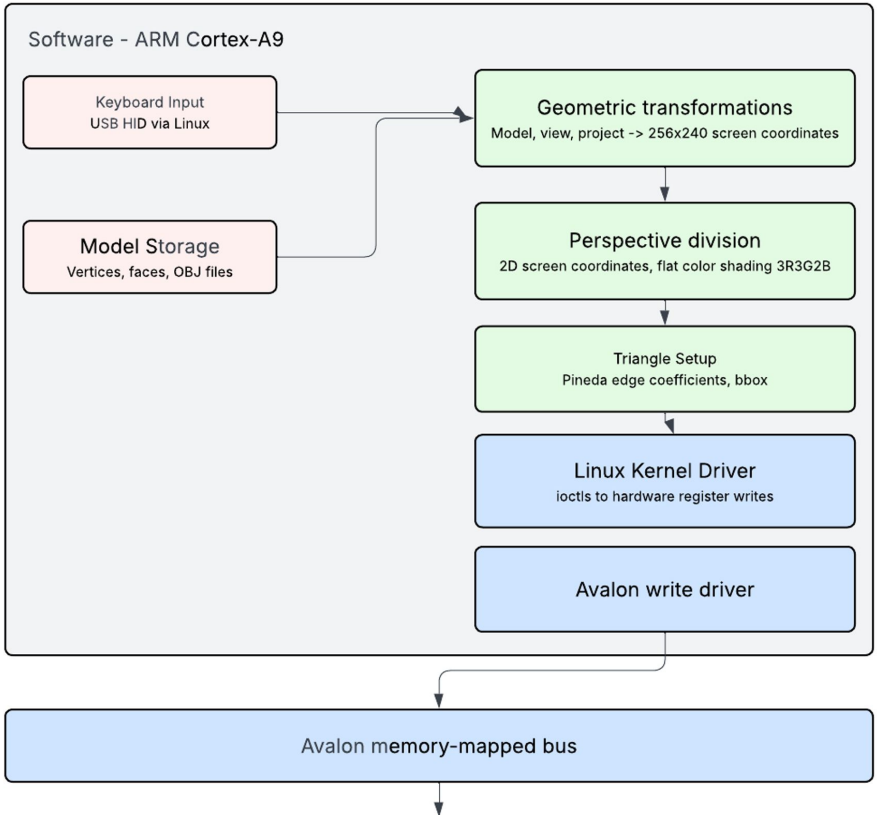
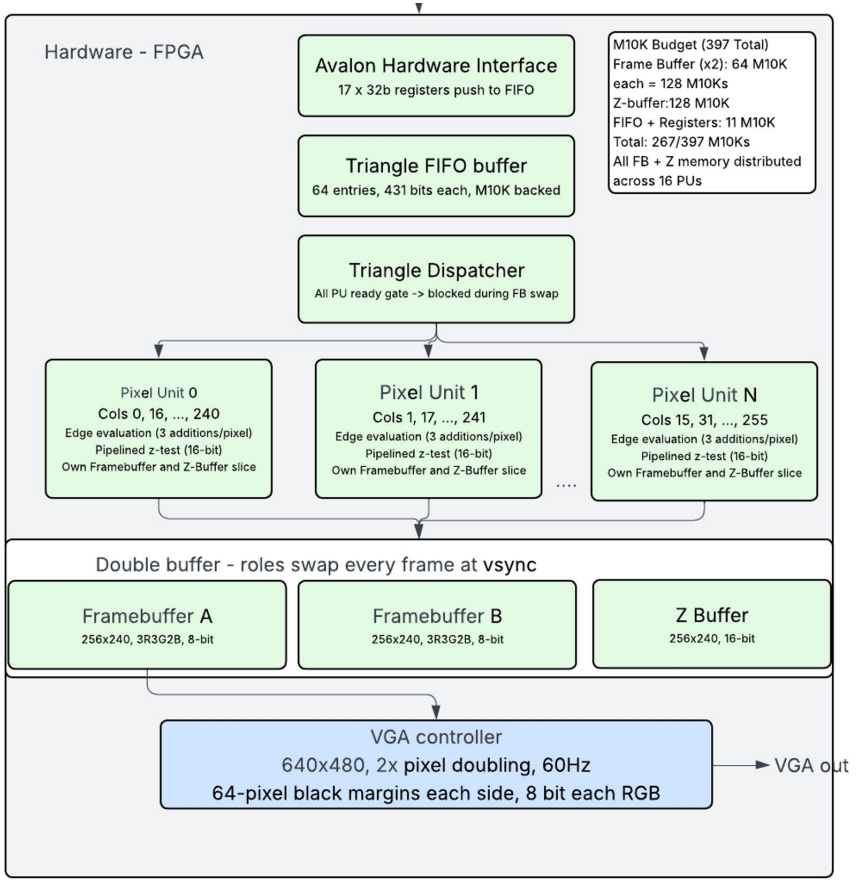
3D Hardware Rasterizer

Gianna Belmont (gb2973), Srika Chagarlamudi (sc5800),
Sathvik Rajampalli (vr2618), Shlok Desai (sbd2150), Aarush Agarwal (aa5763),

System Overview

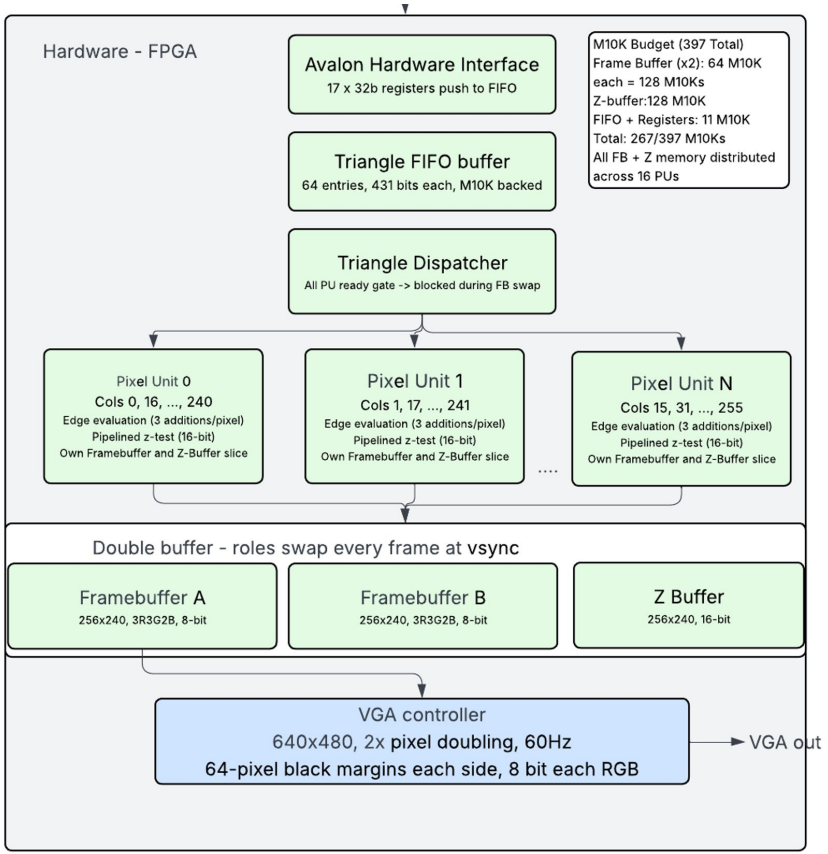




Hardware Algorithm

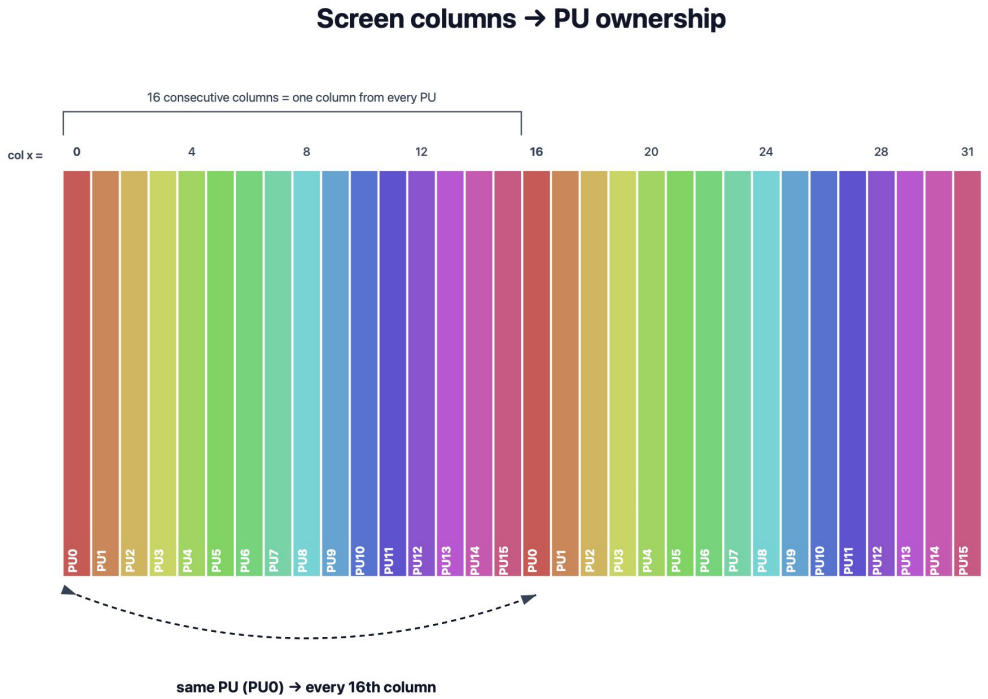
Hardware Rasterizer Overview

- Software submits triangle packets to a hardware FIFO
- Dispatcher waits until all 16 PUs report ready
- PU0 receives the initial seed, chain forwards it down the line
- Each PU owns one column bank: PU_ID = x[3:0]



Pixel Unit Algorithm

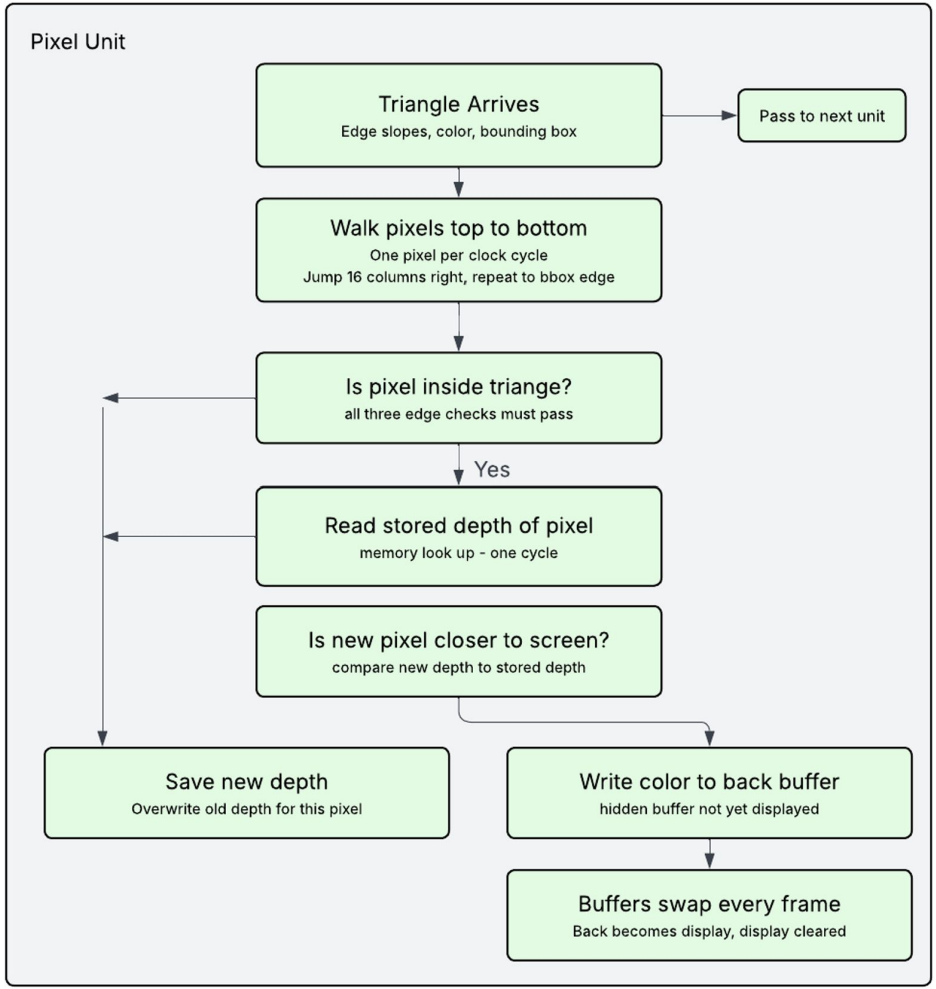
- E(x,y) = ax + by + c
- Latch seed + triangle constants on dispatch
- Walk down rows: e += b, z += z_step_y each cycle
- At bottom, jump +16 columns using 16·a from saved column-top
- Queue one candidate pixel per active cycle for the Z-test



Pixel Unit Walkthrough

4 Stages

- S_INIT_CLEAR: clears local memories after reset (everything)
- S_IDLE: waits for a triangle seed or frame-clear request
- S_ACTIVE: rasterizes the current triangle
- S_CLEAR: clears the Z-buffer and current back buffer between frame



Cycle Overview

Cycle 0: PU0 latches seed/constants, forwards seed + a to PU1 (IDLE)

Cycle 1: PU0 tests pixel 0, queues q_*, reads Z, steps +b. PU1 latches seed

Cycle 2: PU0 Z-tests/writes pixel 0 while testing pixel 1. PU1 tests pixel 0. PU2 latches seed

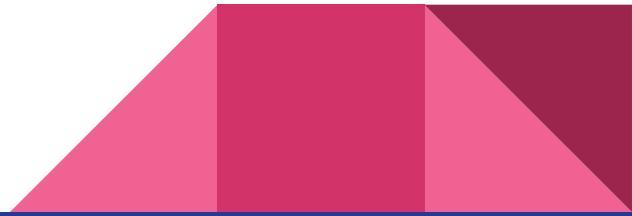
Cycles 3-15: seed wave fills the chain; active PUs keep walking rows

Steady state: each active PU queues 1 pixel/cycle; previous pixel gets Z-tested/written

End column: if col + 16 > last_col, stop; else jump +16 columns using col_top + 16*a

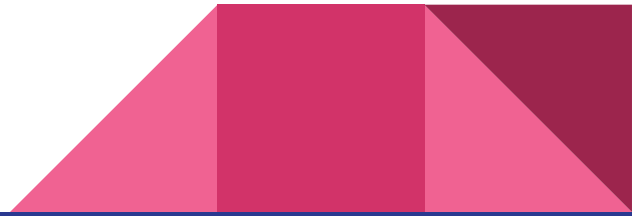
Local memories allocation and Z-test

- Local address: { x[7:4], y[7:0] }
  - 12 bits, 4096 entries
- Local memories: FB_A, FB_B, and Z-buffer (M10K)
- Z-test is pipelined over 2 cycles (sync read -> compare/write)
- Sustains one candidate per active PU per cycle after fill



VGA + frame swap

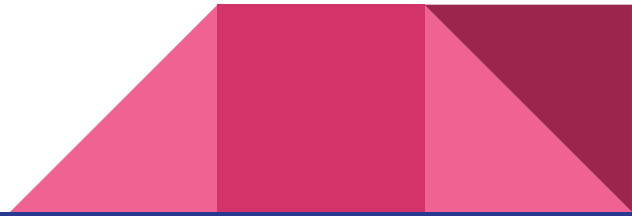
- Internal 256 x 240 buffer scaled 2x to 512 x 480, centered in 640 x 480
- VGA selects PU using fb_x[3:0]
- Local addr uses { fb_x[7:4], fb_y }
- Present waits for: frame done, FIFO empty, all PUs idle
- Then swap buffers + clear new back buffer (FB -> 0, Z -> 0xFFFF)



Software Algorithm

Software Algorithm (Overview of Per-Frame Render)

1. Check Keyboard Status.
2. Update Rotation, Zoom, Model
3. Render Frame
    a. Geometry Calculations
    b. Triangle Packet Setup
    c. Send Triangle Packet



Software Algorithm (Geometry)

- Using Keyboard Status information, find MVP matrix
  - (perspective * translation * rotation x,y,z)
- Using MVP, for each vertex, find the screen pixel coordinates
  - (Object Space -> Clip Space -> NDC Space -> Screen Pixels)
- Triangle Face Shading
  - Calculate the cross product to find the orientation of the face.
  - Calculate the dot product to of that face with light vector to find the shading.

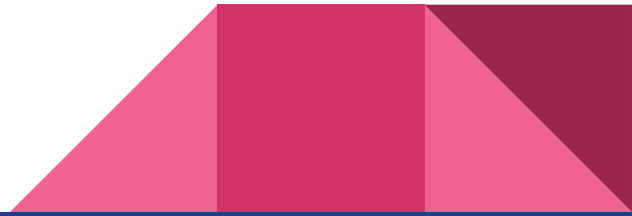
Software Algorithm (Triangle Setup) - 1/

1. Calculate Pineda's Edge Equations
    a. E(x,y) = a*x + b*y + c
    b. a = A.sy − B.sy (row delta)
    c. b = B.sx − A.sx (column delta)
    d. c = A.sx*B.sy − B.sx*A.sy (constant term)
    e. COMPUTE E0(x,y), E1(x,y), E2(x,y):
        i. Edge 0: v1 → v2
        ii. Edge 1: v2 → v0
        iii. Edge 2: v0 → v1

2. Rejecting Triangles
    a. According to Pineda's formulation, area is E0 evaluation with V0 (consistency).
    b. That value is proportional to twice signed area of the triangle.
    c. Calculate Triangle Face Area. If Area is 0 or close to 0, (specifically, 1e-6f), **RETURN -1**

Software Algorithm (Triangle Setup) - 2/

3. Bounding Box Calculations
    a. x-min, x-max (A.x, B.x, C.x)
    b. y-min, y-max (A.y, B.y, C.y)
4. Edge Equation Values
    a. e0_initf
    b. e1_initf
    c. e2_initf
5. Depth Interpolation
    a. z_origin
    b. z_step_y
    c. Z_step_x



Hardware-Software Interface

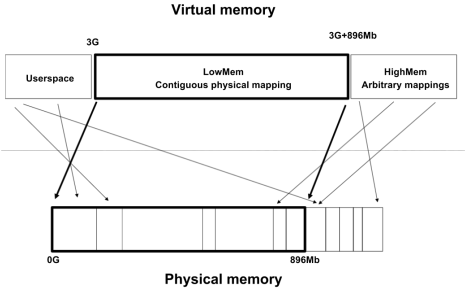
```
static void writeTrianglePacket(const triangle_packet_t *pkt)
{
    const __u32 *words = (const __u32 *)pkt;
    int i;

    for (i = 0; i < 17; i++)
        iowrite32(words[i], dev.virtbase + 0x00 + (i * 4));

    iowrite32(1, dev.virtbase + 0x44);
}

static long rasterizer_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    rasterizer_arg_t ra;

    switch (cmd) {
    case RASTERIZER_SUBMIT:
        copy_from_user(&ra, (rasterizer_arg_t *)arg, sizeof(ra));
        writeTrianglePacket(&ra.packet);
        break;
    }
    return 0;
}
```

Triangle Submission Speedup

```
static const struct file_operations rasterizer_fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = rasterizer_ioctl,
    .mmap           = rasterizer_mmap,
};
```

```
static int rasterizer_mmap(struct file *f, struct vm_area_struct *vma) {
    vma->vm_page_prot = pgprot_writecombine(vma->vm_page_prot);
    return remap_pfn_range(vma,
        vma->vm_start,
        0xFF200000 >> PAGE_SHIFT,
        vma->vm_end - vma->vm_start,
        vma->vm_page_prot);
}
```

```
for (int i = 0; i < 17; i++)
    g_regs[i] = w[i];        // AXI write to 0xFF200000+offset
mmio_barrier();              // dsb sy
g_regs[RAST_COMMIT_OFFSET / 4] = 1;
```



11. PAT (Page Attribute Table)

x86 Page Attribute Table (PAT) allows for setting the memory attribute at the page level granularity. PAT is complementary to t
PAT is more flexible than MTRR due to its capability to set attributes at page level and also due to the fact that there are no hard
for not having memory type aliasing for the same physical memory with multiple virtual addresses.

PAT allows for different types of memory attributes. The most commonly used ones that will be supported at this time are:

WB	Write-back
UC	Uncached
WC	Write-combined
WT	Write-through
UC-	Uncached Minus

DMB

Data Memory Barrier acts as a memory barrier. It ensures that all explicit memory accesses that appear in program order before the `DMB` instruction are observed before any explicit memory accesses that appear in program order after the `DMB` instruction. It does not affect the ordering of any other instructions executing on the processor.

Permitted values of `option` are:

SY
Full system DMB operation. This is the default and can be omitted.

ST
DMB operation that waits only for stores to complete.

ISH
DMB operation only to the inner shareable domain.

ISHST
DMB operation that waits only for stores to complete, and only to the inner shareable domain.

NSH
DMB operation only out to the point of unification.

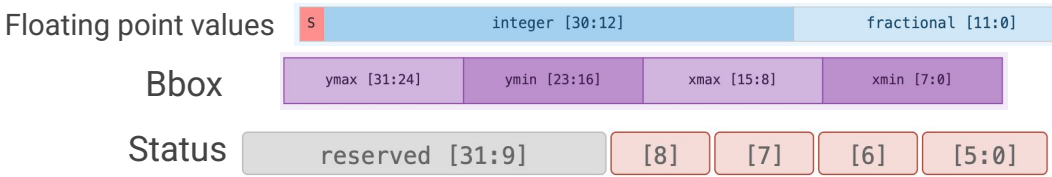
NSHST
DMB operation that waits only for stores to complete and only out to the point of unification.

OSH
DMB operation only to the outer shareable domain.

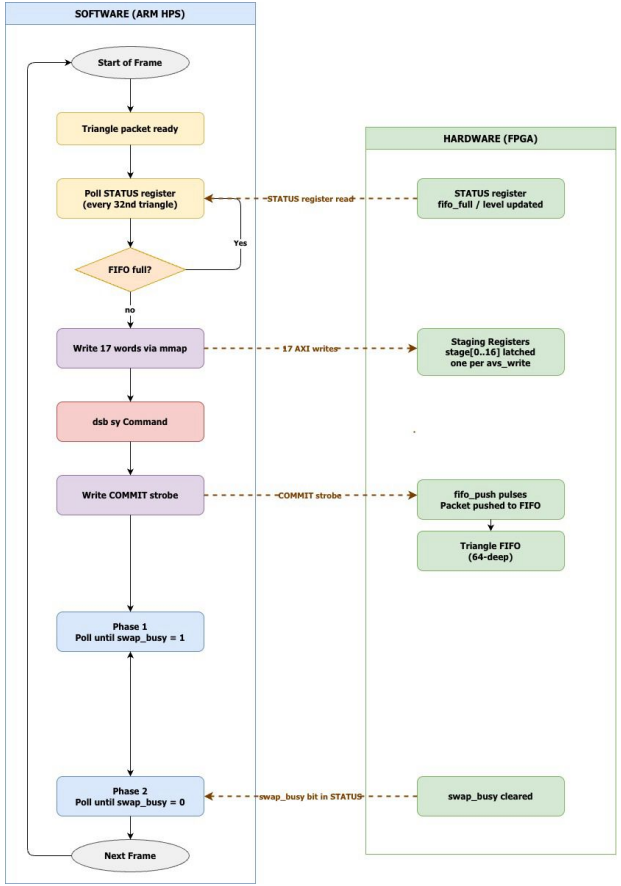
OSHST
DMB operation that waits only for stores to complete, and only to the outer shareable domain.

Hardware-Software Interface

Byte Offset	Value
0x00	a0
0x04	b0
0x08	c0
0x0C	a1
0x10	b1
0x14	c1
0x18	a2
0x1C	b2
0x20	c2
0x24	e0_init
0x28	e1_init
0x2C	e2_init
0x30	z_origin
0x34	z_step_x
0x38	z_step_y
0x3C	bbox_packed
0x40	color_cull
—	—
0x44	COMMIT
0x48	STATUS
0x4C	CONTROL
0x50	PRESENT



Hardware-Software Interface



Resource Summary

Fitter Summary	
Fitter Status	Successful - Sun May 10 16:46:02 2026
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	soc_system
Top-level Entity Name	soc_system_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	8,933 / 32,070 (28 %)
Total registers	13524
Total pins	362 / 457 (79 %)
Total virtual pins	0
Total block memory bits	2,123,520 / 4,065,280 (52 %)
Total RAM Blocks	267 / 397 (67 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

Performance Analysis

Metric	15k model	50k model	160k model
Demo FPS	30	12	4.5
Setup time	23ms	75ms	247ms
Submit time	10ms	40ms	194ms
Present time	16.26 ms	11.95 ms	6.44 ms
FIFO Stalls	0	0	0

Thank You!