

# CEN 598: Reconfigurable Computing

## Lab 2: GEMM accelerator using High-Level Synthesis

Deepesh Sahoo: 1230089330

Debanjalee Roy: 1225660787

Spring 2024

### 1 Modify C++ code and write a test-bench

A testbench was developed for the C++ code, with the following features in mind.

1. Randomized test vectors.
2. Integration with Vitis (To prevent issues in later sections)
3. A GEMM library function to compare against. (uBLAS)

The testbench uses the rand function to generate numbers, and assign each element of the matrix to an integer between 0 and 255.

The results are then compared against the ublas generated data. The testbench runs for a total of 10 iterations, and provides a pass/fail status for each data.

Figure 1 below shows a snapshot of the test-bench output.

```
[dsahoo4@c020:~/Lab2/p1]$ ./matrix_mult.out
Pattern 0
Test Passed!
Pattern 1
Test Passed!
Pattern 2
Test Passed!
Pattern 3
Test Passed!
Pattern 4
Test Passed!
Pattern 5
Test Passed!
Pattern 6
Test Passed!
Pattern 7
Test Passed!
Pattern 8
Test Passed!
Pattern 9
Test Passed!
```

Figure 1: Testbench Output

## 2 Perform HLS and Co-Simulation

The testbench and C++ code are now implemented using VITIS, an HLS development environment for C++ codes. Previously, the float data type was used as the data type for our computations. In this section, we move from fp32 to int8 for inputs and int32 for outputs, specifically ap\_uint data types. . There are a few trade-offs to this conversion.

1. Range: fp32 or single floating point precision data has a wide range of data. unsigned int 8 has a lower range, and can handle a smaller range of values.
2. Performance: int data types are computationally better than float data types, however with a downside for precision.
3. Storage: int8 is smaller in size than the float data type.

C Synthesis is performed and the GEMM function is synthesised using HLS. The working of the synthesized netlist can be understood using the schedule viewer.

Figure 2 shows the schedule for a single multiplier. Notice that the multiplier, denoted by the "(\*)" is dependant on "zext\_ln232" and "zext\_ln10\_1". On further analysis, both the dependancies are arriving from a load and b load to perform computation.

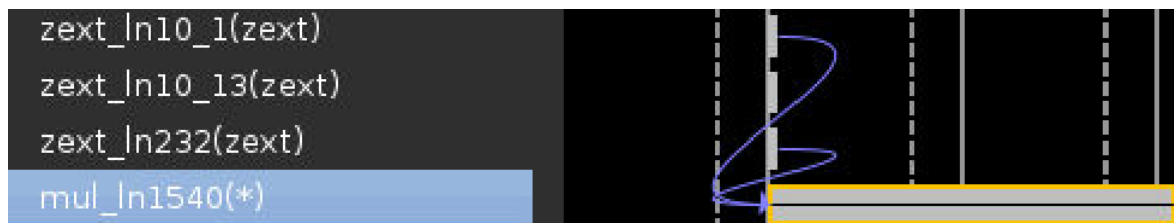


Figure 2: Multiplication schedule viewer

The output of the multiplier in the schedule viewer is going to an addition logic. "zext\_ln232" as shown in Figure 3, shows the movement of the previous multiplier, coming over to the adder. Similarly, the multiplier shown in the figure is another input to the adder. The output of the adder eventually moves to the prod\_addr\_write where the write operation takes place.

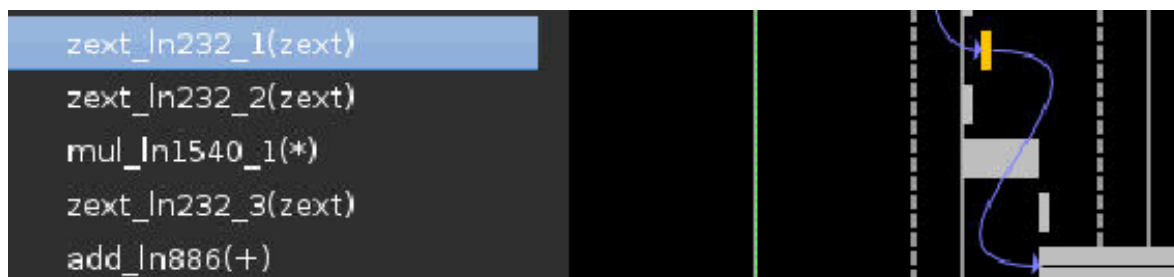


Figure 3: Addition schedule viewer

The synthesis report shows the estimated usage for the design. Figure 4 shows the synthesis report, we've provided a period of 10 ns, but the estimated period is 6.3 ns.

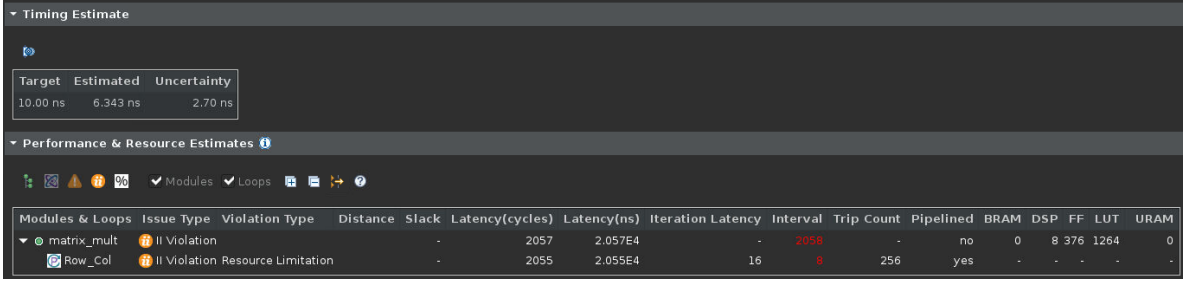


Figure 4: Synthesis Report

HLS generates a few files. They are :

1. matrix\_mult\_flow\_control\_loop\_pipe.v: This is the FSM for all the control signals of the design
2. matrix\_mult\_mac\_muladd.8ns.8ns.16ns.17.4.1.v: This is the mac design(multiply-accumulate) block for multiplication+addition.
3. matrix\_mult\_mul.8ns.8ns.16.1.1.v: This is the multiplication design block for multiplication
4. matrix\_mult.v: This is the top level design.

The next step in the process is to perform co-simulation. The tool generates a testbench called "autotb", which is used to perform simulation on the generated verilog file. Figure 5 shows the waveforms for the design. ap\_start is asserted when the inputs are ready, and ap\_done is asserted when the output is ready.

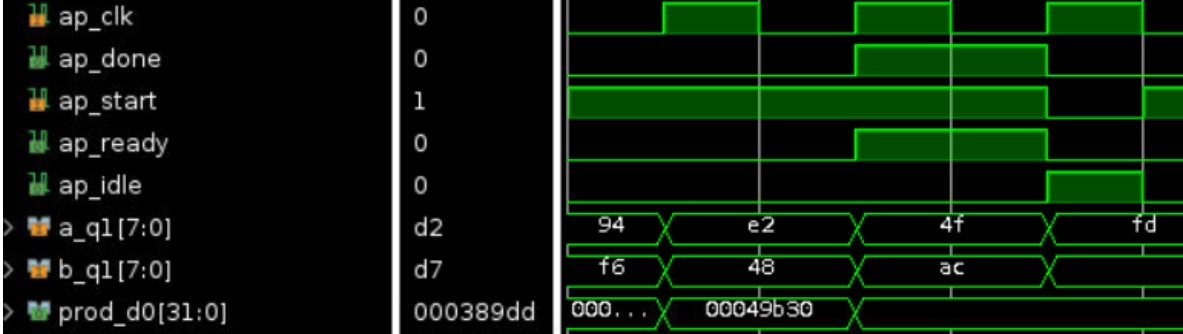


Figure 5: Cosimulation waveforms

Table 1 explains the differences between HLS and PnR implementations are as follows.

	HLS	PnR
Latency (cycles)	2057	2057
II (cycles)	2058	2058
DSP	8	16
FF	376	214
LUTs	1264	254
Period (ns)	6.343	6.798

Table 1: Comparison of HLS and PnR Data

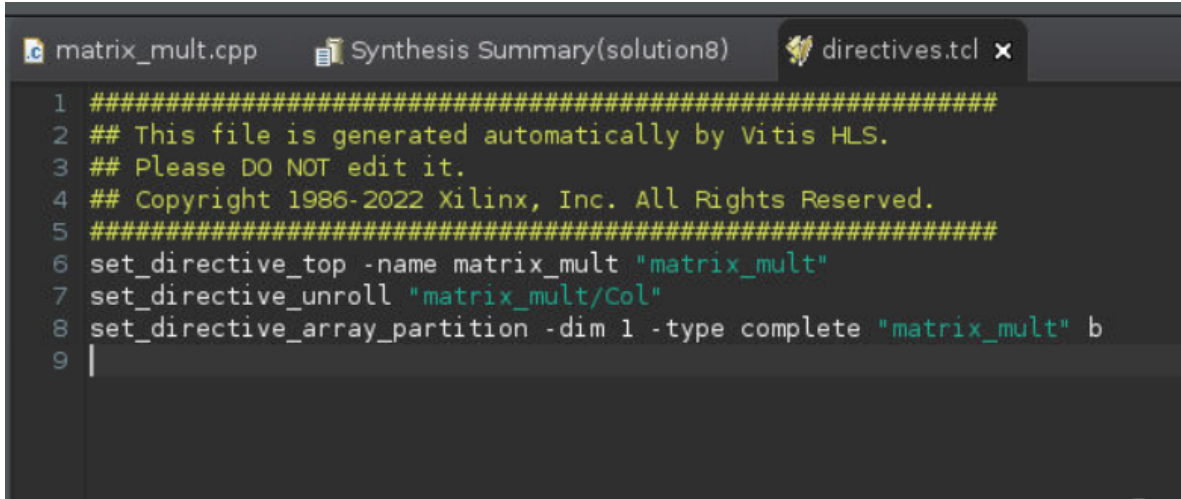
### 3 Design Space Exploration using HLS

We explored multiple architectural alternatives for the GEMM design with the synthesis directives offered by Vitis HLS for unrolling, pipelining, and array partitioning. The following best solutions from 21 solutions that we analysed are presented here.

### 3.1 Solution 1

In this solution, two directive settings are implemented, one with Loop unrolling and another with Array Partition.

- Unroll instructs the synthesis tool to unroll the loop labeled "Col" within the "matrix\_mult" function.
- Loop unrolling is done in the first inner loop to improve the performance by executing multiple iterations of the loop in parallel.
- Array Partition is done in the array "b" along the first dimension into complete partitions to optimize memory access and then we proceeded with the next combinations.



```
matrix_mult.cpp  Synthesis Summary(solution8)  directives.tcl x
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_unroll "matrix_mult/Col"
8 set_directive_array_partition -dim 1 -type complete "matrix_mult" b
9 |
```

Figure 6: Solution 1 Synthesis Directive Setting

The following observations are made with the above settings:-

- Synthesis Directives: LOOP UNROLL, ARRAY PARTITION
- Improved throughput (reduced II) and reduced latency as compared to when no pragmas are used.
- Resource usage is moderate, with low DSP and BRAM utilization as compared to the case when no pragmas are used since it is optimized now.

### 3.2 Solution 2

In solution 2, the same synthesis directives are kept and loop unroll is applied in the inner loop by a factor of 2 and array partition is applied to variables a and b keeping dimension 1.

- Synthesis Directives: LOOP UNROLL, ARRAY PARTITION
- Unroll instructs the synthesis tool to unroll the loop "Product" within the "matrix\_mult" function.
- Loop unrolling is done in the inner loop to understand how it impacts the metrics.
- Array Partition is done on both array "a" and "b" along the first dimension into complete partitions.

```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_array_partition -type complete -dim 1 "matrix_mult" b
8 set_directive_array_partition -type complete -dim 1 "matrix_mult" a
9 set_directive_unroll -factor 2 "matrix_mult/Product"
10 |
```

Figure 7: Solution 2 Synthesis Directive Setting

The following observations are made with the above settings:-

- Loop unrolling in the innermost loop reduces clock frequency and slightly improves throughput slightly than solution 1.
- Array Partition is done on both array "a" and "b" along the first dimension into complete partitions to optimize memory access because the throughput improves significantly when applied to both variables instead of only one.
- Analysis is similar to Solution 1 but with lower resource usage(Low LUTs and FFs) and slightly better latency and throughput metrics.

### 3.3 Solution 3

In solution 3, the PIPELINE directive is applied to the first inner loop "Col" combined with ARRAY PARTITION on variable "b" along the first dimension.

- Synthesis Directives: PIPELINE, ARRAY PARTITION
- Array Partition is done on first one variable and combined with Pipeline to understand the impact on metrics.
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the first inner loop "Col" to reduce latency due to pipelined stages.

```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Col"
8 set_directive_array_partition -dim 1 -type complete "matrix_mult" b
9 |
```

Figure 8: Solution 3 Synthesis Directive Setting

The following observations are made with the above settings:-

- Resource Usage: Moderate LUTs and FFs, moderate DSPs, no BRAM
- Throughput: Significantly less in this combination than solution 1,2.
- Latency: More due to array partition on one variable only and also additional pipelined stages due to  $II=2$ .
- Operating Clock Frequency: Maintained or slightly improved due to improved throughput.
- Analysis: Pipelining and array partitioning resulted in increased throughput and latency.

### 3.4 Solution 4

In solution 4, directives used are similar to solution 3 with increasing array partition to both variables "a" and "b".

- Synthesis Directives: PIPELINE, ARRAY PARTITION
- Array Partition is done on both variables and combined with Pipeline to understand the impact on metrics.
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the first inner loop "Col" to reduce latency due to pipelined stages.

```

1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Col"
8 set_directive_array_partition -type complete -dim 1 "matrix_mult" b
9 set_directive_array_partition -dim 1 -type complete "matrix_mult" a
10

```

Figure 9: Solution 4 Synthesis Directive Setting

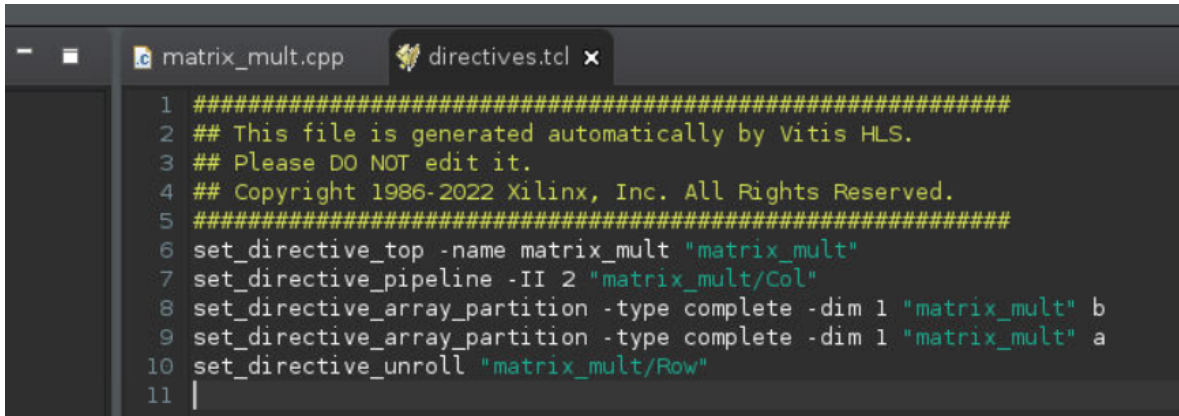
The following observations are made with the above settings:-

- Resource Usage: High LUTs and FFs, moderate DSPs, no BRAM increased as due to array partitioning, it has to create separate memory partitions for two variables now.
- Throughput: Improved significantly due to increased optimization than solution 3.
- Latency: less than solution 3 because of increased optimization in array partitioning.
- Operating Clock Frequency: Maintained or slightly more due to more resource usage.
- Analysis: Similar to Solution 3 but with higher resource usage and significantly better throughput metrics.

### 3.5 Solution 5

In solution 5, the combination is such that PIPELINE is applied to "Col" loop, with loop unroll in outer loop and array partition on both the variables "a" and "b"

- Synthesis Directives: PIPELINE, ARRAY PARTITION, LOOP UNROLL
- Array Partition is done on both variables and combined with Pipeline to understand the impact on metrics.
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the first inner loop "Col" to reduce latency due to pipelined stages.
- Loop unroll is applied to Outer loop to analyse the impact.



```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Col"
8 set_directive_array_partition -type complete -dim 1 "matrix_mult" b
9 set_directive_array_partition -type complete -dim 1 "matrix_mult" a
10 set_directive_unroll "matrix_mult/Row"
11 |
```

Figure 10: Solution 5 Synthesis Directive Setting

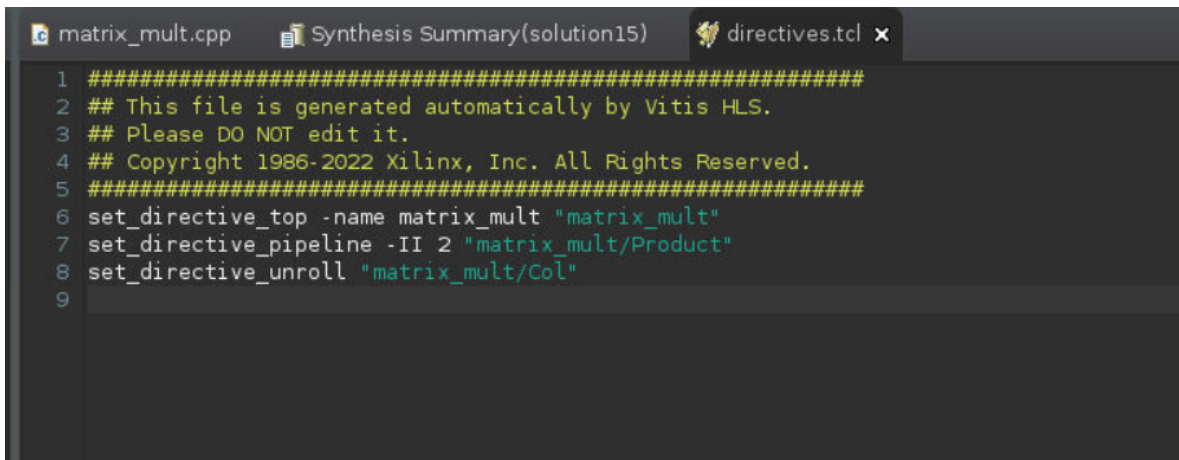
The following observations are made with the above settings:-

- Resource Usage: Very High LUTs and FFs, moderate DSPs, no BRAM increased as due to array partitioning and loop unrolling, it has to create separate memory partitions for two variables now, and overhead increased due to loop unroll.
- Throughput: Improved similar to solution 4 due to increased optimization than solution.
- Latency: Similar to solution 4.
- Operating Clock Frequency: Maintained or slightly more due to more resource usage.
- Analysis: Combination of pipelining, array partitioning, and loop unrolling results in improved throughput with high resource usage and moderate latency.

### 3.6 Solution 6

- Synthesis Directives: LOOP UNROLL, PIPELINE
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the innermost loop "prod" to reduce latency due to pipelined stages.
- Loop unroll is applied to the first inner loop "Col" to analyze the impact.





```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Product"
8 set_directive_unroll "matrix_mult/Col"
9
```

Figure 11: Solution 6 Synthesis Directive Setting

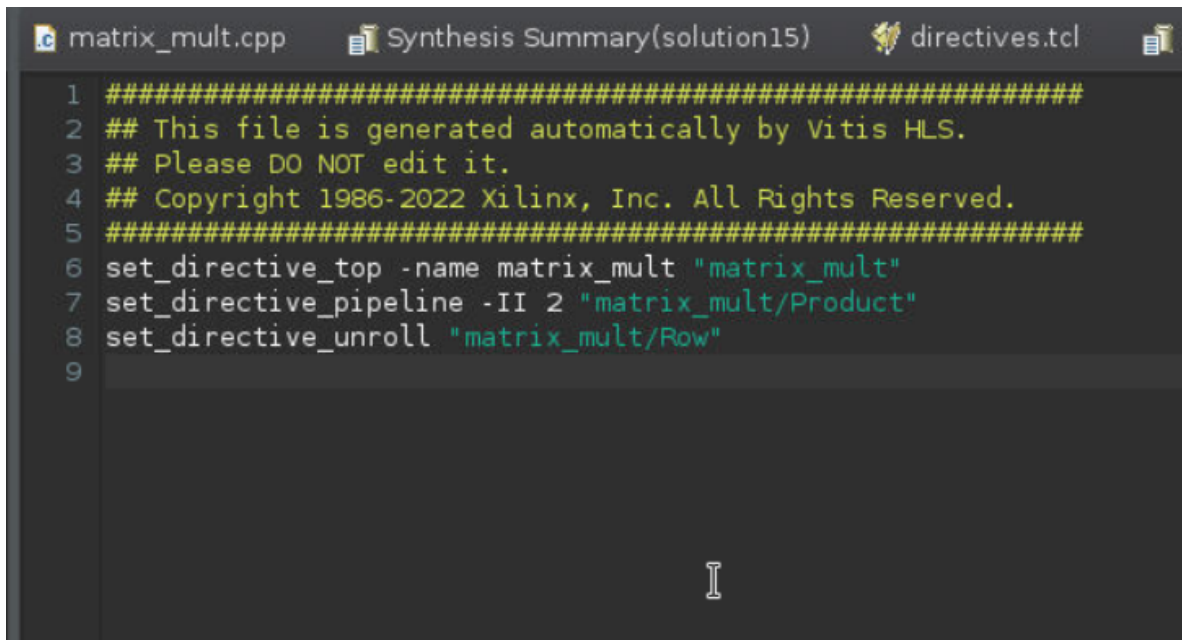
The following observations are made with the above settings:-

- Resource Usage: lesser LUTs and FFs, high DSPs, no BRAM compared to when array partition is used.
- Throughput: A little lesser due to reduced optimization than solution 5.
- Latency: Higher latency since array partitioning is not used.
- Operating Clock Frequency: Maintained or slightly less due to less resource usage.
- Analysis: Loop unrolling and pipelining applied, resulting in significantly improved throughput but with high latency.

### 3.7 Solution 7

- Synthesis Directives: LOOP UNROLL, PIPELINE
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the innermost loop "prod" to reduce latency due to pipelined stages.
- Loop unroll is applied to the outer loop "Row" to analyze the impact.



A screenshot of a code editor window with three tabs: 'matrix\_mult.cpp', 'Synthesis Summary(solution15)', and 'directives.tcl'. The 'directives.tcl' tab is active, showing a Tcl script. The script starts with a header block of comments (lines 1-5) indicating it was generated by Vitis HLS. Lines 6-8 contain three directives: 'set\_directive\_top -name matrix\_mult "matrix\_mult"', 'set\_directive\_pipeline -II 2 "matrix\_mult/Product"', and 'set\_directive\_unroll "matrix\_mult/Row"'. Line 9 is empty. A cursor is visible at the end of line 9.

```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Product"
8 set_directive_unroll "matrix_mult/Row"
9
```

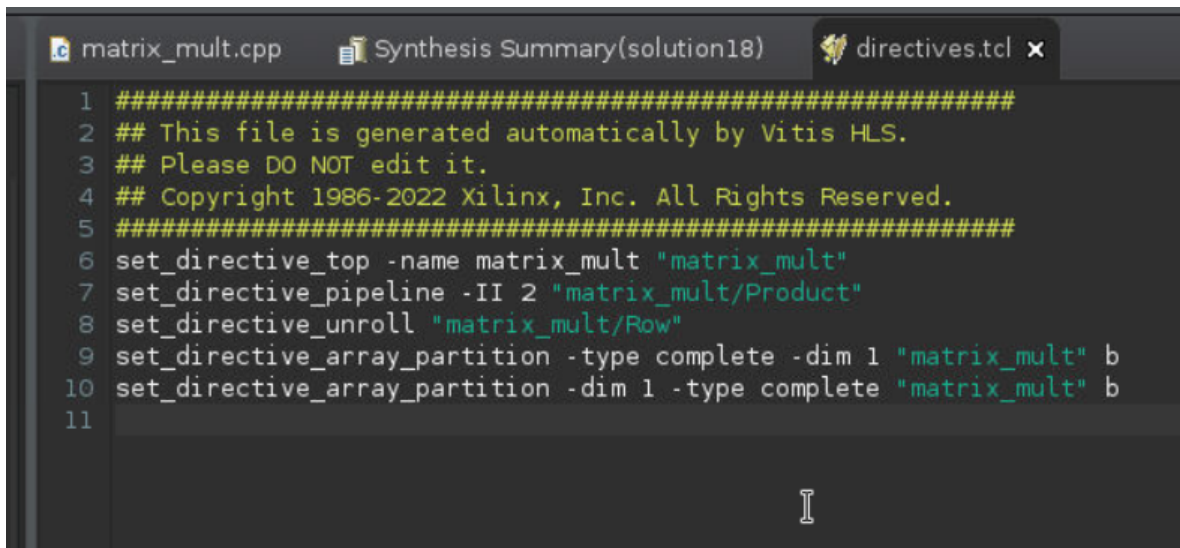
Figure 12: Solution 7 Synthesis Directive Setting

The following observations are made with the above settings:-

- Resource Usage: similar to solution 6, lesser LUTs and FFs, high DSPs, no BRAM compared to when array partition is used.
- Throughput: Similar to solution 6 as not much change w.r.t optimization.
- Latency: Higher latency since array partitioning is not used.
- Operating Clock Frequency: lesser due to optimization to the innermost loop.
- Analysis: Similar to Solution 6 but with slightly different latency and throughput metrics.

### 3.8 Solution 8

- Synthesis Directives: PIPELINE, ARRAY PARTITION, LOOP UNROLL
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the innermost loop "Product" to reduce latency due to pipelined stages.
- Loop unroll is applied to the outer loop "Row" to analyze the impact.
- Array Partition is done on both array "a" and "b" along the first dimension into complete partitions to optimize memory access because the throughput improves significantly when applied to both variables instead of only one.

A screenshot of a code editor window with three tabs: 'matrix\_mult.cpp', 'Synthesis Summary(solution18)', and 'directives.tcl'. The 'directives.tcl' tab is active, showing a Tcl script. The script starts with a header block of 5 lines, followed by 6 lines of directives. Line 6 sets the directive top to 'matrix\_mult'. Line 7 sets the pipeline initiation interval to 2 for the 'Product' loop. Line 8 sets loop unroll to 1 for the 'Row' loop. Line 9 sets array partitioning to complete for dimension 1 of variable 'b'. Line 10 sets array partitioning to complete for dimension 1 of variable 'b'. Line 11 is empty. A cursor is visible at the end of line 11.

```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Product"
8 set_directive_unroll "matrix_mult/Row"
9 set_directive_array_partition -type complete -dim 1 "matrix_mult" b
10 set_directive_array_partition -dim 1 -type complete "matrix_mult" b
11
```

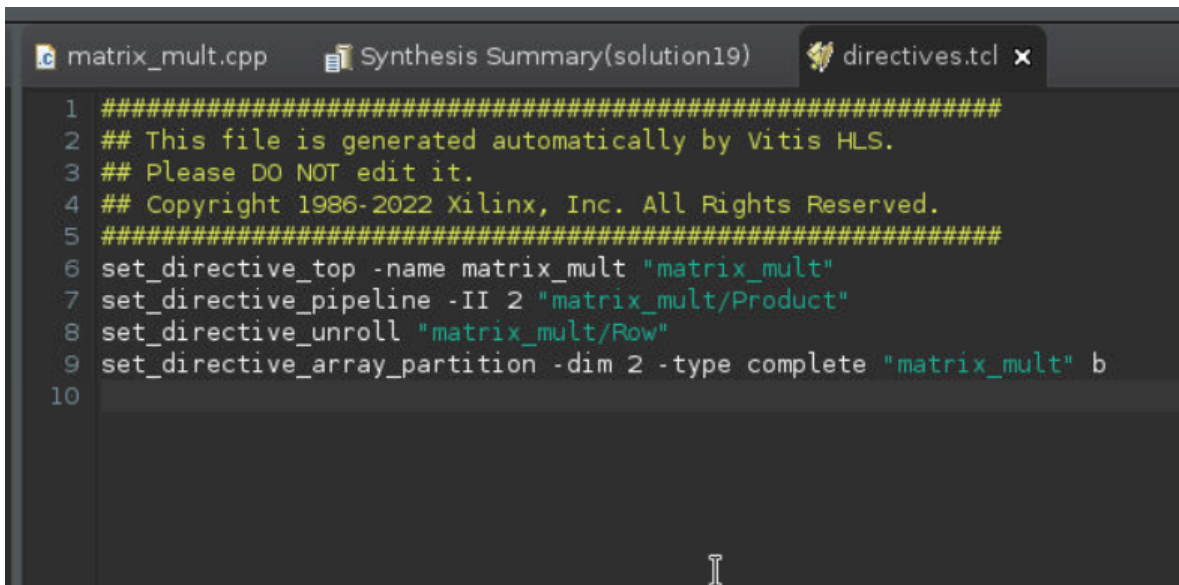
Figure 13: Solution 8 Synthesis Directive Setting

The following observations are made with the above settings:-

- Resource Usage: more than to solution 7, more LUTs and FFs, similar DSPs, no BRAM since array partition is used.
- Throughput: Similar to solution 7 as not much change w.r.t optimization.
- Latency: Higher latency since pipelining is used.
- Operating Clock Frequency: lesser due to optimization with array partitioning.
- Analysis: Combined optimization techniques result in improved throughput with increased resource usage and latency.

### 3.9 Solution 9

- Synthesis Directives: PIPELINE, ARRAY PARTITION, LOOP UNROLL
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the innermost loop "Product" to reduce latency due to pipelined stages.
- Loop unroll is applied to the outer loop "Row" to analyze the impact.
- Array Partition is done on variable "b" along the second dimension into complete partitions to optimize memory access to understand the impact.

A screenshot of a code editor window with three tabs: 'matrix\_mult.cpp', 'Synthesis Summary(solution19)', and 'directives.tcl'. The 'directives.tcl' tab is active, showing a Tcl script. The script starts with a header block containing copyright information and a warning not to edit the file. It then contains four directives: 'set\_directive\_top' to name the matrix\_mult block, 'set\_directive\_pipeline' to set a pipeline initiation interval of 2 for the innermost loop 'Product', 'set\_directive\_unroll' to unroll the outer loop 'Row', and 'set\_directive\_array\_partition' to partition the array 'b' along its second dimension into complete partitions. The cursor is at the end of the fourth directive.

```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Product"
8 set_directive_unroll "matrix_mult/Row"
9 set_directive_array_partition -dim 2 -type complete "matrix_mult" b
10
```

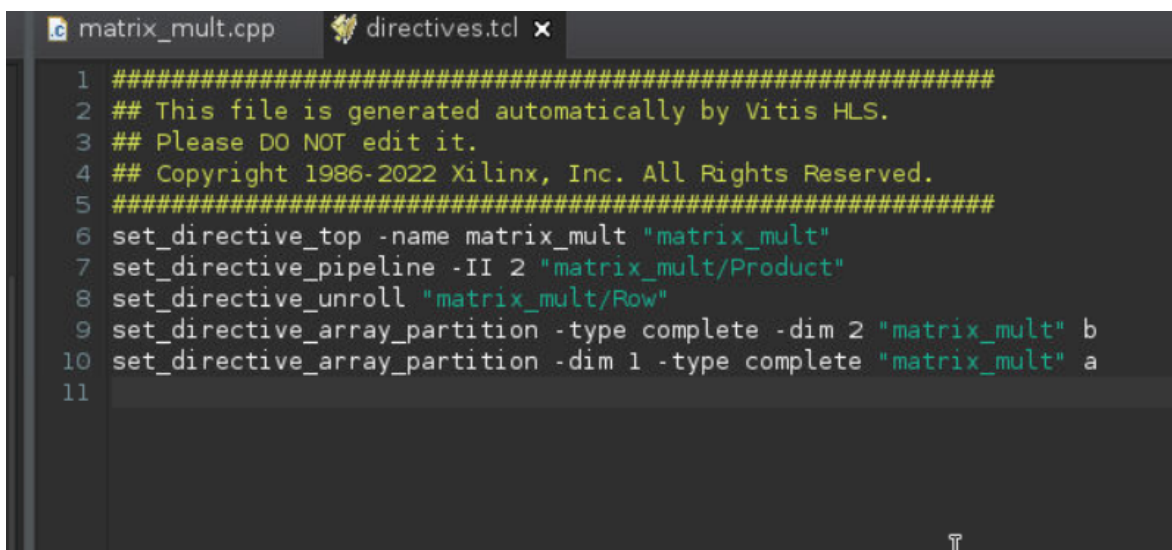
Figure 14: Solution 9 Synthesis Directive Setting

The following observations are made with the above settings:-

- Resource Usage: more than to solution 8, more LUTs and lesser FFs, similar DSPs, no BRAM since array partition is used.
- Throughput: Similar to solution 8 as not much change w.r.t optimization.
- Latency: Similar latency as compared to solution 8.
- Operating Clock Frequency: similar to solution 8 due to similar optimization
- Analysis: Similar to Solution 8 with slightly different latency and throughput metrics.

### 3.10 Solution 10

- Synthesis Directives: PIPELINE, ARRAY PARTITION, LOOP UNROLL
- Pipeline with a pipeline initiation interval (II) of 2 clock cycles is applied to the innermost loop "Product" to reduce latency due to pipelined stages.
- Loop unroll is applied to the outer loop "Row" to analyze the impact.
- Array Partition is done on variable "b" along the second dimension into complete partitions to optimize memory access to understand the impact.
- Array Partition is done on the variable "a" along the first dimension into complete partitions to optimize memory access to understand the impact.



```
1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_pipeline -II 2 "matrix_mult/Product"
8 set_directive_unroll "matrix_mult/Row"
9 set_directive_array_partition -type complete -dim 2 "matrix_mult" b
10 set_directive_array_partition -dim 1 -type complete "matrix_mult" a
11
```

Figure 15: Solution 10 Synthesis Directive Setting

The following observations are made with the above settings:-

- Resource Usage: Similar to solution 9, lesser LUTs and same FFs, similar DSPs, no BRAM since array partition is used.
- Throughput: Similar to solution 9 as not much change w.r.t optimization.
- Latency: Similar latency as compared to solution 8.
- Operating Clock Frequency: similar to solution 9 due to similar optimization
- Analysis: Similar to Solutions 8 and 9 but with a different combination of array partitioning.

### 3.11 Solution 11

- Synthesis Directives: LOOP UNROLL, ARRAY PARTITION
- Loop unroll is applied to the outer loop "Row" with a factor of 4 to analyze the impact.
- Array Partition is done on variable "b" along the first dimension into complete partitions(cyclic does worse) to optimize memory access to understand the impact.
- Array Partition is done on the variable "a" along the first dimension into complete partitions to optimize memory access to understand the impact.

```

1 #####
2 ## This file is generated automatically by Vitis HLS.
3 ## Please DO NOT edit it.
4 ## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_top -name matrix_mult "matrix_mult"
7 set_directive_array_partition -type complete -dim 1 "matrix_mult" b
8 set_directive_array_partition -dim 1 -type complete "matrix_mult" a
9 set_directive_unroll -factor 4 "matrix_mult/Row"
10

```

Figure 16: Solution 11 Synthesis Directive Setting

The following observations are made with the above settings:-

- Resource Usage: High LUTs and FFs, high DSPs, no BRAM due to
- Throughput: Throughput is increased due to an increase in Loop unroll factor i.e., four iterations of the loop are executed in parallel in each clock cycle.
- Latency: Similar latency as compared to solution 8.
- Operating Clock Frequency: similar to solution 9 due to similar optimization
- Analysis: Analysis: Loop unrolling and array partitioning applied, resulting in improved throughput with high resource usage and moderate latency.

### 3.12 Results

The following table presents the resources, latency, operating clock frequency, and area information related to the different solutions that we discussed above:

Solution	LUTs Used	FFs Used	DSPs Used	BRAM	Latency (cycles)	Latency (ns)	Projected Latency (ns)	Interval	Clock Frequency (MHz)	Clock (ns)	Area
Solution 1	10251	5599	128	0	149	1490	945.107	150	157.65	6.343	164908
Solution 2	600	363	1	0	143	1430	896.61	144	159.49	6.27	7126
Solution 3	1076	419	8	0	2057	20570	13047.551	2058	157.65	6.343	14798
Solution 4	4552	2519	8	0	527	5270	3421.284	528	154.04	6.492	53758
Solution 5	15115	7731	128	0	647	6470	4103.921	648	157.65	6.343	217812
Solution 6	2929	587	16	0	9745	97450	63664.085	9746	153.07	6.533	36864
Solution 7	3155	817	16	0	10000	100000	51690	10001	193.46	5.169	39584
Solution 8	5401	1025	16	0	10256	102560	53013.264	10257	193.46	5.169	62460
Solution 9	6625	817	16	0	10000	100000	54360	10001	183.96	5.436	74284
Solution 10	6381	817	16	0	10000	100000	54360	10001	183.96	5.436	71844
Solution 11	9152	3110	32	0	393	3930	2661.396	394	147.67	6.772	110540

Table 2: Directives Data Analysis

Figure 17 shows the Pareto front for different pragmas. These calculations were done with a period of 10ns, as provided while running Vitis.

Solution 9 offers the best solution in our case, with low area and latency. Solution 11 and 12 are close, with either being used for either area/latency trade-offs but worse than solution 9.

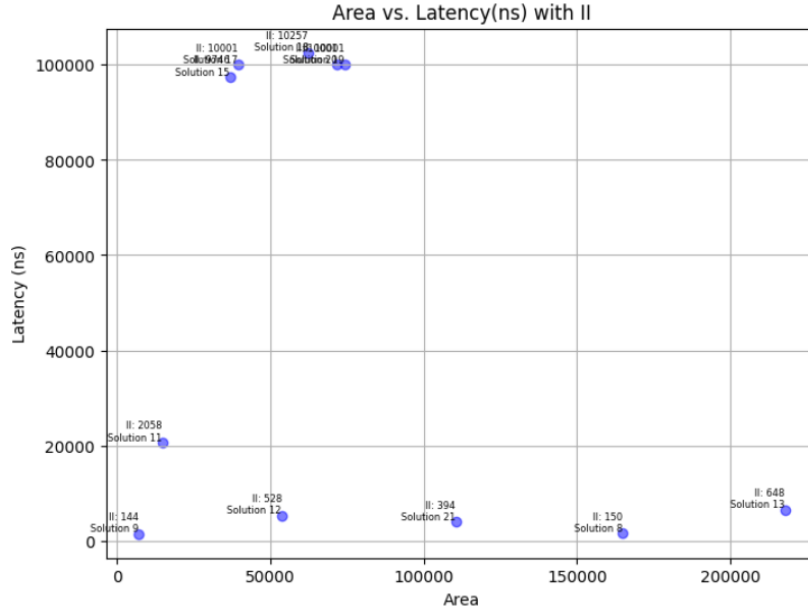


Figure 17: Area Vs Reported Latency(ns) plot

It was imperative to replot the graph with the estimated time periods to get an accurate latency prediction. Figure 18 shows the Pareto front for different pragmas with the new estimated latency numbers. The general trends of low latency and area can be observed with solutions 9, 11 ad 12.

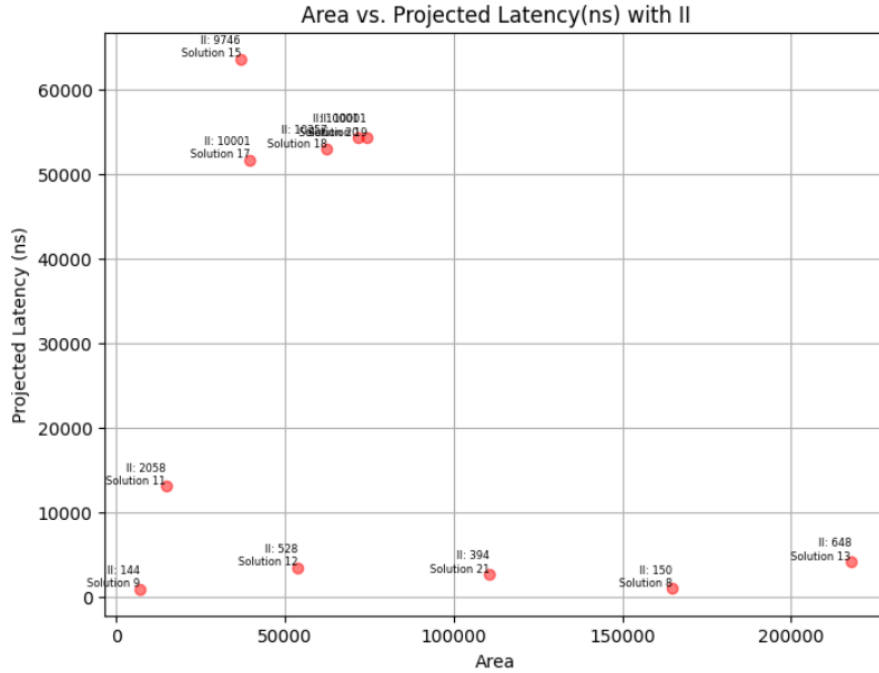


Figure 18: Area Vs Projected Latency(ns) plot

However, we notice a slight difference in the solutions in the top-left corner of the design. Hence, we plotted the graphs of both over each other for a detailed analysis. We observe a 2x reduction in latency for most designs with low area and high latency, and very few difference in low latency designs. This is expected as the latency is directly proportional to clock frequency.

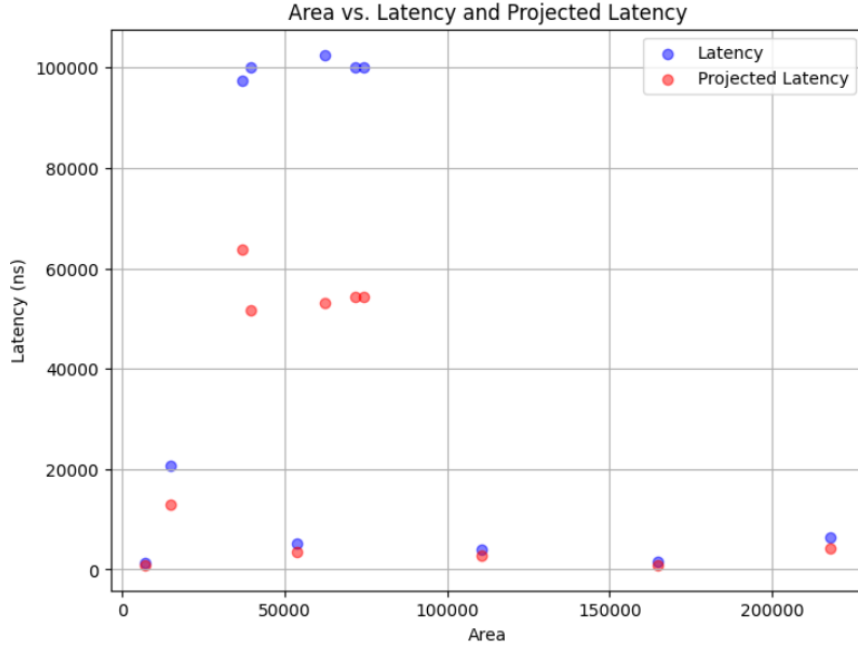


Figure 19: Area Vs Projected Latency(ns) and Reported Latency(ns) plot

Conclusion: From the graphs, we observe that our solution corresponding to  $\Pi=144$  in the bottom left corner is the best solution with lower latency and area.

## 4 Integrating the GEMM accelerator with an on-chip CPU

To integrate the GEMM accelerator on the Pynq board, we did the following:

1. Add pragmas for the different IPs that we use alongside the matrix multiplier. For example: AXI-lite interface and BRAM interface.
2. Export IP from Vitis and import to Vivado
3. Connect the design and generate block diagram
4. Generate wrapper and bitstream.
5. Read bitstream into the board, and implement the driver.

The following pragmas were added to the design to allow AXI-LITE and BRAM interfaces.

```
#pragma HLS INTERFACE s_axilite port=return bundle=control
#pragma HLS INTERFACE mode=bram port=a storage_type=ram_1p
#pragma HLS INTERFACE mode=bram port=b storage_type=ram_1p
#pragma HLS INTERFACE mode=bram port=prod storage_type=ram_1p
```

Figure 20 shows the pragmas implemented in the GEMM code.



Summary						
Clock	Target	Estimated	Uncertainty			
ap_clk	10.00 ns	6.343 ns	2.70 ns			

Latency						
Summary						
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
149	149	1.490 us	1.490 us	150	150	no

Figure 20: Pragmas for IP integration

Two solutions were implemented. In solution 1, the IPs were not used, and a bare-metal python code was written to perform the GEMM multiplication. In solution 2, we wrote and read to the BRAM IPs and performed computation using the IP function. In each case, 10 random patterns were generated, and the worst time was taken for the comparison. The table below shows the delay for the computations.

Solution	Worst Case Time (seconds)
Software	0.058
Software + Hardware	0.041

Table 3: Average Time for Two Runs

The hardware + software solution works better in our case. One reason could be that the hardware+software solution was an optimized design, and hence ran better. However, the hardware+software solution could be improved. Some possible bottlenecks in the non-optimized code is the requirement to poll the "AP\_IDLE" register to figure out when the operation is complete. Another bottleneck could be the dependancy on the writes and reads into a single operation.

To fix these, here are some solutions:

1. Seperate read and write into two processes, allow pipelining to perform parallel computation.
2. Move from busy waiting to sleep-waiting. Sleep waiting allows a process to be awakened when a given condition is met, rather than keeping the compute hostage with busy waiting.
3. Perform a parallel multiplier, one entire row x column could be performed at a single iteration, but with an overhead on area.

We performed the HLS implementation with the driver for our solution