
VectorBlox
embedded supercomputing
FPGA-Optimized
Lightweight
Vector
Extensions
for
VectorBlox ORCA
RISC-V
© 2016 VectorBlox Computing Inc.

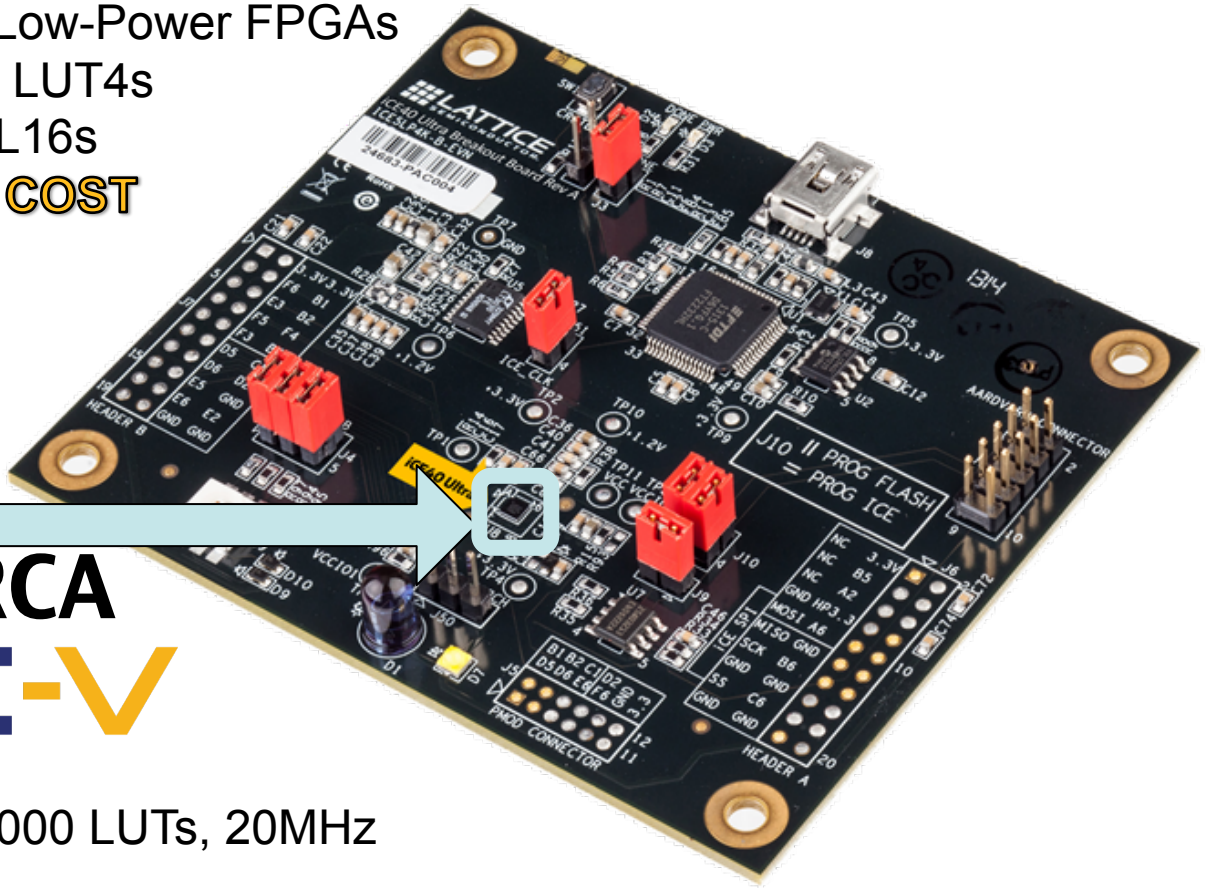
What is ORCA?

- Family of RISC-V implementations
  - Highly parameterized
  - Ideally suited for FPGAs
  - Portable across FPGA vendors
    - Lattice, Altera, Xilinx, Microsemi
  - BSD license open source hardware

VectorBlox ORCA

embedded supercomputing

Tiny, Low-Power FPGAs
3,500 LUT4s
4 MUL16s
LOW COST



VectorBlox ORCA

RISC-V

FPGA-optimized:
RV32IM can be < 2,000 LUTs, 20MHz

Lightweight Vector Extensions

- Goals
  - Area-optimized for tiny FPGAs
    - <<< 10,000 LUTs, no external RAM
  - Performance ~10x
    - Eliminate loads, stores, loop overhead
  - Natural extension to RISC V

- Approach
  - Add dedicated vector data scratchpad (on-chip, fast)
    - Ability to “repeat N times” for arithmetic instructions
    - Address generators to walk through vector data
  - Re-use RISC-V ALU (save area)

Vector Instructions

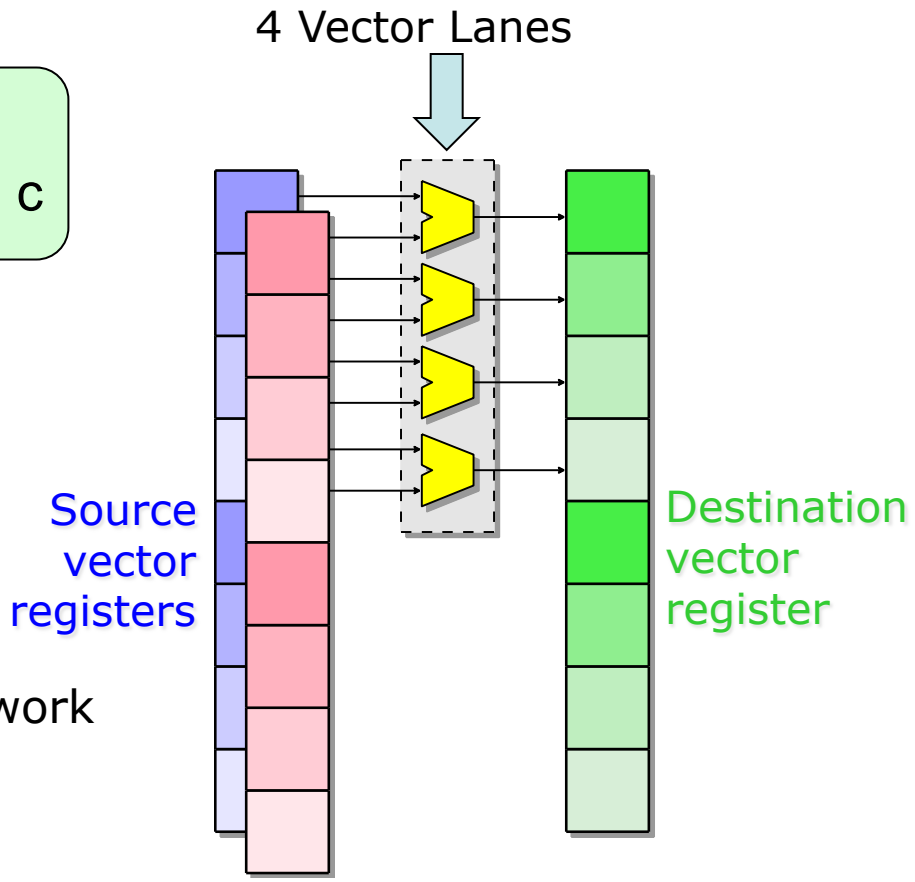
Data-level parallelism

```
for ( i=0; i<8; i++ )
  a[i] = b[i] * c[i];
```

→

```
set_vlen 8
vmult  a, b, c
```

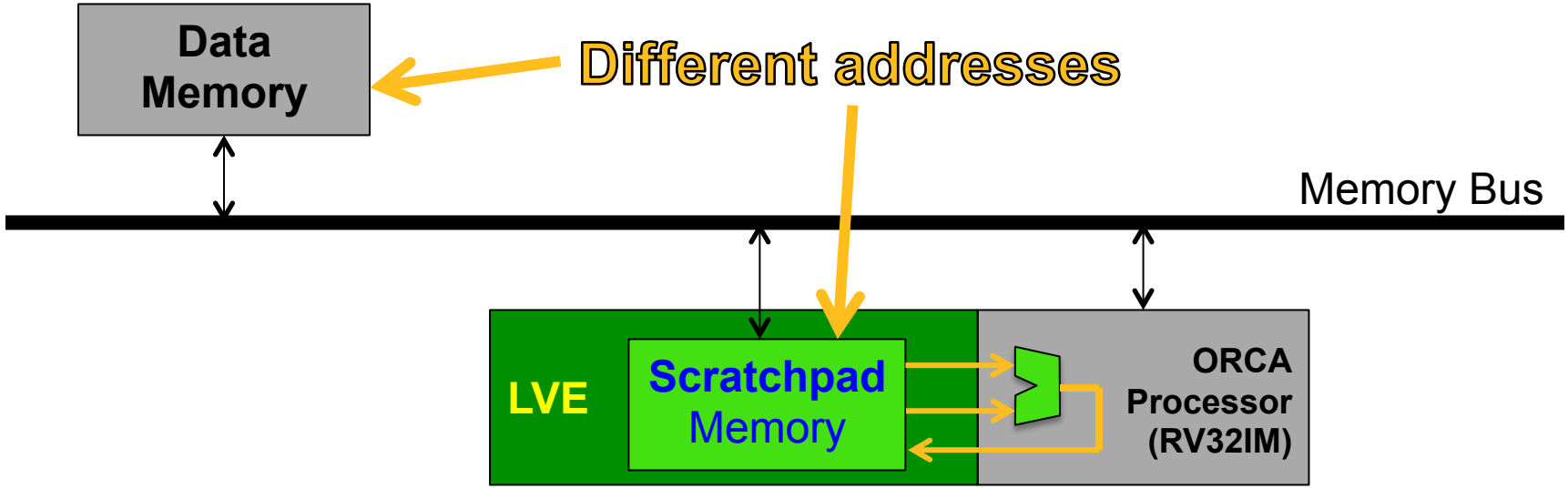
- 1D vectors
  - Extension: 2D, 3D matrices
- Vector operands are RISC-V scalar registers
  - Pointers into scratchpad
  - Address generators do useful work
- 32b data
  - Extension: 8b, 16b
  - Extension: Fixed-point



LVE Vector Instructions

```
Vadd        Vsub              // I base instructions
Vsll        Vsrl    Vsra
Vxor        Vor     Vand

Vslt        Vsltu

Vmul        Vmulh             // M instruction extension
Vdiv        Vrem

Vcmv_z      Vcmv_nz           // conditional move if zero/nonzero
Vtype                         // sets data type, vector length

Vset2Dsrc   Vset2Ddst         // optional extensions for 2D
Vset3Dsrc   Vset3Ddst         //              and 3D matrices
```

System Model



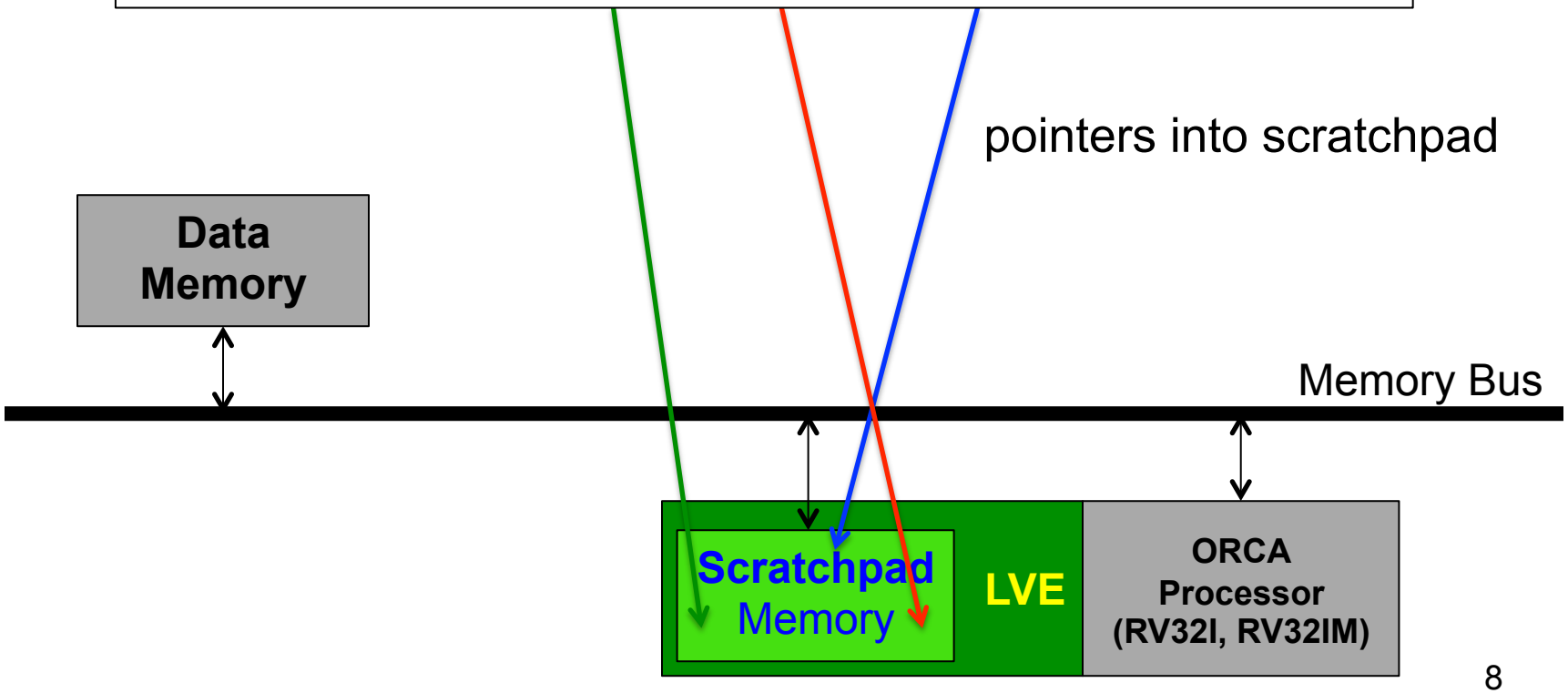
- Vector instructions operate only on scratchpad
- Allocate a vector of 8 words

```
vbx_word_t *vsrc1 = vbx_sp_malloc( 32 );
```

API

Intrinsics

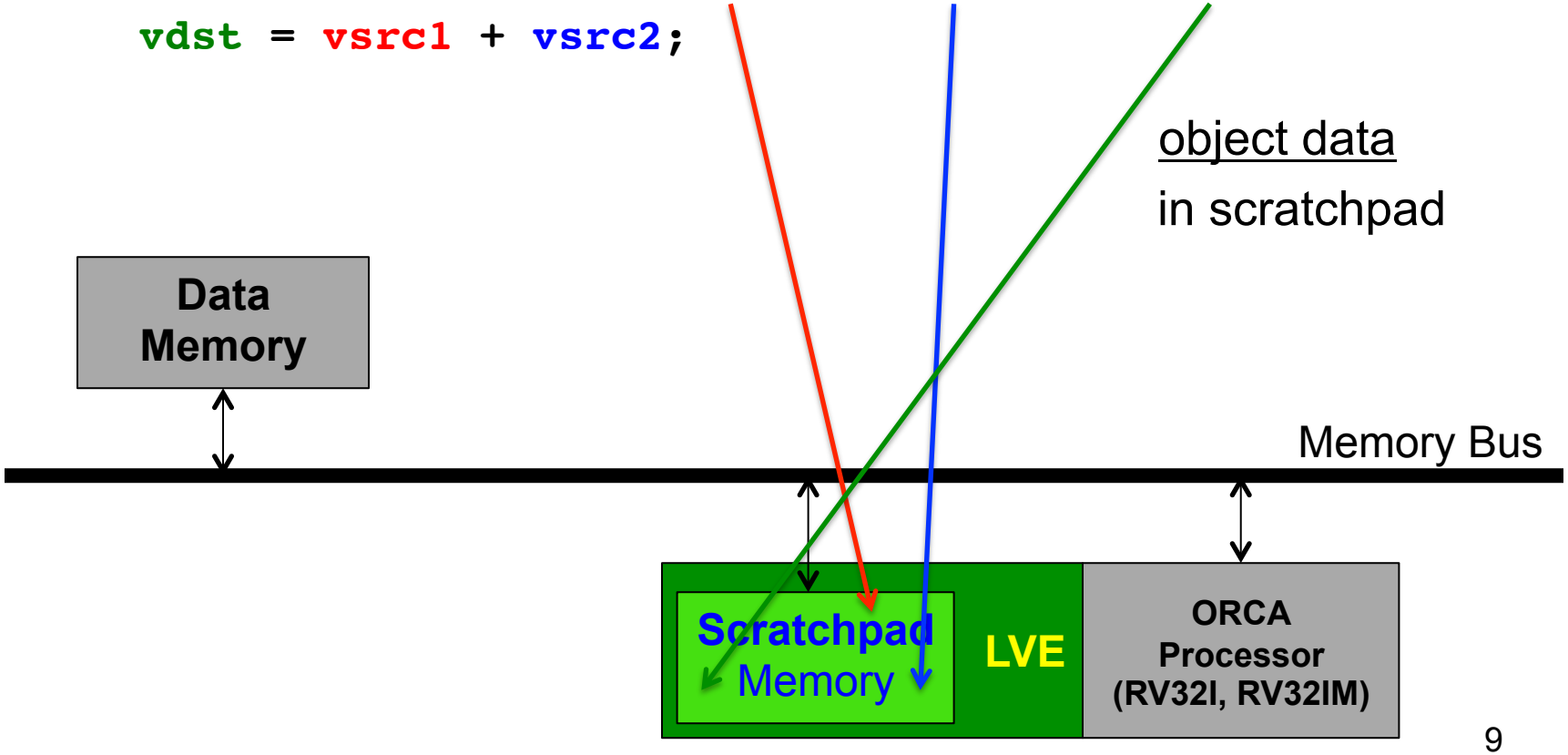
```
vbx_word_t *vsrc1 = vbx_sp_malloc( 32 );
vbx_set_vl( 8 );
vbx( VVW, VADD, vdst, vsrc1, vsrc2 );  // C, or
vbxx(     VADD, vdst, vsrc1, vsrc2 );  // C++
```



API

C++ Objects

```
Vector<vbx_word_t> vsrc1(8), vsrc2(8), vdst(8);
vdst = vsrc1 + vsrc2;
```



Area and Performance

	VectorBlox ORCA RV32IM	+ LVE
Area	2,900 LUTs 20 BRAMs	3,800 LUTs 20 BRAMs + scratchpad
Clock speed	17 MHz	17 MHz
FIR filter (32 taps)	15 cycles/tap	1.25 cycles/tap **12x speedup**

FIR filter

- RV32IM

```
00000030 <scalar_fir(long*, long*, long*, int, int)>:
  30:   40e686b3            sub     a3,a3,a4
  34:   06d05263            blez    a3,98 <.L6>
  38:   00269693            slli    a3,a3,0x2
  3c:   00271e13            slli    t3,a4,0x2
  40:   00d50eb3            add     t4,a0,a3
  44:   01c60e33            add     t3,a2,t3
  48:   00100f13            li      t5,1

0000004c <.L10>:
  4c:   0005a683            lw      a3,0(a1)
  50:   00062803            lw      a6,0(a2)
  54:   00460793            addi    a5,a2,4
  58:   00058893            mv      a7,a1
  5c:   03068833            mul     a6,a3,a6
  60:   01052023            sw      a6,0(a0)
  64:   02ef5263            ble     a4,t5,88 <.L11>

00000068 <.L13>:
  68:   0048a683            lw      a3,4(a7)
  6c:   0007a303            lw      t1,0(a5)
  70:   00478793            addi    a5,a5,4
  74:   00488893            addi    a7,a7,4
  78:   026686b3            mul     a3,a3,t1
  7c:   00d80833            add     a6,a6,a3
  80:   01052023            sw      a6,0(a0)
  84:   fefe12e3            bne     t3,a5,68 <.L13>

00000088 <.L11>:
  88:   00450513            addi    a0,a0,4
  8c:   00458593            addi    a1,a1,4
  90:   faae9ee3            bne     t4,a0,4c <.L10>
  94:   00008067            ret
```

RV32IM:
8 instructions + stalls
72 bytes code

- RV32IM + LVE

```
00000000 <vector_fir(long*, long*, long*, int, int)>:
   0:   000007b7            lui     a5,0x0
   4:   00e7a023            sw      a4,0(a5)
   8:   40e686b3            sub     a3,a3,a4
   c:   02d05063            blez    a3,2c <.L1>
  10:   00269693            slli    a3,a3,0x2
  14:   00d586b3            add     a3,a1,a3

00000018 <.L3>:
  18:   08c5fe2b            vtype.www   a1,a2
  1c:   a6e50cab            vmul.vv.1d.sss.acc  a0,a4
  20:   00458593            addi    a1,a1,4
  24:   00450513            addi    a0,a0,4
  28:   fed598e3            bne     a1,a3,18 <.L3>

0000002c <.L1>:
  2c:   00008067            ret
```

RV32IM + LVE:
1 instruction + no stalls
20 bytes code

Looking Forward

- Scalable / future implementations
  - Add 2D+3D operations (faster outer loops)
  - Add halfword, byte level subword-SIMD (2x or 4x performance)
  - Add second ALU (2x performance)
  - Advanced conditional flags (area/performance TBD)

- Forwards-compatible
  - VectorBlox MXP accelerator for larger systems

- Why not “proposed” RISC-V vector extensions (Hwacha)?
  - Detailed RISC V vector proposal not yet released
  - LVE lower overhead than Hwacha
    - LVE flexible scratchpad: any number of vectors, any length
    - Can write composable vector libraries (not tied to named vector registers)

Conclusions

- ORCA RISC-V family is free, portable, FPGA-optimized
- RISC-V Lightweight Vector Extensions
  - Very low area overhead (< 1,000 LUTs)
  - Good performance (12x on FIR)
  - Scalable (another 2x to 8x faster)
  - Migration path to higher performance (MXP)
- Important advantages
  - Flexible / efficient use of vector data storage
  - Composable vector library functions

BREAK TIME!

So Long, and Thanks for All the Fish !!