An Open Source RISC-V GPU ISA Extension

Atif Zafar

PIXILICA

Atif@Pixilica.com

Abel Bernabeu

Esperanto TECHNOLOGIES

Abel.Bernabeu@Gmail.com

Introduction

RISC-V is the leading open-source ISA with over 300 companies supporting it

- No *open-source* GPU ISA extension currently available for this
- We are a group of enthusiasts who wish to develop such an ISA

Why is another GPU needed?

- Commercial offerings are inaccessible to small projects, makers, startups
- Many use cases in the low-end (embedded displays, handheld gaming…)
- Customers want to be able to change feature sets
  - Open-source drivers will allow this
  - Can avail features like dual-frustum clipping, SLAM, Kalman filters in HW
- The IP can go into ASICs, SoCs, FPGAs or as a software implementation
- Power Management is another issue - this ISA can be very low power

We need your help

We are currently in need of

- RISC-V Vector ISA simulator
- Assistance with LLVM back-end development
- RTL design expertise

Our Timeline

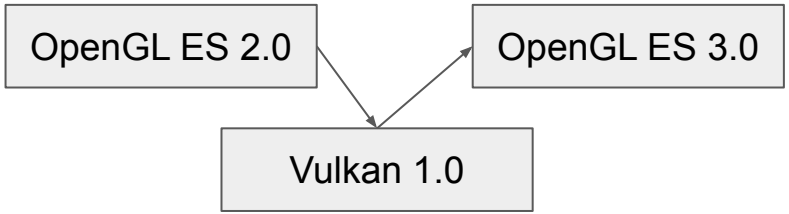
- Initial (alpha) ISA specification - socialize within the RISC-V community
- Software model development + driver and demos
- RTL development (for FPGA and then ASIC)

Roadmap

- We will adhere to a layered extension architecture in the spirit of RISC-V

- Initial support for OpenGL ES 2.0 (programmable shaders)

- Subsequently Vulkan and ES 3.0 (SPIR-V)

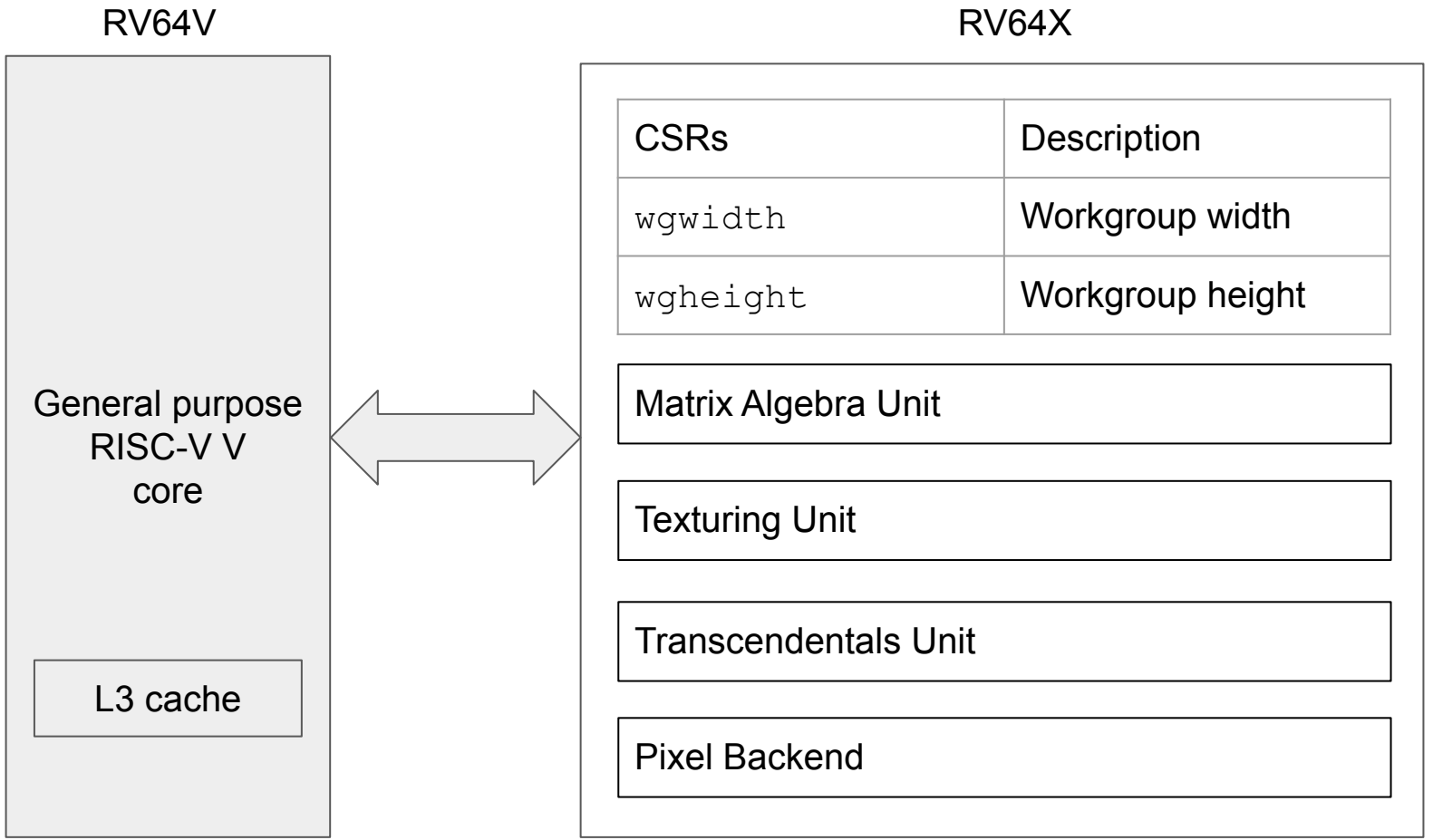


Technical Description of RV64X

Abel Bernabeu

RV64X requirements

- Support programmable graphics (OpenGL ES 2.0 or better)
- Close mapping to SPIR-V, used as IR in the shader compiler
- Prove the fused CPU-GPU ISA concept (unified instruction decoder)
- Fully preemptible
- Featuring a unified memory architecture


RV64V
RV64X
General purpose RISC-V V core
L3 cache
CSRs
Description
wgwidth
Workgroup width
wgheight
Workgroup height
Matrix Algebra Unit
Texturing Unit
Transcendentals Unit
Pixel Backend

Workgroups

Workgroup width and height and not just a vector length?

- Most obvious reason: OpenCL programming model
- Not so obvious: pixel shaders capable of different patch shapes.

Examples:

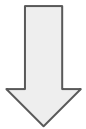
0,0	1,0	2,0	3,0
0,1	1,1	2,1	3,1
0,2	1,2	1,2	3,2
0,3	1,3	2,3	3,3

0,0	1,0
0,1	1,1

Matrix Algebra Unit

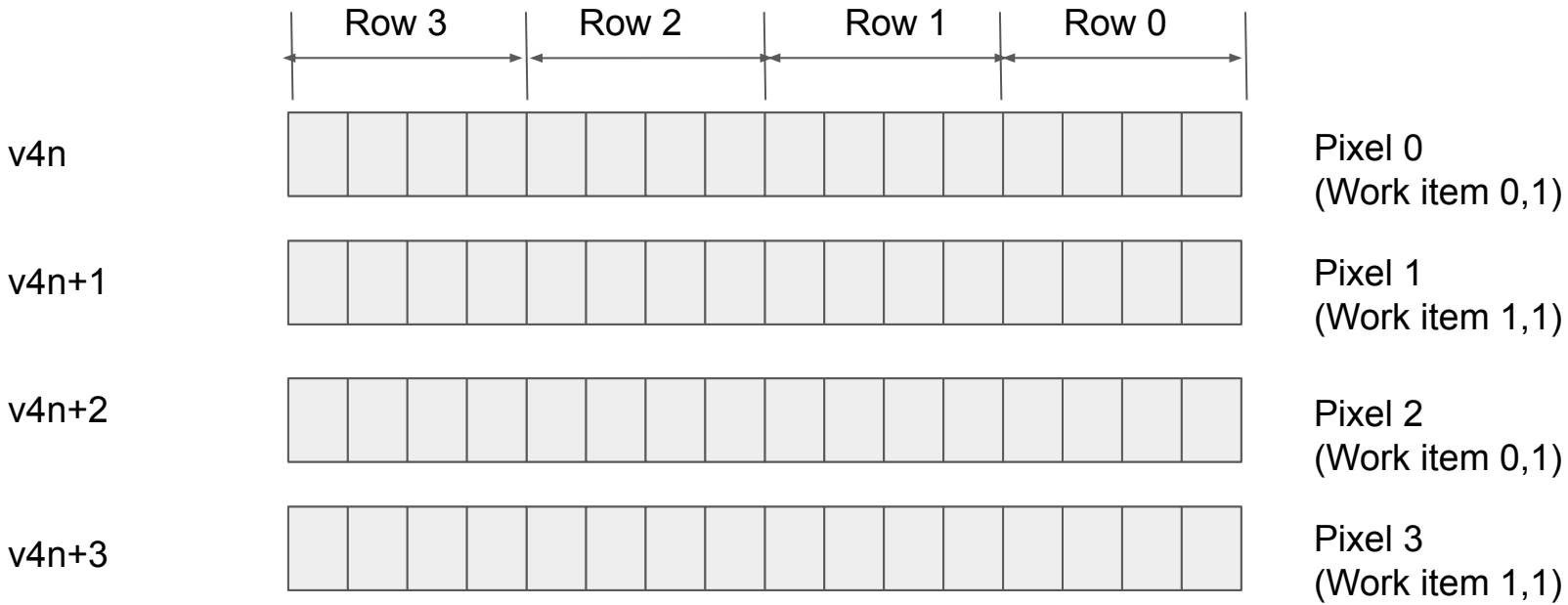
Going from a “vector” to a “matrix” architecture

Vector:	Common length for sources and destination operands



Matrix:	Source operands are two arbitrarily shaped matrices Destination has an implicit shape, that depends on the operation and the sources

Example: a 4x4 row major matrix per pixel




```
2048 bits register group
vtype: VLEN=512 bits, LMUL=4 regs, SEW=32 bits
```

mtype: matrix dimensions

RV64V CSR	RV64X CSRs
`vtype`	`(vtype, wgwidth, wgheight,` **`mtype`**`)`
Variable allocation	Variable allocation and dimensional structure

One product to rule them all

matrix x matrix
matrix x vector
vector x matrix
dot
scalar x matrix
scalar x vector

even a transpose!

`xmatmul`

```
for (i = 0; i < I; ++i) {
  for (j = 0; j < J; ++j) {
    acc = 0
    for (k = 0; k < K; ++k)
      acc += a(i,k) * b(k,j)
    c(i, j) = acc
  }
}
```

Texturing Unit

Texturing CSRs

CSRs	Description
`sdt`	Sampler desc table base address
`ivdt`	Image view desc table base address
`sampler`	Index into sdt for filtering settings
`imgview`	Index into ivdt for image view

Texturing instructions

Mnemonic	Description
`xlivd` `xlsd`	Image view descriptor and sampler load
`xsample`	Sample at a given coordinate and LoD
`xdpd{x|y}`	Partial derivatives
`xlod`	Implicit level of detail
`xproj`	Projection
`xtcinv`	Texture cache invalidation

Trascendentals Unit

Transcendentals

`xsin`	`xasin`	`xexp`	`xrcp`
`xcos`	`xacos`	`xlog`	`xrsq`
`xtan`	`xatan`	`xpow`	`xsqrt`

Pixel backend

Pixel backend (1/2)

CSRs	Description
`crtdt`	Color render target descriptor table base address
`dsrtdt`	Depth/stencil render target descriptor table base address

Pixel backend (2/2)

CSRs	Description
`globalx` `globaly`	Fragment position for upper-left pixel in patch

`stencilop_{front|back}`
`stenciltest_{front|back}`
`stencilref_{front|back}`
`stencilmask_{front|back}`
	Stentil op and test for {front|back} facing polygons

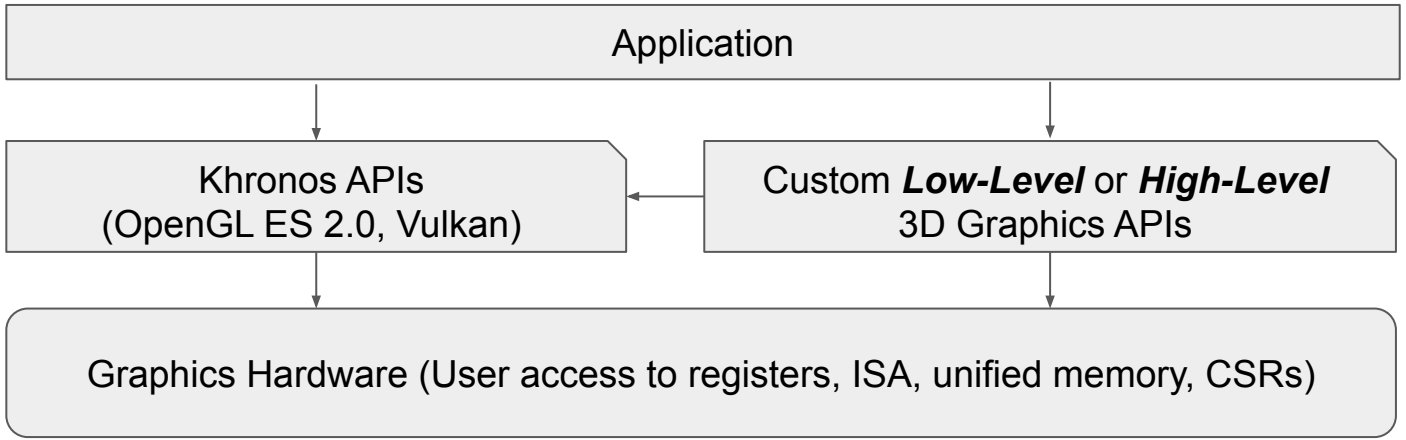
`blendop`	Current blending mode for color

Basic instructions	Instructions for custom blending

`xdstest vd # Depth/stencil test`
`xblend vs # Color blending`
`xlfragc vd # Loads gl_FragCoord`
	`xlcolor vd # Color load`
	`xscolor vs # Color store`

Software Architecture

Software Architecture



Key Advantage:

Access to registers, ISA, memory allows developers to *write their own low-level drivers*. This means custom rasterizers, compute processes and data structures that can be instantiated once and used by multiple shaders.

Summary

Value proposition for RV64X

- Fully Preemptible Architecture
- Unified Memory Design
- Open-Source Drivers
- Full Access to the ISA
- Combined CPU-GPU ISA (Full RISC-V Ecosystem Integration)
- Follows Khronos and other Standards

Acknowledgements

We wish to thank the following individuals for their support:

RV64X Team Members

Mick Thomas Lim
Peter Lieber

Advisor

Dr. Jon Peddie

Backup

mtype CSR layout

STYPE	Scalar type
LPWI	LHS: data per work-item?
LW	LHS: width (power of 2)
LH	LHS: height (power of 2)
LMAJ	LHS: row or col major (default row major)
LMOD	LHS: vreg, scalar splat or IDENTITY
RPWI	…
RW	…
RH	…
RMAJ	…
RMOD	...

RV64X encoding space

major opcode =1100111b (`jalr`), funct3 = 100b

22 bits per instructions

7 bits secondary opcode
5 bits destination
5 bits source 1
5 bits source 2