Mr. Phot Koseekrainiramon z5387411

23T2 COMP4601 Week 3 Lab Hand-in exercises

Exercise 1:

Performance Estimates

Timing

Clock		float	fixed_12	fixed_16
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.286 ns	5.259 ns	5.303 ns

Latency

		float	fixed_12	fixed_16
Latency (cycles)	min	385	65	65
	max	385	65	65
Latency (absolute)	min	3.850 us	0.650 us	0.650 us
	max	3.850 us	0.650 us	0.650 us
Interval (cycles)	min	385	65	65
	max	385	65	65

Utilization Estimates

	float	fixed_12	fixed_16
BRAM_18K	1	0	0
DSP48E	13	2	2
FF	1250	110	142
LUT	1225	305	357
URAM	0	0	0

By using arbitrary precision type (both 12 and 16 bits), the area and the latency are significantly reduced and not using BRAM anymore (because the total number of bits in the memory is less than 1024 bits). The estimated clock period is also reduced. Since more bits would require more operations and increase utilization estimates.

By comparing fixed 12 and 16 bits, they are having the same latency with slightly different estimated clock periods and area usage. They both use no BRAMs, but with fixed 12 bits, it has less area usage (but with the same number of DSP) and estimated clock periods.

In terms of accuracy, the original design is obviously providing the most accurate result. The fewer bits of the arbitrary precision type used the more error there are. This makes sense since the fewer bits would have less accurate due to cutting off the least significant bits. However, the error is way worse when using 12 bits arbitrary precision type compared to using 16 bits (the total error is increased by 11 times) and float (the total error is increased by 2858 times).

Exercise 2:

Performance Estimates

Timing

Clock		fixed_12	optimized
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	5.259 ns	2.745 ns

Latency

		fixed_12	optimized
Latency (cycles)	min	65	65
	max	65	65
Latency (absolute)	min	0.650 us	0.650 us
	max	0.650 us	0.650 us
Interval (cycles)	min	65	65
	max	65	65

Utilization Estimates

	fixed_12	optimized
BRAM_18K	0	0
DSP48E	2	0
FF	110	86
LUT	305	295
URAM	0	0

The new design has significantly fewer estimated clock periods due to having less complex logic (fewer operations leading to fewer components required). The new design also has the same latency as the previous design but with slightly less area usage for flip-flops and look-up tables, however, the new design uses no DSPs while the previous design uses 2 of them.

In terms of accuracy, the new design is more accurate by having 3 times less total error because simple logic is used.

Exercise 3:

The loop has an iteration latency of 2 with a 32-trip count (optimized solution).

Performance Estimates

Timing

Clock		optimized	unroll_2	unroll_4	unroll_8
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	10.00 ns
	Estimated	2.745 ns	4.137 ns	8.255 ns	8.315 ns

Latency

		optimized	unroll_2	unroll_4	unroll_8
Latency (cycles)	min	65	33	17	17
	max	65	33	17	17
Latency (absolute)	min	0.650 us	0.330 us	0.170 us	0.170 us
	max	0.650 us	0.330 us	0.170 us	0.170 us
Interval (cycles)	min	65	33	17	17
	max	65	33	17	17

Utilization Estimates

	optimized	unroll_2	unroll_4	unroll_8
BRAM_18K	0	0	0	0
DSP48E	0	0	0	0
FF	86	100	127	227
LUT	295	502	916	1801
URAM	0	0	0	0

I expected the latency to be reduced by half when unrolling the loop but it would use up more areas in exchange, and since it uses up more areas, the estimated clock period should be higher.

My expectation was correct, but the estimated clock period varied more than I expected. The latency is not exactly half out because it uses 1 cycle for the beginning and terminating the execution (trip count is 16 iterations which is half of not unrolling, with the same iteration latency).

When changing the unrolling factor from 2 to 4, the latency is reduced by half with an increased estimated clock period and area usage. But when changing the factor from 4 to 8, the latency stays the same (uses up more areas and slightly higher estimated clock periods) even though the trip count is reduced by half but the iteration latency is increased by 2 in exchange causing them to have to same overall latency.

The reason why unrolling by a factor of 4 and 8 results the same is that the target clock periods aren't enough for the unrolling by a factor of 8 to execute 8 loops at a time as it's supposed to do. We could change the target clock period to 17 and that would eliminate the bottleneck (double the estimated clock period of unrolling by a factor of 4).

Performance Estimates

Timing

Clock		unroll_2	pipelined
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	4.137 ns	2.745 ns

Latency

		unroll_2	pipelined
Latency (cycles)	min	33	34
	max	33	34
Latency (absolute)	min	0.330 us	0.340 us
	max	0.330 us	0.340 us
Interval (cycles)	min	33	34
	max	33	34

Utilization Estimates

	unroll_2	pipelined
BRAM_18K	0	0
DSP48E	0	0
FF	100	64
LUT	502	323
URAM	0	0

The area usage and estimated clock periods are significantly reduced when pipelining the design. Both of the designs have the same trip count and iteration latency, but the pipelined needs to spend an extra cycle to operate writing in the last stage.

By completely unrolling the loop, the performance in terms of latency would surely be improved but with a significantly increased area usage. Or adjusting the number of iterations, and lowing the values would speed up the design with the same utilization estimate but cause more errors as a trade-off.

Exercise 4:

Performance Estimates

Timing

Clock		optimized	iteration_16	iteration_12	iteration_8
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	10.00 ns
	Estimated	2.745 ns	2.745 ns	2.745 ns	2.745 ns

Latency

		optimized	iteration_16	iteration_12	iteration_8
Latency (cycles)	min	65	33	25	17
	max	65	33	25	17
Latency (absolute)	min	0.650 us	0.330 us	0.250 us	0.170 us
	max	0.650 us	0.330 us	0.250 us	0.170 us
Interval (cycles)	min	65	33	25	17
	max	65	33	25	17

Utilization Estimates

	optimized	iteration_16	iteration_12	iteration_8
BRAM_18K	0	0	0	0
DSP48E	0	0	0	0
FF	86	84	82	82
LUT	295	295	290	292
URAM	0	0	0	0

The area usage is only slightly altered for each number of iterations (causing them to have the same estimated clock period). Since the utilization depends on computation and not the number of iterations, however, the fewer number of iterations provides better performance in terms of latency as well as throughput. Since more iterations would increase the number of clock cycles period.

The error is increased as the number of iterations decreases, however, the number of iterations of 16 and 12 has the same accuracy (same number of total errors). As the more iterations, the smaller angle is used to approach the value, causing to have fewer errors.

It doesn't affect the initial values of current_cos and current_sin as we started with a fixed degree, the only matter things are precisions and the array.

The array cordic_phase doesn't need to be modified since it is not dependent on the number of iterations.

Having fewer bits of a data type but increasing the number of iterations might be feasible, but would need to find an optimal point where the trade-off is reasonable in terms of utilization and speed vs accuracy.