

Spring 2021
California State University, Northridge
Department of Electrical & Computer Engineering



Lab 7
Vivado HLS
Design Flow Lab
April 25, 2021
ECE 520L

Written By: Jose Luis Martinez

Introduction

For this lab I will follow the steps in the lab manual taking me over the whole Vivado HLS flow. In this lab I will create a Vivado HLS project using the GUI and simulate, synthesize, and implement the provided design.

Objective

After completing this lab, you will be able to:

- Create a new project using Vivado HLS GUI
- Simulate a design
- Synthesize a design
- Implement a design
- Perform design analysis using the Analysis capability of Vivado HLS
- Analyze simulator output using Vivado and XSim simulator

Methodology

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

1. Create a Vivado HLS project using the gui.
 - a. For the sources add matrixmul.cpp and matrixmul.h.
 - b. For the test files add matrixmul_test.cpp.
 - c. Specify the xc7z010clg400-1 as the target device.
2. Run C simulation to view the expected output.
 - a. Go to project > Run C Simulation and click ok.
 - b. You will be able to see if your simulation output was correct if in the console it prompts "Test passed."
3. Run the debugger
 - a. Go to project > Run C Simulation but this time select Launch Debugger and then click ok.
 - b. Put breakpoints on lines 105 and 101 in the matrixmul_test.cpp.
 - c. Press F6 until i=j=k=1 and record sw results.
 - d. Click resume.
 - e. Press F5 to step into matrixmult.cpp.
 - f. Press F6 until i=j=k=1 and record the results.
 - g. Press F7 and record the results from sw_result and hw_result.
4. Synthesize the design.
 - a. Switch to Synthesis view.
 - b. Run C Synthesis by clicking the green arrow head.
 - c. View and record the results from the Synthesis Report.
5. Switch to analysis perspective.
 - a. Expand Row, Col, and Product to view the timing schedule for each loop.
 - b. Right click one of the cells and select Goto Source to see where in the code the schedule comes from.
 - c. Expand the Performance Profile to see Iteration Latency.

6. Run C/RTL Co-simulation
 - a. To run the cosimulation go to Solution > Run C/RTL Cosimulation and select auto for Simulator Settings and VHDL for RTL Selection.
 - b. Go to console to check if the test passed.
7. View simulation results in Vivado.
 - a. Go to solution > Run C/RTL Cosimulation and select VHDL and then for the Dump Trace select All.
 - b. Open Vivado and in the tcl console type the following


```
cd C:/Zynq_Book/hls/solution1/sim/vhdl
current_fileset
open_wave_database matrixmul.wdb
open_wave_config matrixmul.wcfg
```


 - i. You should be able to see the waveforms of the entire simulation in one iteration.
 - d. Set a_address0, b_address0, and res_address0 radix to unsigned decimal.
 - e. Set a_q0, b_q0, and res_d0 radix to signed decimal.
 - f. Take a screenshot and then exit Vivado.
8. Back in Vivado HLS export the design by selecting VHDL as the language and run the implementation by selecting the evaluate option.
 - a. In the explorer you can see the implementation in the folder impl/vhdl
 - b. The project is zipped up in /impl/ip for you to use in vivado as an ip.
 - c. Create 3 solutions and compare the results of the Synthesis report base on the directives below:















```
Solution 2: Assign PIPELINE directive with II = 1, II = 2, and II = 3 to the outer loop.
Solution 3: Assign LOOP_FLATTEN constraint to the outer loop.
Solution 4: Assign UNROLL constraint to the outer loop.
```
 - d. Write a conclusion and close Vivado HLS.

Conclusion

In this lab I learned the Vivado HLS flow that goes from C code to RTL. I started off by creating a new project Vivado HLS and including the files provided to us. I then ran the C simulation to confirm that the design is working properly. Then I used the debugger tool to see what happens in code at certain points. Then the solution is synthesized and we can see the results of latency and resource utilization in the Synthesis Report. I also learned how to export the HLS project into vivado as an IP so it can easily be used in Vivado as an IP.

Questions

- 3-1-5. Using the Step Over (F6) button () a few times and observe the execution progress. Note the variables as you step through the code. What are the values of sw_results matrix when i=j=k=1? Provide the proof for your claim by screenshot of the variables i, j, k, and sw_results. Are these values correct? If not, is there any bug in the code? Explain your answer.

(x)= Variables  Breakpoints  1010 0101 Registers  Expressions  Modules 		
Name	Type	Value
(x)= argc	int	1
>  argv	char **	0x1035a00
(x)= k	int	1
(x)= j	int	1
(x)= i	int	1
>  in_mat_a	char [3][3]	0x61fecf
>  hw_result	short [3][3]	0x61feb4
(x)= err_cnt	int	0
>  in_mat_b	char [3][3]	0x61fec6
▼  sw_result	short [3][3]	0x61fea2
▼  sw_result[0]	short [3]	0x61fea2
(x)= sw_result[0][0]	short	870
(x)= sw_result[0][1]	short	906
(x)= sw_result[0][2]	short	942
▼  sw_result[1]	short [3]	0x61fea8
(x)= sw_result[1][0]	short	1086
(x)= sw_result[1][1]	short	308
(x)= sw_result[1][2]	short	28195
▼  sw_result[2]	short [3]	0x61feae
(x)= sw_result[2][0]	short	30239
(x)= sw_result[2][1]	short	1664
(x)= sw_result[2][2]	short	-12862

The values are correct up to `sw_result[1][0]`, but that is due to the fact that the for loop has not finished all of its iterations.

3-1-9. Using the **Step Over (F6)** several times, observe the computed results. Note the variables as you step through the code. What are the values of `res` array when `i=j=k=1`? Provide the proof for your claim by screenshot of the variables `i`, `j`, `k`, and `res`. Are these values correct and do they match with the `sw_results` that you obtained in the first part? If not, is there any bug in the code? If yes, fix it and once satisfied, you can use the **Step Return (F7)** button to return from the function.

(x)= Variables		
Breakpoi 1010 0101 Register Expressi Modules		
Name	Type	Value
> ➔ a	char (*)[3]	0x61fecf
> ➔ b	char (*)[3]	0x61fec6
▼ ➔ res	short (*)[3]	0x61feb4
▼ 🌐 *res	short [3]	0x61feb4
▼ 🌐 *res[0]	short [3]	0x61feb4
(x)= *res[0][0]	short	870
(x)= *res[0][1]	short	906
(x)= *res[0][2]	short	942
▼ 🌐 *res[1]	short [3]	0x61feba
(x)= *res[1][0]	short	1086
(x)= *res[1][1]	short	308
(x)= *res[1][2]	short	0
▼ 🌐 *res[2]	short [3]	0x61fec0
(x)= *res[2][0]	short	7328
(x)= *res[2][1]	short	64
(x)= *res[2][2]	short	-336
(x)= k	int	1
(x)= j	int	1
(x)= i	int	1

The values are correct up to and match the software results up to `sw_result[1][0]`, but after that the numbers do not match beyond that point. This is due to the fact that the for loop has not finished iterating and there are random numbers in those indexes.

3-1-10. The program execution will suspend at line 105 as we had set a breakpoint. Provide the proof such as following to show software and hardware results match in Variables view.

(x)= Variables Breakpoints Registers Expressions Modules		
Name	Type	Value
(x)= argc	int	1
> argv	char **	0x1085a00
> in_mat_a	char [3][3]	0x61fecf
▼ hw_result	short [3][3]	0x61feb4
▼ hw_result[0]	short [3]	0x61feb4
(x)= hw_result[0][0]	short	870
(x)= hw_result[0][1]	short	906
(x)= hw_result[0][2]	short	942
> hw_result[1]	short [3]	0x61feba
> hw_result[2]	short [3]	0x61fec0
(x)= err_cnt	int	0
> in_mat_b	char [3][3]	0x61fec6
▼ sw_result	short [3][3]	0x61fea2
▼ sw_result[0]	short [3]	0x61fea2
(x)= sw_result[0][0]	short	870
(x)= sw_result[0][1]	short	906
(x)= sw_result[0][2]	short	942
> sw_result[1]	short [3]	0x61fea8
> sw_result[2]	short [3]	0x61feae

```

matrixmul.cpp  matrixmul.h  matrixmul_test.cpp
101  matrixmul(in_mat_a, in_mat_b, hw_result);
102  #endif
103
104  // Print result matrix
105  cout << "{" << endl;
106  //cout << setw(6);

```

Q1

Estimated clock period: 8.34ns

Worst case latency: 106 clock cycles

Number of DSP48E used: 80

Number of FFs used: 61

Number of LUTs used: 68

What is your conclusion from the above data?

The current solution for the function matrixmul has a clk per of 8.34ns, with a latency of 106 clock cycles, using 80 DSP48E, 61 FFs, and 68 LUTs.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.34	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
106	106	107	107	none

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	1	0	45
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	23
Register	-	-	61	-
Total	0	1	61	68
Available	120	80	35200	17600
Utilization (%)	0	1	~0	~0

4-1-7. Review the synthesis report and explain what protocols have been used for port level as well as block level synthesis. Explain your answer thoroughly what ports have been added to the matrix multiplication block and why it makes for the synthesis tools to do so.

One protocol has been used for port level being the ap_memory protocol. For the block level synthesis, ap_ctrl_hs is being used.

The ap_ctrl_hs protocol adds the following ports: ap_clk, ap_rst, ap_start, ap_done, anp_idle, and ap_ready. This protocol has been implemented in order to provide a clk signal to the module as well as adding a way to start or restart the function. This protocol outputs signals of the current state of the module and will also have a signal signify when it is ready.

The ap_memory protocol adds address, chip enable, data, and write enable ports. For a and b address, chip enable, and data ports are added with only the data line being an input. The rest of the ports are outputs. For res it has a write enable port as an output and the rest are outputs being address, chip select, and data out.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	matrixmul	return value
ap_rst	in	1	ap_ctrl_hs	matrixmul	return value
ap_start	in	1	ap_ctrl_hs	matrixmul	return value
ap_done	out	1	ap_ctrl_hs	matrixmul	return value
ap_idle	out	1	ap_ctrl_hs	matrixmul	return value
ap_ready	out	1	ap_ctrl_hs	matrixmul	return value
a_address0	out	4	ap_memory	a	array
a_ce0	out	1	ap_memory	a	array
a_q0	in	8	ap_memory	a	array
b_address0	out	4	ap_memory	b	array
b_ce0	out	1	ap_memory	b	array
b_q0	in	8	ap_memory	b	array
res_address0	out	4	ap_memory	res	array
res_ce0	out	1	ap_memory	res	array
res_we0	out	1	ap_memory	res	array
res_d0	out	16	ap_memory	res	array

4-1-8. What is the loop iteration latency for rows and columns?

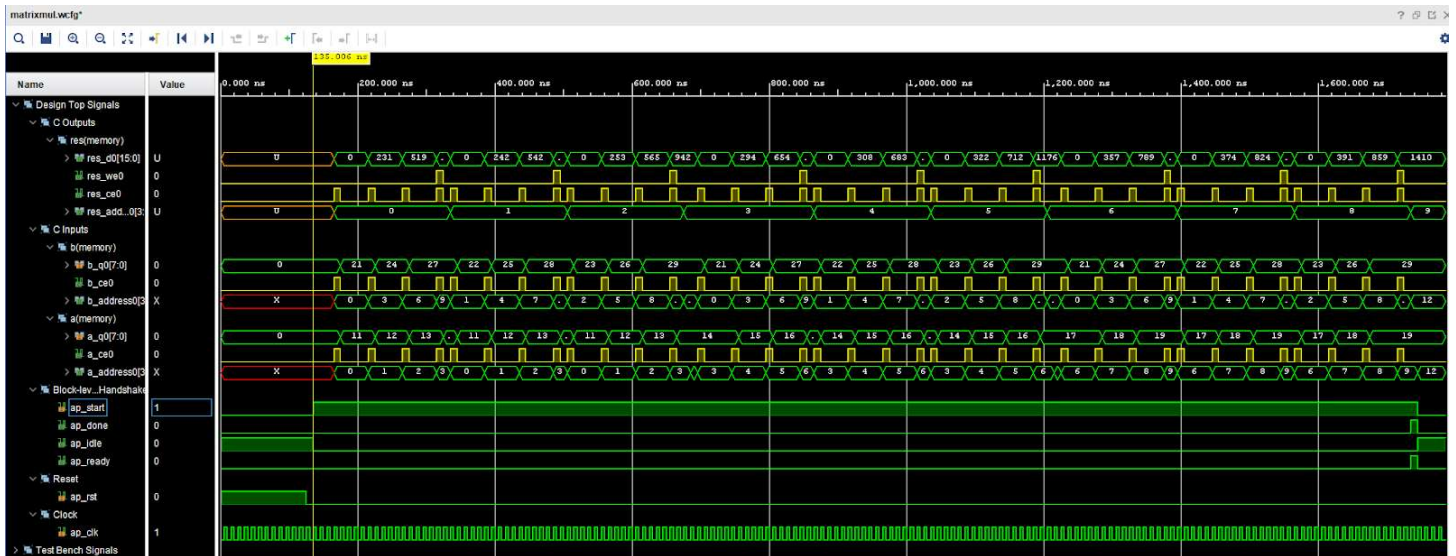
For the row loop the iteration latency is 35 clock cycles and the iteration latency for the col loop is 11 clock cycles.

Loop

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- Row	105	105	35	-	-	3	no
+ Col	33	33	11	-	-	3	no
++ Product	9	9	3	-	-	3	no

4-1-9. The synthesis report does not show the usage of BRAM but port interface uses addressing? Where has been memory been implemented?

This has been implemented for accessing memory contents outside of the module.



8-1-8. Create 3 solutions for the given design with the following specifications and synthesize the code. For each solution, assign a directive based on the guideline given below. Compare the results by analyzing the synthesis reports.

Solution 2: Assign PIPELINE directive with $II = 1$, $II = 2$, and $II = 3$ to the outer loop.

Solution 3: Assign LOOP_FLATTEN constraint to the outer loop.

Solution 4: Assign UNROLL constraint to the outer loop.

Synthesis(solution4) compare reports

Vivado HLS Report Comparison

All Compared Solutions

[solution2_II_1](#): xc7z010clg400-1
[solution2_II_2](#): xc7z010clg400-1
[solution2_II_3](#): xc7z010clg400-1
[solution3](#): xc7z010clg400-1
[solution4](#): xc7z010clg400-1

Performance Estimates

Timing (ns)

Clock		solution2_II_1	solution2_II_2	solution2_II_3	solution3	solution4
ap_clk	Target	10.00	10.00	10.00	10.00	10.00
	Estimated	6.38	6.38	6.38	8.34	8.34

Latency (clock cycles)

		solution2_II_1	solution2_II_2	solution2_II_3	solution3	solution4
Latency	min	19	19	19	106	102
	max	19	19	19	106	102
Interval	min	20	20	20	107	103
	max	20	20	20	107	103

Utilization Estimates

	solution2_II_1	solution2_II_2	solution2_II_3	solution3	solution4
BRAM_18K	0	0	0	0	0
DSP48E	9	9	9	1	3
FF	500	500	500	61	116
LUT	104	104	104	68	189

Export the report(.html) using the [Export Wizard](#)

For all iterations of solution 2 they yielded the same results. Vivado HLS was not able to synthesize the design because it was unable to schedule 'b'. Because of this all of solution2 was not valid. For solution3 its latency is 106, with an interval of 107, and with resource utilization of 1 DSP48E, 61 FFs, and 68 LUTs. For solution4 its latency is 102, with an interval of 103, and with resource utilization of 3 DSP48E, 116 FFs, and 189 LUTs. Solution4 is slightly faster than solution3 but it also is more expensive because it uses 2 more DSP48E, 55 more FFs, and 121 more LUT. Solution4 does not make a lot sense to use because it is 4 clock cycles faster and the extra resources it uses is way too much.