A 28nm RISC-V Vector Processor

Yunsup Lee, Andrew Waterman, Rimas Avizienis,
Brian Zimmer, Ruzica Jevtic, Borivoje Nikolic, Krste Asanovic

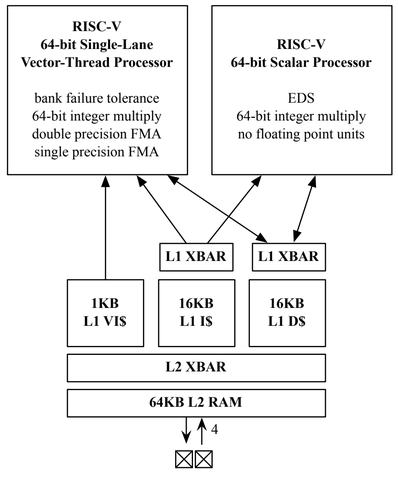
RISC-V Vector

- RISC-V: A New ISA for architecture research
  - Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic, **"The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA"**, TR UCB/EECS-2011-62
- New Vector-Thread extensions to the ISA
- A new microarchitecture
- Capture good ideas from Maven
  - Yunsup Lee, Rimas Avizienis, Alex Bishara, Richard Xia, Derek Lockhart, Chris Batten, Krste Asanovic, **"Exploring the Tradeoffs between Programmability and Efficiency in Data-Parallel Accelerators"**, ISCA '11

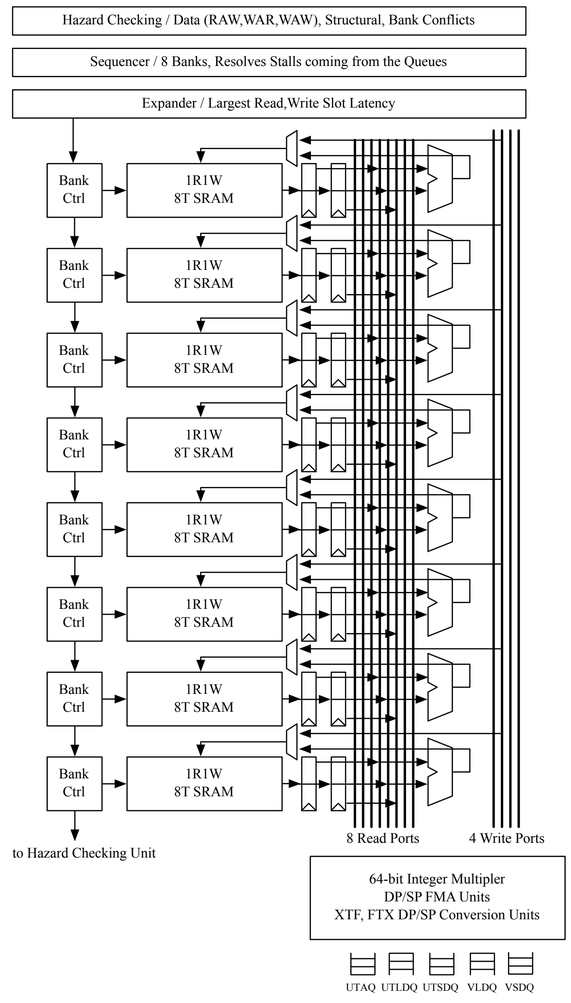
RISC-V Vector Programming Methodology

```
vt::config( 14, 0, ncols-2 );

vt::HardwareVector<uint> vcoeff0( coeff[0] );
vt::HardwareVector<uint> vcoeff1( coeff[1] );
vt::HardwareVector<uint> vnorm( coeff[0] + 4*coeff[1] );

for ( int ridx = 1; ridx < nrows-1; ridx++ ) {

  int vlen = vt::set_vlen( ncols-2 );
  for ( int cidx = 1; cidx < ncols-1; cidx += vlen ) {
    vlen = vt::set_vlen( ncols - 1 - cidx );

    vt::HardwareVector<byte> vpx0, vpx1, vpx2, vpx3, vpx4;
    vpx0.load( &src[ (ridx-1)*ncols + cidx     ] );
    vpx1.load( &src[ ridx*ncols     + (cidx-1) ] );
    vpx2.load( &src[ ridx*ncols     + cidx     ] );
    vpx3.load( &src[ ridx*ncols     + (cidx+1) ] );
    vpx4.load( &src[ (ridx+1)*ncols + cidx     ] );

    vt::HardwareVector<byte> vmask;
    vmask.load( &mask[ ridx*ncols + cidx ] );

    vt::HardwareVector<uint> vdest;
    VT_VFETCH( (vdest), (), (vpx0,vpx1,vpx2,vpx3,vpx4,vmask),
               (vcoeff0,vcoeff1,vnorm),
               (VT_XREG), (), (VT_XREG,VT_XREG,VT_XREG,VT_XREG,VT_XREG,VT_XREG,VT_XREG),
               (VT_XREG,VT_XREG,VT_XREG),
    ({
      uint out = ( vpx0 * vcoeff1 ) + ( vpx1 * vcoeff1 )
               + ( vpx2 * vcoeff0 ) + ( vpx3 * vcoeff1 )
               + ( vpx4 * vcoeff1 );
      vdest = vpx2;
      movn(&vdest, out, !!vmask);
    }));

    vdest.store( &dest[ ridx*ncols + cidx ] );
```

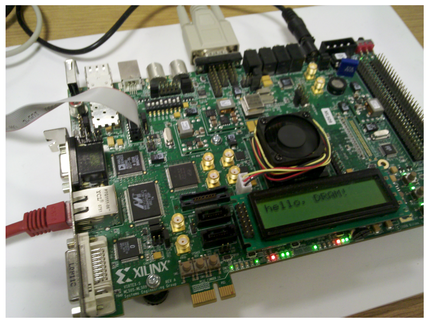
Target Machine



RISC-V Vector Microarchitecture



FPGA Implementation



- Full RTL modeled on XC5VLX110T @33 MHz
- Resource utilization:
  - 85% LUTs
  - 75% BRAMs
  - 64% DSPs

Big decisions for 28nm tapeout

- Use 1R1W 8T SRAM blocks for vector register file
  - 1R1W is normally the best you can get
- Need to add a mechanism to tolerate failures in the SRAM blocks
  - Memory blocks are normally the first things to fail
  - Since the vector lane is banked 8 ways, we can turn off banks and reduce the maximum hardware vector length
  - Add a density time fashion to compress disabled banks
- We are short on time & resources
  - No SRAM generators from the foundry, have to build our own SRAM blocks
  - Need to build our own LVDS pads
  - Need to integrate custom blocks with the digital synthesis flow (make libraries, timing constraints, obstructions on layout)
  - First attempt on physical design
- For this tapeout, we are only going to focus on data parallel code with no branches
- Verify RTL with the FPGA before tapeout
  - Relatively easy, given that we already pushed the control processor through the FPGA toolflow once
  - This will give us a fast target machine, which we can use for software development
- Today's demo is on the FPGA
- Taped out May 29, 2011

Layout