ELEC522 – Fall 2022

Project 1- Xilinx Vitis Model Composer / Nvidia CUDA Tutorial Assignment

By

Hsuan-You (Shaun) Lin

Rice University, Houston, TX

Sep. 16, 2022

Supervised by

Dr. Joseph R. Cavallaro

Contents

List of Figures

List of Tables

Xilinx Lab:

Lab1: Introduction to Vitis Model Composer HDL Library

Step 1: Creating a Design in an FPGA

Lab1_1:

(a) Screen capture of spectrum plots of the initial waveforms

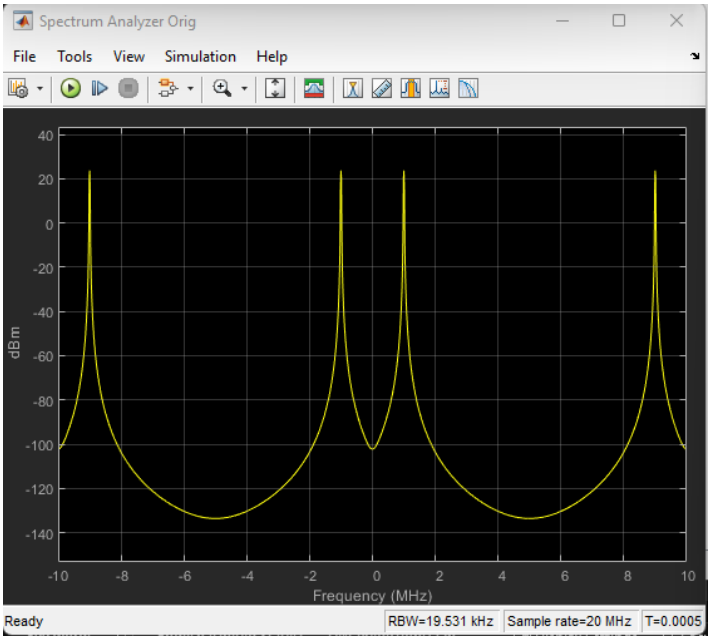


Figure 1.

(b) Screen capture of spectrum plots after adding Digital FIR Filter block

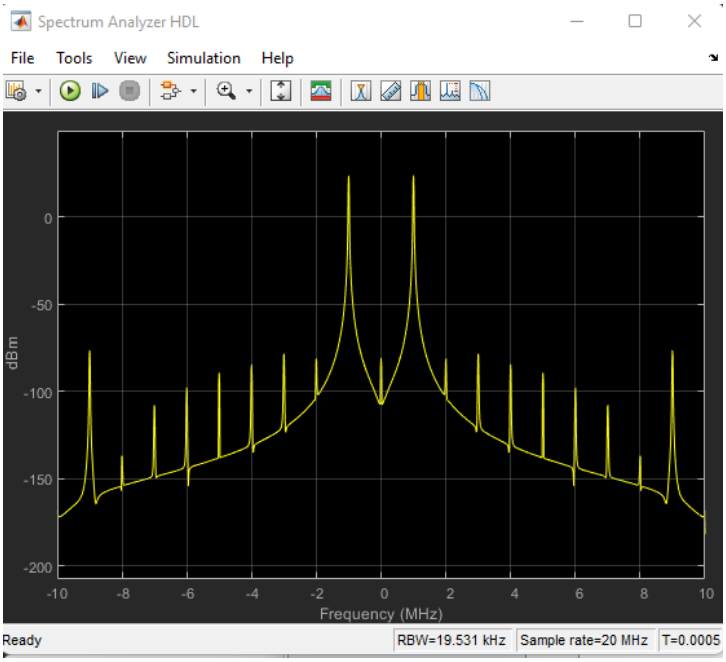


Figure 2.

(c) Screen capture of resource utilization output

Resource Analyzer: Lab1_1

OK | Hide Details

Post Synthesis Resources: Clicking on an instance name highlights corresponding block/subsystem in the model

Name	BRAMs (140)	DSPs (220)	LUTs (53200)	Registers (106400)
Lab1_1	0	6	281	492
Digital FIR Filter	0	6	281	492

Figure 3.

(d) Screen capture of final block diagram

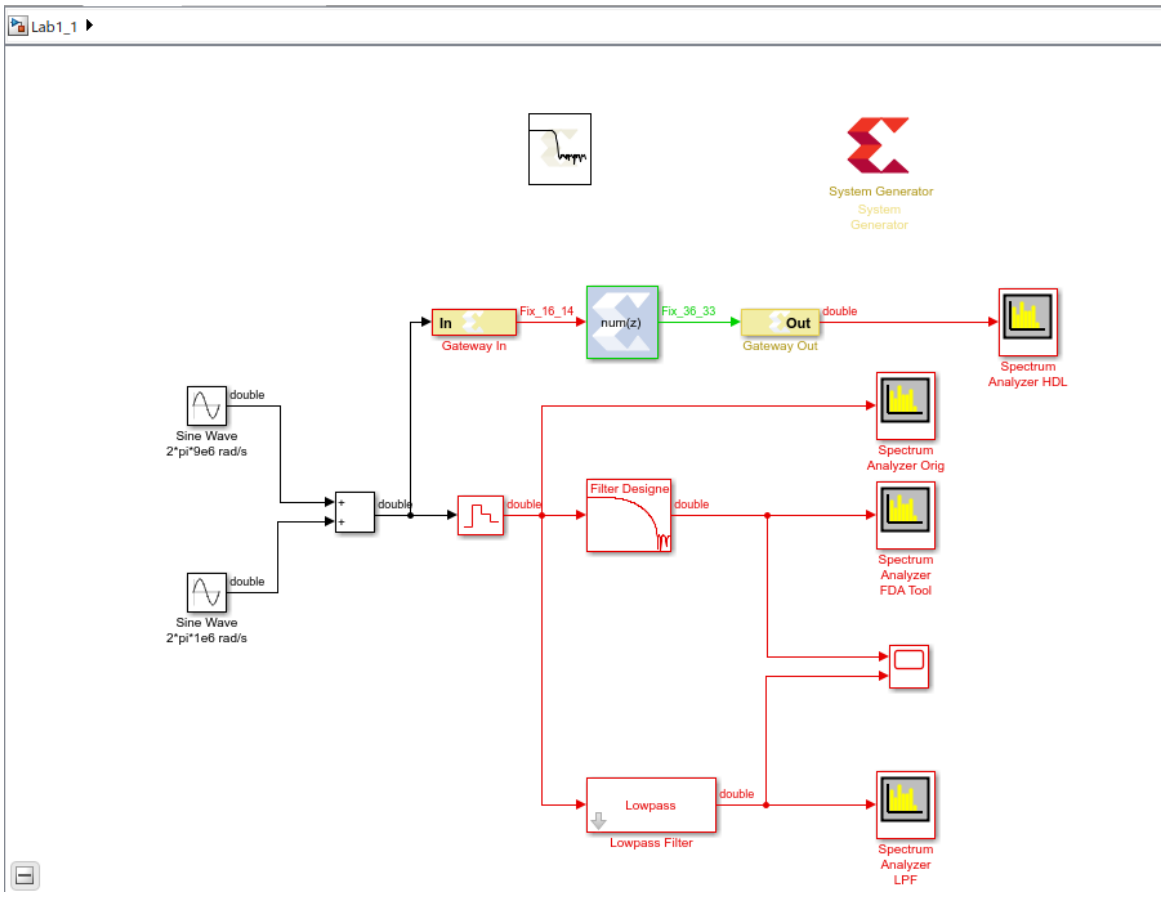


Figure 4.

(e) Compare your results with the tutorial results and note any differences or if the same.

My answer: No differences, I got the same results with the tutorial results.

Step 2: Creating an Optimized Design in an FPGA

Lab1_2:

(a) Screen capture of resource utilization output for higher frequency in Generate step

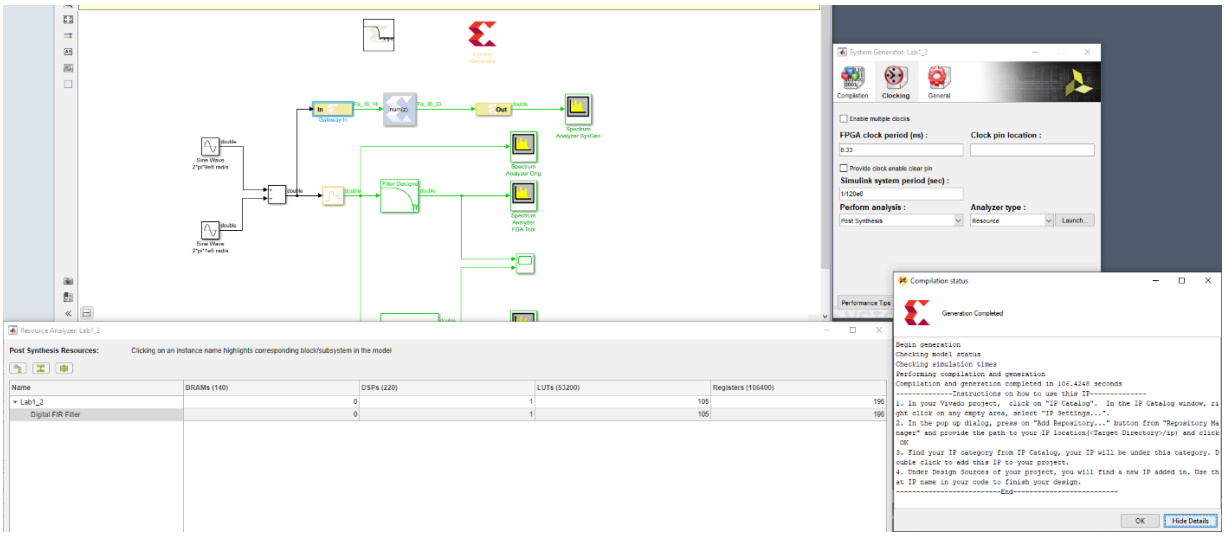


Figure 5.

(b) Compare your results with the tutorial results and note any differences or if the same.

My answer: I got the differences results with the tutorial results, but it seems like the tutorial results is wrong, because the title of the resource utilization output is "Lab1_4_1_sol" with the tutorial results not "Lab1_2".

Step 3: Creating a Design using Discrete Components

Lab1_3:

(a) Screen capture of final block diagram using discrete components

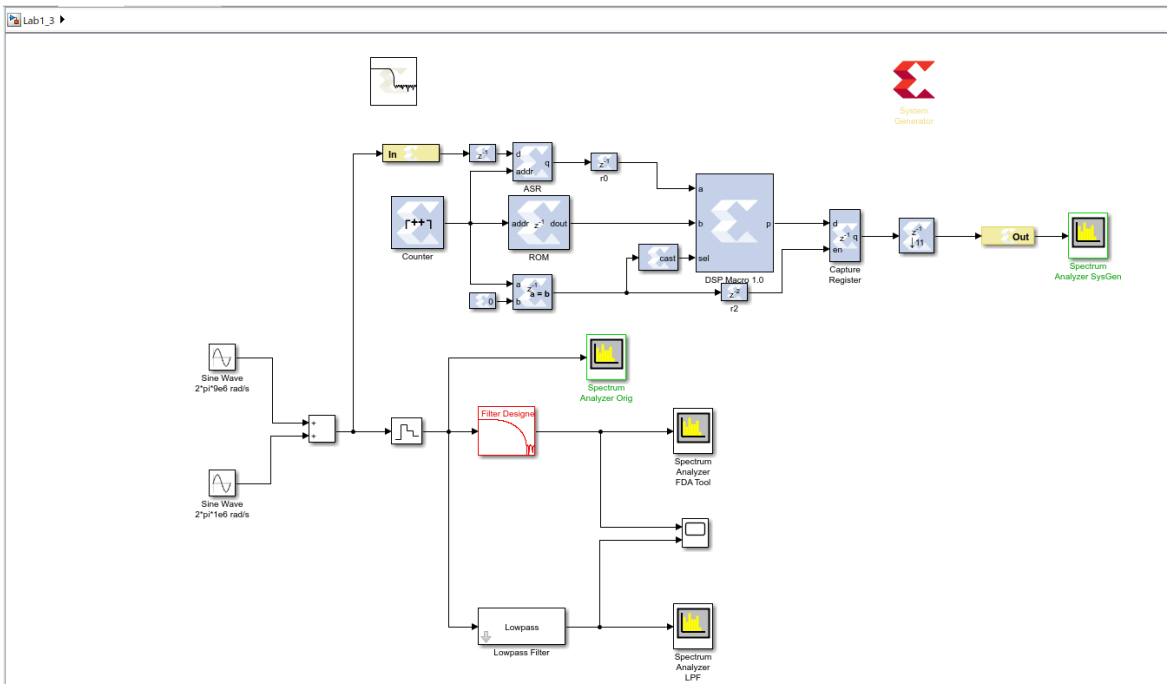


Figure 6.

(b) Screen capture of spectrum plots

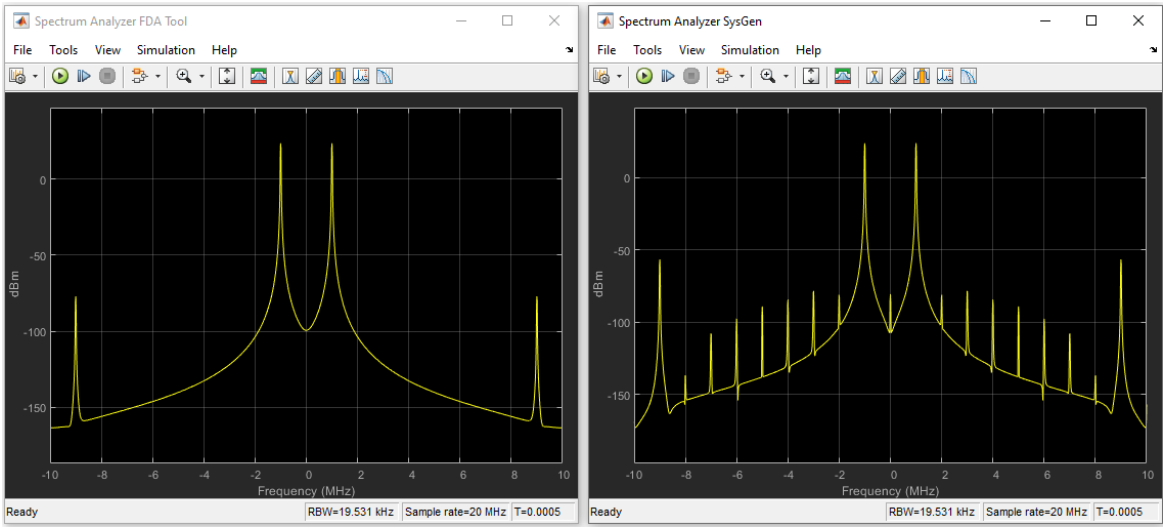


Figure 7.

(c) Screen capture of resource utilization output

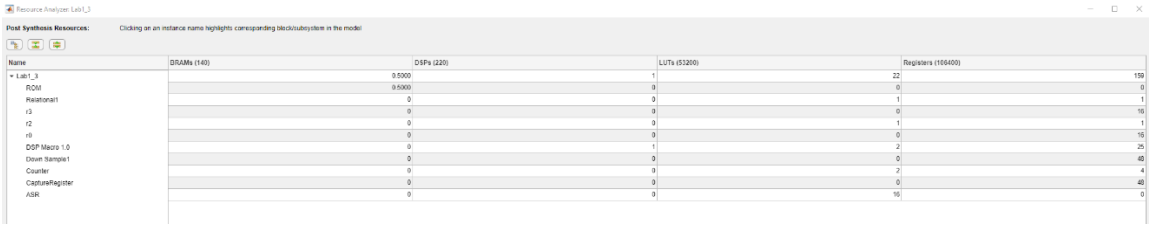


Figure 8.

(d) Compare your results with the tutorial results and note any differences or if the same.

My answer: No differences, I got the same results with the tutorial results.

Step 4: Working with Data Types

Lab1_4: (Part 1: Designing with Floating-Point Data Types)

(a) Screen capture of spectrum plots of **floating-point** version

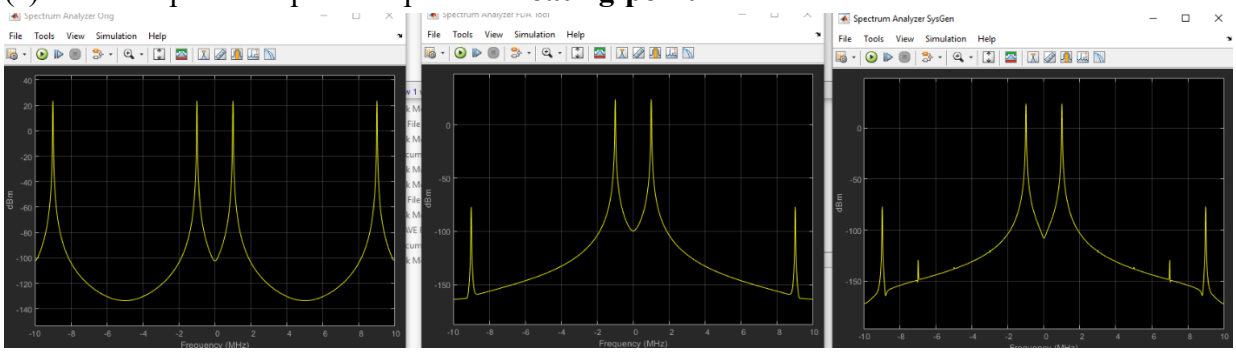
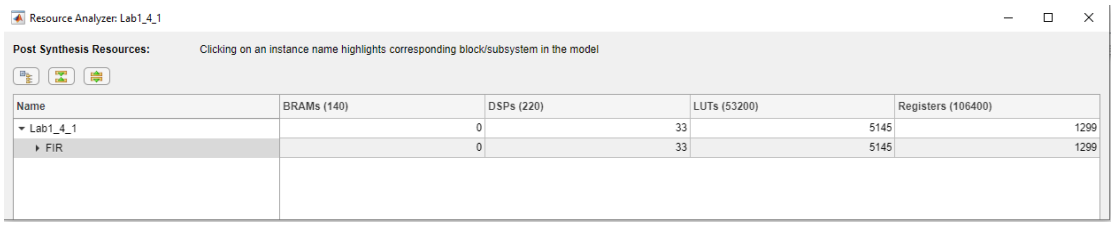


Figure 9.

(b) Screen capture of **floating-point** resource utilization output



Name	BRAMs (140)	DSPs (220)	LUTs (53200)	Registers (106400)
Lab1_4_1	0	33	5145	1299
FIR	0	33	5145	1299

Figure 10.

(c) Compare your results with the tutorial results and note any differences or if the same.

My answer: No differences, I got the same results with the tutorial results.

Lab1_4: (Part 2: Designing with Fixed-Point Data Types)

(d) Screen capture of scope results for **fixed-point** version – initial version

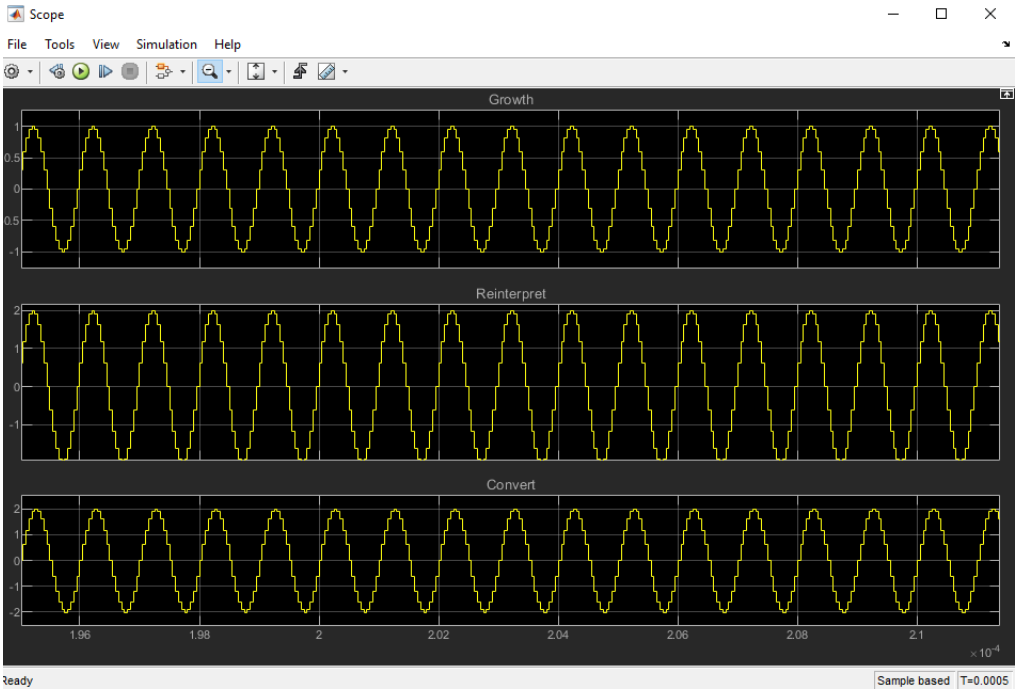


Figure 11.

(e) Screen capture of scope results for **fixed-point** version – after changing reinterpret and convert blocks

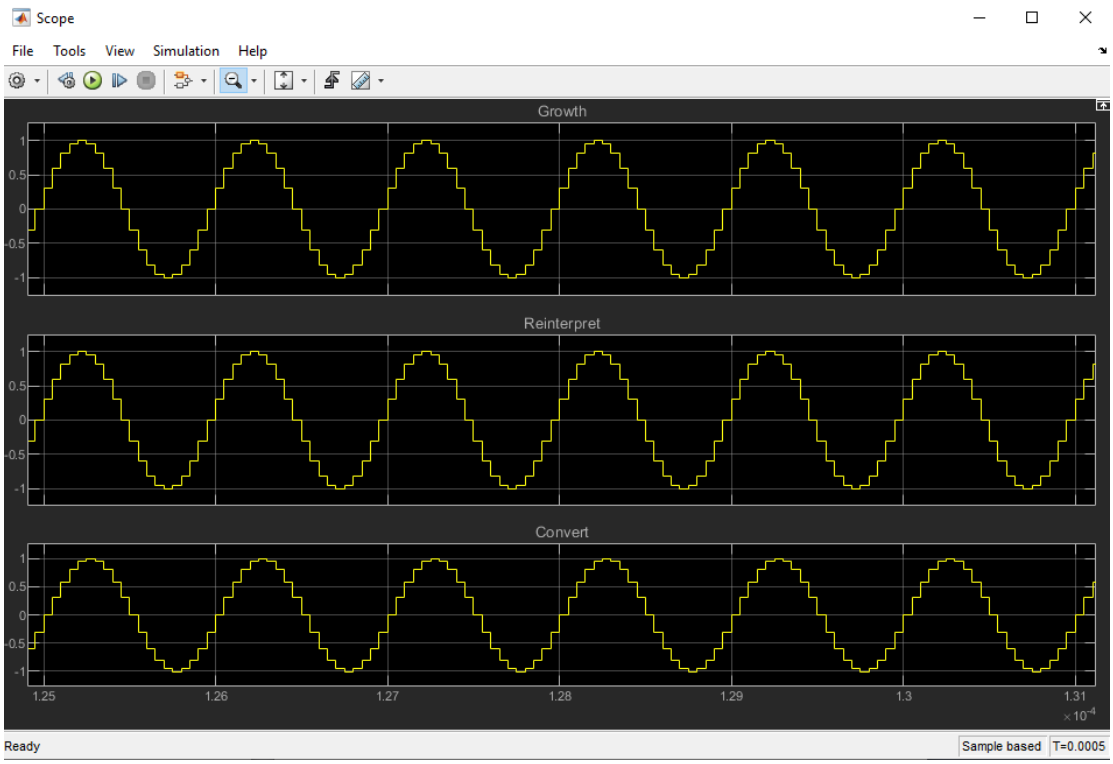
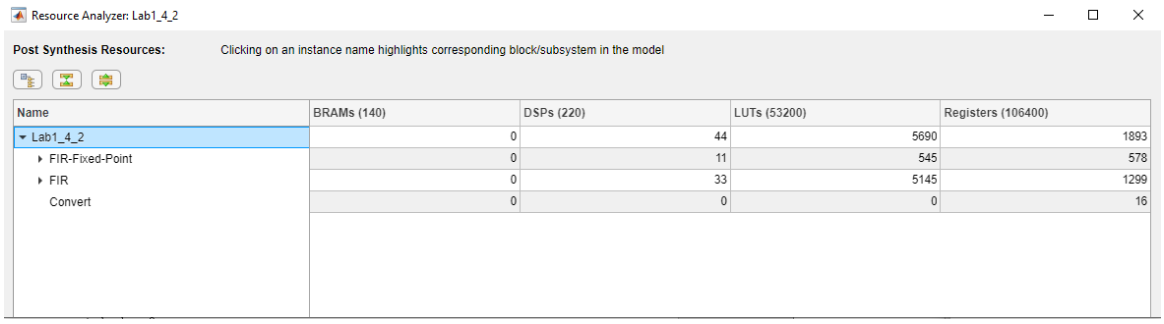


Figure 12.

(f) Screen capture of **fixed-point** resource utilization output



Name	BRAMs (140)	DSPs (220)	LUTs (53200)	Registers (106400)
Lab1_4_2	0	44	5690	1893
FIR-Fixed-Point	0	11	545	578
FIR	0	33	5145	1299
Convert	0	0	0	16

Figure 13.

(g) Compare your results with the tutorial results and note any differences or if the same.

My answer: No differences, I got the same results with the tutorial results.

Lab2: Importing Code into a Vitis Model Composer HDL Design

Step 1: Modeling Control with M-Code

Lab2_1:

(a) state_machine.m code:

```
function matched = state_machine(din)
persistent state, state = xl_state(0,{xlUnsigned, 3, 0});
switch state
    case 0
        if din == 1
            state = 1;
        else
            state = 0;
        end
        matched = 0;
    case 1
        if din == 0
            state = 2;
        else
            state = 1;
        end
        matched = 0;
    case 2
        if din == 1
            state = 3;
        else
            state = 0;
        end
        matched = 0;
    case 3
        if din == 1
            state = 4;
        else
            state = 2;
        end
        matched = 0;
    otherwise
        if din == 1
            state = 1;
        else
            state = 0;
        end
        matched = 1;
end
```

(b) Screen capture of final block diagram

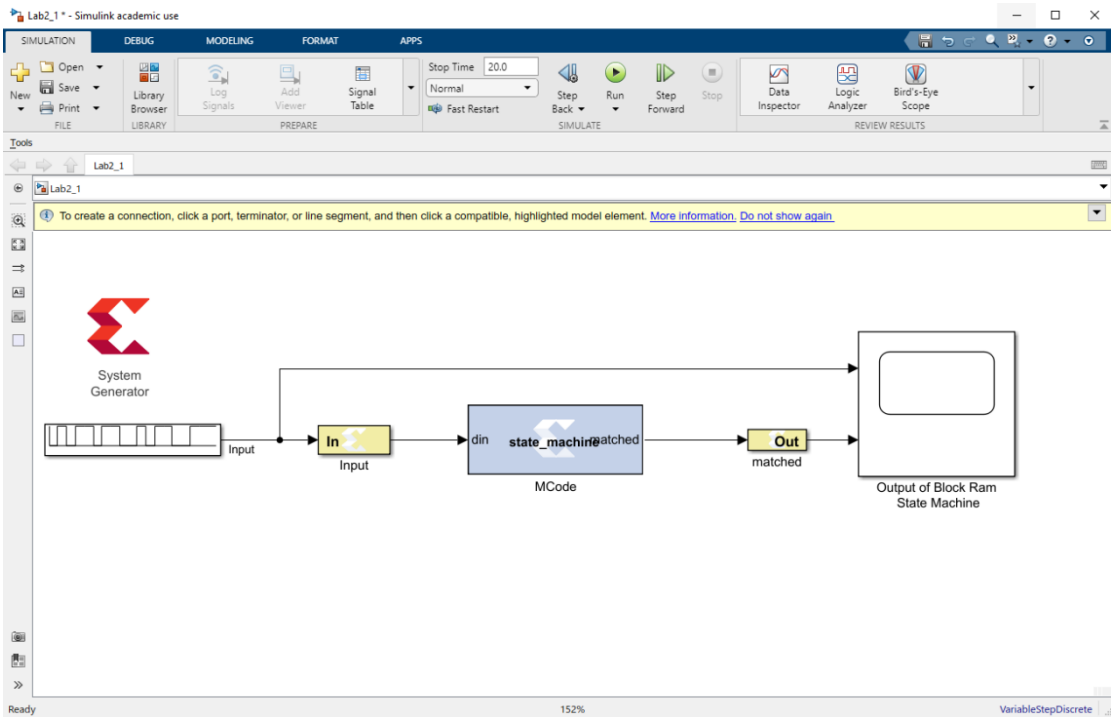


Figure 14.

(c) Screen capture of output waveform

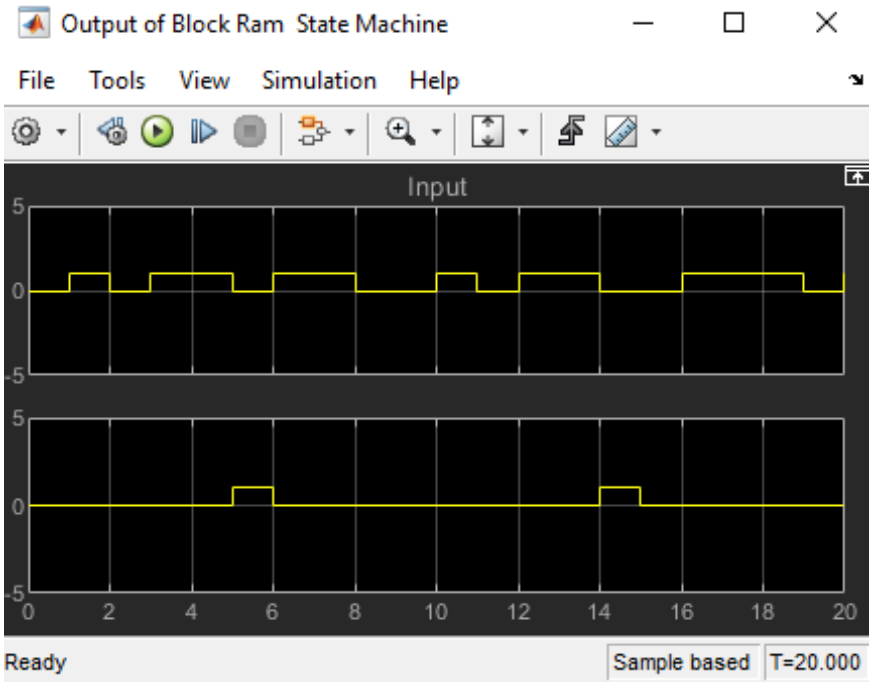


Figure 15.

(d) Compare your results with the tutorial results and note any differences or if the same.

My answer: No differences, I got the same results with the tutorial results.

Lab2_2:

Step 2: Modeling Blocks with HDL

Lab2_2:

(a) Screen capture of final block diagram

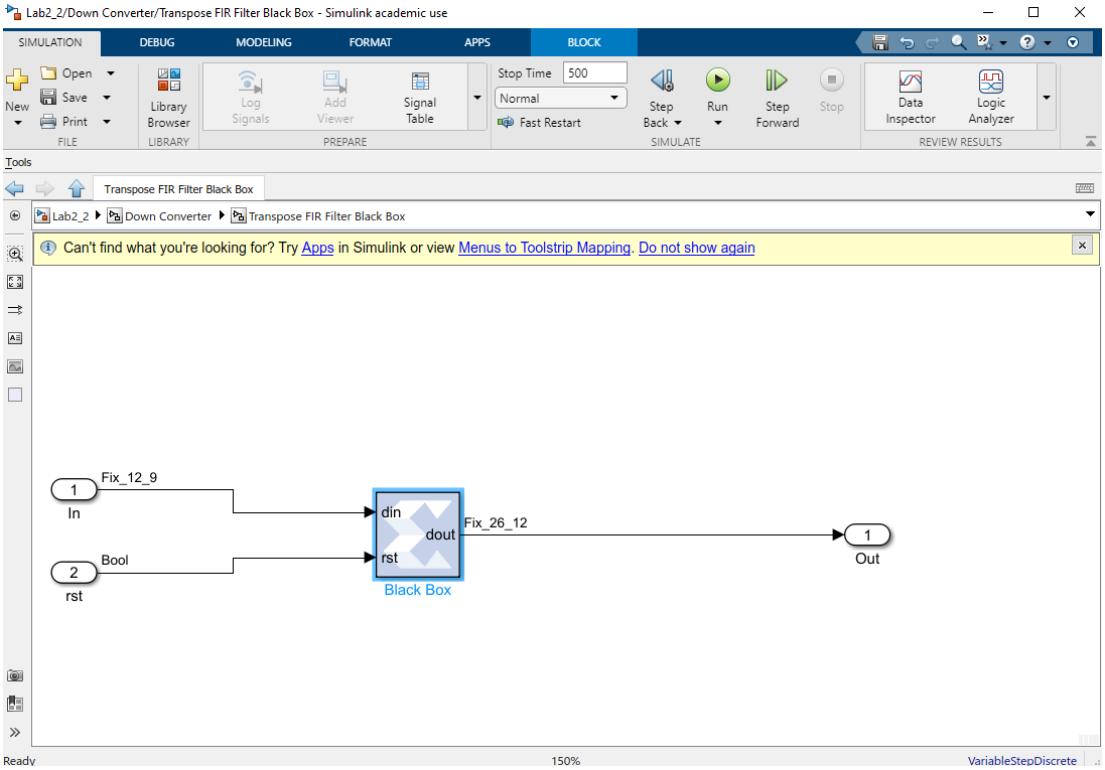


Figure 16.

(b) Screen capture of final output waveform

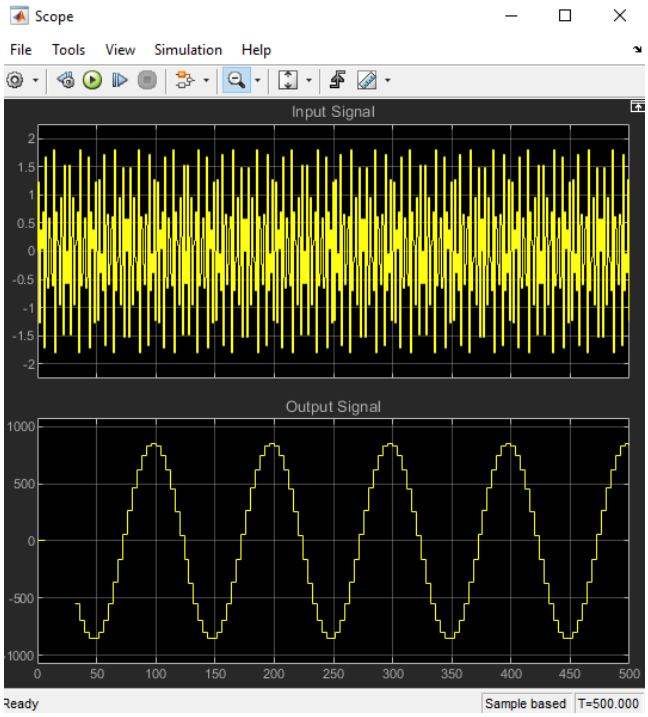


Figure 17.

(c) Compare your results with the tutorial results and note any differences or if the same.

My answer: No differences, I got the same results with the tutorial results.

Lab3: Timing and Resource Analysis

Step 1: Timing Analysis in Vitis Model Composer

Lab3_1:

(a) Screen capture of timing analyzer results – realize that the Kintex -3 part will have less timing error than the Zynq Artix -1 part

Post Synthesis Timing Paths: Clicking on an instance name highlights corresponding block/subsystem in the model

Violation type : setup

Select Columns

Status : FAILED

	Slack (ns)	Delay (ns)	Logic Delay (ns)	Routing Delay (ns)	Levels of Logic	Source	Destination	Source Clock	Destination Clock	Path Constraints
1	-3.2090	9.2990	4.4810	0.8090	14	Lab3/subsystem1/Mult	Lab3/subsystem1/AddSub2	clk	clk	create_clock -name clk -period 2 [get_ports clk]
2	-0.6390	2.0840	2.3850	0.4790	9	Lab3/addr_gen/Register4	Lab3/addr_gen/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]
3	-0.4450	2.0550	1.5940	0.7610	3	Lab3/addr_gen/Register10	Lab3/addr_gen/Relational1	clk	clk	create_clock -name clk -period 2 [get_ports clk]
4	-0.2790	2.3030	1.8030	0.5030	6	Lab3/addr_gen/Register	Lab3/addr_gen/AddSub1	clk	clk	create_clock -name clk -period 2 [get_ports clk]
5	-0.2160	1.9030	0.8870	0.8000	9	Lab3/subsystem1/coef	Lab3/subsystem1/Mult	clk	clk	create_clock -name clk -period 2 [get_ports clk]
6	-0.1450	1.9180	0.5180	0.8000	0	Lab3/Delay4	Lab3/subsystem1/coef	clk	clk	create_clock -name clk -period 2 [get_ports clk]
7	-0.0010	1.8360	0.8130	1.0230	1	Lab3/Delay7	Lab3/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]
8	0.4170	1.5170	1.0170	0	0	Lab3/Delay3	Lab3/Delay3	clk	clk	create_clock -name clk -period 2 [get_ports clk]
9	0.4480	1.5770	1.0690	0.5080	2	Lab3/addr_gen/Register	Lab3/addr_gen/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]
10	0.7100	1.2830	0.8130	0.4700	1	Lab3/Delay7	Lab3/Register1	clk	clk	create_clock -name clk -period 2 [get_ports clk]
11	0.8250	0.8940	0.4190	0.4750	0	Lab3/Delay3	Lab3/Delay7	clk	clk	create_clock -name clk -period 2 [get_ports clk]
12	0.8490	0.8520	0.5180	0.3340	0	Lab3/addr_gen/Relational1	Lab3/Delay3	clk	clk	create_clock -name clk -period 2 [get_ports clk]
13	0.8530	0.8760	0.5180	0.3580	0	Lab3/addr_gen/AddSub1	Lab3/addr_gen/Register10	clk	clk	create_clock -name clk -period 2 [get_ports clk]
14	0.8530	0.8760	0.5180	0.3580	0	Lab3/addr_gen/AddSub1	Lab3/addr_gen/Register5	clk	clk	create_clock -name clk -period 2 [get_ports clk]
15	0.8660	0.8630	0.5180	0.3450	0	Lab3/subsystem1/AddSub2	Lab3/Delay2	clk	clk	create_clock -name clk -period 2 [get_ports clk]
16	0.8770	0.8520	0.5180	0.3340	0	Lab3/addr_gen/Register5	Lab3/Delay4	clk	clk	create_clock -name clk -period 2 [get_ports clk]
17	0.8770	0.8520	0.5180	0.3340	0	Lab3/Delay4	Lab3/Delay4	clk	clk	create_clock -name clk -period 2 [get_ports clk]
18	0.8770	0.8520	0.5180	0.3340	0	Lab3/Delay2	Lab3/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]

Figure 18.

(b) Modify latency in various blocks in order to pass timing. This may be different than the suggestion in the tutorial. Explain your process.

My answer:

After I changed the latency from 1 to 2, I still got a bunch of errors, but I thought it was because the Zedboard couldn't handle so many calculations, so I changed the board to Zynq UltraScale+ and it passed.

(c) Screen capture of timing analyzer results – passing – after modifications

Post Synthesis Timing Paths: Clicking on an instance name highlights corresponding block/subsystem in the model

Violation type : setup

Select Columns

Status : PASSED

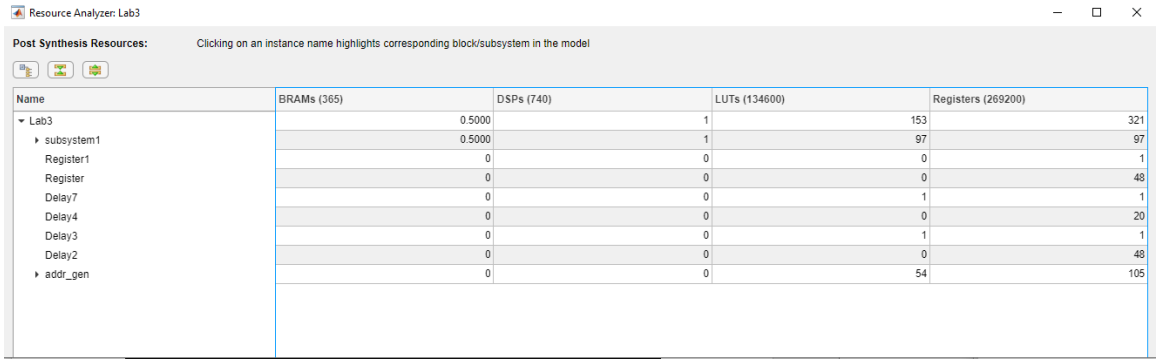
	Slack (ns)	Delay (ns)	Logic Delay (ns)	Routing Delay (ns)	Levels of Logic	Source	Destination	Source Clock	Destination Clock	Path Constraints
1	0.9318	1.0500	0.5880	0.4620	5	Lab3/Delay3	Lab3/subsystem1/AddSub2	clk	clk	create_clock -name clk -period 2 [get_ports clk]
2	1.0385	0.8840	0.8840	0	7	Lab3/subsystem1/Mult	Lab3/subsystem1/Mult	clk	clk	create_clock -name clk -period 2 [get_ports clk]
3	1.1290	0.5140	0.2130	0.3010	0	Lab3/subsystem1/coef	Lab3/subsystem1/Mult	clk	clk	create_clock -name clk -period 2 [get_ports clk]
4	1.1930	0.7690	0.4300	0.2890	2	Lab3/addr_gen/AddSub1	Lab3/addr_gen/Relational1	clk	clk	create_clock -name clk -period 2 [get_ports clk]
5	1.2390	0.6890	0.4740	0.2140	5	Lab3/addr_gen/Register	Lab3/addr_gen/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]
6	1.3270	0.6270	0.3020	0.2350	0	Lab3/Delay3	Lab3/Delay3	clk	clk	create_clock -name clk -period 2 [get_ports clk]
7	1.3500	0.7720	0.6770	0.1950	0	Lab3/Delay4	Lab3/subsystem1/coef	clk	clk	create_clock -name clk -period 2 [get_ports clk]
8	1.4040	0.5770	0.3870	0.1900	3	Lab3/addr_gen/Register5	Lab3/addr_gen/AddSub1	clk	clk	create_clock -name clk -period 2 [get_ports clk]
9	1.4600	0.4270	0.1300	0.2970	1	Lab3/Delay7	Lab3/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]
10	1.5300	0.3310	0.0770	0.2540	0	Lab3/Delay3	Lab3/Delay7	clk	clk	create_clock -name clk -period 2 [get_ports clk]
11	1.5630	0.2550	0.0700	0.1700	0	Lab3/addr_gen/Relational1	Lab3/Delay3	clk	clk	create_clock -name clk -period 2 [get_ports clk]
12	1.7050	0.3760	0.0790	0.1970	0	Lab3/addr_gen/AddSub1	Lab3/addr_gen/Register10	clk	clk	create_clock -name clk -period 2 [get_ports clk]
13	1.7050	0.3760	0.0790	0.1970	0	Lab3/addr_gen/AddSub1	Lab3/addr_gen/Register3	clk	clk	create_clock -name clk -period 2 [get_ports clk]
14	1.7150	0.2860	0.1300	0.1560	1	Lab3/Delay7	Lab3/Register1	clk	clk	create_clock -name clk -period 2 [get_ports clk]
15	1.7170	0.2840	0.0790	0.1850	0	Lab3/subsystem1/AddSub2	Lab3/subsystem1/AddSub2	clk	clk	create_clock -name clk -period 2 [get_ports clk]
16	1.7190	0.2620	0.0770	0.1850	0	Lab3/subsystem1/AddSub2	Lab3/Delay2	clk	clk	create_clock -name clk -period 2 [get_ports clk]
17	1.7190	0.2620	0.0770	0.1850	0	Lab3/addr_gen/Register5	Lab3/Delay4	clk	clk	create_clock -name clk -period 2 [get_ports clk]
18	1.7190	0.2620	0.0770	0.1850	0	Lab3/Delay4	Lab3/Delay4	clk	clk	create_clock -name clk -period 2 [get_ports clk]
19	1.7190	0.2620	0.0770	0.1850	0	Lab3/Delay2	Lab3/Register	clk	clk	create_clock -name clk -period 2 [get_ports clk]

Figure 19.

Step 2: Resource Analysis in Vitis Model Composer

Lab3_2:

(a) Screen capture of resource analyzer results for the Zynq Artix chip



Resource Analyzer: Lab3

Post Synthesis Resources: Clicking on an instance name highlights corresponding block/subsystem in the model

Name	BRAMs (365)	DSPs (740)	LUTs (134600)	Registers (269200)
Lab3	0.5000	1	153	321
subsystem1	0.5000	1	97	97
Register1	0	0	0	1
Register	0	0	0	48
Delay7	0	0	1	1
Delay4	0	0	0	20
Delay3	0	0	1	1
Delay2	0	0	0	48
addr_gen	0	0	54	105

Figure 20.

(b) Compare your results with the tutorial results and note any differences or if the same.

My answer: Basically no differences, there are just a few slightly different in the registers, I got almost the same results with the tutorial results.

CUDA Lab:

<Experiments with the Project 1 CUDA program>

(1) Screen capture of debug results window for arraySize = 10, threads = size, blocks = 1 initial parameters.

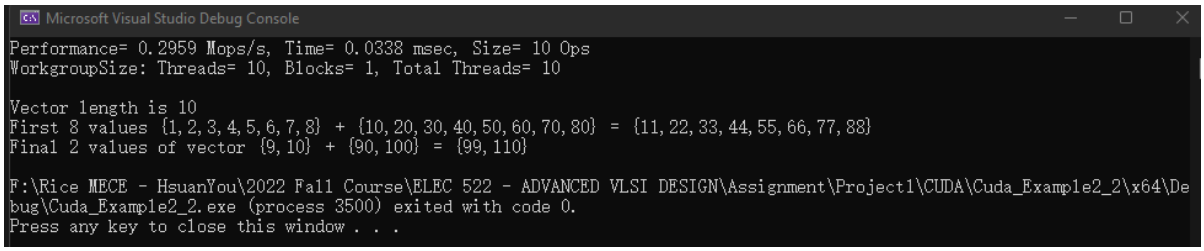


Figure 21.

(2) Include table of results for all of the arraySize values listed in the problem: 100, 200, 500, 750, 1000, 1250, 1500, 2000, 5000, 10000, 25000, and 64000. Table should record: Performance in Mops/s, Time, Size, Threads, Blocks, and Total Threads.

Table 1. arraySize 10 ~ 1250 performance result

arraySize	Performance (Mops/s)	Time (msec)	Size (Ops)	Threads (Set)	Blocks (Set)	Threads (Total)
10	0.2959	0.0338	10	10	1	10
100	3.6295	0.0276	100	100	1	100
200	6.7495	0.0296	200	200	1	200
500	20.4248	0.0245	500	500	1	500
750	26.1579	0.0287	750	750	1	750
1000	37.5601	0.0266	1000	1000	1	1000
1250	50.7965	0.0246	1250	1024	2	2048
1500	54.2535	0.0276	1500	1024	2	2048
2000	69.7545	0.0287	2000	1024	2	2048
5000	195.5570	0.0256	5000	1024	5	5120
10000	338.9371	0.0295	10000	1024	10	10240
25000	915.8852	0.0273	25000	1024	25	25600
64000	2040.8164	0.0314	64000	1024	63	64512

(3) What happens at arraySize = 1250 with the original threads and blocks value? Explain.

My answer:

It will show “Maximum threads/block is 1024. Increase block size and reduce threads/block”, as the result in *Figure1*. This is because the maximum number of threads per block is 1024 in the CUDA specifications, if we set arraySize = 1250, it

will exceed that value, after which there is no register space to save thread state and to limit memory and synchronization contention.

(4) Screen capture of the Debug results for arraySize = 25000

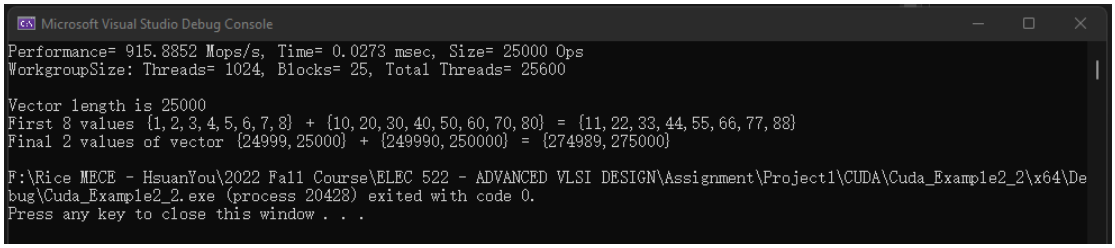


Figure 22.

(5) Final two values of variable c at 64000 with minimum threads and blocks set for 25000. How did you fix these values.

My answer:

If we used the minimum threads and blocks size for the arraySize = 25000, which means threads = 1024 and blocks = 25, then increase the arraySize = 64000 will shows that the final 2 values of vector {63999,64000} + {639990, 640000} = {0, 0}, this means you start blocks with thread count that is not divisible by the warp size, the hardware will simply execute the last warp with some of the threads "masked out" (i.e. they do have to execute, but without any effect on the state of the GPU/memory).

The easiest solution is increasing the number of blocks, let threads*blocks >= arraySize, in this case we set blocks = 63, which means we'll have total 64512 threads as shown in *Table1*.

(6) Analyze your table for the percentage increase in performance, time, and size.

My answer:

From *Table1*, we can see that as the arraySize increases, performance also increases, which means performance and size has the proportional relationship. In contrast, time is almost unchanged, because parallel threads to execute on the GPU, so time and size are irrelevant.