Simple GPU

Anthony Bogedin Michael Lohrer

GPU Architecture





Matrix Multiply Simulation

🕼 dk	0											
1 rst	0											
1∰ s	0											
Va d	0											
🖭 📲 Xin[15:0]	1.0						1.0					
🖽 📲 Yin[15:0]	1.0						1.0					
🖽 📲 Zin[15:0]	1.0						1.0					
🖽 📲 Win[15:0]	1.0						1.0					
😐 📲 Xout[31:0]	1.53125	0.0)	1.54	6875)					1.53	125	
🖽 📲 Yout[31:0]	-3.90234375	0.0)	-0.019	53125	-1.96	09375				-3.90234375	
🖭 📲 Zout[31:0]	-6.89453125	0.0)	-1.00	78125	-2.730	46875	-3.00	78125 X		-6.89	453125
🖭 📲 Wout[31:0]	-5.73046875	0.0)	-0.01	.5625)	-1.44	53125	-0.01	.5625 🗙		-5.73	046875

Matrix Under Test:

-0.0195	-1.0078	-0.0156
-1.9414	-1.7227	-1.4297
-1.9414	-0.2773	1.4297
0.0	-3.8867	-5.7148
	-0.0195 -1.9414 -1.9414 0.0	-0.0195-1.0078-1.9414-1.7227-1.9414-0.27730.0-3.8867

Matlab Output Comparison:

	Х	Y	Z	W
Vertex:	1.5313	-3.9023	-6.8945	-5.7305

Divider

Used the pipelined divider provided by Llamoca, but placed a wrapper around it to handle negative numbers and overflow To get a fractional output, W was truncated by 12 bits ([32 16]/[20 4] = [32 12])

The X, Y, & Z components of the vertex need to be normalized before being mapped to screen space

Dividing all three by W ensures that all values are between -1 and 1 if they are on screen

Since the matrix multiply takes 4 clock cycles, X, Y, & Z are feed into the divider one at a time

After division, they are stored in 3 registers to await processing in the next stage

Divider Tests

Example value with : 1.53125 [32 16] / -5.6875 [20 4] = -0.26904296875 [32 12]

= FFFFBB2 [32 12]

Result from divider: -0.26611328125 [32 12] = FFFFBBE [32 12]

🖬 📲 numerator[31:0]	1.53125	0.0					.53125			
🖽 📲 denominator [19:0]	-5.75	0.0					-5.75			
🖓 dk	0									
¼ rst	0									
U enable	0									
🖬 🔮 result[31:0]	ffffbbe	0000	X X X	XXXXX	XXXXX.	XXXXX.	XXXXX	XXXXX.	() (7f)	fffffbbe
We valid	0									
										1

Camera to Screen Space

Since Xcam and Ycam are from -1 to 1

Just multiply by half of width and height

Truncate fractional bits

Add half of width and height



Clip Detection If the input X and Y coordinates are within the screen dimensions

and there is a valid

vertex, set visible high



Screen to Array Index

frame_address = Xpix + width * Ypix

Multiply Ypix by width to jump to the correct row

Add Xpix to jump to the correct column

Add 2 zeros for AXI word alignment



SGPU Simulation

ી∰ dk	0						הההההההה		חחחחח
🕼 rst	0								
Vie s	0								
🗓 rdy	1								
🖽 📲 Xin[15:0]	ff00	0100		χ		ff00			
🖽 📲 Yin[15:0]	0100				010	0			
🖽 📲 Zin[15:0]	0100				010	0			
V∰ d	0	n							
mem_addr_out[31:0]	00366b68	00000000			003£570	0		00301	00366
T. M Vio[15:0]		ff00				4400			
M. SULTING		1100	\rightarrow			TIUU			
🖽 📲 Yin[15:0]		0100				0100			
🛨 📲 Zin[15:0]		0100				0100			
14a d		0							
🖪 📲 mem_addr_ou	t[31:0]	00366b68	=	003£5700	X	301500	00 X 0:)366Ъ68	
u d		0							
Ua md		0							
10.									
₩ md_reg		1							

Integer Line Drawing

Bresenham's line algorithm:

plotLine(x0,y0, x1,y1) dx = x1 - x0dy=y1-y0 $DIF = 4^*dy - dx$ y=y0 for x from x0 to x1 plot(x,y) $DIF = DIF + (2^*dy)$ if DIF > 0y = y + 1 $DIF = DIF - (2^*dx)$

Line Drawing State Machine

Using two coordinates in Screen Space, trace line between them

Takes two clock cycles

This can saturate the AXI bus



Line Drawing Data Path



Limits of Simple Line Drawing

Only increments

Y increments at most once per X increment

Only draws lines in the first 45° of the first quadrant



Folding Logic

Uses inputs and their negations

MUXs folds the inputs to draw line to always be positive and ensure that x is larger than y

x0

Final MUX folds values back into appropriate quadrant slice



Additional details

When line drawing is active, two fifos are added into the data flow.

Drawing a line can take much longer than transforming a vertex into pixel space.

Other simple constructs and state machine glue logic from the labs and class.

32 to 64 bit temporal multiplexing for input

SM to give the next set of vertices to the line drawer when it is ready and there are two vertices available in the fifo's

Line Drawing Simulation

+--*****

	ų, d	k	0															
	Us rs	t	0															
	Ue st	art_line	0															
	Vie pi	xel_valid	1		տոու	กกกกกก	սոսո	hunnun	NNNN		nnn	սսող		ເທດ				
	내 lin	e_done	0												Л			
	≞≣ ∦ x0	0[15:0]	0000	0					0000									
	≞ ⊶ ≣ ∦ y(0[15:0]	0000	0					0000									
	≞% x:	1[15:0]	0000	0					0000						X			
	≞ ⊶ ‱ y ∶	1[15:0]	0000	0					0000									
	±¶∦ x_	_out[15:0]	005d	(0000)	8										1			
	≞ ¶∦ y_	_out[15:0]	007a	0000		фх <u>э</u> хс	X X											
ik	0																	
st	0																کھ	
tart_line	0																	
ixel_valid	1																	
ne_done	0																	
0[15:0]	0000		0000		000a												000	0
0[15:0]	0000		0000		0032												000	0
:1[15:0]	0000		0000		0046												000	0
1[15:0]	0000		0000		001e												000	0
_out[15:0]	004d			0000		<u> </u>	Da X	000Ъ	<u> </u>	00c	X	000d	o)0De	_X	000f		00
_out[15:0]	010c			0000		X	0032		0	031		_X			0030			$ \rightarrow $

GPU in the Big Picture

Now that we've got a GPU, we add AXI Interfaces to talk to the CPU and directly to memory and then wrap it up as an IP core.



System Architecture





From Zynq TRM UG585 (v1.10) p. 64

AXI Lite Example Simulation

The AXI Lite interface is used for configuring the video frame's height and width, the base frame address, and the PV matrix.

E S_AXI_LITE								
14 S_AXI_LITE_ACLK	0							
B. S_AXI_LITE_AWADDR[31:0]	00000000	00000000	X	30000040	X		00000	0000
U S_AXI_LITE_AWVALID	0							
U S_AXI_LITE_AWREADY	0							
B. S_AXI_LITE_WDATA[31:0]	00000000	00000000		07800438	X		00000	0000
U S_AXI_LITE_WVALID	0							
U S_AXI_LITE_WREADY	0							
U S_AXI_LITE_BVALID	0							
Us S_AXI_LITE_BREADY	1							
🖬 📲 height_width[31:0]	07800438		0000000		X		07800	0438

AXI Slave Full Example Simulation



First write loads X and Y

Second write loads Z and Color

AXI Master Full Example Simulation



Vivado ILA (Chipscope Pro)

The exact processor GP and HP AXI interfaces are hard to simulate since we don't have a testbench for them. Using Vivado Integrated Logic Analyzer (ILA) we can debug hardware while it is running.



Problems

AXI has two separate handshake signals

1 for the address

1 for the data

When the system first initializes, it seems that the address "ready" goes high but the data does not causing the two output fifos to get out of sync

Line Drawing still has issues when a line leaves the screen edges

The screen clip module ensures that a pixel is never sent to a memory location outside of the frame buffer, but the drawing starts to flicker as if vertices are being lost

Demo and Questions