EE271 Winter 2017 Project Report

Name1: Erik Augustine SUID1: 06236230

Name2: Liangcheng Tao SUID2: 06118615

Project Results

Total Dynamic Power (mW): _____65.3404_____

Total Leakage Power (mW): _____59.0256_____

Total Power (mW): ____124.3652_____

Total Area (mm^2): ____0.0836788_____

Total Performance (triangles/second): ____510,000,000_____

Number of Rasterization Units: _____2_____

Actual Clock Used (ns): _____0.75_____

Vector used for performance tests: vec_271_02_sv.dat

Design Optimizations:

1. Backface culling
2. Multi-sample testing
3. Reduced multiplication precision
4. Retiming - increasing clock speed

Optimization 1: Backface culling

Backface culling is a fairly straightforward optimization that allows micropolys to be skipped over completely by testing the backwards facing condition. This was chosen as the initial optimization as it is both simple to implement and has potential to greatly improve throughput.

Before implementing the optimization, our guess for overall performance improvement was for the throughput of the unit to approximately double, due to about half of sample being culling, on average. Power and area was predicted to increases slightly per unit, as only a small amount of hardware (a comparator and some adders and multipliers) is added to implement the backface cull.

Implementing the optimization was fairly simple, just adding the logic for culling in parallel with the bounding box calculation. To properly verify the design, the gold model had to be tweaked to reflect the new optimization.

Looking at the actual performance compared to our prediction, the throughput was about what we expected, nearly doubling after the change. Power and area increased more than expected, however, most likely because the additional multipliers and comparator increased the critical path through the bbox module, requiring more area and power to meet the clock period we had set.

Performance changes (Using vec_271_02_sv.dat):

	Before optimization	After optimization	Change (%)
Cycles / uPoly	26.94	14.26	-47.1%
Clock period (ns)	0.5	0.5	0%
Throughput (uPoly/ns)	0.0742	0.1403	+89.1%
# Rasterization units	7	4	-42.9%
Power per unit (mW)	61.88	87.29	+41.1%
Area per unit (um^2)	35795	46747	+30.6%
Total power (mW)	433.2	349.16	-19.4%
Total area (mm^2)	0.2506	.1870	-25.4%

Optimization 2: Multi-sampling

Mutli-sampling seemed to be a straightforward way to greatly increase throughput without having to increase the area as much. The main assumption made with this technique was that we could make the state machine slightly larger, but iterate our rasterizer less times.

Our code tests four samples at the same time, so the assumption was that throughput would increase 4x at most. Because we are iterating the sample tester and bbox module 4 times over in each module, but not the test iterator, we should expect less than 4x increase in area and power.

In our implementation we test 4 “x” direction samples at once. This didn’t prove to be as difficult to keep track of as expected. We used two incrementers. The small one was to initialize the test points into 4 adjacent points. The large incrementer is 4x as big, and jumps all samples by 4 points. This implementation we used also explains why throughput is not a 4 fold increase. If a bounding box is not a multiple of four, we waste samples. The worst case is the bounding box is size 4k+3, in which case we waste 3*height boxes.

This begged the question of testing 4 in one direction, or a 2x2 box. The worst case for the 2x2 box is length * width, which may be greater than 3*height or 3*length. Thus we chose to implement the “4 in a line” approach.

The harder part of implementation was modifying the verification and testbench code. There was a point where it was not clear if the module that generates “sv_out.ppm” was incorrect, or the state machine was incorrect.

The results were slightly less than expected, but still significant. The exact numbers are below, but we saw nearly 3x improvement in throughput, for less than 2x power increase and slightly more than 2x area increase.

Performance changes (Using vec_271_02_sv.dat):

	Before optimization	After optimization	Change (%)
Cycles / uPoly	14.26	5.220	-63.4%
Clock period (ns)	0.5	0.70	+40%
Throughput (uPoly/ns)	0.1403	0.274	+95.3%
# Rasterization units	4	2	-50%
Power per unit (mW)	87.29	164.06	+87.9%
Area per unit (um^2)	46747	110965	+137.4%
Total power (mW)	349.16	328.12	-6.0%
Total area (mm^2)	.1870	.222	+18.7%

Optimization 3: Reduced precision multiplication

After the previous optimizations we still didn't have an effective design - we had significantly leakage power. We decided to optimize our combinational logic to reduce this, starting with our biggest combinational blocks - multipliers.

The default multipliers take in 2 24 bit vectors and produce a 48 bit result. However, because our testing is centered around zero, we don't need that many bits - we do not ever exceed a certain amount of bits. We didn't have a good metric to see how much the area would be reduced by trimming bits from the multipliers, so we used some very rough calculations.

We had assumed it was a significant chunk of the leakage power, as it is by far the largest combinational block in our circuit. Furthermore, there are 3 of them. We wanted to cut the multipliers in half, which should reduce the leakage power somewhere a little south of 50%.

We found that we could cut 11 of the 24 bits without affecting the output. This greatly improved our power performance, and our area. Furthermore we were able to reduce our critical path and reduce our clock period.

Performance changes (Using vec_271_02_sv.dat):

	Before optimization	After optimization	Change (%)
Cycles / uPoly	5.220	5.220	0%
Clock period (ns)	0.70	0.60	-14.3%
Throughput (uPoly/ns)	0.274	.319	+16.4%
# Rasterization units	2	2	0%
Power per unit (mW)	164.06	75.62	-53.9%
Area per unit (um^2)	110965	47344	-57.3%
Total power (mW)	328.12	151.24	-53.9%
Total area (mm^2)	.222	.0947	-57.3%

Optimization 4: Retiming - Increasing Clock Period

This was the last, and a simple optimization. With Tc = 0.6ns, we were over performing. We decided it was best to reduce area and power by reducing the clock period so throughput would just meet the spec. So we reduced to Tc = 0.75ns (we would need 0.76 to meet spec).

Performance changes (Using vec_271_02_sv.dat):

	Before optimization	After optimization	Change (%)
Cycles / uPoly	5.220	5.220	0%
Clock period (ns)	0.60	0.75	+25%
Throughput (uPoly/ns)	0.319	.255	-20.06274%
# Rasterization units	2	2	0%
Power per unit (mW)	75.62	62.18	-17.8%
Area per unit (um^2)	47344	41893	-11.5%
Total power (mW)	151.24	124.3652	-17.8%
Total area (mm^2)	.0947	.08379	-11.5%

Analysis:

1) There were interesting quirks involved in changing certain bits of logic. A very interesting quirk was trying to optimize some of the logic in the state machine:

```
next_state_R14H     = at_end_box_R14H ? WAIT_STATE : TEST_STATE;
```

To

```
next_state_R14H     = !at_end_box_R14H;
```

Increased the area by 50%.

Contrarily changing:

```
next_halt_RnnnnL    = validPoly_R13H ? 1'b0:1'b1;
```

To

```
next_halt_RnnnnL    = !validPoly_R13H;
```

Produced a negligible difference.

2) As we experimented with clock period as a parameter during optimizations, the general trend was (as in optimization 4) for power and area per unit to decrease as max clock period increases. However, there were some exceptions during testing where the opposite occurred. In all cases, the synthesizer would give the smallest possible slack relative to the clock period.

This is an indication of inefficiency in the synthesizer, as a slower design should always have at least the same power and area as a faster design (as in theory, you could simply run the faster design at a slower clock to achieve the same area and lower power).

3) In moving towards our final design, we experimented with various combinations of our optimizations. Notably, we implemented backface culling and multitest separately, and when both ran individually, both produced substantial benefits in overall power and area. However, when the two optimizations were combined, the result was that the overall performance was worse than either optimization by itself. The reason was due to the multiplicative increase in area and power per unit, the required increase in clock frequency to accommodate the longer critical path between two modules, and the relatively small impact of increased throughput on the number of units required (almost tripling the throughput only halved the number of units).

However, when we added in the precision reduction optimization, the three optimizations together produced the best result compared to all other combinations (with only multitest and precision reduction a close second). This was due to the massive reduction in area and power from shrinking the multipliers in both bbox and sample test.

Conclusions:

This project was a good introduction into the trade-offs between power and area. Many optimizations ended up improving both simultaneously, multiplication precision most noticeably. Another useful insight was which optimizations are worth the time, and which are wastes of time.

Optimizing the state machine, reducing your total workload, or making your multipliers just large enough are endeavours that can make a chip small and fast. While worrying about if you can implement a multiplexer with an AND gate isn't really worth the effort.