ABHISHEK NAYAK
nayak19@purdue.edu
Username - 695r48

Behavioral Synthesis

ECE 695R: System-on-Chip Design (Homework #5)

10th November 2021

OVERVIEW

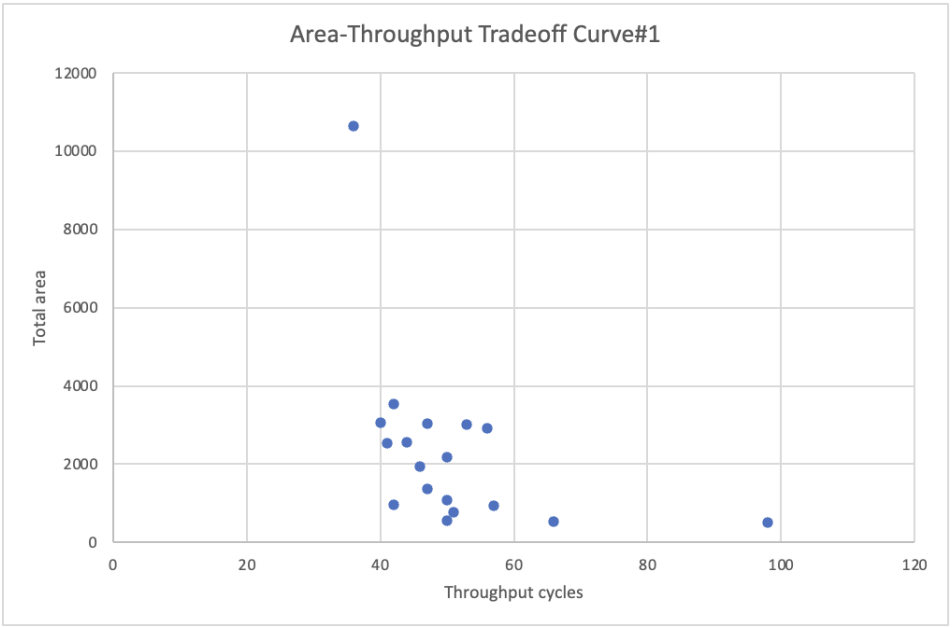
In this homework, I have used the Catapult Behavioral Synthesis tool from Mentor Graphics to perform behavioral synthesis of a Finite Impulse Response (FIR) filter. I have explored the area vs. performance tradeoff space and tried out advanced optimizations such as loop unrolling and pipelining to understand their impact on the synthesized design. I have also implemented the modified behavioral descriptions for the circular buffer implementation and the symmetric coefficient case.

ANALYSIS#1

Varying MAC loop's unrolling factor

In this exercise, I have synthesized the FIR filter C++ description (Files here) into an RTL implementation. With the SHIFT loop fully unrolled, I have varied the MAC loop's unrolling factor.

Area-Throughput tradeoff curve



Table

MAC unroll factor	Throughput cycles	Throughput time (in ns)	Total area
1	98	980	511.46
2	66	660	533.67
3	57	570	929.44
4	50	500	560.93
5	51	510	771.08
6	50	500	1075.83
7	47	470	1376.14
8	42	420	962.4
9	46	460	1935.57
10	50	500	2166.95
11	41	410	2526.37
12	44	440	2551.3
13	47	470	3025.5
14	53	530	3021.76
15	56	560	2918.34
16	40	400	3049.74
17	42	420	3536.39
Full	36	360	10634.51

Analysis

Complete loop unrolling exposes a maximal amount of parallelism at the cost of creating an implementation that requires a significant amount of resources. The cost of the increased area is due to multiple internal registers, being required to store the various intermediate and final variables in parallel. Thus, it ok to perform a complete loop unroll on “smaller” for loops. But completely unrolling a loop with a large number of iterations (e.g., one that iterates a million times) is typically infeasible.

Also, the results indicate that the relation between the unrolling factor and the overall area growth is non-linear. This indicates that for systems where the area is a constraint, designers can gain more throughput with less area by applying loop unrolling. We observed that the overall area either shrinks or increases at a very low rate for the first few times the loops are unrolled. This shows that there are different optimal times to unroll for different programs.

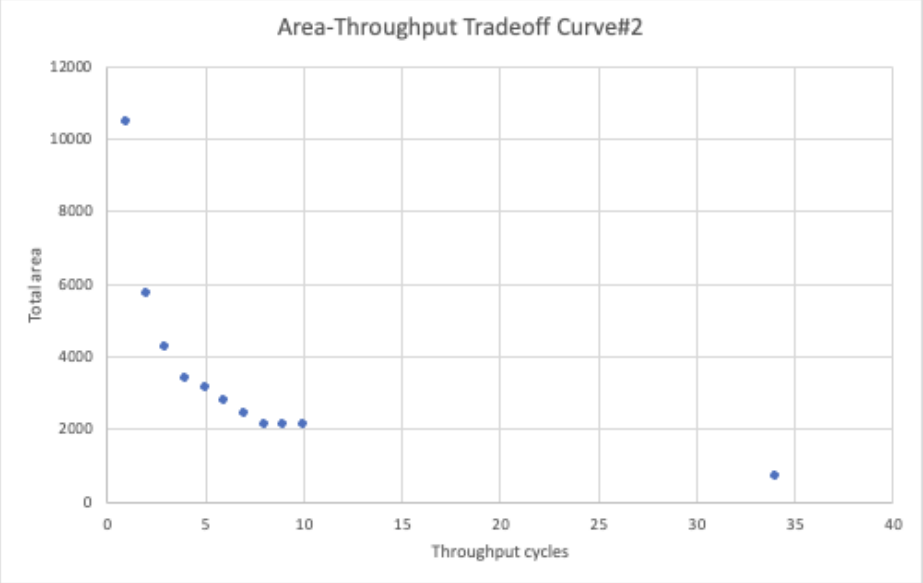
Working directory - /home/ecegridfs/a/695r48/ece695r/hw5/graph1/...

ANALYSIS#2

Varying pipeline initiation intervals

In this exercise, I have used pipelining to optimize the FIR filter's throughput. With the SHIFT and MAC loops fully unrolled, I have varied the 'main' pipeline initiation intervals and used the design points to generate the second area-throughput tradeoff curve.

Area-Throughput tradeoff curve



Table

Pipeline initiation interval cycles	Throughput cycles	Throughput time (in ns)	Total area
No pipeline	34	340	693.34
1	1	10	10446.3
2	2	20	5747.4
3	3	30	4264.5
4	4	40	3374.83
5	5	50	3137.62
6	6	60	2800.86
7	7	70	2400.13
8	8	80	2139.92
9	9	90	2139.95
10	10	100	2139.91

Analysis

Loop pipelining is an optimization that overlaps multiple iterations of a 'for' loop. The first data point is the non-pipelined version (different from the last row of Analysis#1's data since 'mgc_in_wire' is used).

Unrolling both SHIFT & MAC and then pipelining allows improved throughput but at an increased cost of the area. This is because to accommodate the logic, wiring, and pipeline registers.

The loop initiation interval is defined as the number of clock cycles until the next iteration of the loop can start. The best throughput is observed for the row with pipeline initiation interval as 1 cycle. With the subsequent increase in initiation interval, we see a gradual decrease in the area score.

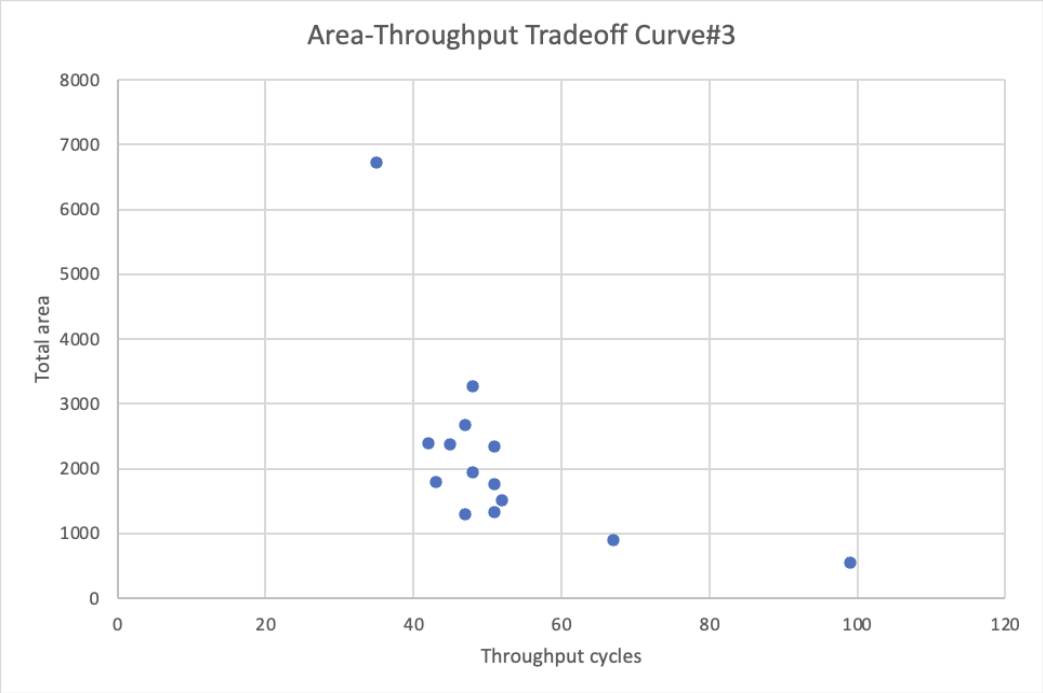
Directory - /home/ecegridfs/a/695r48/ece695r/hw5/graph2/...

ANALYSIS#3

Circular buffer implementation

In this exercise, I have implemented a circular buffer for the 32-tap FIR filter. I have modified the C++ description of the FIR filter that was provided to use a circular buffer and synthesized it using Catapult. For generating the third Area-Throughput tradeoff curve, I have varied the MAC loop's unrolling factor while keeping the SHIFT loop fully unrolled in all cases.

Area-Throughput tradeoff curve



Table

MAC Unroll factor	Throughput cycles	Throughput time (in ns)	Total area
1	99	990	546.8
2	67	670	899.9
3	47	470	1296.11
4	51	510	1326.81
5	52	520	1508.99
6	51	510	1760.86
7	48	480	1944.8
8	43	430	1791.74
9	47	470	2680.08
10	51	510	2341.22
11	42	420	2399.25
12	45	450	2377.72
13	48	480	3277.4
Full	35	350	6728.16

Analysis

Working with circular addressing mode, when the pointer to the buffer reaches the end or 'bottom' location of the circular buffer, and incremented, it is automatically wrapped around to point to the beginning or 'top' location of the buffer that contains the first element.

Modified code location - /home/ecegridfs/a/695r48/ece695r/hw5/graph3/fir.cpp

We see similar performance and area to be almost similar to the register's usage in Analysis#1. This is because the Dual-port RAM is slower than the register implementation.

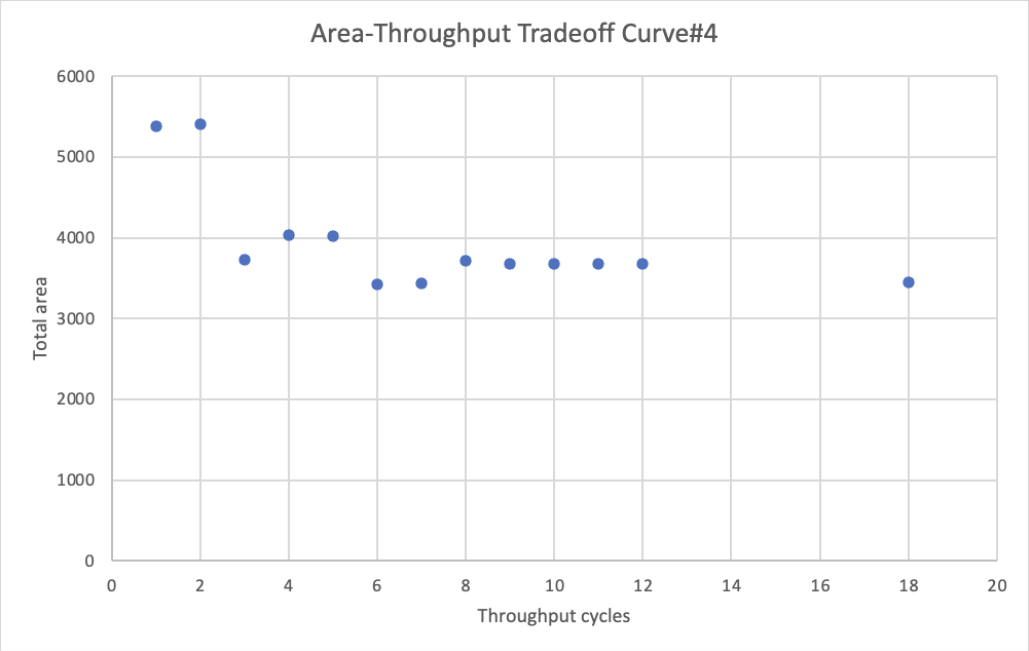
Directory - /home/ecegridfs/a/695r48/ece695r/hw5/graph3/...

ANALYSIS#4

Symmetrical co-efficient values of FIR Filter

In this exercise, I have reduced the number of multiplications in half to exploit the symmetry of the co-efficient values. I have re-written the original C++ description to reflect this transformation and repeated the steps in Analysis#2 to generate the fourth area-throughput tradeoff curve.

Area-Throughput tradeoff curve



Table

Pipeline initiation interval cycles	Throughput cycles	Throughput time (in ns)	Total area
No pipeline	18	180	3443.72
1	1	10	5375
2	2	20	5407.52
3	3	30	3729.05
4	4	40	4035.09
5	5	50	4019.48
6	6	60	3426.09
7	7	70	3428.59
8	8	80	3711.35
9	9	90	3680.13
10	10	100	3680.13
11	11	110	3680.13
12	12	120	3680.13

Analysis

We observe that with no-pipeline, the #throughput_cycle has dropped to 18, which is exactly half of what was implemented in the no-pipeline row of Analysis#2. This is because the number of MAC operations has been halved, owing to the symmetrical co-efficient data.

Also, the area has significantly dropped as well. This is because of fewer resource requirements.

Modified code location - /home/ecegridfs/a/695r48/ece695r/hw5/graph4/fir.cpp

Finally, we observe that the pipeline initiation interval can be set to 1 since there are not many resource constraints and/or dependencies in the code.

Directory - /home/ecegridfs/a/695r48/ece695r/hw5/graph4/...

Suggestions on tool improvements

- Catapult supports most C++ constructs including classes, structs, arrays, and pointers to statically allocated objects. However, it does not support dynamic memory allocation.
- I could specify parallelism, throughput, and memory configuration at high levels of abstraction using attributes and annotation; however, an in-depth understanding of the underlying hardware is required to achieve a good result.
- Catapult does not support speculative execution.
- Catapult can be improved by showing the percentage of utilization by resources.