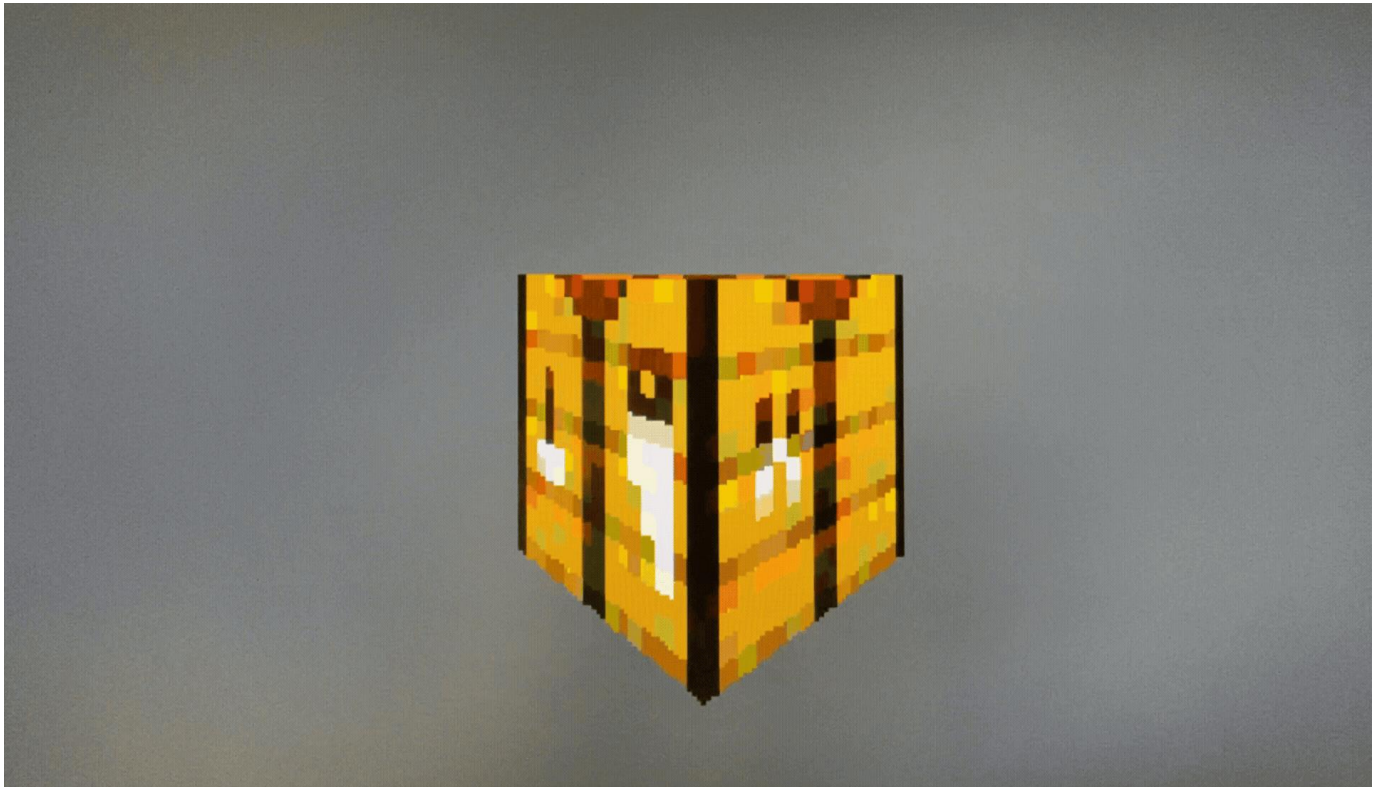
Embedded 3D GPU Design

SENIOR DESIGN GROUP 24, MAY 2026: Digital ASIC Fabrication

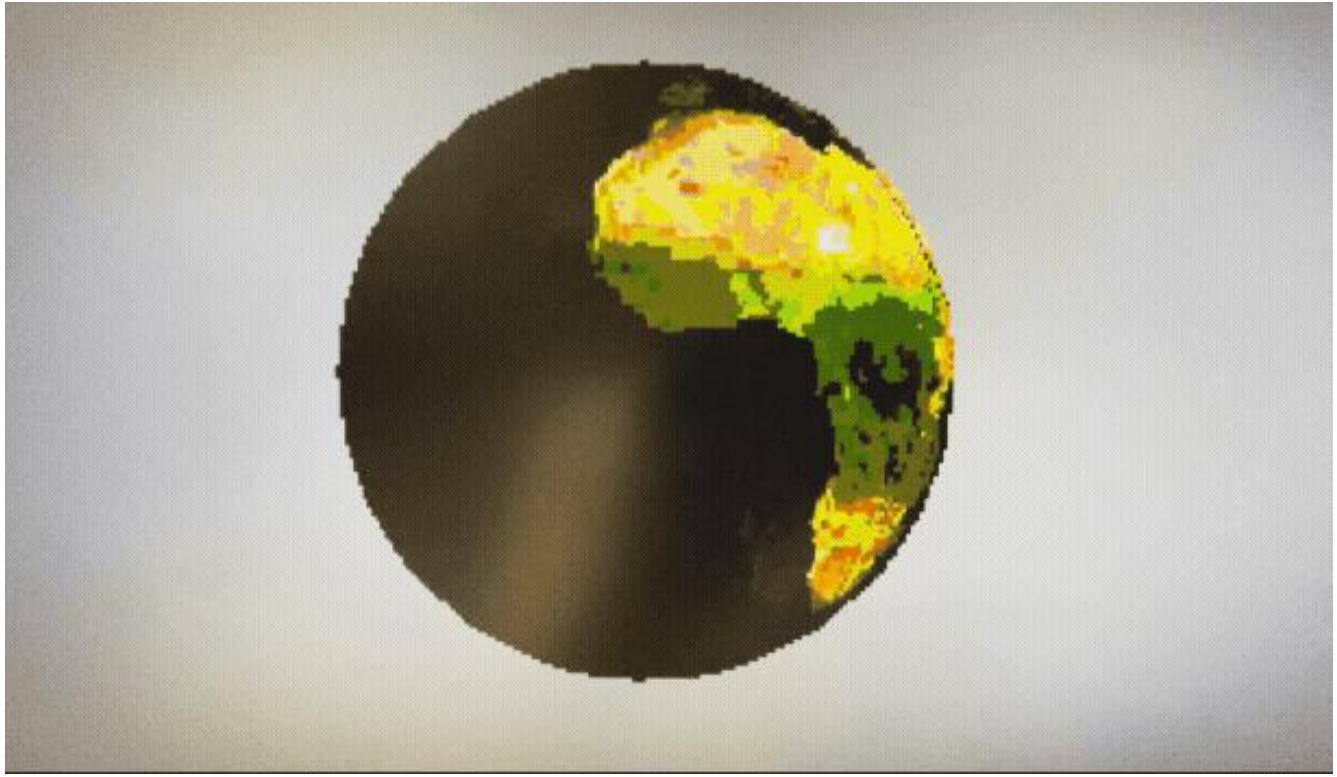
Colin McGann, Michael Drobot, Jack Tonn, Dawud Benedict,
Joshua Arceo, Samuel Forde, Emil Kosic

IOWA STATE UNIVERSITY

This... is a Crafting Table

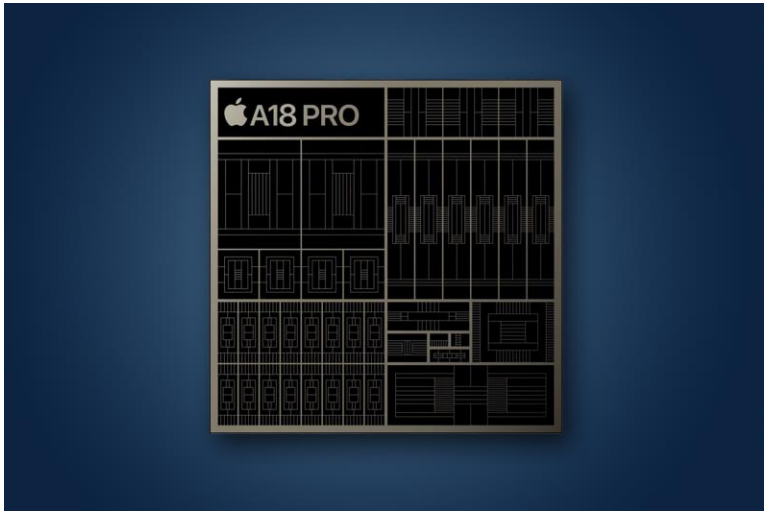


And this... is the World



Problem Context

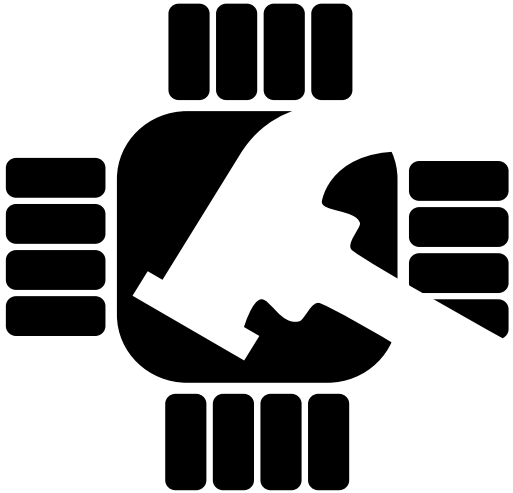
- GPUs are the backbone of the tech industry
  - AI, parallel computing, gaming
- Embedded GPU use increasing
    - Mobile Devices
    - Edge Computing and AI
    - Autonomous Systems
- Considerable gap between early hardware architecture courses and advanced hardware design courses



Apple A18 Pro Chip

µGPU

- Small footprint open-source educational GPU
- Utilizes Iowa State Chip Forge club infrastructure
- Gives students an opportunity to explore application-specific integrated circuit (ASIC) design
- Supplements hardware design for graphics
- Designed to support a variety of workloads beyond rasterization



Chip Forge Club Logo

- Identified user groups
  - Embedded GPU enthusiasts
  - Chip Forge members
  - ISU faculty

Requirements

- Functional Requirements
  - Render textured 3D models at 24Hz
  - Support 320 x 240 resolution
  - Output to a monitor over VGA
- Fabrication Requirements
  - Written in Verilog HDL
  - Max core clock frequency of 50 MHz
  - Pass DRC & LVS for the SkyWater 130nm process
  - Fit design within 3.0 x 3.6 mm die

Suzanne Test Model

Dataflow

Model

Vertices and Connectivity

Vertex Shader Program

Placing objects in the world and projecting them onto the screen

Rasterizer

Checking visibility, applying textures

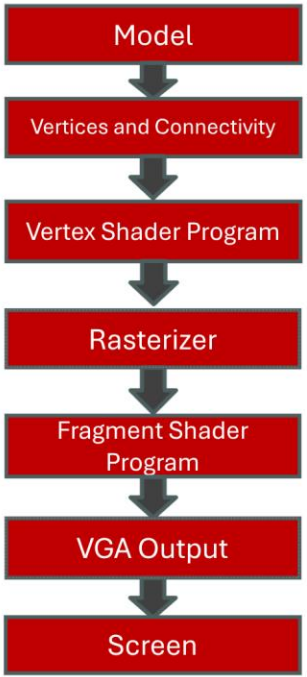
Fragment Shader Program

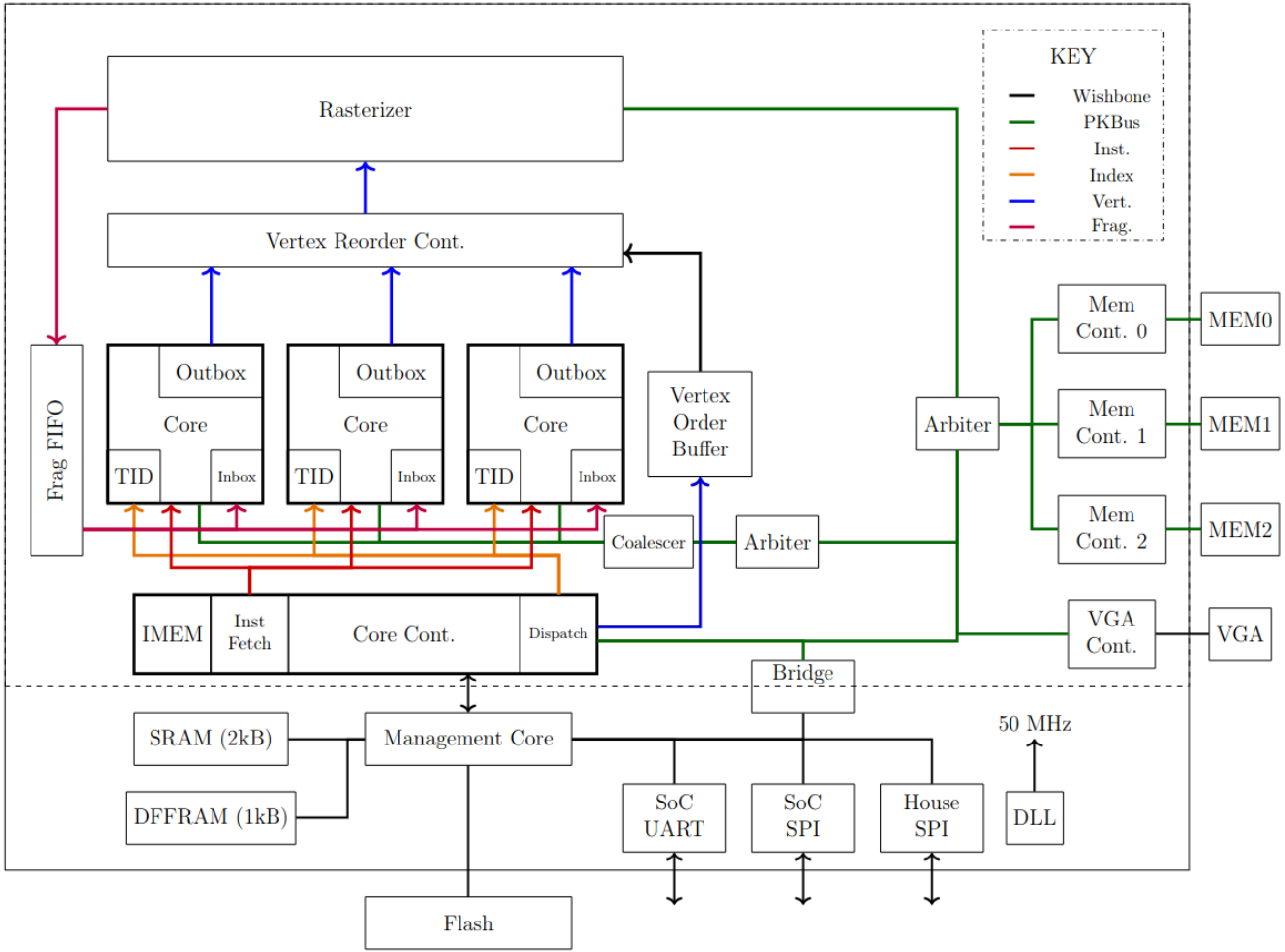
Lighting, smoothing, writing the frame

VGA Output

Screen

Design – High Level





Design - ISA

- Custom shader core ISA similar to MIPS or RISC-V
- Designed to implement a software ray tracer
  - A good test for supporting as many use cases as reasonably possible
- Uses instruction predication instead of branching
- Supports complex conditionals, loops, and procedure calls
- Custom assembler using CustomASM
- Custom Python simulator used for software testing

add	and	xori	srlv	mac	spltu	srltu	jump
addi	andi	sll	srav	maccl	**clrp**	lw	jal
sub	or	srl	lui	macrd	**spr**	lb	**jret**
mul	ori	sra	lli	speq	sreq	sw	**halt**
muli	xor	sllv	out	splt	srlt	sb	in

List of supported instructions

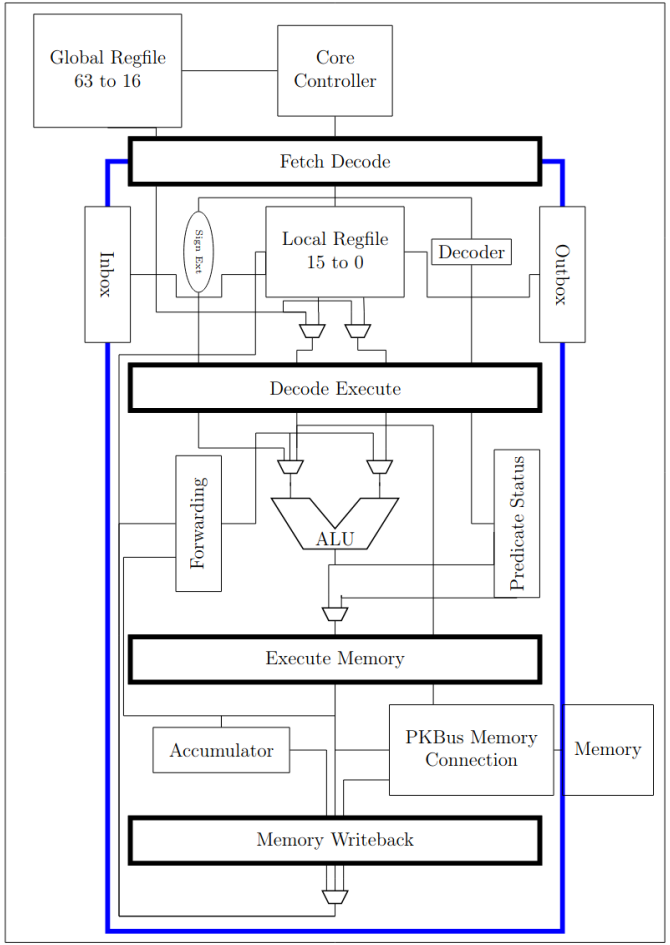
```
if (r0 == r1) {
    r10 = r10 + r11;
} else if (r0 == r2) {
    r10 = r10 << 2;
    r11 = r10;
} else {
    r10 = 0;
}
```

```
; Assume predicate bits are initially 000
; and r0, r1, r2, r10, r11 have different
; data on each core.
()    speq $p0,  $r0,   $r1
(001) add  $r10, $r10,  $r11 ; if (r0 == r1)
(000) speq $p1,  $r0,   $r2
(010) sll  $r10, $r10,  2    ; else if (r0 == r2)
(010) addi $r11, $r10,  0
(000) addi $r10, $zero, 0    ; else
```

Figure 9: Simple predicate example

Design - Shader Cores

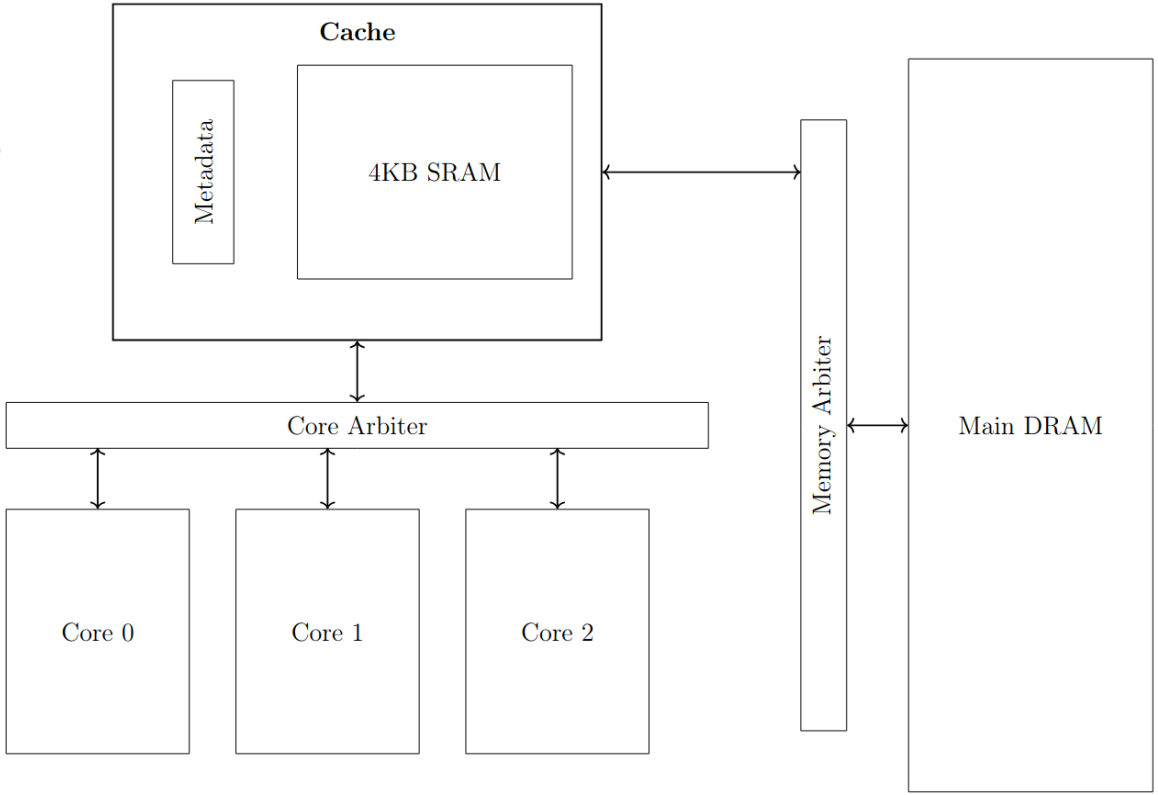
- 5 stage pipeline
  - Core handles decode, execute, memory, writeback
- Each core has a local regfile with a size of 16 words
- Predicate status handles instruction predication at the core level
  - 3 predicate bits
- ALU supports add, sub, single cycle fixed point mult, shift, and bitwise operations
  - Software has a divide procedure



Design – Memory Hierarchy

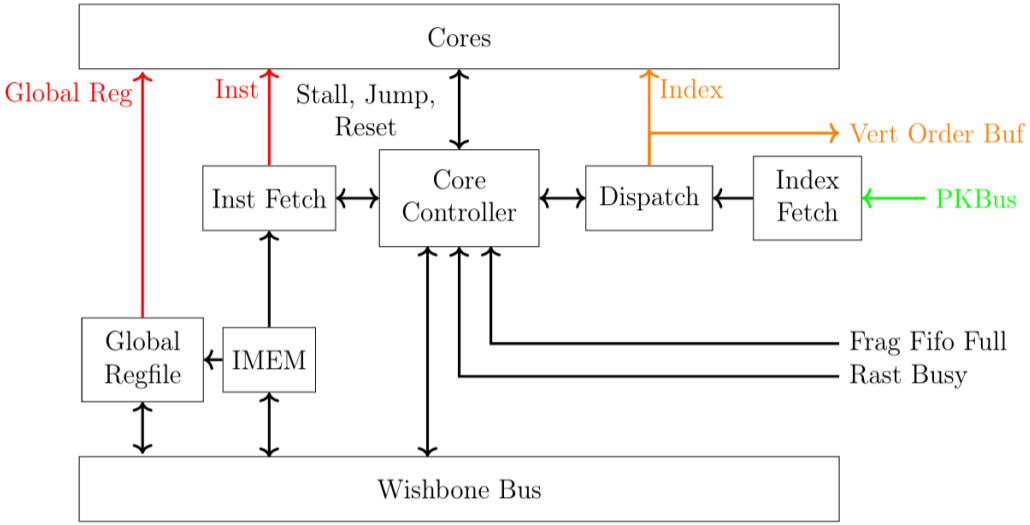
A shared L1 data cache for all cores

- High spatial locality of indices and pixels.
- Moderate temporal and spatial locality of vertices
- ~90% Hit Rate for pixel writes to frame-buffer
  - *3 cores, cube model
- ~15% increase in FPS
  - *3 cores, shaded cow model



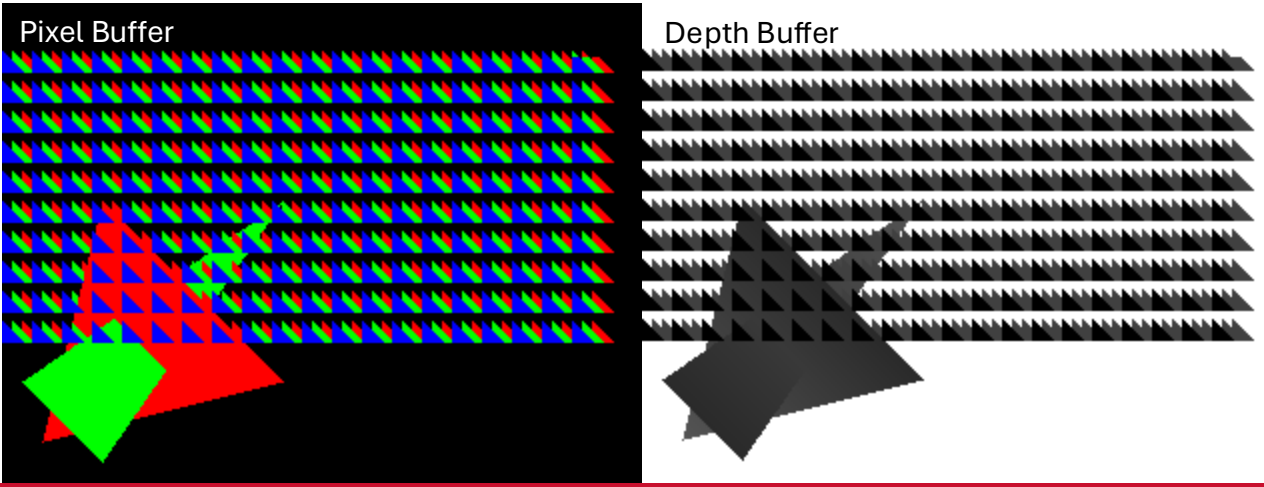
Design - Core Controller

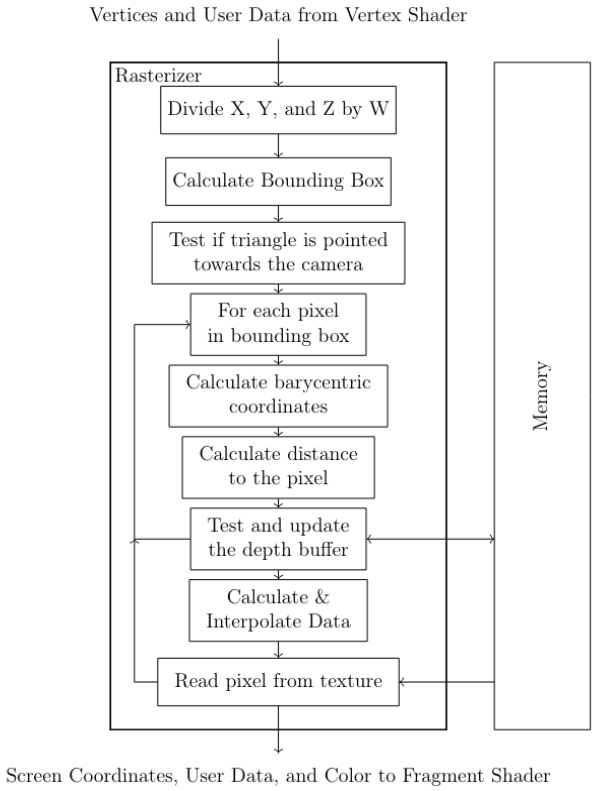
- Fetches instructions and handles the PC
  - Contains the IMEM, call stack, and jump logic
- Contains the global register file
- Dispatches jobs to the shader cores
- Interfaces between the management core and shader cores
- Handles switching between vertex and fragment shading, deadlocks



Design - Rasterizer

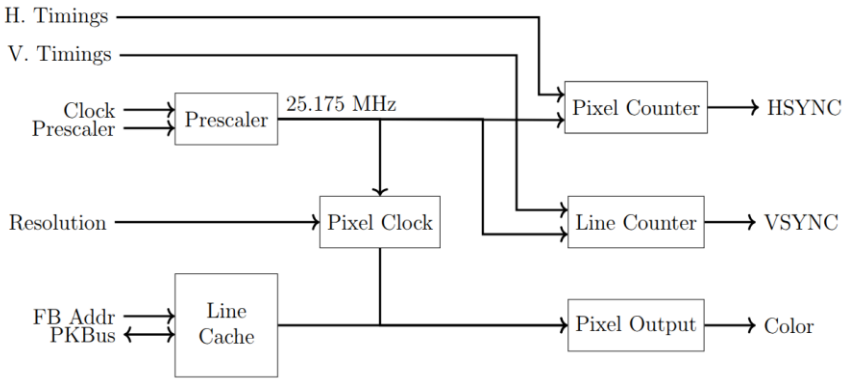
- Translates sets of 3 vertices to the collection of pixels that would fill the triangle
- Handles triangles at different distances using a depth buffer





Design - VGA

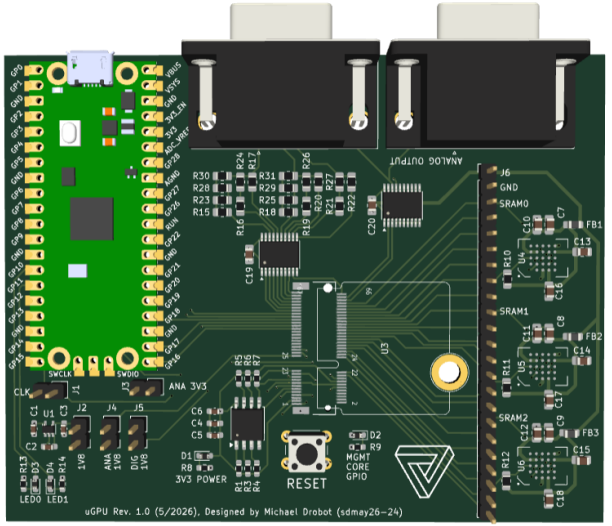
- Generates VGA HSYNC and VSYNC signals
- Outputs color data
- Outputs only 640 x 480 @ 60Hz (standard VGA timings)
    - Supports subsampling down to 320 x 240, 160 x 120, or 80 x 60
- Interfaces with analog DAC

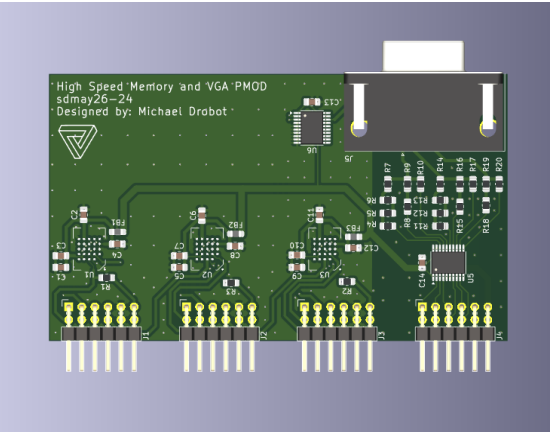


VGA Block Diagram

Design - PCB

- Contains the memory chips, VGA DAC, and connectors
- MemoryVGAPmod
  - Designed for our Arty A7-100T FPGAs with a quad PMOD interface
- uGPU
  - Supports bringup with the Caravel M.2 carrier
  - Contains jumpers for analog and digital power
  - Uses the Chip Forge Pi Pico interface
  - Allows for on chip and off chip VGA DACs





Right: MemoryVGAPmod, Top: uGPU

Simulation Testing

- All Verilog modules are functionally verified with SVUnit testing framework
  - Tests have randomized input vectors to maximize the coverage
- High-level testing done in Caravel DV
- All modules are test-hardened before merge
- ISA verified with Python simulator

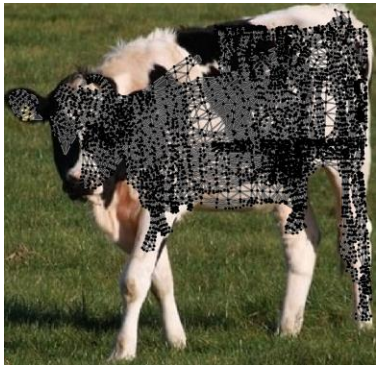
Model	Model Name	Author	Num. Triangles	Reqd. FPS
	Cube	Michael Drobot	12	24
	Suzanne [6]	Blender Foundation	500	24
	Cow [4] [8]	Viewpoint Animation Engineering, Sun Microsystems	5804	24
	Utah Teapot [9]	Martin Newell, Univ. of Utah	6320	24

Subset of Test Models

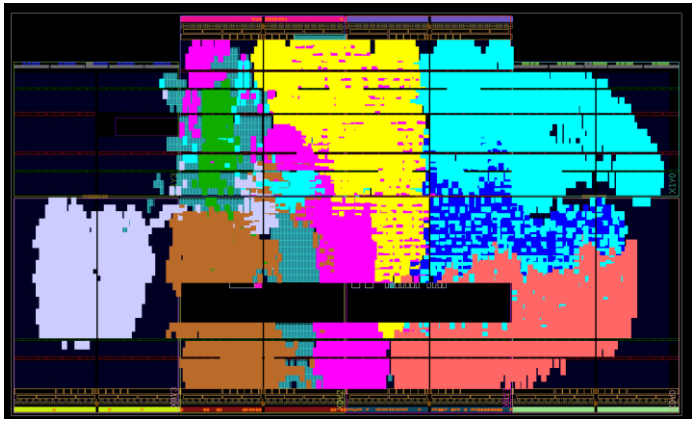
A Commonly Used Test Model

FPGA Testing

- Final µGPU design tested on Chip Forge Arty A7-100T FPGAs
- Passes Sky130A DRC, LVS, and ChipFoundry precheck
- Test models were textured by hand, and converted to C header files containing vertices, triangles, normals, and texture data via custom conversion script



UV Texture Map for Cow



Arty A7-100T Vivado Usage

Test Results

Demo	Num Triangles	Target FPS	Measured FPS
Cube	12	24	22.2
Suzanne	967	24	9.37
Cow	5804	24	3.7
Teapot	6320	24	3.3
Cheburashka	13334	24	1.7
Bunny	69663	10	0.3
Dragon	871414	2	DNF

*Reference screenshots taken in Blender

Test Results

Cube	Suzanne	Cow	Teapot	Chebureshka	Bunny
22.2 FPS	9.37 FPS	3.7 FPS	3.3 FPS	1.7 FPS	0.3 FPS

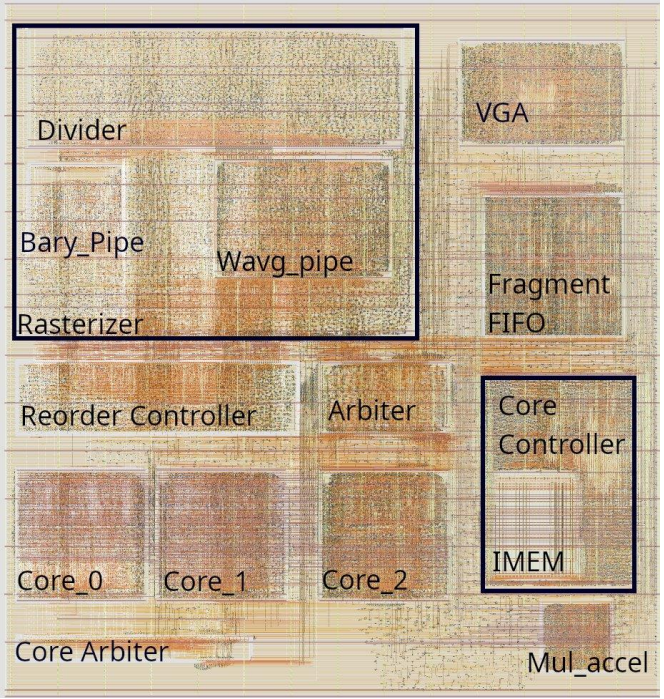
Hardening

- OpenLane hardening flow
  - Synthesis
  - Routing and Optimizations
  - DRC & LVS
  - Timing verification
- OpenROAD GUI was used to verify that PDN layers were correctly synthesized.
- Total area of hardened macros were minimized using our custom python optimization script.

Module	Dimensions (μm)	Area (mm^2)
Rasterizer	1700x1300	2.21
Fragment FIFO	800x800	0.64
Core Controller	650x900	0.585
VGA	700x450	0.36
Cores (Each)	550x550	0.3025

Area Table

Floor Planning

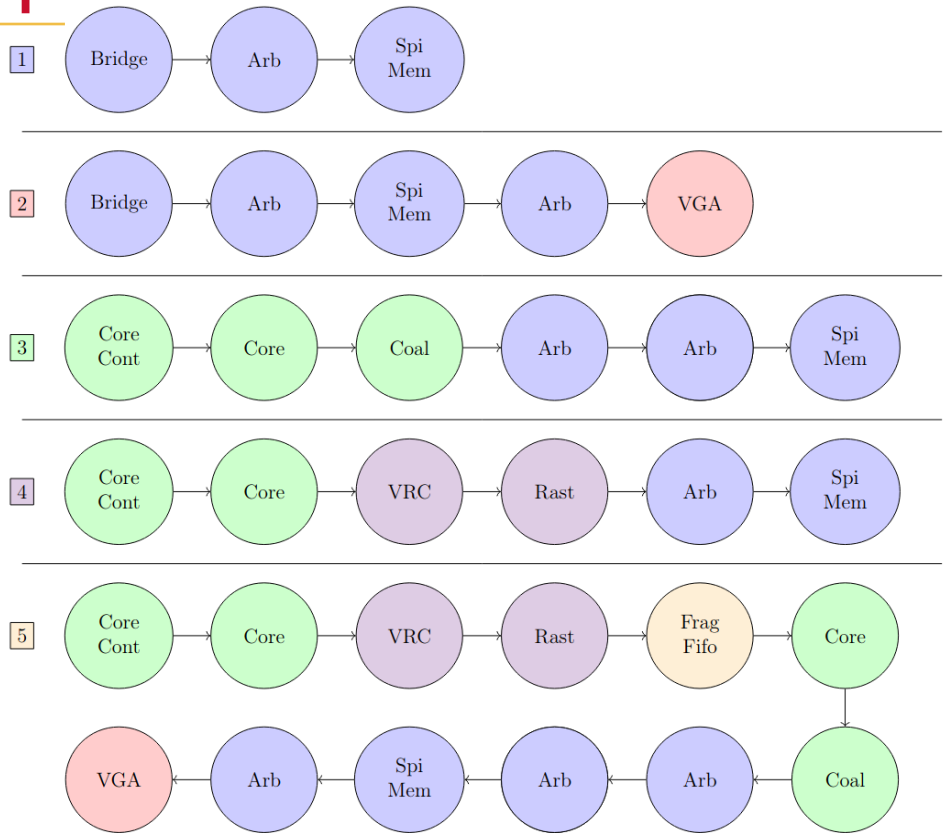


Top level layout

- We decided to synthesize our top level with 3 cores
  - This was due to having to share die space and lack of time before tapeout
- Issues with the open source toolflow
  - Routing tools can be hard to guide
  - Sub-macros can be synthesized with issues which are not discovered until integration

Post Fabrication Bringup

- Identified 5 chains of modules to test
  - Each chain given a color to represent which modules it tests
- If system does not work, this can point to which module is failing
- Chain 5 shows the entire dataflow of the system
- C code has been written for each distinct chain
- C code has been verified with FPGA

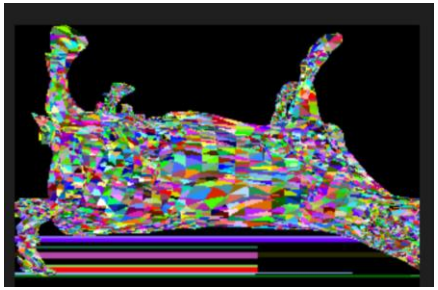


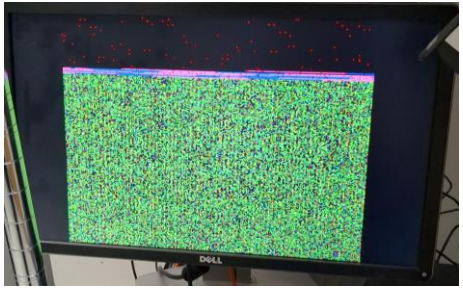
Conclusions

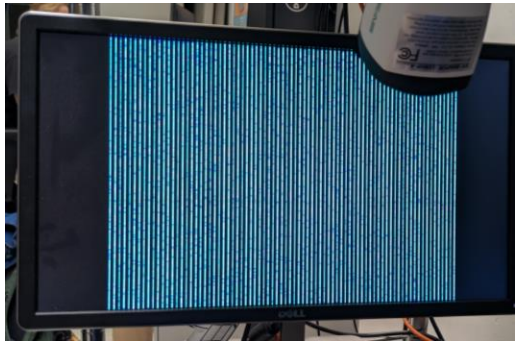
- Design is fully tested and working on FPGA
- Payment has been deposited and design and will be taped out on May 13th, 2026
- Post-fabrication bringup will be done by returning members of senior design and ChipForge club members

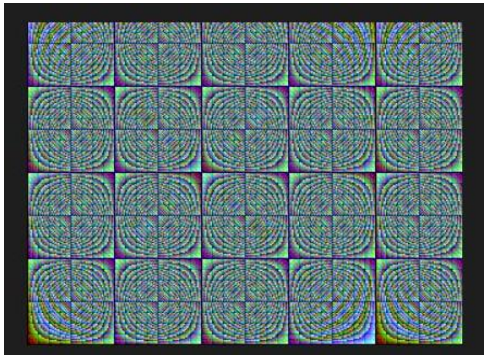
FPGA Test Setup Running A Demo of a Familiar Cat

Notable Test Renders

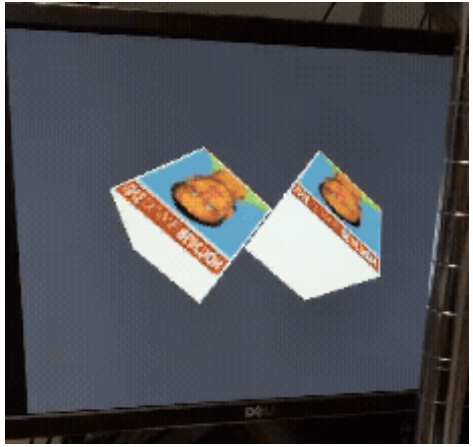


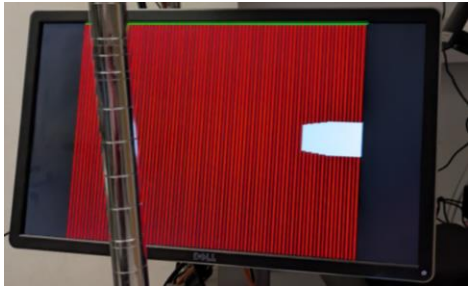


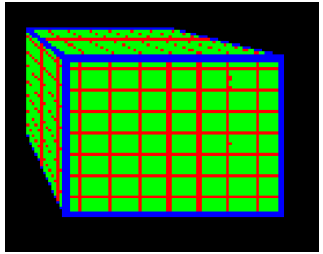




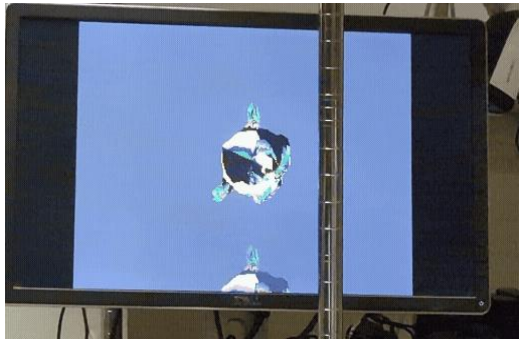
Notable Test Renders

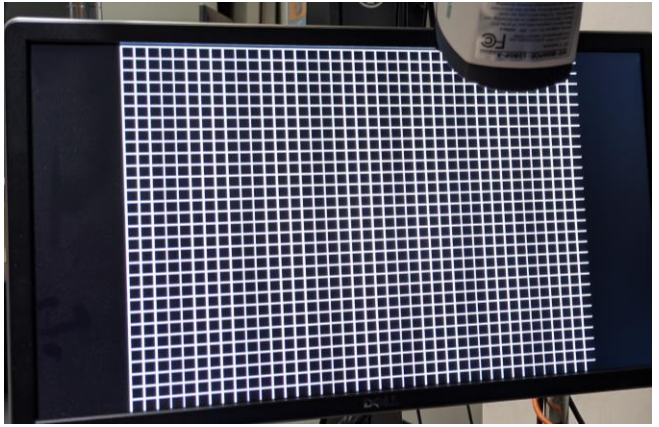


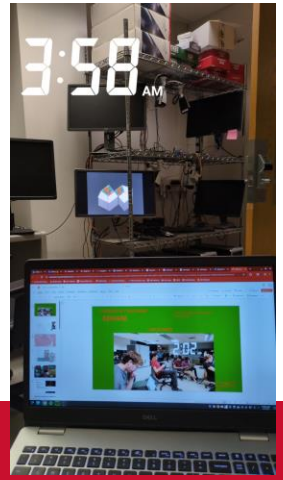




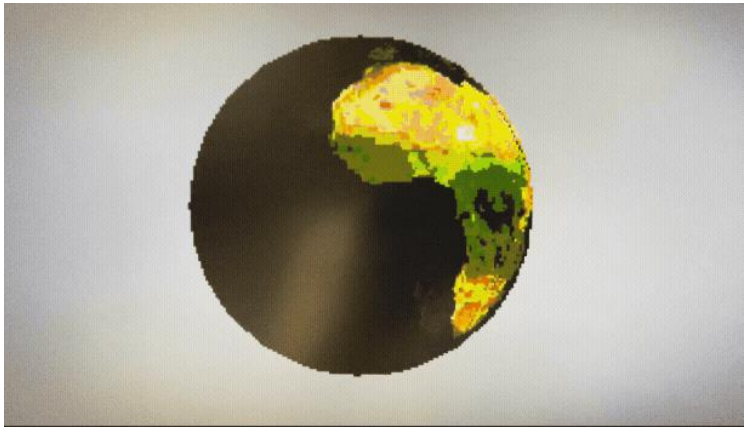
Notable Test Renders







Notable Test Renders



Notable Test Renders