Spring 2021

California State University, Northridge

Department of Electrical & Computer Engineering

Lab 6

High Level Synthesis

April 25, 2021

ECE 520L

Written By: Jose Luis Martinez

Introduction

For this lab we are to follow the instructions in the Zynq Book Tutorial on designing with Vivado HLS. This software allows us to write C code and then verify and convert it into HDL. This lab will walk us through the steps of compiling and changing the constraints.

Objective

After completing this lab, students will be able to:
- Create a new project using Vivado HLS
- Simulate, synthesize, and implement a high level design a design
- Perform design analysis using the analysis capability of Vivado HLS using directives
- Create different solutions and study trade-offs
- Exercise interface synthesis process, and create HLS IP

Methodology

The following are the steps I took to complete this lab:

1. Follow the instructions for creating a Vivado HLS project by following the steps in Zynq Book Exercise 3A.
2. Optimize the code from tut3a by following the steps in Zynq Book Exercise 3B.
3. Analyze the interface summary by following the steps in Zynq Book Exercise 3C.

Conclusion

In this exercise/lab I learned how to create a project in Vivado HLS, how to create multiple solutions, how to apply directives, and how to compare the results of multiple solutions. For solution2 we applied a pipeline directive to the Product loop and HLS was not able to apply the constraints. For solution3 we applied a pipeline directive to the Col loop however due to ‘a’ and ‘b’ being read from more than 2 times at a time, HLS was not able to apply the constraints. In solution we alleviated the situation by resizing ‘a’ and ‘b’ allowing HLS to pipeline with the specified constraints. For solution5 I removed the pipeline directive for Col and instead applied it to the top level function. As we went from solution 1 to 5 the designs interval got faster but resource cost also increased. In the end the resource cost from solution 4 to 5 was too high for the speed that we were gaining.

Questions

1. For the 5x5 matrix multiplication (solution 1), approximately how many clock cycles does it take to read each matrix element? What part of the report shows this information?

According to the synthesis report, it takes 35 clock cycles to read each matrix element.

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row	935	935	187	-	-	5	no
+ Col	185	185	37	-	-	5	no
++ Product	35	35	7	-	-	5	no

2. For the 5x5 matrix multiplication (solution 2), approximately how many clock cycles does it take to read each matrix element? What part of the report shows this information?

According to the synthesis report, after pipelining the product loop it takes 14 clock cycles to read each matrix element.

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Row_Col	475	475	19	-	-	25	no
+ Product	14	14	7	2	1	5	yes

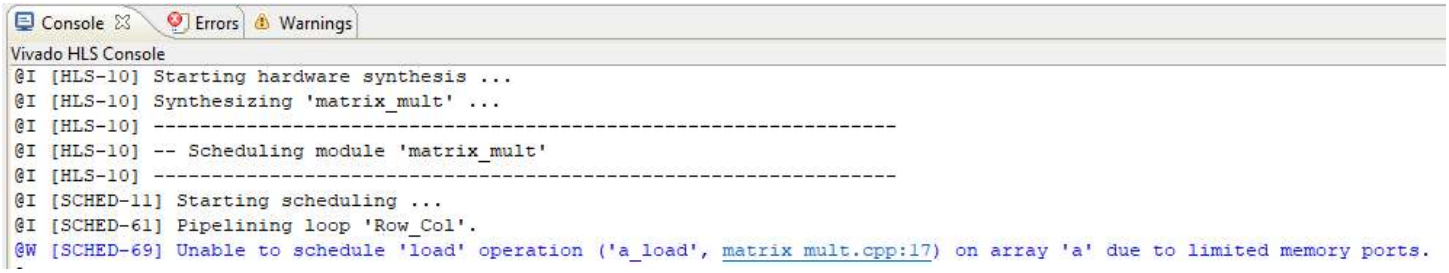
3. Has any pipelining been applied to any of the loops in Solutions 2 at high level in C code? If yes, which loop? If yes, is HLS able to apply this constraint to the hardware?

Pipelining has been applied to the product loop. According to the console logs HLS was unable to enforce the constraints.

```
@W [SCHED-68] Unable to enforce a carried dependency constraint (II = 1, distance = 1)
   between 'store' operation (matrix_mult.cpp:17) of variable 'tmp_8', matrix_mult.cpp:17 on array 'prod' and 'load' operation ('prod_load', matrix_mult.cpp:17) on array 'prod'
@I [SCHED-61] Pipelining result: Target II: 1, Final II: 2, Depth: 7.
@W [SCHED-21] Estimated clock period (6.74ns) exceeds the target (target clock period: 5ns, clock uncertainty: 0.625ns, effective delay budget: 4.38ns).
@W [SCHED-21] The critical path consists of the following:
        'load' operation ('prod_load', matrix_mult.cpp:17) on array 'prod' (2.39 ns)
        'add' operation ('tmp_8', matrix_mult.cpp:17) (1.96 ns)
        'store' operation (matrix_mult.cpp:17) of variable 'tmp_8', matrix_mult.cpp:17 on array 'prod' (2.39 ns)
```

4. What part of the report proves that solution 3 failed meeting the requirement? Present your proof.

In the console log, it mentions that it was unable to schedule ‘a_load’ due to ‘a’ having a limited memory space.



5. Has any pipelining been applied to any of the loops in Solutions 3 at high level in C code? If yes, which loop? If yes, is HLS able to apply this constraint to the hardware?

Pipelining has been applied to the Col loop with a II: 1. HLS was not able to apply the constraint due to the limitations in resources.

6. Explain what exactly the issue is with initial settings of solution 3 (only II=1).

Due to ‘a’ only having two memory ports, HLS is unable to schedule more than 2 reads from ‘a’.

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
1-6	⊞I/O Ports												
7-12	⊞Instances												
13	⊟Memory Ports												
14	b(p1)			read	read								
15	b(p0)			read	read	read							
16	a(p0)			read	read	read							
17	a(p1)				read	read							
18	prod(p0)												write
19-43	⊞Expressions												

7. Explain why solution 4 is offered as an optimization process with respect to solution 3.

Because we resized the arrays to smaller portions we are able to access more elements at the same time therefore allowing HLS to pipeline loop Col with II: 1.

8. In Solution 4, explain why dimension for arrays a and b are selected as 2 and 1 respectively?

The product loop uses k for its index so we should partition array a and b along the k dimension. In matrix_mult.cpp we are able to see a[i][k] and b[k][j]. So we would partition ‘a’ at dimension 2 and ‘b’ at dimension 1.

9. Explain improvement in performance from solution 1 to solution 4? What has been achieved? What is the overall speed up in performance in terms of processing the same data size?

The improvement from solution1 to solution4 is that ‘a’ and ‘b’ have been resized in order to pipeline the loop Col therefore reducing the amount of clock cycles required to complete the operation. From the Vivado HLS Report Comparison we can see that solution1 has an interval of 937 cycles and solution 5 35 cycles. Solution 4 is faster by 932 cycles or roughly 27 times faster solution1.

Vivado HLS Report Comparison

All Compared Solutions

solution4: xc7z010clg400-1

solution1: xc7z010clg400-1

Performance Estimates

Timing (ns)

Clock		solution4	solution1
ap_clk	Target	5.00	5.00
	Estimated	3.89	3.89

Latency (clock cycles)

		solution4	solution1
Latency	min	34	936
	max	34	936
Interval	min	35	937
	max	35	937

Utilization Estimates

	solution4	solution1
BRAM_18K	0	0
DSP48E	5	1
FF	260	102
LUT	92	84

10. What is the difference between applying the PIPELINE directive for Solution 5 against solution 4?

The PIPELINE directive for solution5 is applied to the entire function while the PIPELINE directive for solution4 is applied to the Col loop with an II=1.

11. Present your analysis by comparing solution 4 and 5 and study the trade-offs. State what you are gaining against what you are losing by moving from solution 4 to solution 5.

Solution5 is faster than Solution4 but it comes at the cost of hardware resources. Solution5 has an interval of 13 clock cycles while Solution4 is 35 clock cycles. From the Utilization Estimates we are also able to see that compared to Solution4, Solution5 uses 120 more DSP48E, 3691 more FF, and 1254 more LUT. So by speeding up our design by almost 3 times we are spending around 15 times more resources. Solution5 is not worth the extra speed as we spending way more in resources.

Vivado HLS Report Comparison

All Compared Solutions

solution5: xc7z010clg400-1

solution4: xc7z010clg400-1

Performance Estimates

Timing (ns)

Clock		solution5	solution4
ap_clk	Target	5.00	5.00
	Estimated	3.89	3.89

Latency (clock cycles)

		solution5	solution4
Latency	min	23	34
	max	23	34
Interval	min	13	35
	max	13	35

Utilization Estimates

	solution5	solution4
BRAM_18K	0	0
DSP48E	125	5
FF	3951	260
LUT	1346	92

12. Explain the logical flow of the optimization flow from solution 1 to solution 5. How did the directives change and how they affected the optimization process. Be clear and succinct in why it made sense going from one solution to another and how to change the directives to improve the performance.

Solution1 did not have any directives so the loops were not pipelined at all. HLS has to schedule everything one after the other there having a very long interval of 937 clock cycles. So as we progressed from solution1 to solution5 we started to apply pipeline directives from the inner loop to the most outer one. As we went from inner to the outer loop, the design got faster but also the hardware cost increased.

13. Explain why there is a difference between the interfaces from solution 1 to solution 5? How does this affect the overall design? Justify your answer.

Solution5 has a bigger interface because ‘a’ and ‘b’ have been resized to smaller sizes. Pipelining has been applied to solution5 increasing the number of RTL ports.

matrix_mult.cpp | compare reports | Synthesis(solution5)

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	matrix_mult	return value
ap_rst	in	1	ap_ctrl_hs	matrix_mult	return value
ap_start	in	1	ap_ctrl_hs	matrix_mult	return value
ap_done	out	1	ap_ctrl_hs	matrix_mult	return value
ap_idle	out	1	ap_ctrl_hs	matrix_mult	return value
ap_ready	out	1	ap_ctrl_hs	matrix_mult	return value
a_address0	out	3	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	40	ap_memory	a	array
a_address1	out	3	ap_memory	a	array
a_ce1	out	1	ap_memory	a	array
a_q1	in	40	ap_memory	a	array
b_address0	out	3	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	40	ap_memory	b	array
b_address1	out	3	ap_memory	b	array
b_ce1	out	1	ap_memory	b	array
b_q1	in	40	ap_memory	b	array
prod_address0	out	5	ap_memory	prod	array
prod_ce0	out	1	ap_memory	prod	array
prod_we0	out	1	ap_memory	prod	array
prod_d0	out	16	ap_memory	prod	array
prod_address1	out	5	ap_memory	prod	array
prod_ce1	out	1	ap_memory	prod	array
prod_we1	out	1	ap_memory	prod	array
prod_d1	out	16	ap_memory	prod	array

matrix_mult.cpp | compare reports | Synthesis(solution5) | Synthesis(solution1)

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	matrix_mult	return value
ap_rst	in	1	ap_ctrl_hs	matrix_mult	return value
ap_start	in	1	ap_ctrl_hs	matrix_mult	return value
ap_done	out	1	ap_ctrl_hs	matrix_mult	return value
ap_idle	out	1	ap_ctrl_hs	matrix_mult	return value
ap_ready	out	1	ap_ctrl_hs	matrix_mult	return value
a_address0	out	5	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	8	ap_memory	a	array
b_address0	out	5	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	8	ap_memory	b	array
prod_address0	out	5	ap_memory	prod	array
prod_ce0	out	1	ap_memory	prod	array
prod_we0	out	1	ap_memory	prod	array
prod_d0	out	16	ap_memory	prod	array