ESE 532 Homework 3

Liwen Zhong

Partner: Sheil Sarda

**1. Initial CPU implementation and HLS Kernel.**

**a.**

The visualized application timeline is shown below:

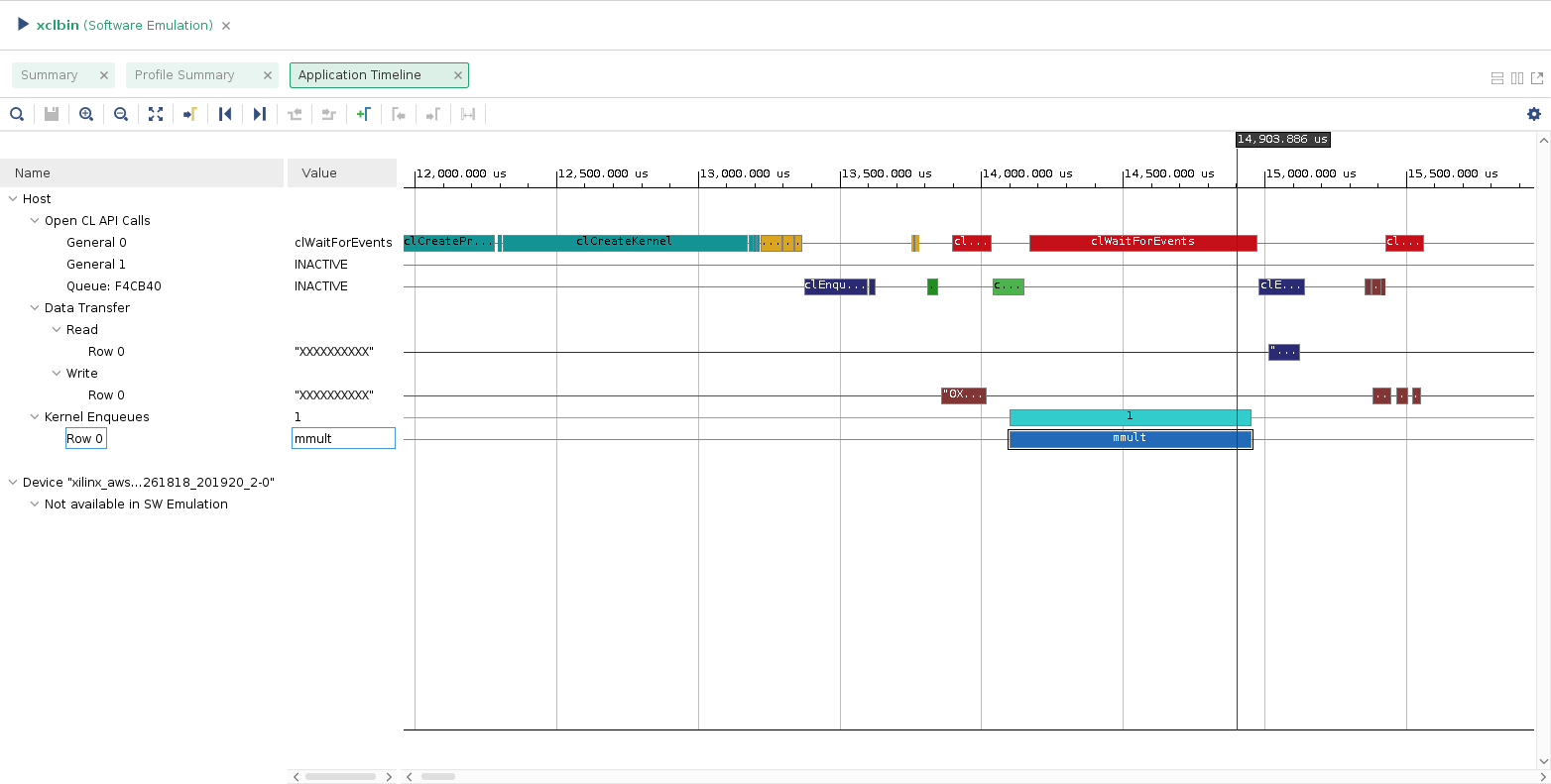


Figure 1

The latency of mmult kernel is 0.85060 ms.

**b.**

Starting C simulation ...

/opt/Xilinx/Vitis/2020.1/bin/vitis\_hls /home/centos/hw5/solution1/csim.tcl

INFO: [HLS 200-10] Running '/opt/Xilinx/Vitis/2020.1/bin/unwrapped/lnx64.o/vitis\_hls'

INFO: [HLS 200-10] For user 'centos' on host 'ip-172-31-76-23.ec2.internal' (Linux\_x86\_64 version 3.10.0-1127.10.1.el7.x86\_64) on Tue Oct 13 13:47:57 UTC 2020

INFO: [HLS 200-10] On os "CentOS Linux release 7.7.1908 (Core)"

INFO: [HLS 200-10] In directory '/home/centos'

Sourcing Tcl script '/home/centos/hw5/solution1/csim.tcl'

INFO: [HLS 200-10] Opening project '/home/centos/hw5'.

INFO: [HLS 200-10] Adding design file 'ese532\_code/hw5/hls/MatrixMultiplication.cpp' to the project

INFO: [HLS 200-10] Adding design file 'ese532\_code/hw5/hls/MatrixMultiplication.h' to the project

INFO: [HLS 200-10] Adding test bench file 'ese532\_code/hw5/hls/Testbench.cpp' to the project

INFO: [HLS 200-10] Opening solution '/home/centos/hw5/solution1'.

INFO: [SYN 201-201] Setting up clock 'default' with a period of 8ns.

INFO: [HLS 200-10] Setting target device to 'xcvu9p-flga2104-1-i'

INFO: [HLS 200-1505] Using flow\_target 'vitis'

Resolution: For help on HLS 200-1505 see www.xilinx.com/html\_docs/xilinx2020\_1/hls-guidance/200-1505.html

INFO: [HLS 200-435] Setting 'config\_flow -target' configuration: config\_interface -m\_axi\_latency=64

INFO: [HLS 200-435] Setting 'config\_flow -target' configuration: config\_interface -m\_axi\_alignment\_byte\_size=64

INFO: [HLS 200-435] Setting 'config\_flow -target' configuration: config\_interface -m\_axi\_max\_widen\_bitwidth=512

INFO: [HLS 200-435] Setting 'config\_flow -target' configuration: config\_interface -default\_slave\_interface=s\_axilite

INFO: [HLS 200-435] Setting 'config\_flow -target' configuration: config\_rtl -register\_reset\_num=3

INFO: [HLS 200-1464] Running solution command: config\_compile -dump\_cfg=false

INFO: [HLS 200-1464] Running solution command: config\_compile -name\_max\_length=80

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [HLS 200-1464] Running solution command: config\_compile -no\_signed\_zeros=false

INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its signedness.

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [HLS 200-1464] Running solution command: config\_compile -pipeline\_loops=0

INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or its parent loop when its trip count is less than or equal 0.

INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its signedness.

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [HLS 200-1464] Running solution command: config\_compile -pipeline\_style=stp

INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or its parent loop when its trip count is less than or equal 0.

INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its signedness.

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [HLS 200-1464] Running solution command: config\_compile -pragma\_strict\_mode=false

INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or its parent loop when its trip count is less than or equal 0.

INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its signedness.

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [HLS 200-1464] Running solution command: config\_compile -unsafe\_math\_optimizations=false

INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or its parent loop when its trip count is less than or equal 0.

INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its signedness.

INFO: [XFORM 203-1173] Reordering floating point operations aggressively.

INFO: [XFORM 203-1176] Optimizing floating point comparison without checking NaN.

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [SYN 201-201] Setting up clock 'default' with a period of 8ns.

INFO: [XFORM 203-1171] Pipeline the innermost loop with trip count more than 0 or its parent loop when its trip count is less than or equal 0.

INFO: [XFORM 203-1172] Optimizing floating point zeros and discarding its signedness.

INFO: [XFORM 203-1173] Reordering floating point operations aggressively.

INFO: [XFORM 203-1176] Optimizing floating point comparison without checking NaN.

INFO: [XFORM 203-1161] The maximum of name length is set into 80.

INFO: [SIM 211-2] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CSIM start \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INFO: [SIM 211-4] CSIM will launch GCC as the compiler.

Compiling ../../../../ese532\_code/hw5/hls/Testbench.cpp in debug mode

Compiling ../../../../ese532\_code/hw5/hls/MatrixMultiplication.cpp in debug mode

Generating csim.exe

TEST PASSED

INFO: [SIM 211-1] CSim done with 0 errors.

INFO: [SIM 211-3] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CSIM finish \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Finished C simulation.

**c.**

We are doing matrix multiplying in our application. To test the functionality of the code, we test the code by giving it a test input, just like how we test a circuit with a test input signal. The Testbench.cpp does the job by giving two test matrix and compare the actual output with the expected output.

**d.**

The synthesis report is shown below:

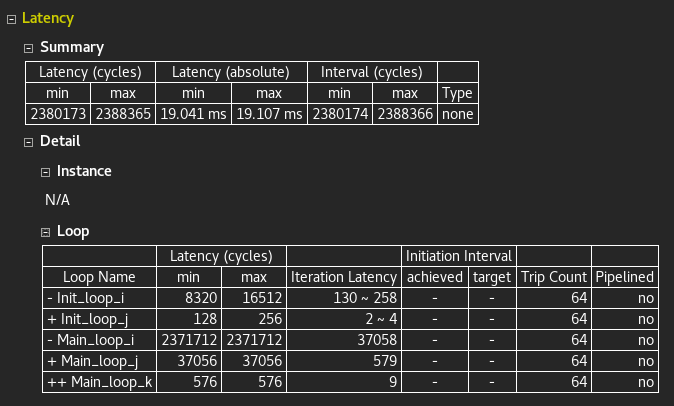


Figure 2

The expected latency is 19.041 ~ 19.107ms.

**e.**

The resource utilization estimates are shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | BRAM | DSP | FF | LUT | URAM |
| **Total** | 76 | 5 | 7600 | 5532 | 0 |

**f.**

The time schedule of multiplication (the yellow box) is shown below:

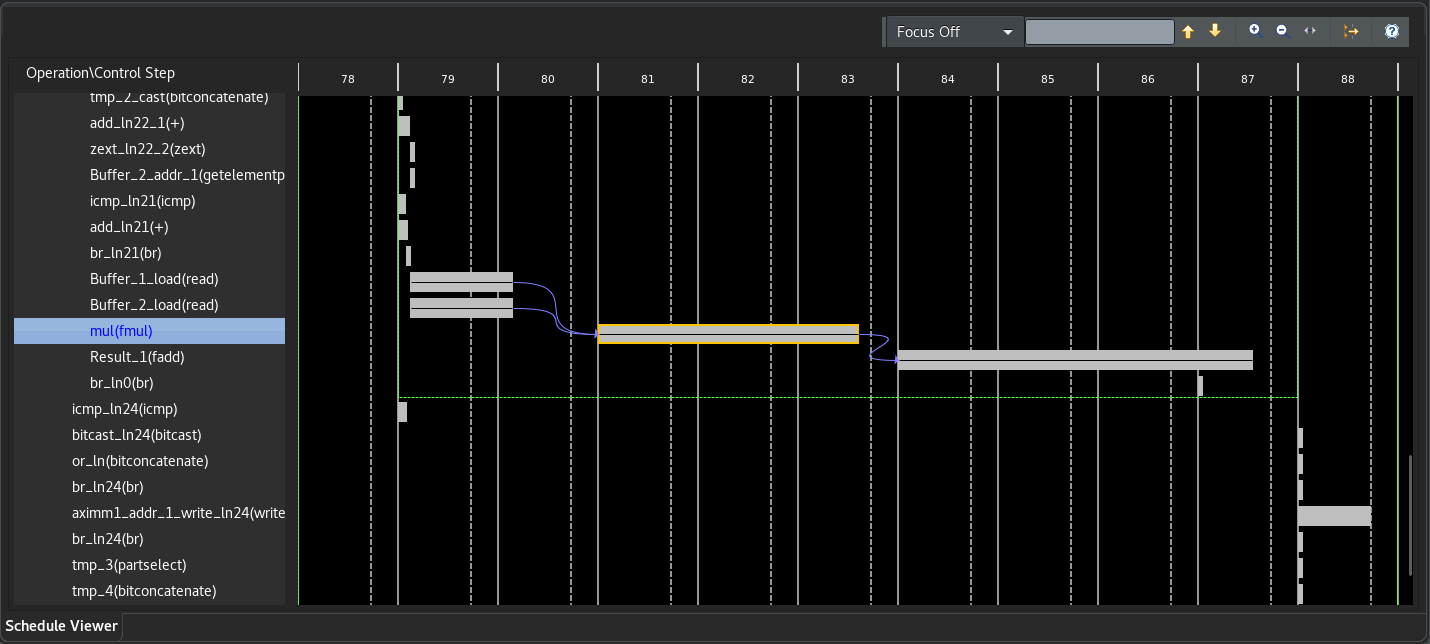


Figure 3

The multiplication takes approximately 2.6 cycles.

**g.**

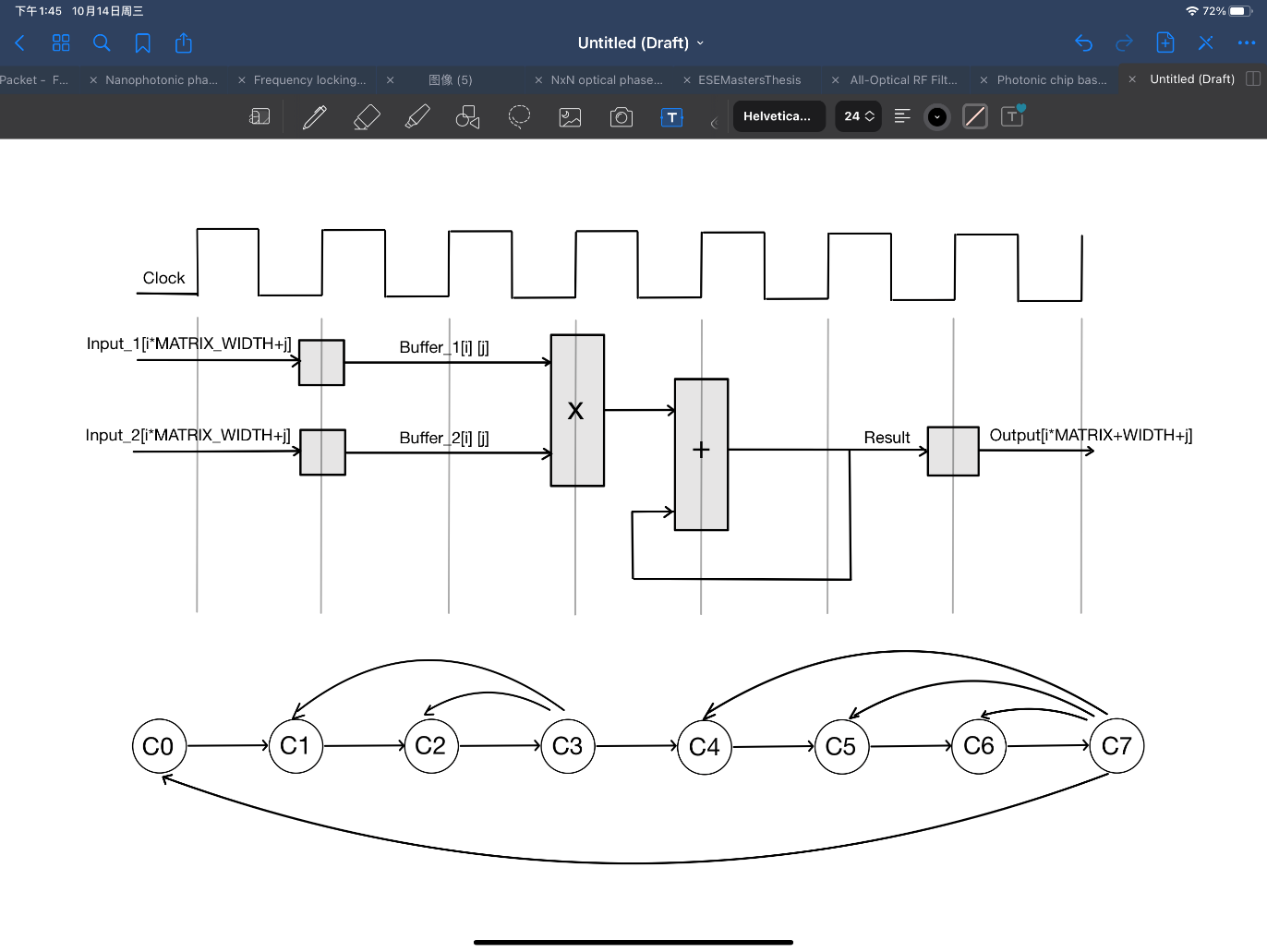


Figure 4

**h.**

There are many factors that slow down the accelerator. For example, the loops are unpipelined and unparalleled. The reading and writing to the public memory also take a relatively long time.

**2. HLS Kernel Optimization: Loop Unrolling**

**a.**

The code is shown below:

Main\_loop\_k: for (int k = 0; k < MATRIX\_WIDTH; k++) {

#pragma HLS unroll factor=2

Result += Buffer\_1[i][k] \* Buffer\_2[k][j];

}

The synthesis report is shown below:

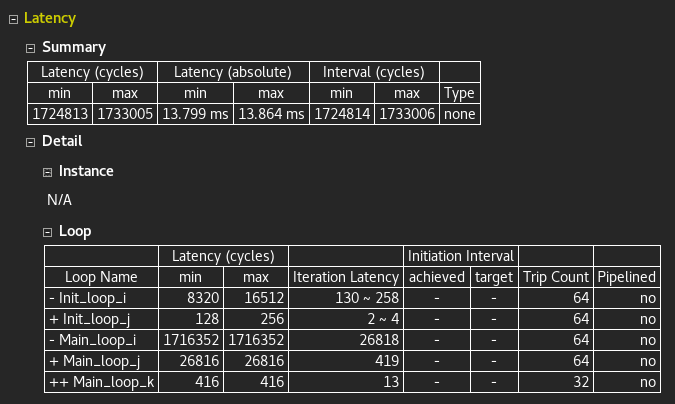


Figure 5

The expected latency is 13.799 ~ 13.861ms.

Loops in the C/C++ functions are kept rolled by default. When loops are rolled, synthesis creates the logic for one iteration of the loop, and the RTL design executes this logic for each iteration of the loop in sequence. A loop is executed for the number of iterations specified by the loop induction variable. Using the UNROLL pragma we can unroll loops to increase data access and throughput.

**b.**

The unrolled loop will look like the following:

Main\_loop\_k: for (int k = 0; k < MATRIX\_WIDTH; k+=2) {

    Result += Buffer\_1[i][k] \* Buffer\_2[k][j];

    if (k + 1 >= MATRIX\_WIDTH) break;

    Result += Buffer\_1[i][k + 1] \* Buffer\_2[k + 1][j];

}

**c.**

The resource utilization estimates are shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | BRAM | DSP | FF | LUT | URAM |
| **Total** | 76 | 8 | 7860 | 5650 | 0 |

The time schedule of multiplication (the yellow box) is shown below:

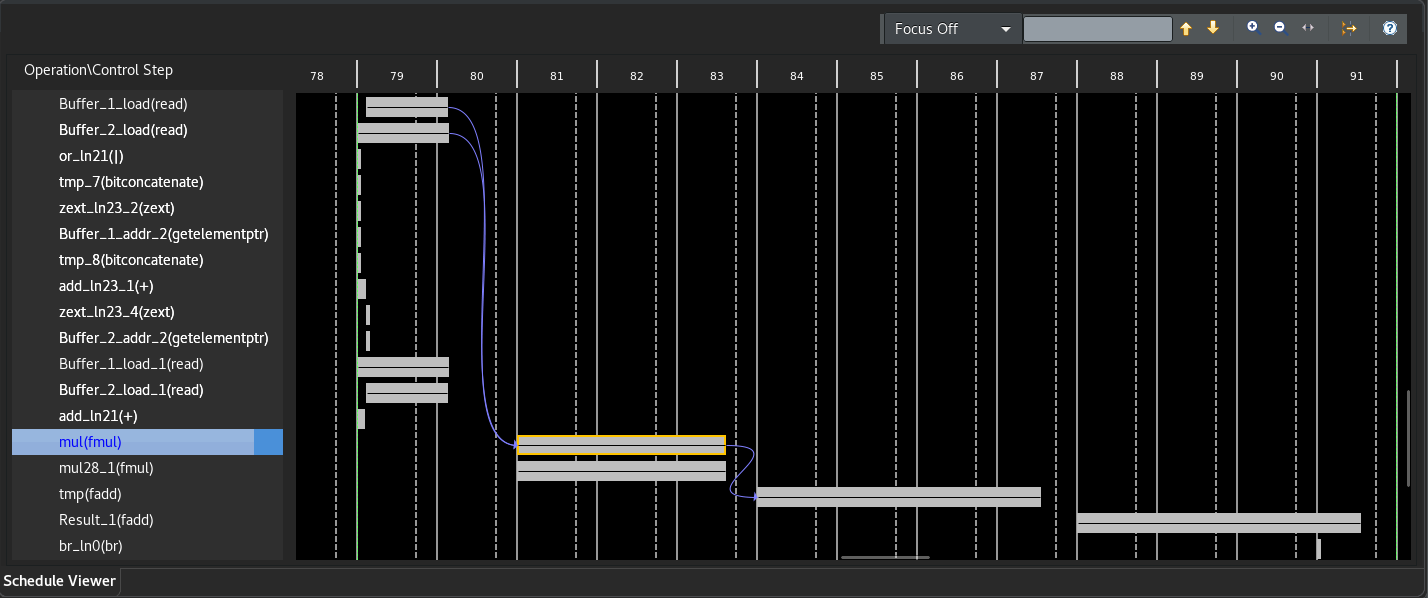


Figure 6

We can see the buffer loading (Buffer\_1\_load\_1 and Buffer\_2\_load\_2) operations and multiplication (fmul) operations are shared by multiple operations.

**d.**

The code is shown below:

Main\_loop\_k: for (int k = 0; k < MATRIX\_WIDTH; k++) {

#pragma HLS unroll

Result += Buffer\_1[i][k] \* Buffer\_2[k][j];

}

The synthesis report is shown below. The latency estimation is shown correctly and is 2.084 ~ 2.149ms.

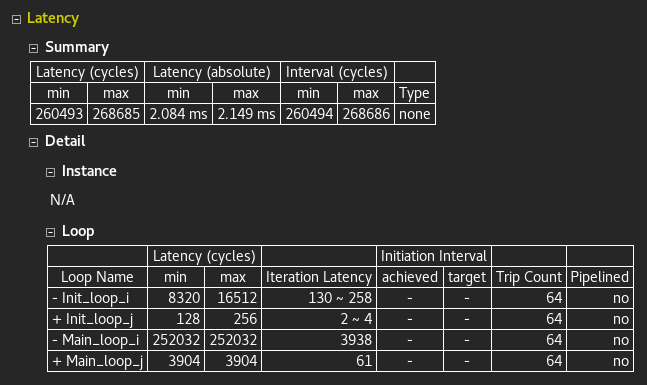


Figure 7

**e.**

The synthesis report is shown below. The estimated latency is 4.063 ~ 4.227ms.

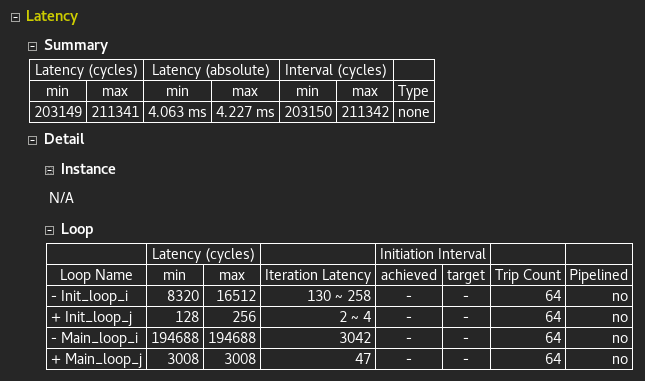


Figure 8

**f.**

The resource utilization estimates are shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | BRAM | DSP | FF | LUT | URAM |
| **Total** | 76 | 14 | 11517 | 8760 | 0 |

**g.**

The results show that unrolling loops does not parallelize floating-point additions.

**f.**

Based on the performance of the accelerators in 1d and 2d, we choose the one in 2b for more parallelism and can achieve the throughput of approximately 500 operations/s.

**3. HLS Kernel Optimization: Pipelining**

**a.**

The code is shown below:

Main\_loop\_i: for (int i = 0; i < MATRIX\_WIDTH; i++)

    Main\_loop\_j: for (int j = 0; j < MATRIX\_WIDTH; j++) {

        #pragma HLS pipeline

        matrix\_type Result = 0;

        Main\_loop\_k: for (int k = 0; k < MATRIX\_WIDTH; k++) {

            Result += Buffer\_1[i][k] \* Buffer\_2[k][j];

        }

        Output[i \* MATRIX\_WIDTH + j] = Result;

    }

The synthesis report is shown below:

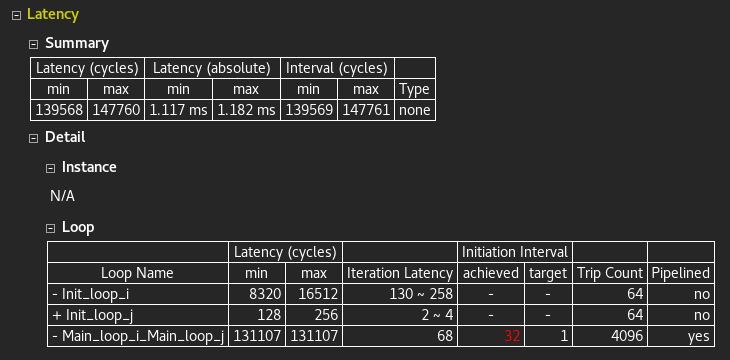


Figure 9

We achieved an II of 32 cycles.

**b.**

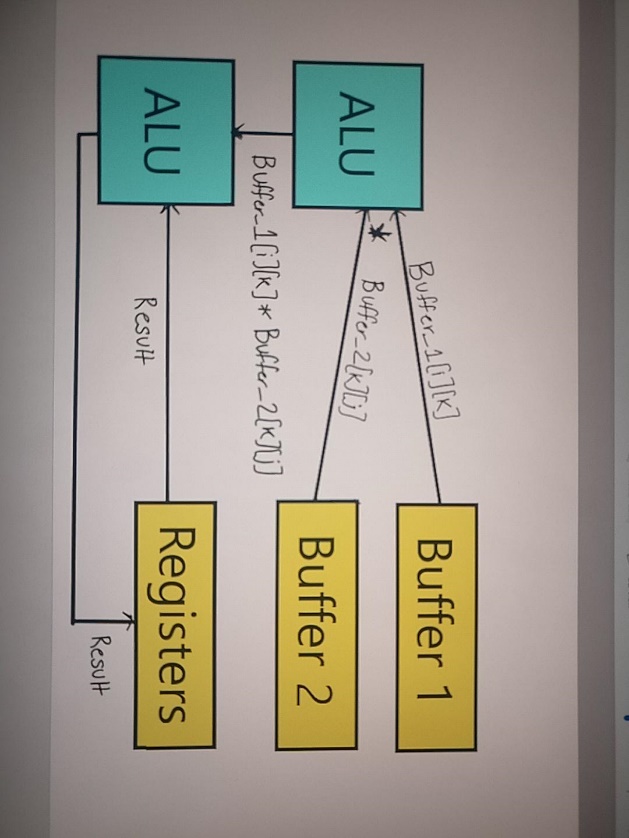


Figure 10

**c.**

4096 words/cycle

**d.**

The minimal initiation interval is limited by the number of words loaded from the buffer per cycle. We can achieve better performance by increasing the loading speed.

**e.**

Arrays are implemented as block RAM which only has a maximum of two data ports. This can limit the throughput of a read/write (or load/store) intensive algorithm. The bandwidth can be improved by splitting the array (a single block RAM resource) into multiple smaller arrays (multiple block RAMs), effectively increasing the number of ports.

**f.**

The code is shown below:

void mmult(const matrix\_type Input\_1[MATRIX\_WIDTH \* MATRIX\_WIDTH],

        const matrix\_type Input\_2[MATRIX\_WIDTH \* MATRIX\_WIDTH],

        matrix\_type Output[MATRIX\_WIDTH \* MATRIX\_WIDTH]) {

#pragma HLS INTERFACE m\_axi port=Input\_1 bundle=aximm1

#pragma HLS INTERFACE m\_axi port=Input\_2 bundle=aximm2

#pragma HLS INTERFACE m\_axi port=Output bundle=aximm

    matrix\_type Buffer\_1[MATRIX\_WIDTH][MATRIX\_WIDTH];

    matrix\_type Buffer\_2[MATRIX\_WIDTH][MATRIX\_WIDTH];

#pragma HLS array\_partition variable=Buffer\_1 block factor=32 dim=0

#pragma HLS array\_partition variable=Buffer\_2 block factor=32 dim=0

    Init\_loop\_i: for (int i = 0; i < MATRIX\_WIDTH; i++)

        Init\_loop\_j: for (int j = 0; j < MATRIX\_WIDTH; j++) {

            Buffer\_1[i][j] = Input\_1[i \* MATRIX\_WIDTH + j];

            Buffer\_2[i][j] = Input\_2[i \* MATRIX\_WIDTH + j];

        }

    Main\_loop\_i: for (int i = 0; i < MATRIX\_WIDTH; i++)

        Main\_loop\_j: for (int j = 0; j < MATRIX\_WIDTH; j++) {

            #pragma HLS pipeline

            matrix\_type Result = 0;

            Main\_loop\_k: for (int k = 0; k < MATRIX\_WIDTH; k++) {

                Result += Buffer\_1[i][k] \* Buffer\_2[k][j];

            }

            Output[i \* MATRIX\_WIDTH + j] = Result;

        }

}

The synthesis report is shown below:

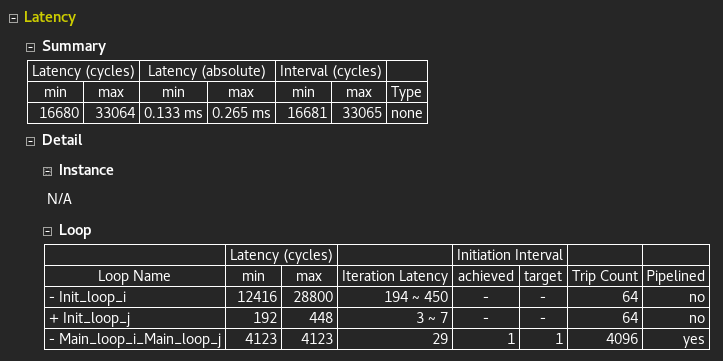


Figure 11

The estimated latency is 0.133 ~ 0.265ms.

**g.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | BRAM | DSP | FF | LUT | URAM |
| **Total** | 90 | 318 | 299579 | 884981 | 0 |

**h.**

Completed.

**4. Vitis Analyzer**

**a.**

Completed.

**b.**

Completed.

**c.**

Completed.

**d.**

The latency of the mmult kernel is 1.02ms.

**e.**

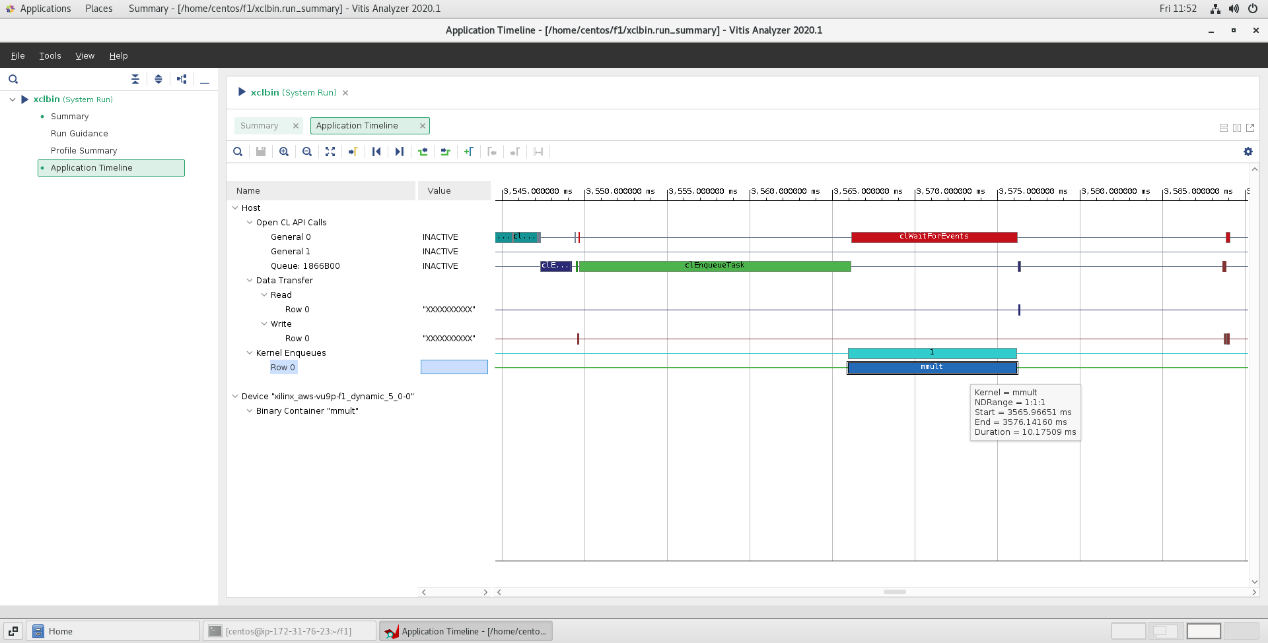


Figure 12

**5. Breakeven and Net Acceleration**

**a.**



**b.**



**c.**



**d.**

 should be more than 24.1ms.

**e.**



Where  is the constant for the processor.

**f.**



Where  is the constant for the FPGA.

**g.**



Where  is the constant for the FPGA.

**h.**

We have:



Solve the equation and we get the minimum integer N = 578.

**i.**

To achieve a 10× speedup, the minimum N is 4170.

**j.**



**k.**

Large invocation of the accelerators will largely reduce N. N = 411.

**6. Reflection**

**a.**

In part 1, we analyze the baseline performance and some basic optimization (unrolling). In part 2, we exploit optimizing the performance by pipelining the loop. In part 3, we exploit the bottleneck that slows down part 2 and remove that bottleneck by partitioning the buffers and increase the speed of data access.

**b.**

The time-DSPs plot is shown below:

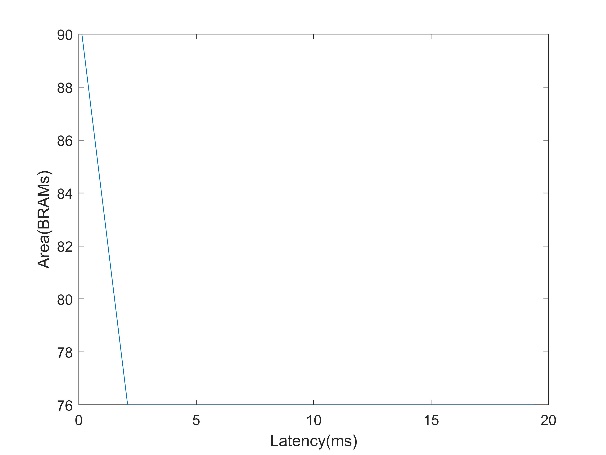
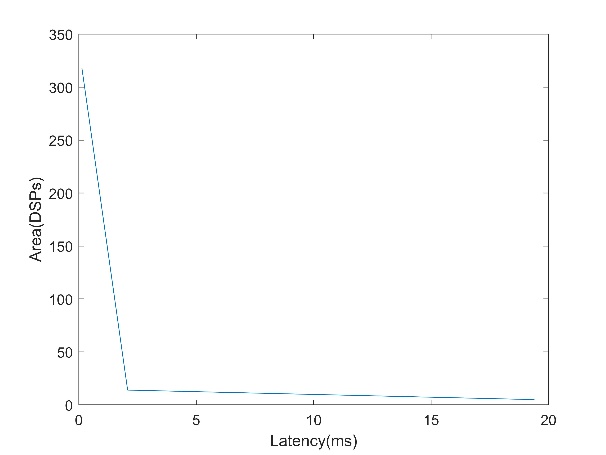


Figure 13

**c.**

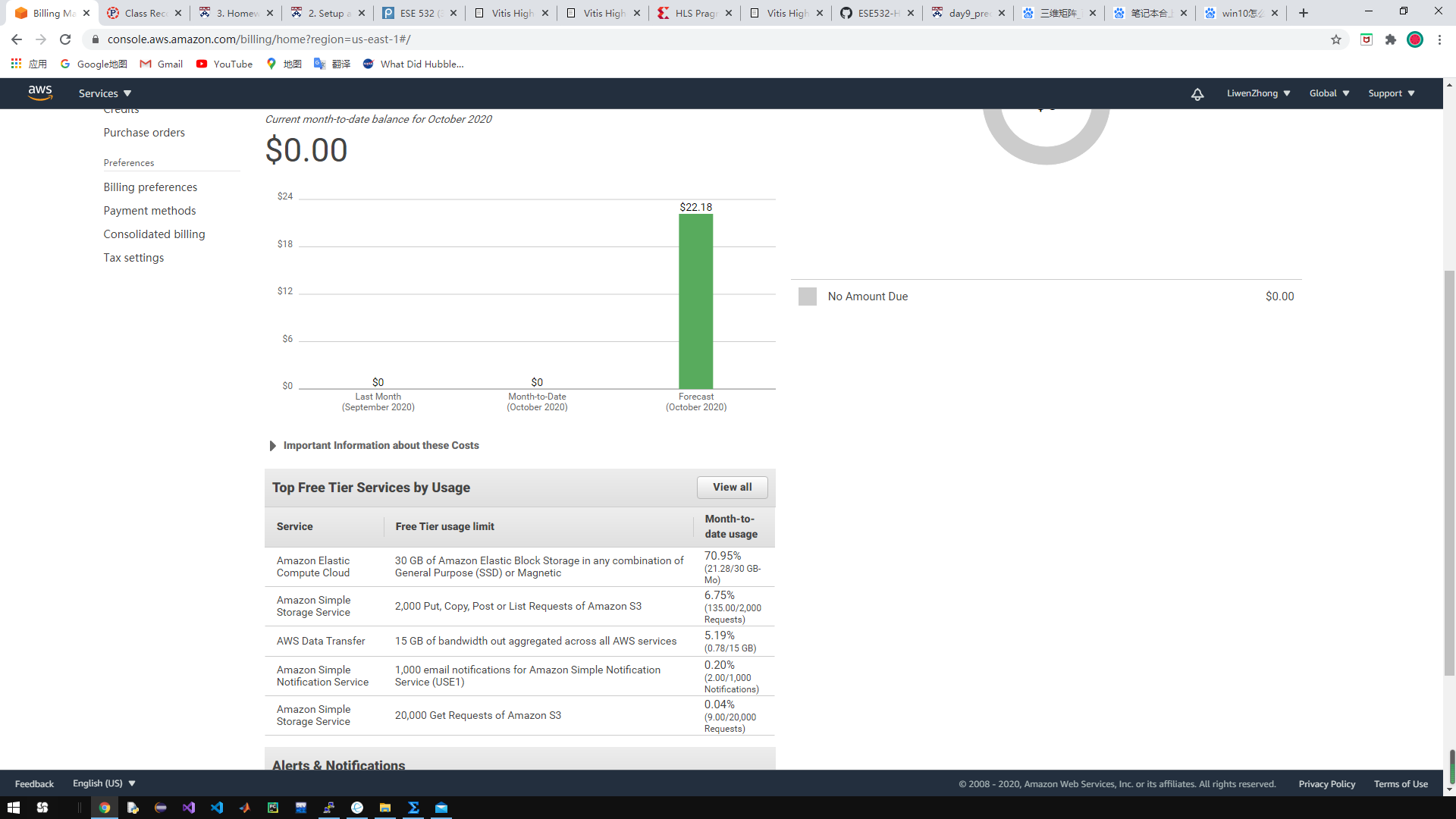


Figure 14