complex floating point program - nonideal
 - complete renderer or the Mandelbrot set fractal
 - verifies our architecture can be targeted by a programmer



Tasks

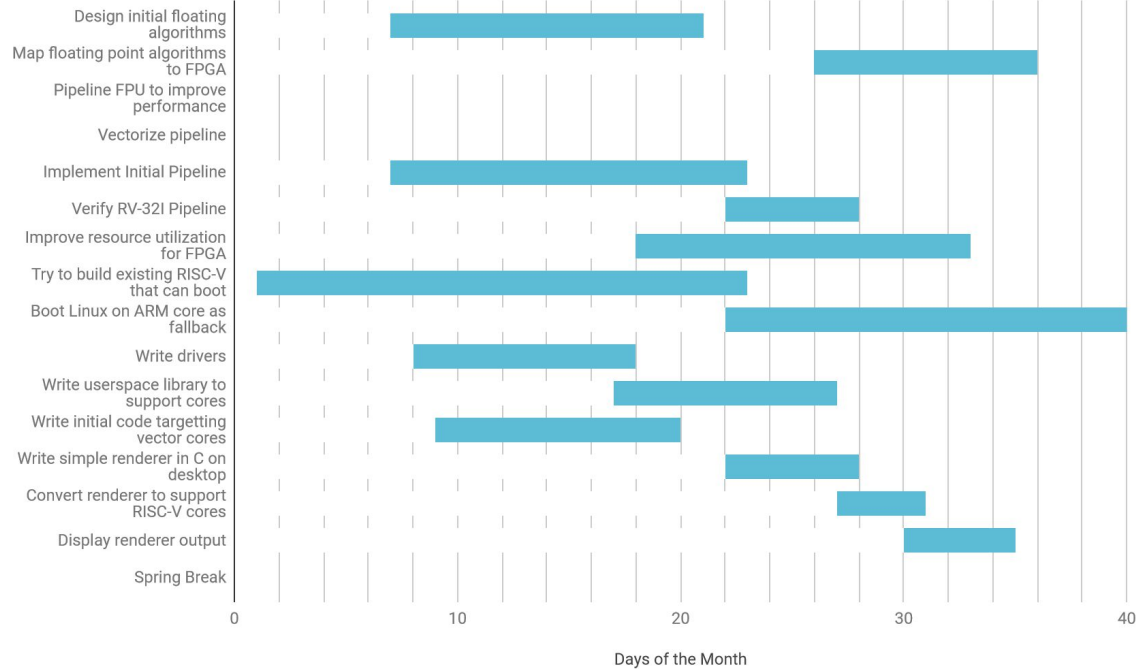
- Implement the RV-32I core
 - Implement a RISC-V compliant FPU
 - Expand the throughput of the FPU to support vector operations
- Compile example programs for the vector instruction set
- Try building existing RISC-V cores that can boot Linux
 - Start by running Linux on the ARM core and write code to manage the vector co-processors
 - If time allows augment the vector core to also boot Linux and be the supervisor



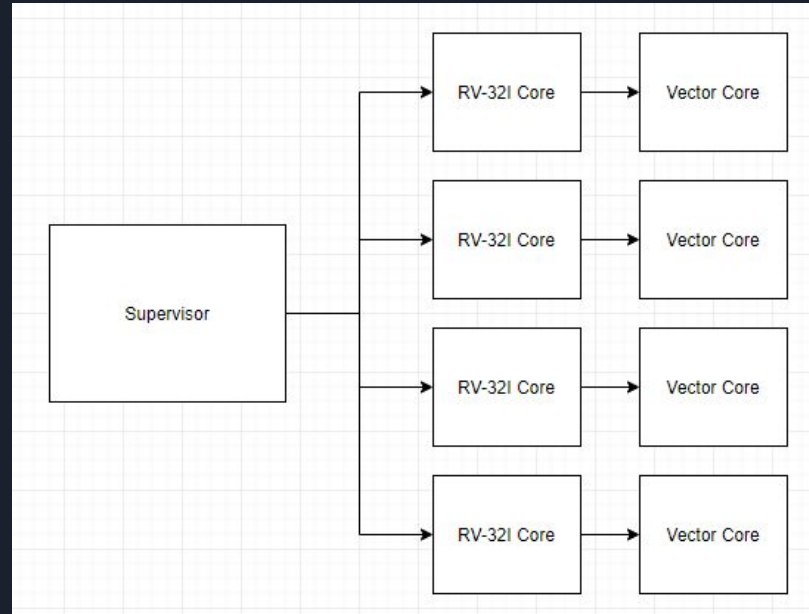
Division of Labor

- David
 - Write basic RV-32I core
 - Pipeline floating point algorithms
 - Design FPU super-scalar augmentations to allow for vector operations
- Cyril
 - Implement and verify SystemVerilog for required floating point operations
 - Assist with designing FPU and vector units
- Alex
 - Write drivers for Linux to support managing and using the vector co-processors
 - Work with and integrate compilers for targeting the vector cores

Schedule



Block Diagram





Stretch Goals

- Implement RISC-V core that can also boot Linux
- Transition to a bigger FPGA
 - Preferably use an Ultrascale part as well, would make transitioning easier