Mr. Phot Koseekrainiramon z5387411

23T2 COMP4601 Week 4&5 Lab Hand-in exercises

Exercise 1:

It has an estimated clock period of 6.435ns with 145 cycles of latency (1.160us absolute latency since we are using an 8ns target clock period, not 10ns). In terms of resource utilization, 3 DSP48Es are used (0.002% of available), with 60 flip-flops (0.0002% of available) and 174 look-up tables (0.001% of available).

dot_product_loop which is inside data_loop has an iteration latency of 2 with a trip count of 8, causing it to take 16 cycles of latency. data_loop has an iteration latency of 18 with the same number of trip counts, causing it to take 144 cycles of latency.

Only one multiplier and one adder are required.

Exercise 2:

Performance Estimates

Timing

Clock		solution1	unroll_dot_product
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns

Latency

		solution1	unroll_dot_product
Latency (cycles)	min	145	49
	max	145	49
Latency (absolute)	min	1.160 us	0.392 us
	max	1.160 us	0.392 us
Interval (cycles)	min	145	49
	max	145	49

Utilization Estimates

	solution1	unroll_dot_product
BRAM_18K	0	0
DSP48E	3	6
FF	60	115
LUT	174	437
URAM	0	0

The new design has the same number of estimated clock periods but significantly fewer cycles of latency since the iteration latency of the loop is reduced to 6 with the same number of trip counts. Even though the new design is faster, it used up more resources in exchange. 3 more DSP48Es, 55 more flip-flops, and 263 more look-up tables.

It needs to do 8 multipliers but only 2 multipliers are required (6 DSP48Es) because they are done simultaneously, almost the same for adders but not all are shared.

By unrolling the inner loop, each of the multiplications could be done simultaneously with an adder tree for additions. Since loading an element from an array takes approximately 1 cycle, with a SIZE of 8, we need to load 16 elements (to do multiplication). However, we could load 4 elements at a time and perform 2 sets of multiplication before adding them up. This takes approximately 4-5 cycles since you could start loading the next set of elements while performing multiplication. Then another 1-2 cycles performing multiplication and addition of the last set causing 6 cycles of iteration.

Exercise 3:

Performance Estimates

Timing

Clock		unroll_dot_product	pipeline_dot_product
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.663 ns

Latency

		unroll_dot_product	pipeline_dot_product
Latency (cycles)	min	49	67
	max	49	67
Latency (absolute)	min	0.392 us	0.536 us
	max	0.392 us	0.536 us
Interval (cycles)	min	49	67
	max	49	67

Utilization Estimates

	unroll_dot_product	pipeline_dot_product
BRAM_18K	0	0
DSP48E	6	3
FF	115	106
LUT	437	283
URAM	0	0

The new design has little higher estimated clock period with 18 more cycles of latency. Overall better resource utilization with 3 fewer DSP48Es since only 1 multiply needs to be done in one of the stages, 9 fewer flip-flops, and 154 fewer look-up tables.

The loop nest is gone from completely unrolling.

By performing only 1 set of multiplication in one iteration, loading 2 elements from arrays take approximately 2 cycles with 1 cycle of multiplication and addition at the end of each iteration causing 3 iteration latency. Since a new multiplication could start every clock cycle on a pipelined, the initial interval is only 1 cycle.

Exercise 4:

Performance Estimates

Timing

Clock		solution1	unroll_dot_product	pipeline_dot_product	unroll_data_loop
ap_clk	Target	8.00 ns	8.00 ns	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns	6.663 ns	6.435 ns

Latency

		solution1	unroll_dot_product	pipeline_dot_product	unroll_data_loop
Latency (cycles)	min	145	49	67	136
	max	145	49	67	136
Latency (absolute)	min	1.160 us	0.392 us	0.536 us	1.088 us
	max	1.160 us	0.392 us	0.536 us	1.088 us
Interval (cycles)	min	145	49	67	136
	max	145	49	67	136

Utilization Estimates

	solution1	unroll_dot_product	pipeline_dot_product	unroll_data_loop
BRAM_18K	0	0	0	0
DSP48E	3	6	3	6
FF	60	115	106	337
LUT	174	437	283	975
URAM	0	0	0	0

This is totally not worth it since the number of cycles of latency is only reduced by 9 with little less absolute latency than compared to the original design but with a tremendously increased number of resources required.

Compared with unrolling dot_product_loop and pipelining dot_product_loop, unrolling data_loop is worse in every perspective (both in terms of performance and utilization). So, it is always better to choose to unroll dot_product loop than data_loop.

With 8 dot_product_loop operations that need to be performed sequentially, 16 latency each, this is because the inner loop isn't being unrolled. Only if the inner loop is unrolled, the performance would be better than other designs but comes with a tremendously increased number of resources used.

Exercise 5:

Performance Estimates

Timing

Clock		solution1	unroll_dot_product	pipeline_dot_product	unroll_data_loop	pipeline_data_loop
ap_clk	Target	8.00 ns	8.00 ns	8.00 ns	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns	6.663 ns	6.435 ns	6.435 ns

Latency

		solution1	unroll_dot_product	pipeline_dot_product	unroll_data_loop	pipeline_data_loop
Latency (cycles)	min	145	49	67	136	35
	max	145	49	67	136	35
Latency (absolute)	min	1.160 us	0.392 us	0.536 us	1.088 us	0.280 us
	max	1.160 us	0.392 us	0.536 us	1.088 us	0.280 us
Interval (cycles)	min	145	49	67	136	35
	max	145	49	67	136	35

Utilization Estimates

	solution1	unroll_dot_product	pipeline_dot_product	unroll_data_loop	pipeline_data_loop
BRAM_18K	0	0	0	0	0
DSP48E	3	6	3	6	6
FF	60	115	106	337	122
LUT	174	437	283	975	454
URAM	0	0	0	0	0

The inner loop is automatically unrolled (completely) causing the iteration latency to be 6 as explained in exercise 2. But with the initial interval of 4 (since 4 cycles are required to load 16 elements from arrays as explained in exercise 2), even though the trip count is 8 (SIZE), the total latency is only 35 and not 8*6.

By ranking all of these designs, pipelining the inner loop would be the first since it has the third-best performance (not much differences), however, it only needs 3 DSPs with fewer LUTs and FFs compared to other designs (except the original). Second and third places go for pipelining the outer loop and unrolling the inner loop, with the first and second-best performances almost the same in terms of utilization. The original design goes forth and unrolling the outer loop goes last as explained in exercise 4.

Exercise 6:

Performance Estimates

Timing

Clock		unroll_dot_product	unroll_inner
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns

Latency

		unroll_dot_product	unroll_inner
Latency (cycles)	min	49	49
	max	49	49
Latency (absolute)	min	0.392 us	0.392 us
	max	0.392 us	0.392 us
Interval (cycles)	min	49	49
	max	49	49

Utilization Estimates

	unroll_dot_product	unroll_inner
BRAM_18K	0	0
DSP48E	6	6
FF	115	115
LUT	437	437
URAM	0	0

Everything is the same in terms of performance and utilization comparing manually unrolling the inner loop and unrolling the inner loop using a directive. Since the logic behind the code is the same, one is letting Vivado HLS design from the given directive, another is manually editing from the source code.

Exercise 7:

Performance Estimates

Timing

Clock		pipeline_data_loop	solution7
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns

Latency

		pipeline_data_loop	solution7
Latency (cycles)	min	35	19
	max	35	19
Latency (absolute)	min	0.280 us	0.152 us
	max	0.280 us	0.152 us
Interval (cycles)	min	35	19
	max	35	19

Utilization Estimates

	pipeline_data_loop	solution7
BRAM_18K	0	0
DSP48E	6	12
FF	122	184
LUT	454	503
URAM	0	0

The new design is faster (with the same number of estimated clock periods but fewer cycles of latency) but with little more resources used (double DSPs since extra multiplications are required, and more flip-flops for array partition that split elements of an array into their own registers) for a trade-off.

With only 4 iteration latency due to half time of loading elements from arrays and performing twice sets of multiplication since we have them as separated arrays. With the initial interval of 2 due to 2 cycles of loading elements from arrays before performing a multiplication at the end, only 19 cycles are required.

Exercise 8:

Performance Estimates

Timing

Clock		solution7	solution8
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns

Latency

		solution7	solution8
Latency (cycles)	min	19	19
	max	19	19
Latency (absolute)	min	0.152 us	0.152 us
	max	0.152 us	0.152 us
Interval (cycles)	min	19	19
	max	19	19

Utilization Estimates

	solution7	solution8
BRAM_18K	0	0
DSP48E	12	12
FF	184	152
LUT	503	503
URAM	0	0

They have the same performance and iteration latency (slightly different multiplication order) but block partitioning requires fewer flip-flops (for registers). This might be because some elements are being stored somewhere else that Vivado HLS doesn't provide more details, or might be because of some limitations of the device part we use.

Exercise 9:

Performance Estimates

Timing

Clock		pipeline_data_loop	solution7	solution9
ap_clk	Target	8.00 ns	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.435 ns	6.762 ns

Latency

		pipeline_data_loop	solution7	solution9
Latency (cycles)	min	35	19	11
	max	35	19	11
Latency (absolute)	min	0.280 us	0.152 us	88.000 ns
	max	0.280 us	0.152 us	88.000 ns
Interval (cycles)	min	35	19	11
	max	35	19	11

Utilization Estimates

	pipeline_data_loop	solution7	solution9
BRAM_18K	0	0	0
DSP48E	6	12	24
FF	122	184	372
LUT	454	503	480
URAM	0	0	0

It's obvious that, in terms of performance, the array partition is better (both complete and non-complete). However, as explained in Exercise 7 that the trade-off couldn't be avoided. As a complete array partition, double flip-flops and DSP are required for extra multiplications and registers to store elements from the arrays separately from that non-complete array. The logic is the same as Exercise 7 in that we could load more elements from those array partitions at the time causing less iteration latency

(3) and initial interval (1). (Even though array partitioning has a higher estimated clock period, with less number of cycles of latency, the performance is better than that in solution 7).

Exercise 10:

Performance Estimates

Timing

Clock		solution7	solution9	solution10
ap_clk	Target	8.00 ns	8.00 ns	8.00 ns
	Estimated	6.435 ns	6.762 ns	6.762 ns

Latency

		solution7	solution9	solution10
Latency (cycles)	min	19	11	18
	max	19	11	18
Latency (absolute)	min	0.152 us	88.000 ns	0.144 us
	max	0.152 us	88.000 ns	0.144 us
Interval (cycles)	min	19	11	18
	max	19	11	18

Utilization Estimates

	solution7	solution9	solution10
BRAM_18K	0	0	0
DSP48E	12	24	24
FF	184	372	339
LUT	503	480	484
URAM	0	0	0

By forcing an extra initial interval from 1 to 2, the advantage we got from complete partitioning is gone (loading more elements at a time to have less iteration latency) while the resource usage is almost the same from having more registers like solution 9. The performance is almost no different from that of solution 7 since we are wasting 1 extra cycle for nothing when it is ready to start the next operation.

Exercise 11:

Performance Estimates

Timing

Clock		solution9	solution11
ap_clk	Target	8.00 ns	5.00 ns
	Estimated	6.762 ns	3.880 ns

Latency

		solution9	solution11
Latency (cycles)	min	11	12
	max	11	12
Latency (absolute)	min	88.000 ns	60.000 ns
	max	88.000 ns	60.000 ns
Interval (cycles)	min	11	12
	max	11	12

Utilization Estimates

	solution9	solution11
BRAM_18K	0	0
DSP48E	24	24
FF	372	1168
LUT	480	544
URAM	0	0

The iteration latency becomes 4, however, the order and the behavior of the operations are the same except for immediately performing multiplications after loading elements, it waits for a cycle of loading elements to pass and then performs the multiplication. This might be the way it tries to meet the target clock cycle. The new design is faster but with a significantly increased number of flip-flops. With significantly less estimated clock period, even with 1 more cycle of latency but in terms of absolute latency, solution 11 is faster than that solution 9.

Using dual-port BRAM might be a way to improve the utilization since that would not require many extra registers.

DFT Exercise 1:

In terms of performance, the inner loop interval has an iteration latency of 51-54 and the outer loop has a 13070-13838 iteration latency, then the last loop has an iteration latency of 2 with all of them having 256 (N) trip counts. With 3346434- 3543042 total latency. Most of the resources are used as instances, 18 BRAMs (6%), 1248 DSPs (17%), 234240 FFs (3%), and 117120 LUTs (12%) with a 6.841ms estimated clock period.

DFT Exercise 2:

Performance Estimates

Timing

Clock		solution1	solution2
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.841 ns	6.675 ns

Latency

		solution1	solution2
Latency (cycles)	min	3346434	341250
	max	3543042	341250
Latency (absolute)	min	26.771 ms	2.730 ms
	max	28.344 ms	2.730 ms
Interval (cycles)	min	3346434	341250
	max	3543042	341250

Utilization Estimates

	solution1	solution2
BRAM_18K	18	12
DSP48E	215	110
FF	7432	5242
LUT	14617	8209
URAM	0	0

After pipelining the inner loop, the number of cycles becomes stable with better performance (less estimated clock period as well). Not only performance but in terms of utilization, the new design requires fewer resources from every component.

By pipelining, even if it is impossible to achieve 1 initial interval (we got 5) with only 42 iteration latency, this is ~10 fewer iterations compared to not pipelining causing to have less total number of cycles of iteration. Since there are data dependencies in each floating-point addition (expensive) which takes ~5 cycles, it is necessary to wait for the result before moving on.

DFT Exercise 3:

Performance Estimates

Timing

Clock		solution2	solution3
ap_clk	Target	8.00 ns	8.00 ns
	Estimated	6.675 ns	6.955 ns

Latency

		solution2	solution3
Latency (cycles)	min	341250	138498
	max	341250	138498
Latency (absolute)	min	2.730 ms	1.108 ms
	max	2.730 ms	1.108 ms
Interval (cycles)	min	341250	138498
	max	341250	138498

Utilization Estimates

	solution2	solution3
BRAM_18K	12	12
DSP48E	110	109
FF	5242	4300
LUT	8209	9888
URAM	0	0

With a little more estimated clock period but significantly fewer cycles of latency, the new design is better in terms of performance with some trad-off on FFs and LUTs causing not much difference in terms of utilization. The reason behind this is that the new design has less initial interval (only 2) since floating-point addition is the main part that makes solution 2 has 5 II, and with fewer bits operation (far less expensive), we could achieve less initial interval.

We forgot to consider the impact of the output in terms of accuracy which is necessary when changing datatype.

We could write a testbench and compare the results from the original design with the new datatype design and see if the errors are minor or not.

DFT Exercise 4:

```
// Calculate each frequency domain sample iteratively
dft_label1: for (i = 0; i < N; i += 1) {
    temp_real[i] = 0;
    temp_imag[i] = 0;

    // (2 * pi * i)/N
    w = (2.0 * 3.141592653589  / N) * (TEMP_TYPE)i;
    w = (2.0 * 3.141592653589  / N) * i;

    // Calculate the jth frequency sample sequentially
    dft_label0: for (j = 0; j < N; j += 1) {
        // Utilize HLS tool to calculate sine and cosine values
        c = cos(j * w);
        s = -sin(j * w);

        // Multiply the current phasor with the appropriate input sample and keep
        // running sum
        temp_real[j] += (sample_real[i] * c - sample_imag[i] * s);
        temp_imag[j] += (sample_real[i] * s + sample_imag[i] * c);
    }
}
```

Performance Estimates

Timing

Clock		solution1	solution2	solution3	solution4
ap_clk	Target	8.00 ns	8.00 ns	8.00 ns	8.00 ns
	Estimated	6.841 ns	6.675 ns	6.955 ns	6.625 ns

Latency

		solution1	solution2	solution3	solution4
Latency (cycles)	min	3346434	341250	138498	73218
	max	3543042	341250	138498	73218
Latency (absolute)	min	26.771 ms	2.730 ms	1.108 ms	0.586 ms
	max	28.344 ms	2.730 ms	1.108 ms	0.586 ms
Interval (cycles)	min	3346434	341250	138498	73218
	max	3543042	341250	138498	73218

Utilization Estimates

	solution1	solution2	solution3	solution4
BRAM_18K	18	12	12	22
DSP48E	215	110	109	203
FF	7432	5242	4300	7100
LUT	14617	8209	9888	15643
URAM	0	0	0	0

By swapping the inner loop and the outer loop, we achieved an initial interval of 1 resulting in a better performance but significantly increased number of resources required (almost double that of solution 3). The output shouldn't be affected since the S matrix is diagonally symmetric.