EE 271: Project Report

Edward Kim, Austin Rothschild

December 2024

Abstract

Part III of the final project investigates various optimizations in RTL to minimize a desired FoM which is a function of the synthesized area, power consumption, and number of rasterization units needed to achieve a given throughput specification. This report details the various optimizations that were explored and implemented, highlighting the performance gains achieved from each.

1 Introduction

In Part III, the main objective was to optimize the performance of the RTL-based rasterization pipeline implemented in Part II based on the key performance metrics of power, area, and throughput. These are quantified through a specific figure of merit (FoM).

1.1 Figure of Merit (FoM)

The design objective is to minimize the FoM defined below:

$$\text{FoM} = \left(\frac{\text{Area [mm}^2\text{]}}{\text{Unit}}\right) \cdot \left(\frac{\text{Power [mW]}}{\text{Unit}}\right) \cdot (\text{Num Rast Units})^2 \tag{1}$$

where (Num Rast Units) is the number of cycles per triangle multiplied by the CLK period and divided by the specification requirement of 2 ns/cycle. We then round this up to the nearest integer to obtain the number of required rasterization units.

2 Key Performance Indicators

2.1 FoM & Performance Plots

Below we summarize the performance of our design across different implementation strategies. We stack each optimization on top of each other and record the relative key performance indicators (KPIs) as a reuslt of the changes.

We found that we were able to achieve the best throughput and hence FoM on test vector 02_short. Using our design achieving best FoM (Design # 6), the FoM for test vector 02 is given below. Note that the absolute FoM value achieved for test vector 02 is not different than what is seen for test vector 01 since we take the ceiling of the number of rasterization units.

Design	Method	Power [mW]			Area [mm^2]	No. of Cycles/Tri	CLK Period [ns]	Throughput [ns/Tri]	No. of Rast Units	FoM
		Dynamic	Leakage	Total						
1	No Optimization	26.140	0.677	26.817	0.0343	26.17	1.2	31.40	16	235.25
2	Reducing Precision	19.140	0.613	19.753	0.0310	26.17	1.2	31.40	16	156.52
3	Backface Culling	22.960	0.682	23.642	0.0349	13.91	1.2	16.70	9	66.76
4	Clock Gating	19.990	0.635	20.625	0.0330	13.91	1.2	16.69	9	54.97
5	Bubble Smashing	22.639	0.634	23.273	0.0330	13.88	1.2	16.66	9	62.04
6	Retiming + Selective Zeroing	23.919	0.614	24.533	0.0320	13.91	1.0	13.91	7	37.87

Table 1: Performance comparison of optimized design implementations for Test Vector 01.

Design	Method	Power [mW]			Area [mm^2]	No. of Cycles/Tri	CLK Period [ns]	Throughput [ns/Tri]	No. of Rast Units	FoM
		Dynamic	Leakage	Total						
6	Retiming + Selective Zeroing	23.919	0.614	24.533	0.0320	13.41	1.0	13.41	7	37.87

Table 2: Performance of the optimal design for Test Vector 02_short.

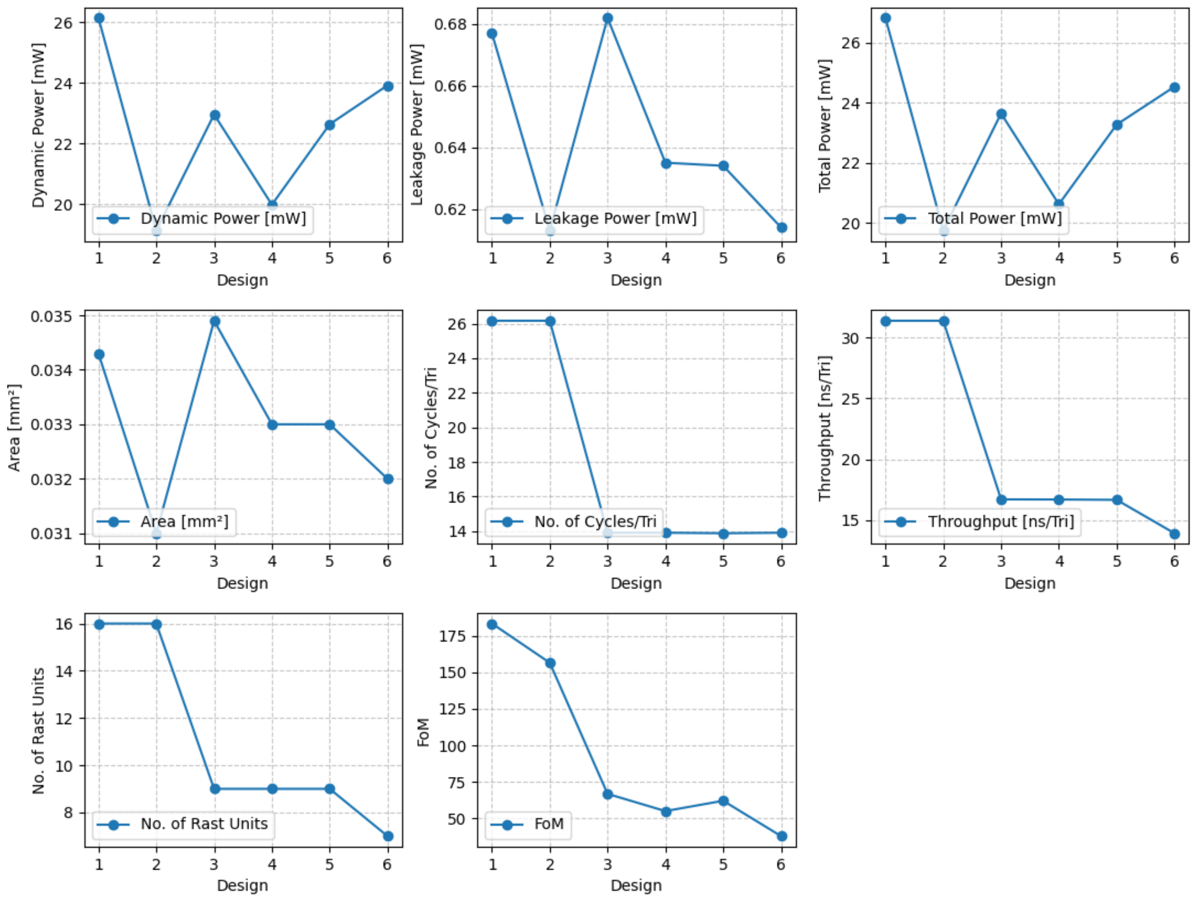

Figure 1: Performance comparison of optimized design implementations for Test Vector 01.

3 Optimizations

Below we summarize the optimizations that were implemented in each design as highlighted in Table. 1 and Figure. 1

3.1 Default Design

With no optimizations (essentially just Part II RTL) the design yielded a FoM of around 235.

3.2 Back-Face Culling

With back-face culling [1], [2], the polygons that are processed by the bounding-box module are first checked to determine if they are back-facing. A polygon that is back-facing is one that is not visible to the camera because they face away from it. In the rasterization pipeline, these polygons are removed from the pipeline as they do not contribute to the final rendered image, hence the computational load is relaxed. In a typical 3D scene, approximately 50% of the polygons are back-facing. Thus, by implementing back-face culling, we expect a 2X improvement in overall throughput in the rasterization pipeline, since half the geometry is no longer being processed.

A triangle has a front-face and a back-face. To determine which face is being processed, the typical calculation is to find the surface normal vector perpendicular to the triangle's face and then compute the inner-product of this with the camera's normal vector. This can be simplified to consider only the triangle's 2D area and then checking its sign:

$$\text{Area} = (x_2 - x_1)(y_3 - y_2) - (y_2 - y_1)(x_3 - x_2) \tag{2}$$

After the bbox module determines if the triangle is backfaced, this modified valid signal is sent through the pipeline into the test iterator module where the rasterizer enables the pipeline and never iterates through the bounding box if the incoming valid signal is low (hence the triangle is "culled" before the sample test stage). Implementing this was straightforward and we did not run into any unexpected issues. The main challenge was being able to minimize the delay of the culling calculation.

Backface culling approximately doubled the throughput and halved the FoM, giving by far the greatest performance gain out of all other optimizations used. This was as expected.

3.3 Reduced Precision

We expected the use of precision reduction to mainly reduce the latency of some expensive calculations like multiplication like in the sample_test.sv and bbox.sv modules (after backface culling). This would decrease the critical path delay and enable us to decrease the clock period in order to make throughput gains (of course taking into account power, area for FoM). We estimated this would reduce around 0.1 to 0.4 ns in the critical path, which was most often coming up in sample_test.sv. We found that within bbox.sv we are able to truncate the backface culling precision by 11 bits and still pass all the test vectors. In sample_test.sv, we were only able to truncate by 5 bits in order to pass all the test vectors.

The implementation of this was straightforward and involved zeroing out a certain number of bits incrementally in brute force fashion until the test vectors no longer passed in simulation. The unexpected part of this was that only by taking away some of the MSBs in calculations were we able to still maintain accuracy. Typically, the LSBs are the targeted bits for precision reduction.

As expected, reducing the precision of multiplication operations reduced the critical path latency in sample_test.sv by around 0.3ns. This also led to a significant (around 26%) reduction in total power, particularly in dynamic power consumption.

3.4 Clock-Gating

The main optimization provided by clock-gating is typically less power consumption due to enabling of the clock only to areas that are active. The expectation was that total power consumption would reduce by at around at least 10%, but the extent of the expected performance benefit was difficult to gauge by just looking at the RTL. The implementation of clock-gating was easy and involved simply adding a flag to a line in rast_dc.tcl. The result after implementation was around a 13% decrease in total power consumption, which was slightly more than initially expected. Additionally, we found the area slightly improved which is likely due to some synthesis optimizations of under the hood.

3.5 Bubble Smashing

The expected performance gain for bubble-smashing was a decrease in cycles per triangle (more throughput) as it would remove "bubbles" of invalid triangles between valid triangles within the pipeline. This would help deal with idle cycles or pipeline stalls and thus allow the rasterizer to process more triangles in less time. It was difficult to predict the expected gain from bubble-smashing as it would depend on the test vector being simulated, but a throughput improvement of around 10% was expected.

The implementation of this came from the perspective of the FSM in the test iterator module. When the FSM is in the TEST state it has received a valid bounding box and triangle and iterates through the bounding box. During this, the bounding box pipeline is halted to prevent valid triangles from coming in during the TEST state and not being processed. Bubble-smashing was implemented to maintain this goal and increase throughput by letting the bounding box pipeline run when the triangle seen at the test iterator input was invalid. This way, after the TEST state the FSM wouldn't potentially have to run many more cycles until it receives the next valid triangle and bounding box. The original halt signal was sent from the test iterator to the bbox module, modified in the bbox module, and send out from the bbox module to rast.sv so that the enable signals for bbox and the incoming triangles to rast.sv could be synchronized.

Bubble smashing yielded significantly higher power consumption and a marginal improvement (less than 1%) in cycles per triangle. The throughput improvement was so negligible that the number of rasterizer units needed for FoM did not change. This was quite unexpected since we thought the throughput improvement would be far greater. Since bubble smashing made FoM worse, due to the power tradeoff being more significant, it was not used.

3.6 Final Optimizations

The last two optimizations made to decrease FoM were relatively minor for implementation but gave significant performance improvements.

3.6.1 Selective Zeroing

Selective zeroing is simply zeroing out signals when the incoming input triangle or sample is invalid. We noticed that many wasteful computations were being done for invalid triangles even when the signals could just be set to 0 if the triangle was invalid. We expected this to primarily improve dynamic power consumption by around 10% since depending on the number of invalid inputs, the amount of switching could be cut down significantly.

The implementation was straightforward and was essentially using multiplexer if-else logic to zero out relevant signals when the input was invalid. This was done in rast.sv with the incoming triangles so that for tor the rest of the pipeline the rasterizer wouldn't have to waste power doing computations on invalid triangles.

3.6.2 Retiming

Retiming was enabled by setting the retime flag high on appropriate registers in the pipeline. This would help optimize the critical path delay and thus enable a lower clock period. The best throughput we were getting at this point was around 13.9, and we deduced that the most reasonable target number of rasterizer units we could decrease to was 7 (previously 9 and 16 before that). To get down to 7 units, we decreased CLK period in the MAKEFILE to 1.0 ns. Meeting timng for 1.0 ns involved adding an additional stage to the sample test pipeline, which was the location of the critical path. While decreasing the clock period would help increase throughput, we expected this to increase power. It was difficult to gauge by how much, but we estimated around a 2 to 3 mW increase in total power. We hoped that this power increase could be offset significantly by power saving from the selective zeroing optimization as previously discussed.

The results of combined retiming and selective zeroing allowed us to decrease the number of rasterizer units to 7, meet the timing constraint for 1.0 ns clock period, and maintain relatively low power. It seems the power increase from decreasing clock period outmatched the power decrease from selective zeroing, but not by very much. Selective zeroing was able to decrease power by around 16%, which was more than expected.

4 Conclusions

In part III, we were able to reduce our FoM from a starting value of 235.25 to a final value of 37.87, an 83.9% reduction. We found that reducing precision, backface culling, and retiming allowed us to achieve the most significant gains in performance. Additional optimizations that could lead to greater performance gains include implementing a more efficient & parallelized bounding box algorithm, testing multiple samples per cycle, and further simplyfing the design logic in order to drive down power and area while retaining throughput improvements.

References

[1] Wikipedia, “Back-face culling.” [Online]. Available: https://en.wikipedia.org/wiki/Back-face_culling

[2] R. Sedgewick, “Geometric primitives - princeton algorithms,” n.d., accessed: December 3, 2024. [Online]. Available: