ELEC522 – Fall 2022

Project 3: Using Vitis HLS to Implement Matrix Multiplication

By

Hsuan-You (Shaun) Lin

Rice University, Houston, TX

Oct. 20, 2022

Supervised by

Dr. Joseph R. Cavallaro

Contents

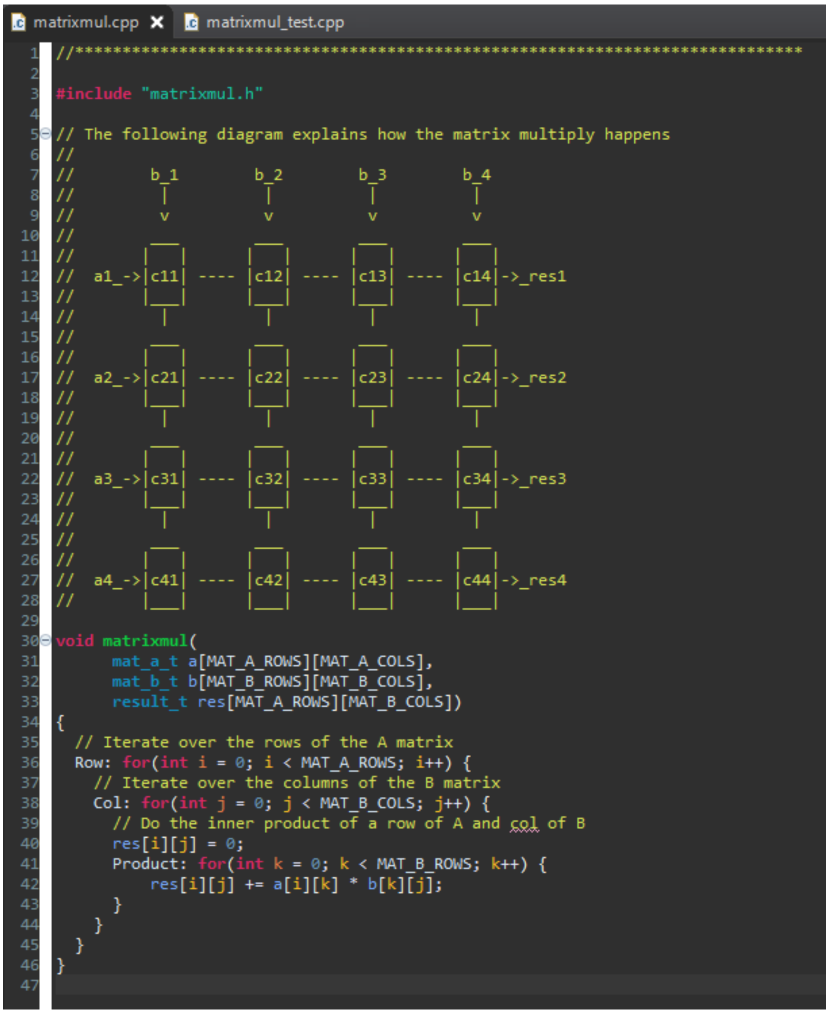
List of Figures

Use the Vitis HLS tool to design a matrix multiplication system:

1. C++ code for 4x4 matrix multiplication:

Here I modify the Lab 1 C++ code to adapt it to my 4x4 matrix multiplication, just use the three for loop function to calculate the result matrix, I also write a diagram in the code to visualize the process.

(a) Screen capture of 4x4 matrixmul c++ code.




```
//******************************************************************************

#include "matrixmul.h"

// The following diagram explains how the matrix multiply happens
//
//          b_1        b_2        b_3        b_4
//           |          |          |          |
//           v          v          v          v
//          ___        ___        ___        ___
//         |   |      |   |      |   |      |   |
//  a1_->|c11| ---- |c12| ---- |c13| ---- |c14|->_res1
//         |___|      |___|      |___|      |___|
//           |          |          |          |
//          ___        ___        ___        ___
//         |   |      |   |      |   |      |   |
//  a2_->|c21| ---- |c22| ---- |c23| ---- |c24|->_res2
//         |___|      |___|      |___|      |___|
//           |          |          |          |
//          ___        ___        ___        ___
//         |   |      |   |      |   |      |   |
//  a3_->|c31| ---- |c32| ---- |c33| ---- |c34|->_res3
//         |___|      |___|      |___|      |___|
//           |          |          |          |
//          ___        ___        ___        ___
//         |   |      |   |      |   |      |   |
//  a4_->|c41| ---- |c42| ---- |c43| ---- |c44|->_res4
//         |___|      |___|      |___|      |___|

void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS])
{
  // Iterate over the rows of the A matrix
  Row: for(int i = 0; i < MAT_A_ROWS; i++) {
    // Iterate over the columns of the B matrix
    Col: for(int j = 0; j < MAT_B_COLS; j++) {
      // Do the inner product of a row of A and col of B
      res[i][j] = 0;
      Product: for(int k = 0; k < MAT_B_ROWS; k++) {
        res[i][j] += a[i][k] * b[k][j];
      }
    }
  }
}
```

Figure 1.

(b) Screen capture of original Directive. (No optimization for Directive)

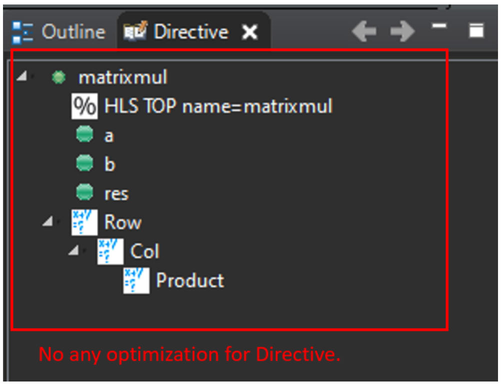


Figure 2.

(c) Screen capture of 4x4 matrixmul testbench c++ code.

matrixmul.cpp   matrixmul_test.cpp

```cpp
//*****************************************************************************
#include <iostream>
#include "matrixmul.h"

using namespace std;

int main(int argc, char **argv)
{
   mat_a_t in_mat_a[4][4] = {
      {11, 12, 13, 14},
      {14, 15, 16, 17},
      {17, 18, 19, 20},
      {20, 21, 22, 23}
   };
   mat_b_t in_mat_b[4][4] = {
      {21, 22, 23, 24},
      {24, 25, 26, 27},
      {27, 28, 29, 30},
      {30, 31, 32, 33}
   };
   result_t hw_result[4][4], sw_result[4][4];
   int err_cnt = 0;

   // Generate the expected result
   // Iterate over the rows of the A matrix
   for(int i = 0; i < MAT_A_ROWS; i++) {
      for(int j = 0; j < MAT_B_COLS; j++) {
          sw_result[i][j] = in_mat_a[i][j] * in_mat_b[i][j];
         // Iterate over the columns of the B matrix
         sw_result[i][j] = 0;
         // Do the inner product of a row of A and col of B
         for(int k = 0; k < MAT_B_ROWS; k++) {
             sw_result[i][j] += in_mat_a[i][k] * in_mat_b[k][j];
         }
      }
   }

#ifdef HW_COSIM
   // Run the AutoESL matrix multiply block
   matrixmul(in_mat_a, in_mat_b, hw_result);
#endif

   // Print result matrix
   cout << "{" << endl;
   //cout << setw(6);
   for (int i = 0; i < MAT_A_ROWS; i++) {
      cout << "{";
      for (int j = 0; j < MAT_B_COLS; j++) {
#ifdef HW_COSIM
         cout << hw_result[i][j];
         // Check HW result against SW
         if (hw_result[i][j] != sw_result[i][j]) {
            err_cnt++;
            cout << "*";
         }
#else
         cout << sw_result[i][j];
#endif
         if (j == MAT_B_COLS - 1)
            cout << "}" << endl;
         else
            cout << ",";
      }
   }
   cout << "}" << endl;

#ifdef HW_COSIM
   if (err_cnt)
      cout << "ERROR: " << err_cnt << " mismatches detected!" << endl;
   else
      cout << "Test passed." << endl;
#endif
   return err_cnt;
}
```

Figure 3.

(d) Screen capture of 4x4 matrixmul .h file.

C matrixmul.h ×

F: > VLSI > project3_v1 > C matrixmul.h

```c
//*****************************************************************************
#ifndef __MATRIXMUL_H__
#define __MATRIXMUL_H__

#include <cmath>
using namespace std;

// Uncomment this line to compare TB vs HW C-model and/or RTL
//#define HW_COSIM

#define MAT_A_ROWS 4
#define MAT_A_COLS 4
#define MAT_B_ROWS 4
#define MAT_B_COLS 4

typedef short mat_a_t;
typedef short mat_b_t;
typedef short result_t;

// Prototype of top level function for C-synthesis
void matrixmul(
      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],
      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],
      result_t res[MAT_A_ROWS][MAT_B_COLS]);

#endif // __MATRIXMUL_H__ not defined
```

Here I defined 4 number for 4x4matrix

Using short type for input a, b matrix

Figure 4.

(e) Screen capture of C simulation result.

matrixmul.cpp | matrixmul_test.cpp | matrixmul_csim.log ×

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../matrixmul_test.cpp in debug mode
   Compiling ../../../../matrixmul.cpp in debug mode
   Generating csim.exe
{
{1290,1340,1390,1440}
{1596,1658,1720,1782}
{1902,1976,2050,2124}
{2208,2294,2380,2466}
}
Test passed.
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

res matrix: <= a matrix: {11, 12, 13, 14, 14, 15, 16, 17, 17, 18, 19, 20, 20, 21, 22, 23} X b matrix: {11, 12, 13, 14, 14, 15, 16, 17, 17, 18, 19, 20, 20, 21, 22, 23}

Figure 5.

2. Optimization C++ code for 4x4 matrix multiplication:

First, I've tried many optimization directives to configure the synthesis results, and PIPELINE is the most common method for the optimization process in this case.

As professor mentioned in the lecture, pipelining the inner-most loop gives the smallest hardware with generally acceptable throughput, pipelining the upper-levels of the hierarchy unrolls all sub-loops and can create many more operations to schedule, but typically gives the highest performance design in terms of throughput and latency.

In conclusion, I choose pipelining the sub-inner loop, to get a better balance between system throughput and hardware resource costs, my constraint configurations as shown in *Figure 6*.

(a) Screen capture of my first optimization method. (PIPELINE II = 2)

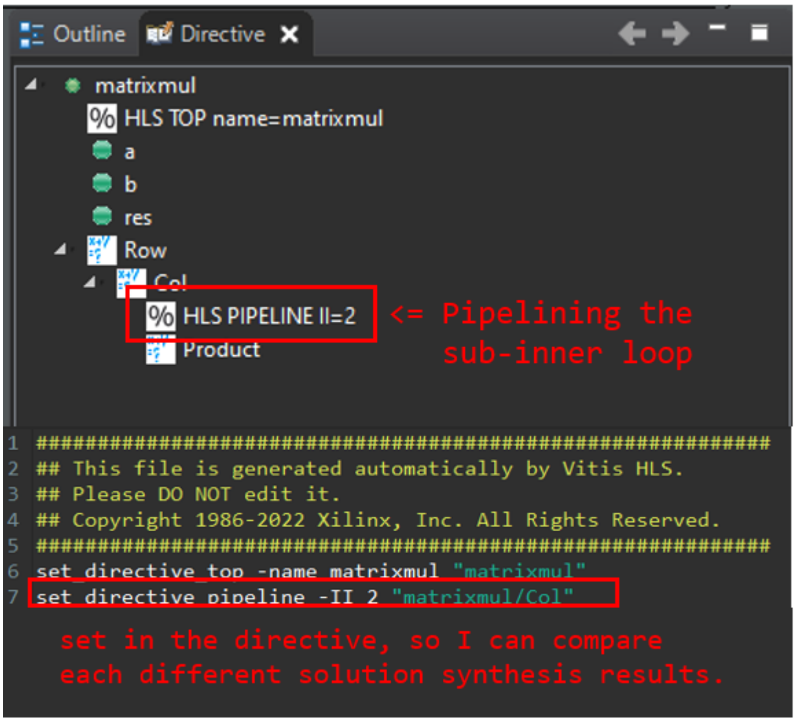
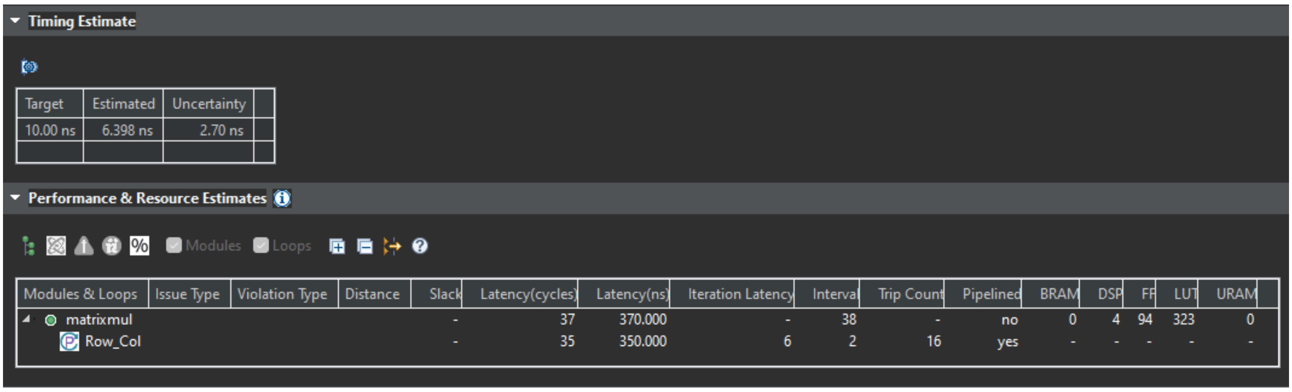


Figure 6.

(b) Screen capture of pipelining the sub-inner loop synthesis result.


Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	6.398 ns	2.70 ns

Performance & Resource Estimates

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul				-	37	370.000	-	38	-	no	0	4	94	323	0
Row_Col				-	35	350.000	6	2	16	yes	-	-	-	-	-

Figure 7.

(c) Screen capture of pipelining the sub-inner loop cosimulation result.

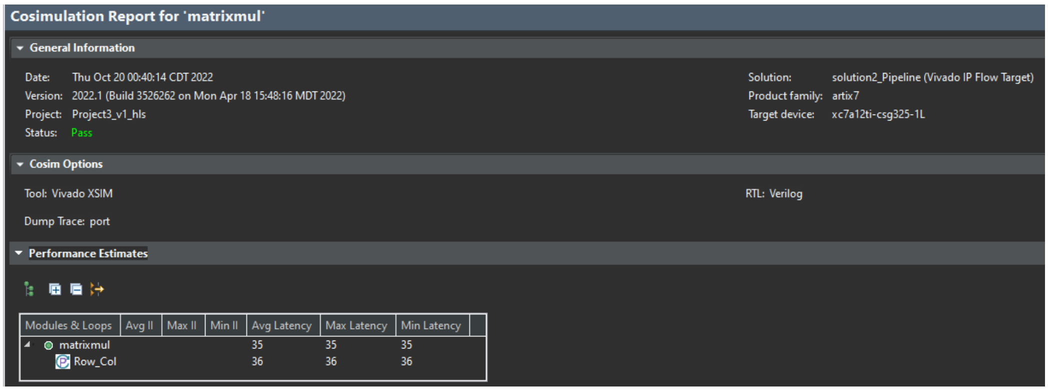


Figure 8.

(d) Screen capture of pipelining the sub-inner loop schedule viewer result.

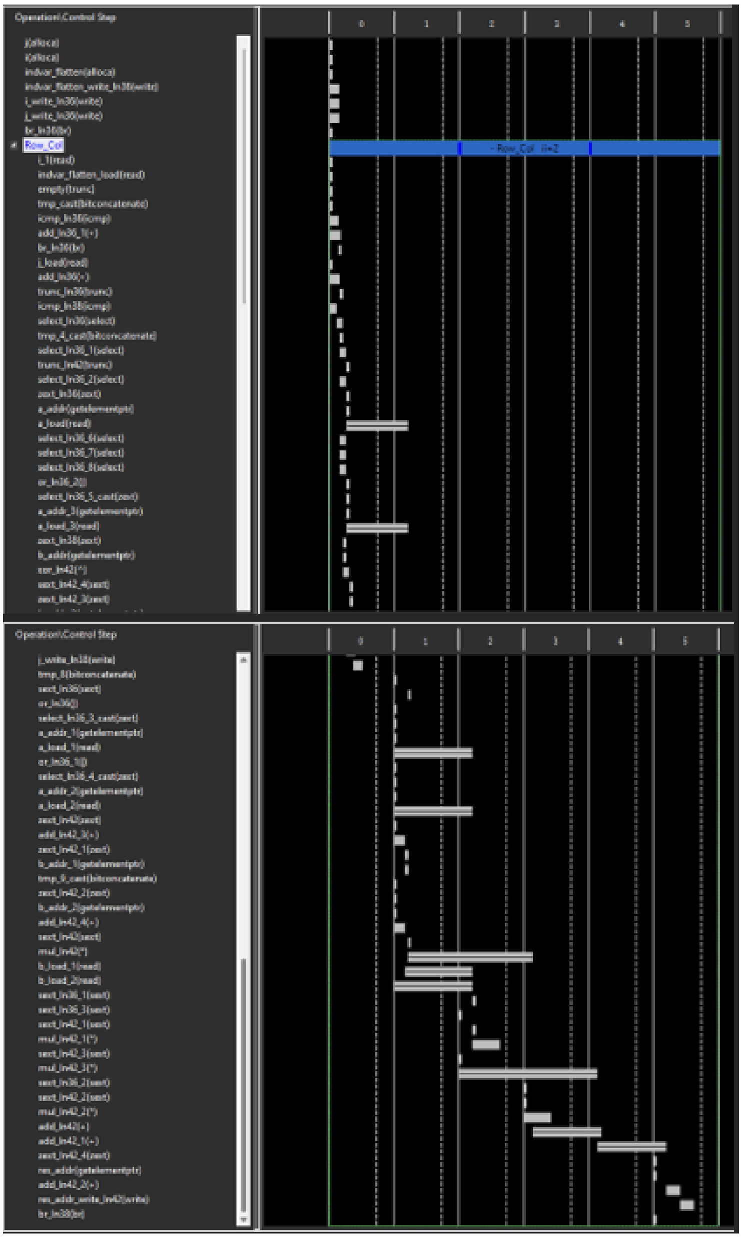
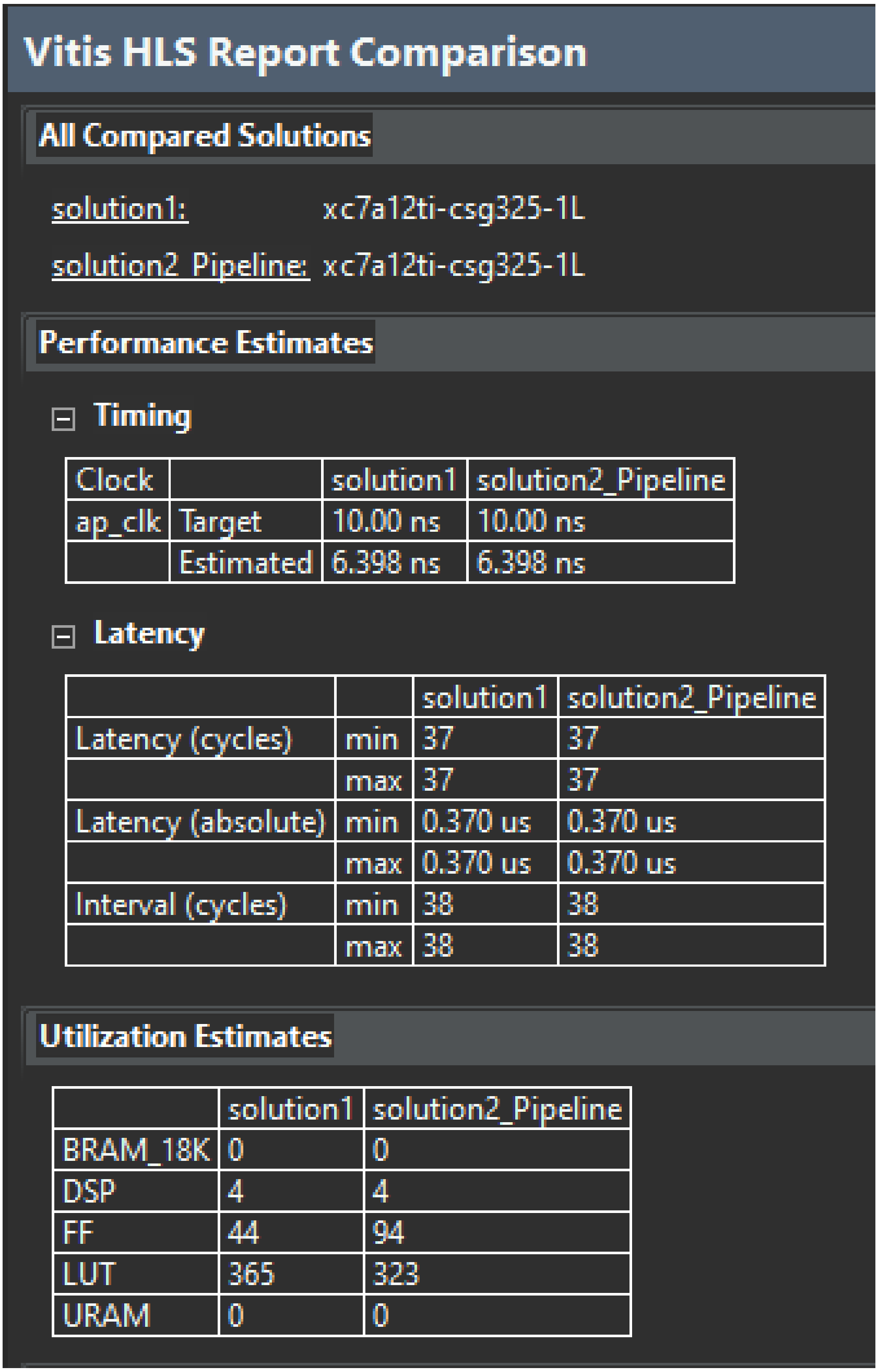


Figure 9.

(e) Screen capture of compare the default result and pipelining sub-inner loop result.



Vitis HLS Report Comparison

All Compared Solutions

solution1: xc7a12ti-csg325-1L

solution2_Pipeline: xc7a12ti-csg325-1L

Performance Estimates

Timing

Clock		solution1	solution2_Pipeline
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	6.398 ns	6.398 ns

Latency

		solution1	solution2_Pipeline
Latency (cycles)	min	37	37
	max	37	37
Latency (absolute)	min	0.370 us	0.370 us
	max	0.370 us	0.370 us
Interval (cycles)	min	38	38
	max	38	38

Utilization Estimates

	solution1	solution2_Pipeline
BRAM_18K	0	0
DSP	4	4
FF	44	94
LUT	365	323
URAM	0	0

Figure 10.

As you can see from the *Figure 10*, we reduce the hardware resource costs from **365 LUT** to **323 LUT**, but latency didn't change.

Second, instead of pipelining the sub-inner loop, this time I reshape the input matrix, so that these data can be read at the same time after splitting, then do the pipeline operations later, this method should get a better latency result, since we let the data read at the same time.

(a) Screen capture of my second optimization method. (ARRAY_RESHAPE)

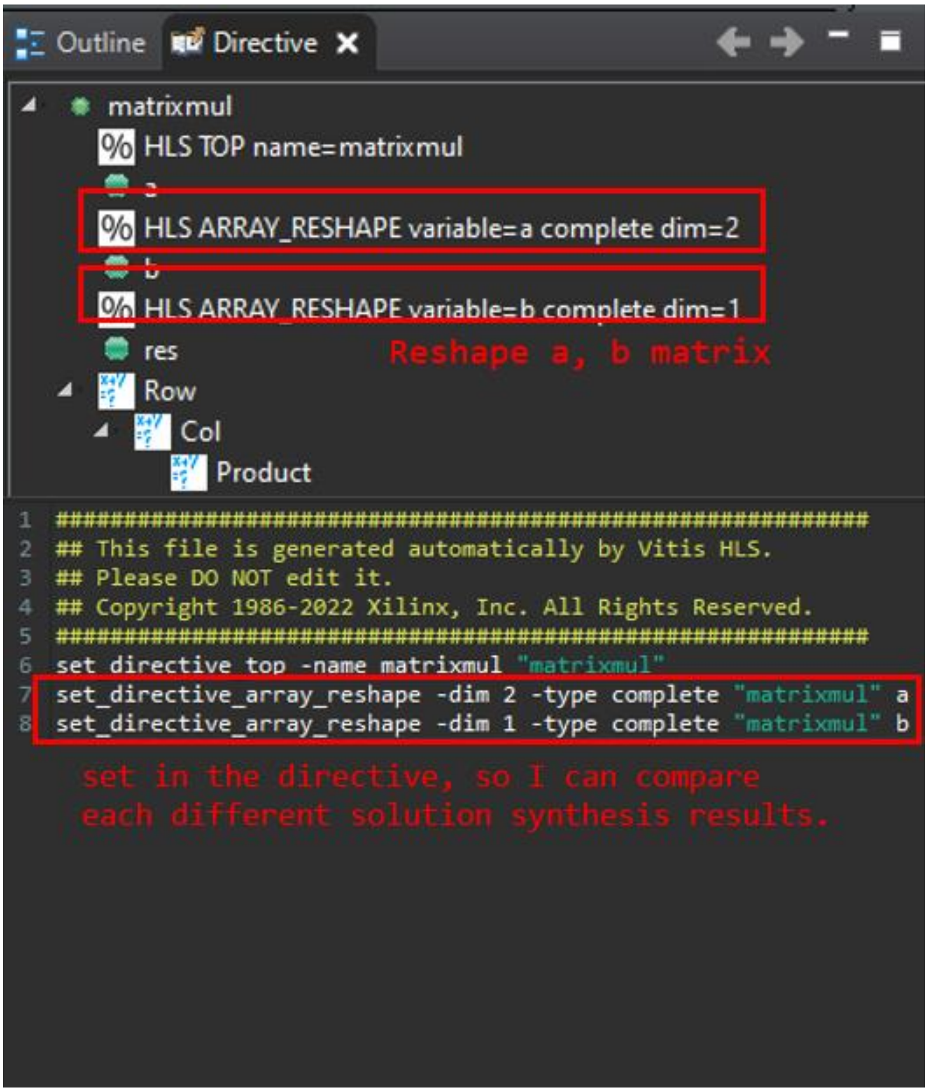
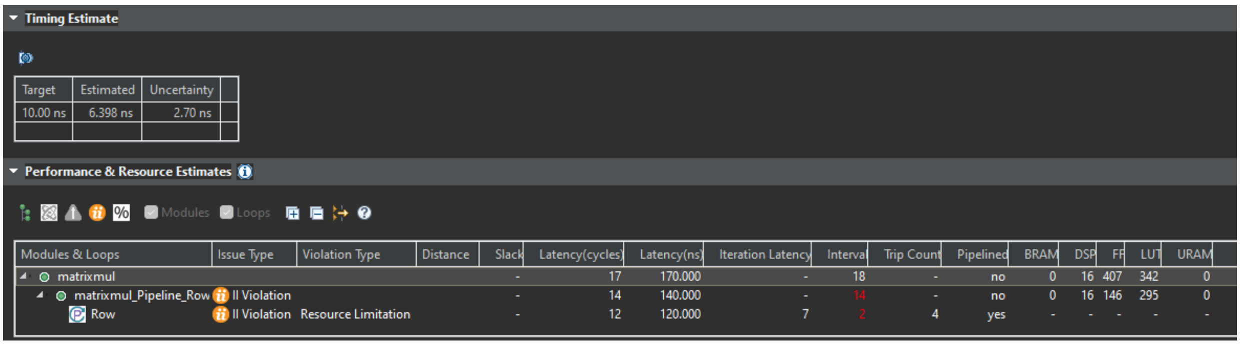


Figure 11.

(b) Screen capture of array_reshape synthesis result.



Target	Estimated	Uncertainty
10.00 ns	6.398 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul				-	17	170.000	-	18	-	no	0	16	407	342	0
matrixmul_Pipeline_Row	II Violation			-	14	140.000	-	14	-	no	0	16	146	295	0
Row	II Violation	Resource Limitation		-	12	120.000	7	2	4	yes	-	-	-	-	-

Figure 12.

(c) Screen capture of array_reshape cosimulation result.

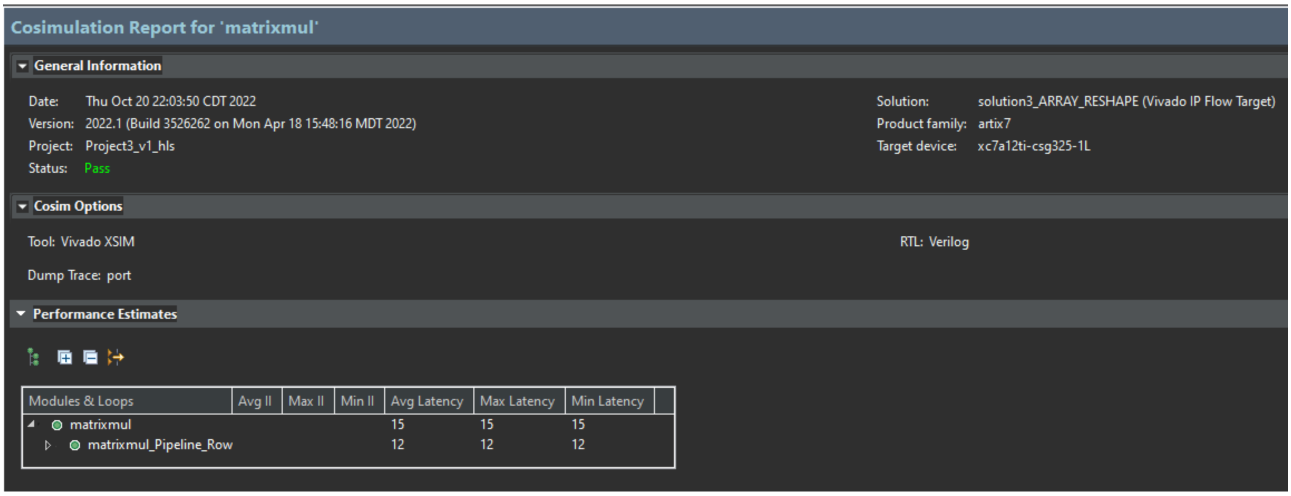


Figure 13.

(d) Screen capture of array_reshape schedule viewer result.

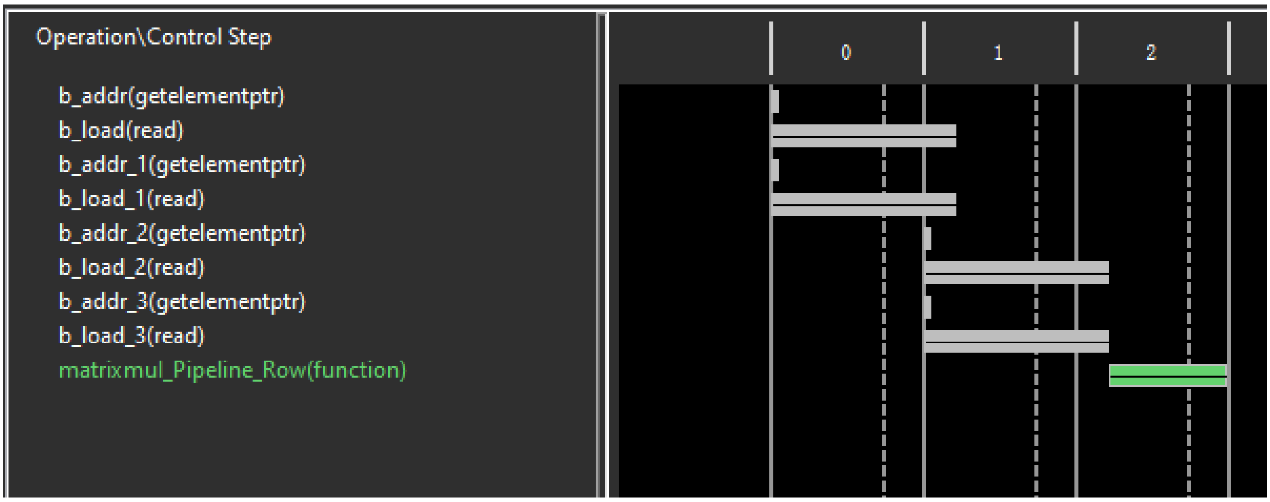


Figure 14.

As you can see from the *Figure 14*, schedule viewer results show that we reduced many operations, because we reduce the number of block RAM consumed while providing parallel access to the data.

(e) Screen capture of compare the default result, pipelining sub-inner loop result and array_reshape result.

Vitis HLS Report Comparison

All Compared Solutions

solution1: xc7a12ti-csg325-1L

solution2_Pipeline: xc7a12ti-csg325-1L

solution3_ARRAY_RESHAPE: xc7a12ti-csg325-1L

Performance Estimates

Timing

Clock		solution1	solution2_Pipeline	solution3_ARRAY_RESHAPE
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	6.398 ns	6.398 ns	6.398 ns

Latency

		solution1	solution2_Pipeline	solution3_ARRAY_RESHAPE
Latency (cycles)	min	37	37	17
	max	37	37	17
Latency (absolute)	min	0.370 us	0.370 us	0.170 us
	max	0.370 us	0.370 us	0.170 us
Interval (cycles)	min	38	38	18
	max	38	38	18

Utilization Estimates

	solution1	solution2_Pipeline	solution3_ARRAY_RESHAPE
BRAM_18K	0	0	0
DSP	4	4	16
FF	44	94	407
LUT	365	323	342
URAM	0	0	0

Figure 15.

As you can see from the *Figure 15*, we reduce throughput and latency from **37 cycles** to **17 cycles**, but hardware resource increase from **4 DSP** to **16 DSP** and **44 FF** to **407 FF**.

3. My final optimization method:

Finally, I used professor recommend optimization methods to my directives, and I successfully reduce the hardware resource costs, and a acceptable throughput and latency.

(a) Screen capture of final directive.

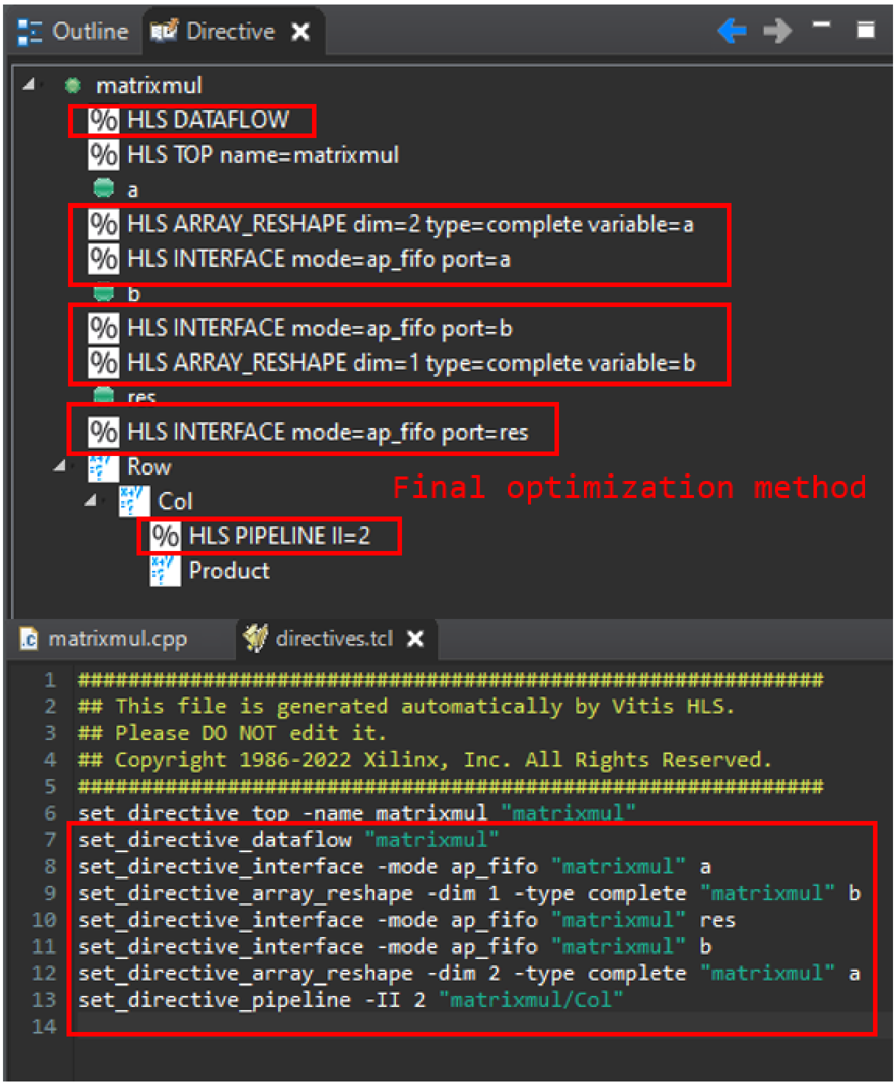
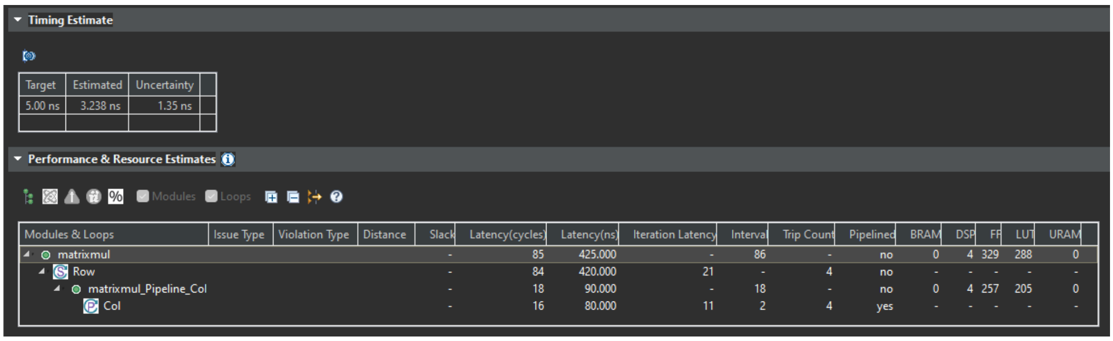


Figure 16.

(b) Screen capture of final optimization synthesis result.



Timing Estimate

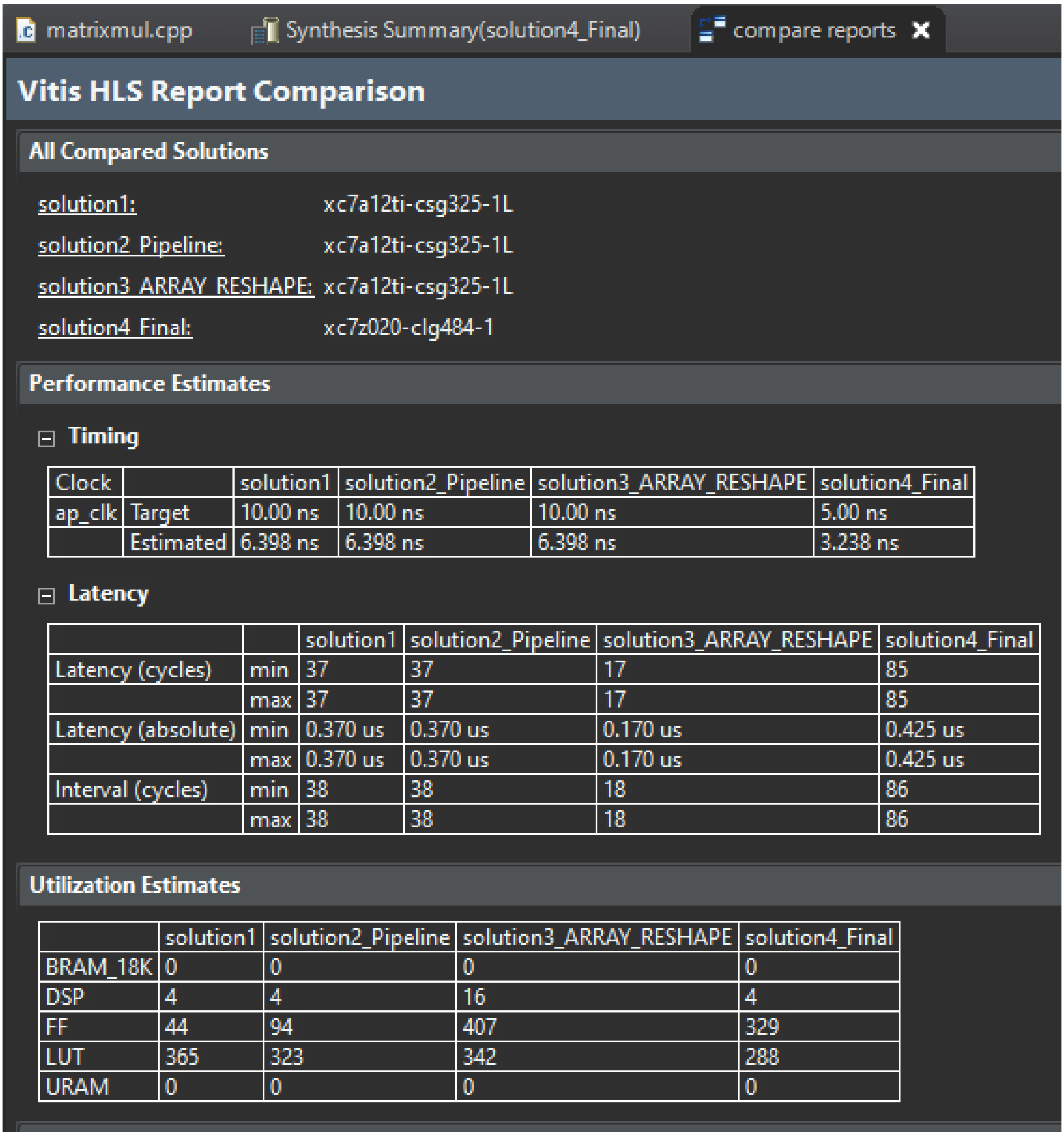
Target	Estimated	Uncertainty
5.00 ns	3.238 ns	1.35 ns

Performance & Resource Estimates

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
matrixmul				-	85	425.000	-	86	-	no	0	4	329	288	0
Row				-	84	420.000	21	-	4	no	-	-	-	-	-
matrixmul_Pipeline_Col				-	18	90.000	-	18	-	no	0	4	257	205	0
Col				-	16	80.000	11	2	4	yes	-	-	-	-	-

Figure 17.

(c) Screen capture of compare the default result, pipelining sub-inner loop result, array_reshape result and final optimization result.


matrixmul.cpp | Synthesis Summary(solution4_Final) | compare reports

Vitis HLS Report Comparison

All Compared Solutions

solution1: xc7a12ti-csg325-1L

solution2_Pipeline: xc7a12ti-csg325-1L

solution3_ARRAY_RESHAPE: xc7a12ti-csg325-1L

solution4_Final: xc7z020-clg484-1

Performance Estimates

Timing

Clock		solution1	solution2_Pipeline	solution3_ARRAY_RESHAPE	solution4_Final
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	5.00 ns
	Estimated	6.398 ns	6.398 ns	6.398 ns	3.238 ns

Latency

		solution1	solution2_Pipeline	solution3_ARRAY_RESHAPE	solution4_Final
Latency (cycles)	min	37	37	17	85
	max	37	37	17	85
Latency (absolute)	min	0.370 us	0.370 us	0.170 us	0.425 us
	max	0.370 us	0.370 us	0.170 us	0.425 us
Interval (cycles)	min	38	38	18	86
	max	38	38	18	86

Utilization Estimates

	solution1	solution2_Pipeline	solution3_ARRAY_RESHAPE	solution4_Final
BRAM_18K	0	0	0	0
DSP	4	4	16	4
FF	44	94	407	329
LUT	365	323	342	288
URAM	0	0	0	0

Figure 18.

Model Composer:

1. Matrix multiplication system architecture in model composer

I designed this matrix multiplication with 8 inputs for a, b matrix, also 5 Boolean values set and 7 output, but only res_din will output the matrix multiply result, as shown in *Figure 19*.

(a) Screen capture of my matrix multiplication system architecture. (Model composer)

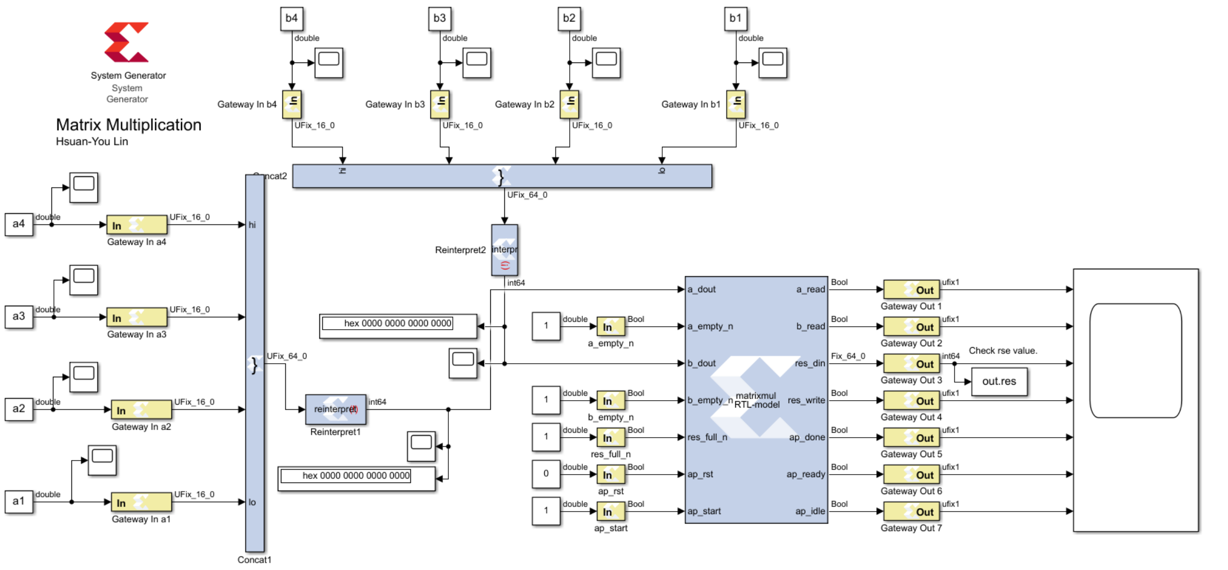


Figure 19.

2. Random input matrix m-code file:

The input matrix will randomly generate for a and b matrix, as shown in *Figure 21*.

(a) Screen capture of random input matrix in command window.

```
>> Input_Matrix
A Input Matrix:
     3     6     5     9
     8     6     3     3
     9     2     9     7
    10     9     8     7

B Input Matrix:
     2     3     5     6
     5     9     7     4
     3     9     9     5
     8     4     7     8

A*B Matrix result:
   123   144   165   139
    79   117   130   111
   111   154   189   163
   145   211   234   192

>>
```

Figure 20.

(b) Screen capture of my random input matrix m-code.

Input_Matrix.m | Output_Matrix.m | Continuous_Input_Matrix.m | Continuous_Output_Matrix.m

```matlab
% MATLAB developed at the Rice University.
% Electrical and Computer Engineering Department.
% Hsuan-You (Shaun) Lin (Oct. 20, 2022)

% Program to build random input data for 4x4 Matrix Multiplication.

% Clears all variables previously assigned.
clear
% Closes all graphs and windows open.
close all

matrix_size = 4;

% Generating first set of random matrices to be inserted to HLS block
a = randi([1 10], matrix_size);
b = randi([1 10], matrix_size);
a1_t = [zeros(4,1); a(1,1); a(2,1); a(3,1); a(4,1); zeros(9,1)];
a2_t = [zeros(4,1); a(1,2); a(2,2); a(3,2); a(4,2); zeros(9,1)];
a3_t = [zeros(4,1); a(1,3); a(2,3); a(3,3); a(4,3); zeros(9,1)];
a4_t = [zeros(4,1); a(1,4); a(2,4); a(3,4); a(4,4); zeros(9,1)];

b1_t = [b(1,1); b(1,2); b(1,3); b(1,4); zeros(18,1)];
b2_t = [b(2,1); b(2,2); b(2,3); b(2,4); zeros(18,1)];
b3_t = [b(3,1); b(3,2); b(3,3); b(3,4); zeros(18,1)];
b4_t = [b(4,1); b(4,2); b(4,3); b(4,4); zeros(18,1)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Constructing the inputs to be inserted to the HLS block
a1 = timeseries(a1_t);
a2 = timeseries(a2_t);
a3 = timeseries(a3_t);
a4 = timeseries(a4_t);
b1 = timeseries(b1_t);
b2 = timeseries(b2_t);
b3 = timeseries(b3_t);
b4 = timeseries(b4_t);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp("A Input Matrix:");
disp(a);
disp("B Input Matrix:");
disp(b);
result = a * b;
disp("A*B Matrix result:");
disp(result);
```

Figure 21.

3. Output Matrix m-code file:

In professor’s example used Fix_16 for HLS result output, but I used FIX_64 for the output, then decode hex to dec number easier to visualize the result, as shown in *Figure 23*.

(a) Screen capture of output matrix result in command window.

```
>> Output_Matrix
Matrix result after simulation:
   123   144   165   139
    79   117   130   111
   111   154   189   163
   145   211   234   192

>>
```

Figure 22.

(b) Screen capture of output matrix m-code.

Input_Matrix.m | Output_Matrix.m | Continuous_Input_Matrix.m | Continuous_Output_Matrix.m

```
% MATLAB developed at the Rice University.
% Electrical and Computer Engineering Department.
% Hsuan-You (Shaun) Lin (Oct. 20, 2022)

% Program to show output data for 4x4 Matrix Multiplication.

matrix_size = 4;

arr = getdatasamples(out.res, [13:16]);
c = dec2hex(arr, matrix_size^2);

c11 = hex2dec(c(1, 13:16));
c12 = hex2dec(c(1, 9:12));
c13 = hex2dec(c(1, 5:8));
c14 = hex2dec(c(1, 1:4));
c21 = hex2dec(c(2, 13:16));
c22 = hex2dec(c(2, 9:12));
c23 = hex2dec(c(2, 5:8));
c24 = hex2dec(c(2, 1:4));
c31 = hex2dec(c(3, 13:16));
c32 = hex2dec(c(3, 9:12));
c33 = hex2dec(c(3, 5:8));
c34 = hex2dec(c(3, 1:4));
c41 = hex2dec(c(4, 13:16));
c42 = hex2dec(c(4, 9:12));
c43 = hex2dec(c(4, 5:8));
c44 = hex2dec(c(4, 1:4));

result_after_sim = [c11, c12, c13, c14;
                    c21, c22, c23, c24;
                    c31, c32, c33, c34;
                    c41, c42, c43, c44];
disp("Matrix result after simulation:");
disp(result_after_sim);
```

Figure 23.

As you can see from the workspace, the out.res saved result in 12~16 clock, so I set this range in my m-code, then decode them.

(c) Screen capture of out.res from workspace.

out.res

Time series name:

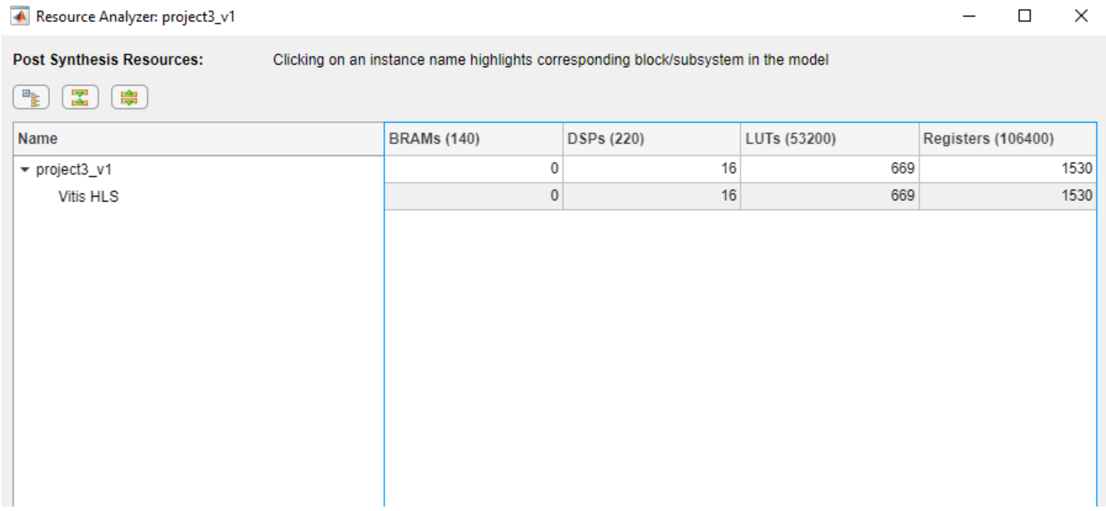
Time	Data:1
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	3.9126e+16
13	3.1244e+16
14	4.5881e+16
15	5.4044e+16
16	5.4044e+16

Figure 24.

4. System Generator - Resources:

In my Vitis HLS project, I not only used PIPELINE and Array_reshape optimization methods, I also used DATAFLOW and ap_fifo.

(a) Screen capture of system generator resources analyzer result.



Name	BRAMs (140)	DSPs (220)	LUTs (53200)	Registers (106400)
project3_v1	0	16	669	1530
Vitis HLS	0	16	669	1530

Figure 25.

(b) Screen capture of system generator completed generation.

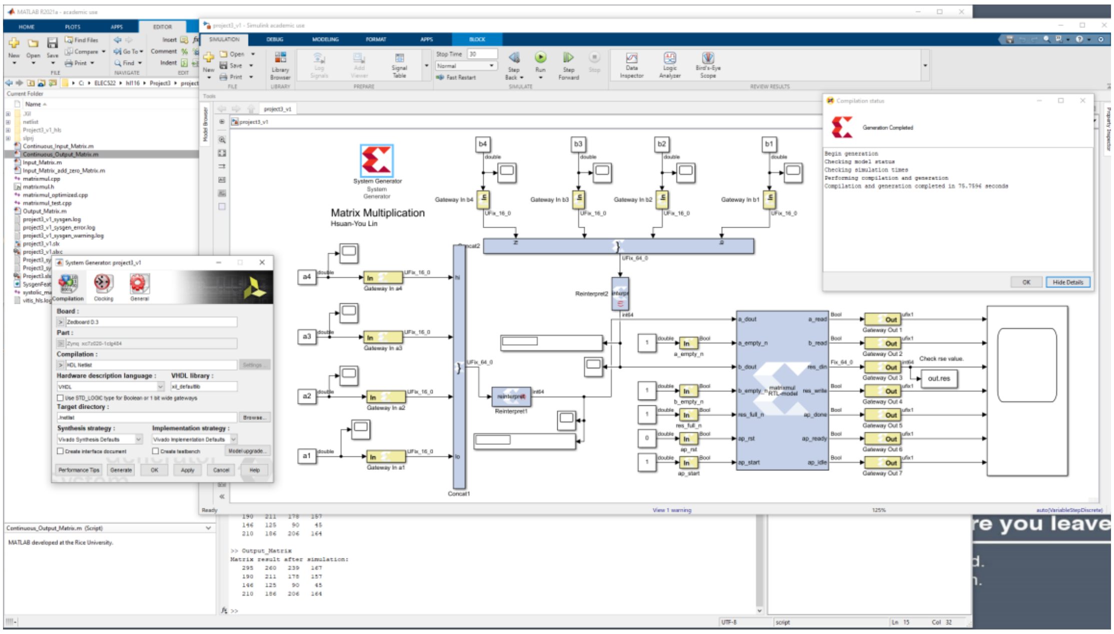


Figure 26.

Continuous Matrix Multiplication:

1. Continuous random input/output matrix m-code file:

We just need to modify the original random input matrix, add the continuous matrix after 1st matrix, as shown in *Figure 27, and the result shown in Figure 28*.

(a) Screen capture of my continuous random input matrix m-code.

Input_Matrix.m | Output_Matrix.m | Continuous_Input_Matrix.m | Continuous_Output_Matrix.m

```
% MATLAB developed at the Rice University.
% Electrical and Computer Engineering Department.
% Hsuan-You (Shaun) Lin (Oct. 20, 2022)

% Program to build random continuous input data for 4x4 Matrix Multiplication.

% Clears all variables previously assigned.
clear
% Closes all graphs and windows open.
close all

matrix_size = 4;

% Generating first set of random matrices to be inserted to HLS block
a_1st = randi([1 10], matrix_size);
b_1st = randi([1 10], matrix_size);

a_2nd = randi([1 10], matrix_size);
b_2nd = randi([1 10], matrix_size);

a1_t = [zeros(4,1); a_1st(1,1); a_1st(2,1); a_1st(3,1); a_1st(4,1);
    zeros(1,1); a_2nd(1,1); a_2nd(2,1); a_2nd(3,1); a_2nd(4,1); zeros(15,1)];
a2_t = [zeros(4,1); a_1st(1,2); a_1st(2,2); a_1st(3,2); a_1st(4,2);
    zeros(1,1); a_2nd(1,2); a_2nd(2,2); a_2nd(3,2); a_2nd(4,2); zeros(15,1)];
a3_t = [zeros(4,1); a_1st(1,3); a_1st(2,3); a_1st(3,3); a_1st(4,3);
    zeros(1,1); a_2nd(1,3); a_2nd(2,3); a_2nd(3,3); a_2nd(4,3); zeros(15,1)];
a4_t = [zeros(4,1); a_1st(1,4); a_1st(2,4); a_1st(3,4); a_1st(4,4);
    zeros(1,1); a_2nd(1,4); a_2nd(2,4); a_2nd(3,4); a_2nd(4,4); zeros(15,1)];

b1_t = [b_1st(1,1); b_1st(1,2); b_1st(1,3); b_1st(1,4);
    b_2nd(1,1); b_2nd(1,2); b_2nd(1,3); b_2nd(1,4); zeros(15,1)];
b2_t = [b_1st(2,1); b_1st(2,2); b_1st(2,3); b_1st(2,4);
    b_2nd(2,1); b_2nd(2,2); b_2nd(2,3); b_2nd(2,4); zeros(15,1)];
b3_t = [b_1st(3,1); b_1st(3,2); b_1st(3,3); b_1st(3,4);
    b_2nd(3,1); b_2nd(3,2); b_2nd(3,3); b_2nd(3,4); zeros(15,1)];
b4_t = [b_1st(4,1); b_1st(4,2); b_1st(4,3); b_1st(4,4);
    b_2nd(4,1); b_2nd(4,2); b_2nd(4,3); b_2nd(4,4); zeros(15,1)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Constructing the inputs to be inserted to the HLS block
a1 = timeseries(a1_t);
a2 = timeseries(a2_t);
a3 = timeseries(a3_t);
a4 = timeseries(a4_t);
b1 = timeseries(b1_t);
b2 = timeseries(b2_t);
b3 = timeseries(b3_t);
b4 = timeseries(b4_t);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp("1st A Input Matrix:");
disp(a_1st);
disp("1st B Input Matrix:");
disp(b_1st);
result1 = a_1st * b_1st;
disp("1st A * B Matrix result:");
disp(result1);

disp("2nd A Input Matrix:");
disp(a_2nd);
disp("2nd B Input Matrix:");
disp(b_2nd);
result2 = a_2nd * b_2nd;
disp("2nd A * B Matrix result:");
disp(result2);
```

Figure 27.

(b) Screen capture of continuous input matrix in command window.

```
Command Window
>> Continuous_Input_Matrix
1st A Input Matrix:
     9     5     4     6
     8     5     3     6
     1     2     4     4
     7     9     4     4

1st B Input Matrix:
     6     5     1     1
     7     9     2     6
    10     2     4     1
     8     1     4     2

1st A * B Matrix result:
   177   104    59    55
   161    97    54    53
    92    35    37    25
   177   128    57    73

2nd A Input Matrix:
     7    10     5     1
     9     6     5     5
    10     6     1     9
     6     4     9     4

2nd B Input Matrix:
     7     2     6     3
     9     3     7     5
     9     9     9     5
    10     6     6    10

2nd A * B Matrix result:
   194    95   163   106
   212   111   171   132
   223   101   165   155
   199   129   169   123

fx >>
```

Figure 28.

(c) Screen capture of output matrix result in command window.

```
Command Window
>> Continuous_Output_Matrix
Continuous matrix result after simulation:
   177   104    59    55
   161    97    54    53
    92    35    37    25
   177   128    57    73

Continuous matrix result after simulation:
   194    95   163   106
   212   111   171   132
   223   101   165   155
   199   129   169   123

fx >>
```

Figure 29.

(d) Screen capture of output matrix m-code.

```
% MATLAB developed at the Rice University.
% Electrical and Computer Engineering Department.
% Hsuan-You (Shaun) Lin (Oct. 20, 2022)

% Program to show continuous output data for 4x4 Matrix Multiplication.

matrix_size = 4;
number_of_matrix = 2;

for count = 0:(number_of_matrix - 1)
    arr = getdatasamples(out.res,[((matrix_size)^2-3) + 5 * count : (matrix_size)^2 + 5 * count]);
    c=dec2hex(arr, matrix_size^2);

    c11 =hex2dec(c(1, 13:16));
    c12 =hex2dec(c(1, 9:12));
    c13 =hex2dec(c(1, 5:8));
    c14 =hex2dec(c(1, 1:4));
    c21 =hex2dec(c(2, 13:16));
    c22 =hex2dec(c(2, 9:12));
    c23 =hex2dec(c(2, 5:8));
    c24 =hex2dec(c(2, 1:4));
    c31 =hex2dec(c(3, 13:16));
    c32 =hex2dec(c(3, 9:12));
    c33 =hex2dec(c(3, 5:8));
    c34 =hex2dec(c(3, 1:4));
    c41 =hex2dec(c(4, 13:16));
    c42 =hex2dec(c(4, 9:12));
    c43 =hex2dec(c(4, 5:8));
    c44 =hex2dec(c(4, 1:4));

    result_after_sim = [c11, c12, c13, c14;
                        c21, c22, c23, c24;
                        c31, c32, c33, c34;
                        c41, c42, c43, c44];
    disp("Continuous matrix result after simulation:");
    disp(result_after_sim);
end
```

Figure 30.

Hardware co-simulation:

1. Co-Simulation matrix multiplication system in model composer

After successfully generate the system in JTAG compilation mode, I connect the input and output into gray block system, to simulate the system in Zedboard as shown in *Figure 31*.

(a) Screen capture of co-simulation matrix multiplication system architecture (model composer)

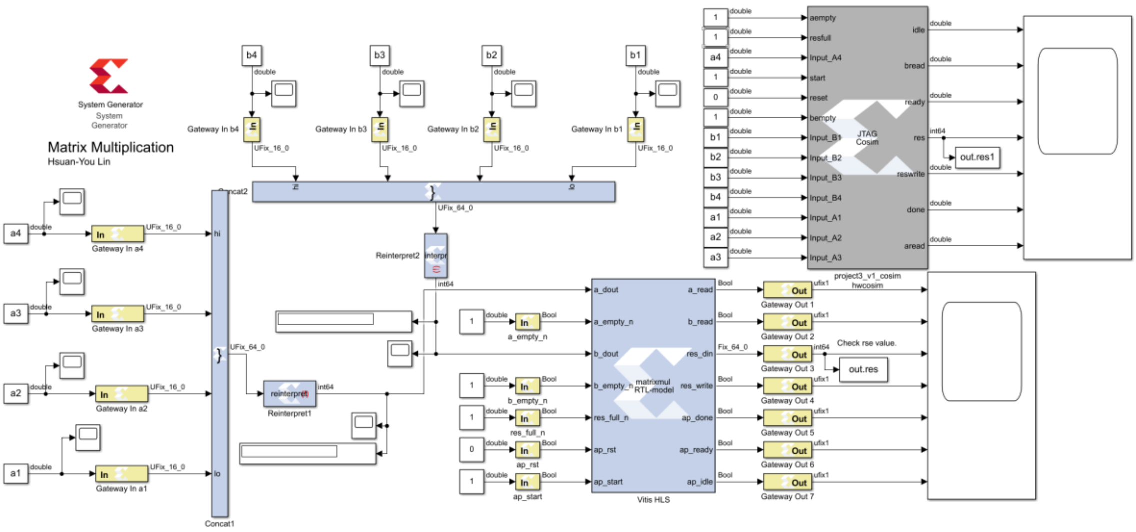


Figure 31.

(b) Screen capture of co-simulation resources analyzer.

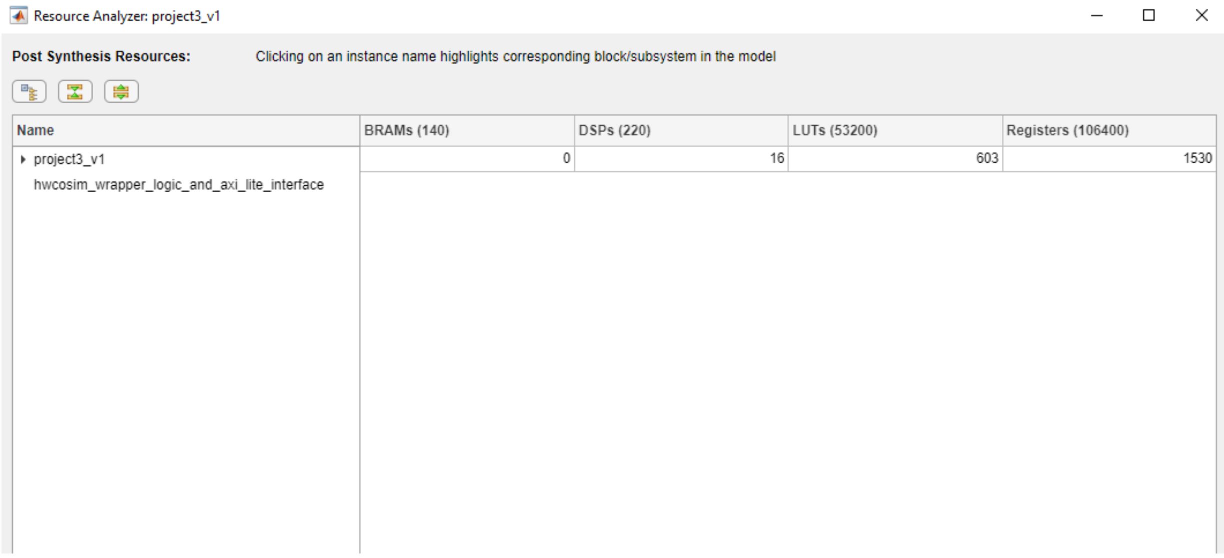


Figure 32.

Compare result:

Compared to the Project2, I think use Vitis HLS is more easier to develop the system, because we can use C/C++ which is more familiar to us, and we can also simply add the optimization methods into directives.