RISC-V Graphics ISA

Preliminary Design

Who we are?

- We are a group of enthusiasts who espouse powerful but low-cost open-source hardware for students, makers, commercial users and anyone else interested in customizable hardware

- Atif Zafar (Pixilica)
  - www.pixilica.com
- Grant Jennings (GOWIN Semiconductor)
  - www.gowinsemi.com
- Ted Marena (CHIPS Alliance and Western Digital)
  - www.chipsalliance.org

RISC-V Graphics ISA

- 3D graphics is now a standard part of many processor designs (Intel, ARM, Qualcomm, Samsung etc.) for consumer devices
- The RISC-V ISA is rapidly gaining industry backing due to its open-source license
- Some industry support for graphics
  - Imagination Technologies PowerVR
  - Libre Open-Source GPU effort

- A key part of the RISC-V ISA is that it is “extensible”

RISC-V ISA

- RISC-V has incremental modular ISA's that add functionality and complexity to a core design:
  - Base Integer: RV32I and RV64I
  - Add:
    - M: Integer Multiply/Divide
    - A: Atomic Instructions (for thread synchronization)
    - F: Single-Precision Floating Point
    - D: Double-Precision Floating Point
    - G: IMAFD (all of the above extensions combined)
    - C: Compressed Instruction Set (for atomic thread sync ops)
    - Q: 128-bit quad-wide floating point format

    - **V: Vector Extensions (proposed)**

Free & Open RISC-V Reference Card ①

Base Integer Instructions: RV32I, RV64I, and RV128I

Category	Name	Fmt	RV32I Base	+RV{64,128}
Loads	Load Byte	I	LB rd,rs1,imm	
	Load Halfword	I	LH rd,rs1,imm	
	Load Word	I	LW rd,rs1,imm	L{D|Q} rd,rs1,imm
	Load Byte Unsigned	I	LBU rd,rs1,imm	
	Load Half Unsigned	I	LHU rd,rs1,imm	L{W|D}U rd,rs1,imm
Stores	Store Byte	S	SB rs1,rs2,imm	
	Store Halfword	S	SH rs1,rs2,imm	
	Store Word	S	SW rs1,rs2,imm	S{D|Q} rs1,rs2,imm
Shifts	Shift Left	R	SLL rd,rs1,rs2	SLL{W|D} rd,rs1,rs2
	Shift Left Immediate	I	SLLI rd,rs1,shamt	SLLI{W|D} rd,rs1,shamt
	Shift Right	R	SRL rd,rs1,rs2	SRL{W|D} rd,rs1,rs2
	Shift Right Immediate	I	SRLI rd,rs1,shamt	SRLI{W|D} rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRA{W|D} rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI rd,rs1,shamt	SRAI{W|D} rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADD{W|D} rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDI{W|D} rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUB{W|D} rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Logical	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Compare	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Branches	Branch =	SB	BEQ rs1,rs2,imm	
	Branch ≠	SB	BNE rs1,rs2,imm	
	Branch <	SB	BLT rs1,rs2,imm	
	Branch ≥	SB	BGE rs1,rs2,imm	
	Branch < Unsigned	SB	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	SB	BGEU rs1,rs2,imm	
Jump & Link	J&L	UJ	JAL rd,imm	
	Jump & Link Register	UJ	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
System	System CALL	I	SCALL	
	System BREAK	I	SBREAK	
Counters	ReaD CYCLE	I	RDCYCLE rd	
	ReaD CYCLE upper Half	I	RDCYCLEH rd	
	ReaD TIME	I	RDTIME rd	
	ReaD TIME upper Half	I	RDTIMEH rd	
	ReaD INSTR RETired	I	RDINSTRET rd	
	ReaD INSTR upper Half	I	RDINSTRETH rd	

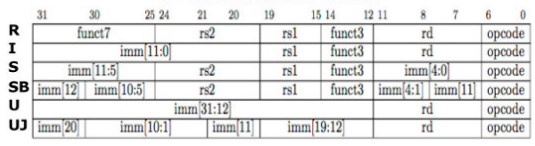
RV Privileged Instructions

Category	Name	RV mnemonic
CSR Access	Atomic R/W	CSRRW rd,csr,rs1
	Atomic Read & Set Bit	CSRRS rd,csr,rs1
	Atomic Read & Clear Bit	CSRRC rd,csr,rs1
	Atomic R/W Imm	CSRRWI rd,csr,imm
	Atomic Read & Set Bit Imm	CSRRSI rd,csr,imm
	Atomic Read & Clear Bit Imm	CSRRCI rd,csr,imm
Change Level	Env. Call	ECALL
	Environment Breakpoint	EBREAK
	Environment Return	ERET
Trap Redirect	to Supervisor	MRTS
	Redirect Trap to Hypervisor	MRTH
	Hypervisor Trap to Supervisor	HRTS
Interrupt	Wait for Interrupt	WFI
MMU	Supervisor FENCE	SFENCE.VM rs1

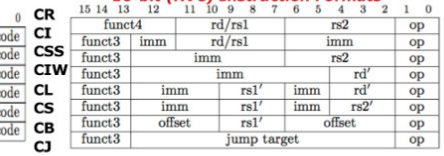
Optional Compressed (16-bit) Instruction Extension: RVC

Category	Name	Fmt	RVC	RVI equivalent
Loads	Load Word	CL	C.LW rd',rs1',imm	LW rd',rs1',imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Load Double	CL	C.LD rd',rs1',imm	LD rd',rs1',imm*8
	Load Double SP	CI	C.LDSP rd,imm	LD rd,sp,imm*8
	Load Quad	CL	C.LQ rd',rs1',imm	LQ rd',rs1',imm*16
	Load Quad SP	CI	C.LQSP rd,imm	LQ rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1',rs2',imm	SW rs1',rs2',imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Store Double	CS	C.SD rs1',rs2',imm	SD rs1',rs2',imm*8
	Store Double SP	CSS	C.SDSP rs2,imm	SD rs2,sp,imm*8
	Store Quad	CS	C.SQ rs1',rs2',imm	SQ rs1',rs2',imm*16
	Store Quad SP	CSS	C.SQSP rs2,imm	SQ rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Word	CR	C.ADDW rd,rs1	ADDW rd,rd,imm
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD Word Imm	CI	C.ADDIW rd,imm	ADDIW rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm	ADDI rd',sp,imm*4
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm
	Load Upper Imm	CI	C.LUI rd,imm	LUI rd,imm
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
Shifts	Shift Left Imm	CI	C.SLLI rd,imm	SLLI rd,rd,imm
Branches	Branch=0	CB	C.BEQZ rs1',imm	BEQ rs1',x0,imm
	Branch≠0	CB	C.BNEZ rs1',imm	BNE rs1',x0,imm
Jump	Jump	CJ	C.J imm	JAL x0,imm
	Jump Register	CR	C.JR rd,rs1	JALR x0,rs1,0
Jump & Link	J&L	CJ	C.JAL imm	JAL ra,imm
	Jump & Link Register	CR	C.JALR rs1	JALR ra,rs1,0
System	Env. BREAK	CI	C.EBREAK	EBREAK

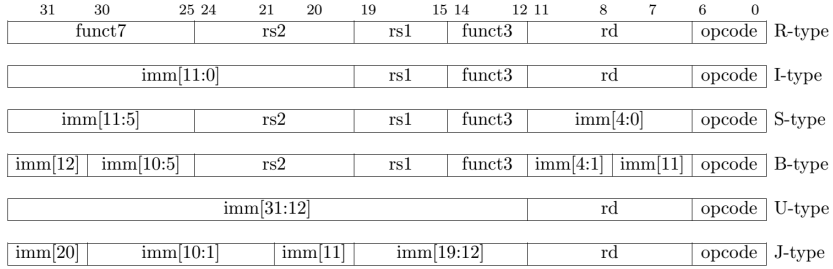
32-bit Instruction Formats



16-bit (RVC) Instruction Formats



RISC-V Integer Base (RV32I/64I/128I), privileged, and optional compressed extension (RVC). Registers x1-x31 and the pc are 32 bits wide in RV32I, 64 in RV64I, and 128 in RV128I (x0=0). RV64I/128I add 10 instructions for the wider formats. The RVI base of <50 classic integer RISC instructions is required. Every 16-bit RVC instruction matches an existing 32-bit RVI instruction. See risc.org.



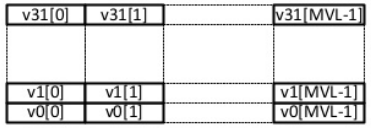
Proposed V Extension State

Standard RISC-V scalar x and f registers

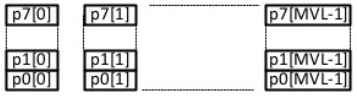
Vector configuration CSR vcfg

Vector length CSR vlr

Up to 32 vector data registers, v0-v31, of at least 4 elements each, with variable bits/element (8,16,32,64,128)



MVL is maximum vector length, implementation and configuration dependent, but MVL >= 4



8 vector predicate registers, with 1 bit per element

RISC-V Vector Extensions

- Instructions are defined for vector:
  - Data Movement
    - Move/load/store
  - Arithmetic
    - Add/Sub
    - Mul/Madd/Msub/Div
    - Min/Max/Clip
    - Sqrt/Dot prod
  - Logical ops
    - Shift/Rotate
    - And/Or/Xor/Merge
  - Branching
    - Gr/Le/Eq

- https://github.com/riscv/riscv-v-spec

RISC for Graphics

- We propose a new set of ***graphics*** instructions designed for 3d graphics and media processing
- *These new instructions build on the base vector instruction set. We **add** support for new data types that are graphics specific.*
- We will denote this as the “X” extension so it doesn’t interfere with other RISC-V nomenclature.
- Advantage is this is a ***fused CPU-GPU ISA***
- We call this **RV32X**

RV32X

- Motivation and Goals:
  - We want small, area-efficient designs
  - Custom programmability and extensibility
  - Low cost of IP ownership and development
  - Does not compete with commercial offerings
  - FPGA and ASIC targets
  - Free and open-source
  - Targeted to low-power microcontrollers
  - Strive to be DirectX (Shader Model 5) and OpenGL/ES and Vulkan compliant as we progress development

RV32X ISA

- Instructions will be 32-bits long (compliant with RISC-V ISA)
- Programming model is SIMD with some scalar operations for special functions
- Hardware will be a graphics vector accelerator attached to a host RISC-V core
- Instruction decode will happen partially on the host and partially in the accelerator using a special bit field in the instruction
- Architecture will support FPGA and ASIC designs as with the base RISC-V ISA
  - 16-bit fixed point (ideal for FPGAs)
  - 32-bit floating point (ASICs or FPGAs)
- Architecture defines a set of:
  - Hardwired base graphics and media instructions
  - User-Configurable RAM based micro-coded instructions for a run-time application-defined ISA extension

RV32X Data Types

- Scalars (8, 16, 24 and 32 bit fixed and floats)
  - Transcendentals (sincos, atan, pow, exp, log, rcp, rsq, sqrt etc.)
- Vectors (RV32-V)
  - We will support 2-4 element (8, 16 or 32 bits/element) vector operations along with specialized instructions for a general 3d graphics rendering pipeline
  - Points, Pixels, Texels (essentially “special” vectors)
    - XYZW points (64 and 128 bit fixed and floats)
    - RGBA pixels (8, 16, 24 and 32 bit pixels)
    - UVW texels (8, 16 bits per component)
    - Lights and Materials (Ia, ka, Id, kd, Is, ks…)
- Matrices
  - 2x2, 3x3 and 4x4 matrices will be supported as a native data type along with memory structures to support them
  - Attribute Vectors (XYZWRGBAUVWHNxNyNz…)
    - Essentially represented in a 4x4 matrix

RV32X Register Set

- 136-bit “configurable” vector registers
  - Splits: 4x32b, 8x16b or 16x8b
  - 8-bit Configuration bits define data types:
    - Pixels (RGBA)
    - Points (XYZW)
    - Vectors (2,3,4 components)
    - Matrix stacks (16 component across registers)
    - Lighting, Viewing, Projection Parameters
    - Texture Coordinates
    - Pixel and Frame-buffer Operations (ROPs, bitblt)
    - Attribute Arrays and Differentials
    - Scalar constants and variables (i.e. math functions)
    - Any other user defined types
      - Have 256 possible data types that can be application defined
      - For example AOS or SOA formats (ArrayOfStructure or StructureOfArray)
- Register Files will support random access by register name or using a push-pop FIFO like functionality.

RV32X Vector/Math Instructions

- Vector/Matrix Processing: 2,3,4 components
  - SetVec/SetMat
  - Push/Pop (vec/mat)
  - MatAddSub/MatMul
  - VecMat/ScalarVec
  - Dot/Cross
  - Dist/Len
  - Trans/Inv/Det/Norm
  - Swz (swizzle components, bits)
  - Lerp/Slerp

- Transcendental Math (scalar)
  - Sincos, atan, exp, pow, log, rcp, rsq, sqrt, cordic
  - Min/Max/Rnd/Floor/Ceil/Lerp/Slerp

RV32X Pixel/Texture Instructions

- Pixel Instructions
  - SetPix/ClrPix/GetPix
  - Blend
  - Ztest
  - ROP
- Texture Instructions
  - Tex2d, Tex3d
  - TexEnv
  - TexGen
  - MipMap
  - Persp
  - TexLoad
  - TexCodec

RV32X Frame Buffer Instructions

- Frame Buffer Instructions
  - SetZ/ClrZ
  - SetArea/ClrArea
  - Sync/Scanout
  - Compress/Decomp
  - BitBlt
  - Improc
  - ConfigBuffer

- Frame Buffer itself can be configured:
  - Pixel Buffer, Geometry Buffer, Texture Buffer, A-Buffer, etc.

RV32X Graphics Instructions

- Optional Graphics Instructions (micro-coded)
  - ModelView
  - Backface
  - Lookat
  - Proj
  - Clip2/Clip3
  - Lit (a,d,s)
  - Persp
  - InterpStep (i.e. Bresenham DDA or scanline attributes)
  - Window
  - TexMap
  - Z-Test
  - AlphaBlend
  - FragMerge

- MicroCode Instructions
  - LoadMicroCode (instruction to load custom micro-instructions into ucode RAM)
  - ClearMicroCode

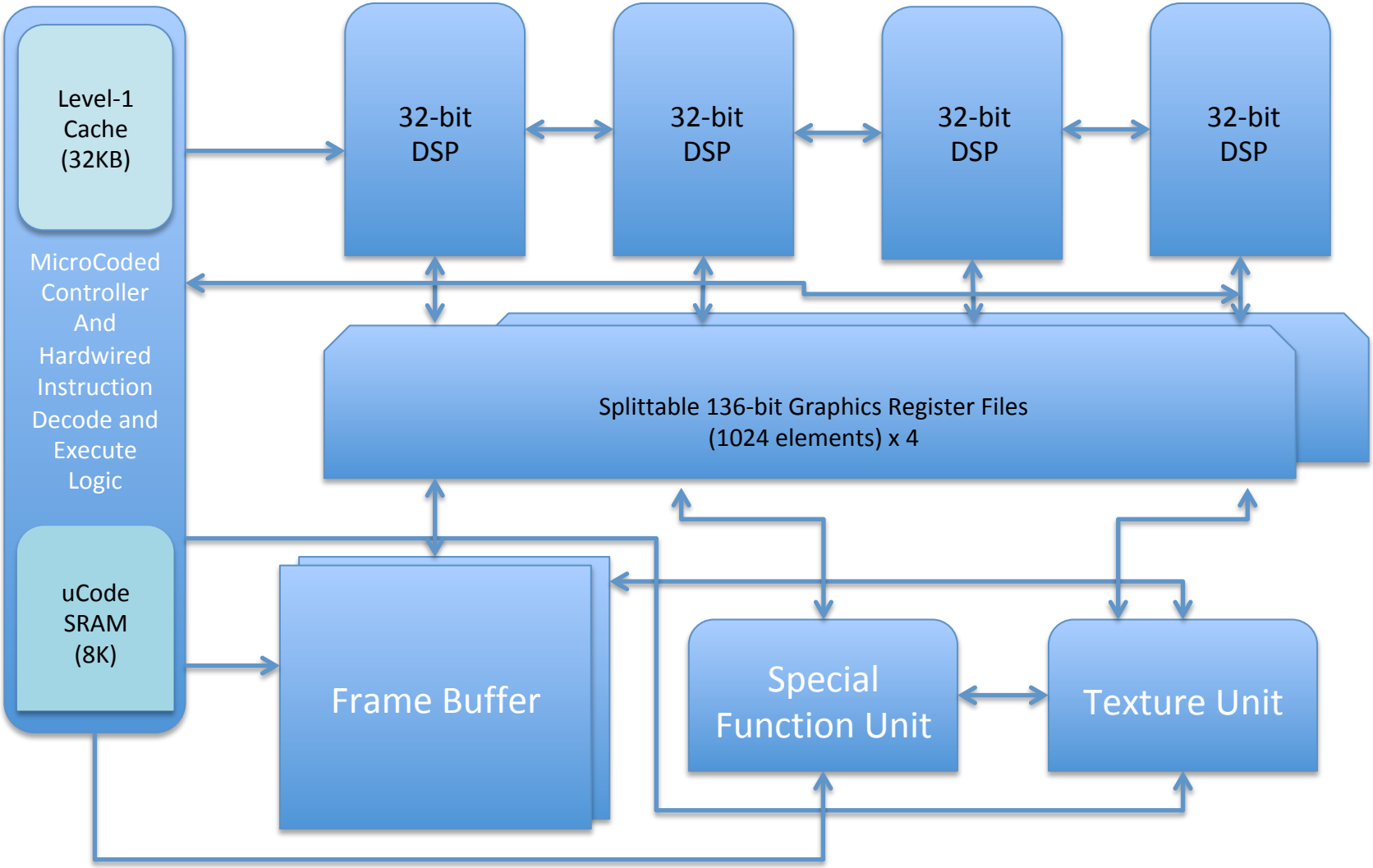
Advantages of Fused CPU-GPU ISA

- Can implement a standard graphics pipeline in microcode
- Support for Custom Shaders
- Can implement Ray-Tracing extensions
- Vector support for Numerical Simulations
- 8-bit integer data types for AI/Machine Learning
- Can implement Custom rasterizers
  - Splines
  - SubDiv Surfaces
  - Patches
- Can implement Custom pipeline “stages”
  - Custom geometry/pixel/frame buffer stages
  - Custom tessellators
  - Custom instancing operations

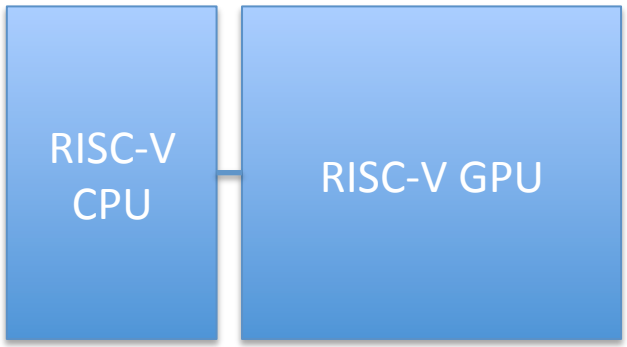
RV32X Reference Implementation

- Instruction/Data SRAM Cache (32KB)
- Microcode SRAM(8KB)
- Dual Function Instruction Decoder
  - Hardwired implementing RV32V and X
  - Micro-coded Instruction Decoder for custom ISA
- Quad Vector ALU (32 bits/ALU – fixed/float)
- 136-bit Register Files (1K elements)
- Special Function Unit
- Texture Unit
- Configurable “local” Frame Buffer

RV32X Hardware

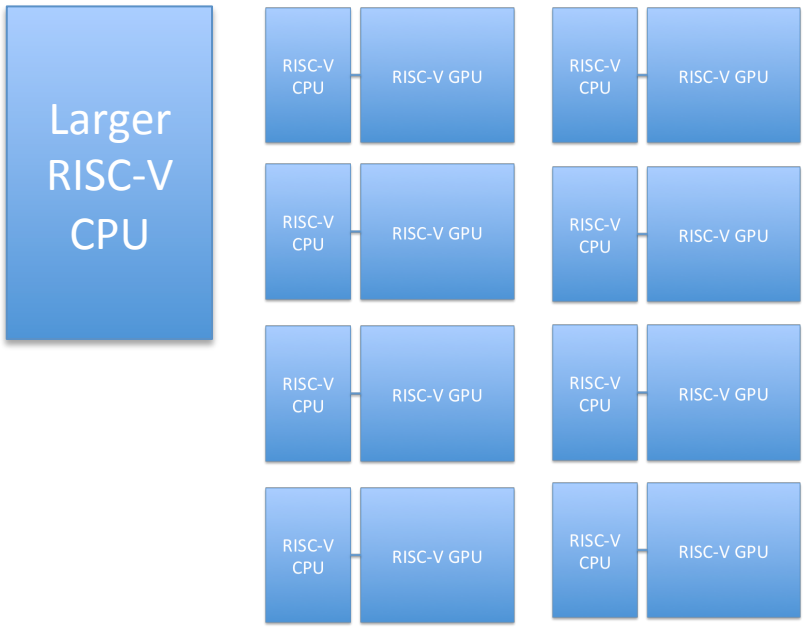


Scalable Design



Stand-Alone Low-End Graphics Microcontroller

or



Used as “shaders” in a multicore design

Novel Ideas

- Fused unified CPU-GPU ISA
- Configurable registers for custom data types
- User-defined SRAM based micro-code for application defined custom hardware extensions
  - Custom rasterizer stages
  - Ray tracing
  - Machine Learning
  - Computer Vision
- Same design serves both as a stand-alone graphics microcontroller or scalable shader unit
- Data formats support FPGA-native or ASIC implementations

Key Points and Next Steps

- This is a very early spec in development, subject to change based on stakeholder and your input. We will have a discussion forum set up.

- Immediate goal is to build a sample implementation
  - Instruction Set Simulator
  - FPGA implementation using open-source IP
  - Custom IP designed as open-source project
  - Demos and Benchmarks

- If anyone interested in helping with this please reach out to one of us. Thank you!

Thank You! Questions?

Atif Zafar

Atif@Pixilica.com