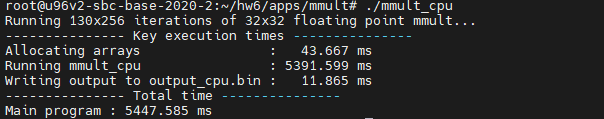
HW6 Write up

TEAM: Songqi Jing and Che Shize (Write Up by Songqi Jing)

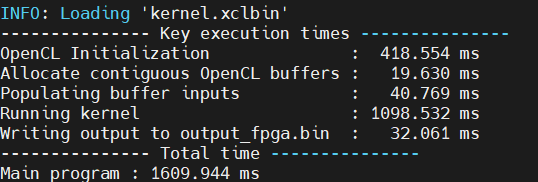
October 14, 2024

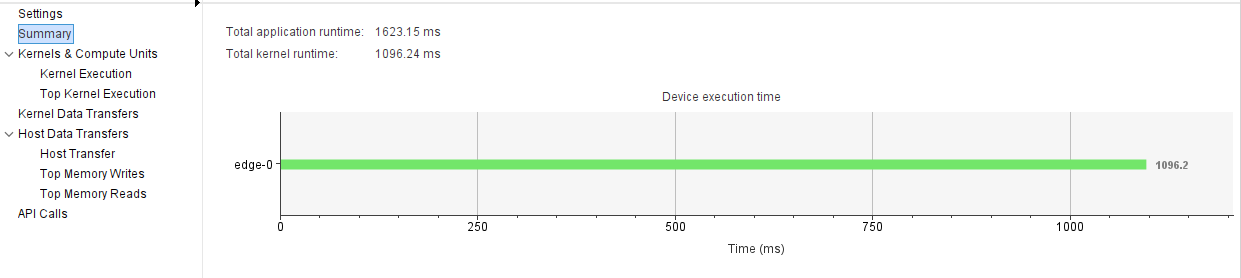
Q1.

1.

Done the latency is 5447.585 ms.

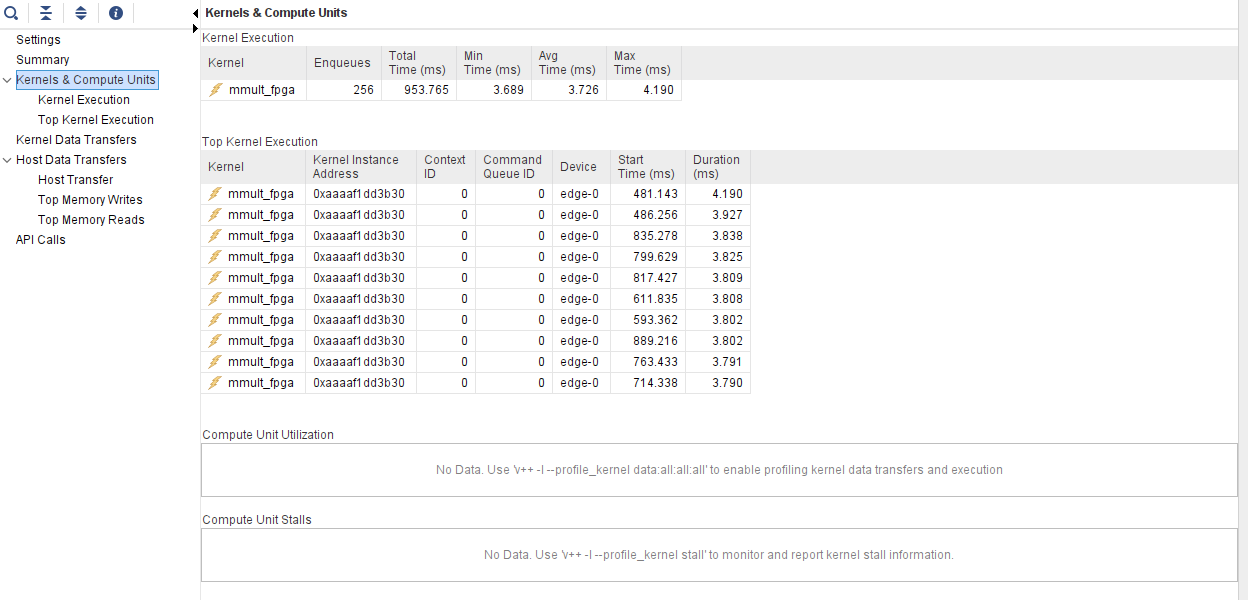
2.

The latency is 1609.944ms



The total application runtime is 1623.15 ms.

The total Kernel runtime is 1096.24 ms.



The total time of mmult\_fpga is 953.765 ms.

3.



a. We can use multi-cores to do the calculation in pipeline

b. We can unroll the loop if we got enough resources

c. We can use muti ports to transfer the data/

4.

Done

5

5.1

#include "MMult.h"

#include "hls\_stream.h"

/\*\*

\*

\* Design principles to achieve II = 1

\* 1. Stream data into local RAM for inputs (multiple access required)

\* 2. Partition local RAMs into N/2 sub-arrays for fully parallel access

\* (dual-port read)

\* 3. Pipeline the dot-product loop, to fully unroll it

\* 4. Separate multiply-accumulate in inner loop to force two FP operators

\*

\*/

void read(const float \*A, const float \*B,

hls::stream<float> &A\_tmp, hls::stream<float> &B\_tmp, int itr)

{

// #pragma HLS array\_partition variable=A\_tmp block factor=16 dim=2

// #pragma HLS array\_partition variable=B\_tmp block factor=16 dim=1

for (int c = 0; c < itr; c++) {

A\_tmp.write(A[c]);

B\_tmp.write(B[c]);

}

}

void execution(hls::stream<float> &A\_tmp, hls::stream<float> &B\_tmp, hls::stream<float> &result)

{

float A\_temp1[N][N];

float B\_temp1[N][N];

#pragma HLS array\_partition variable=A\_temp1 block factor=32 dim=2

#pragma HLS array\_partition variable=B\_temp1 block factor=32 dim=1

for (int c = 0; c < CHUNKS; c++) {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

A\_temp1[i][j]=A\_tmp.read();

B\_temp1[i][j]=B\_tmp.read();

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

#pragma HLS PIPELINE II=1

float res = 0;

for (int k = 0; k < N; k++) {

float term = A\_temp1[i][k] \* B\_temp1[k][j];

res += term;

}

result.write(res);

}

}

}

}

void write(hls::stream<float> &result, float \*output ,int itr){

for (int i = 0; i < itr; i++) {

output[i] = result.read();

}

}

void mmult\_fpga(float A[CHUNKS \* N \* N], float B[CHUNKS \* N \* N], float C[CHUNKS \* N \* N]) {

#pragma HLS INTERFACE m\_axi port=A bundle=b0

#pragma HLS INTERFACE m\_axi port=B bundle=b1

#pragma HLS INTERFACE m\_axi port=C bundle=b2

#pragma HLS INTERFACE ap\_ctrl\_chain port=return

#pragma HLS DATAFLOW

hls::stream<float> A\_tmp;

hls::stream<float> B\_tmp;

hls::stream<float> result;

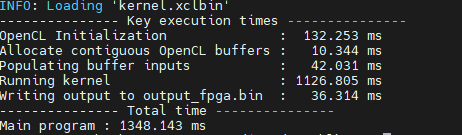
read(A, B, A\_tmp, B\_tmp, N\*N\*CHUNKS);

execution(A\_tmp, B\_tmp, result);

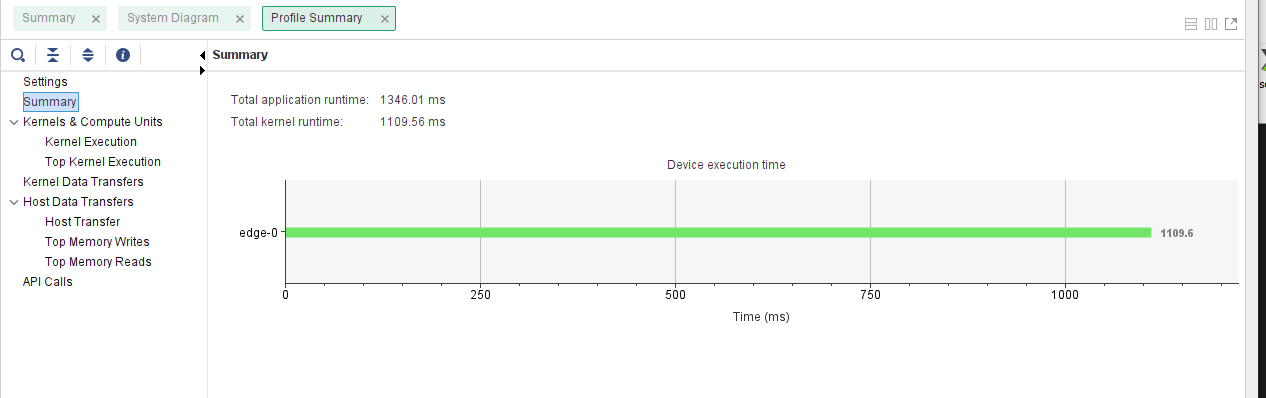
write(result, C, N\*N\*CHUNKS);

}

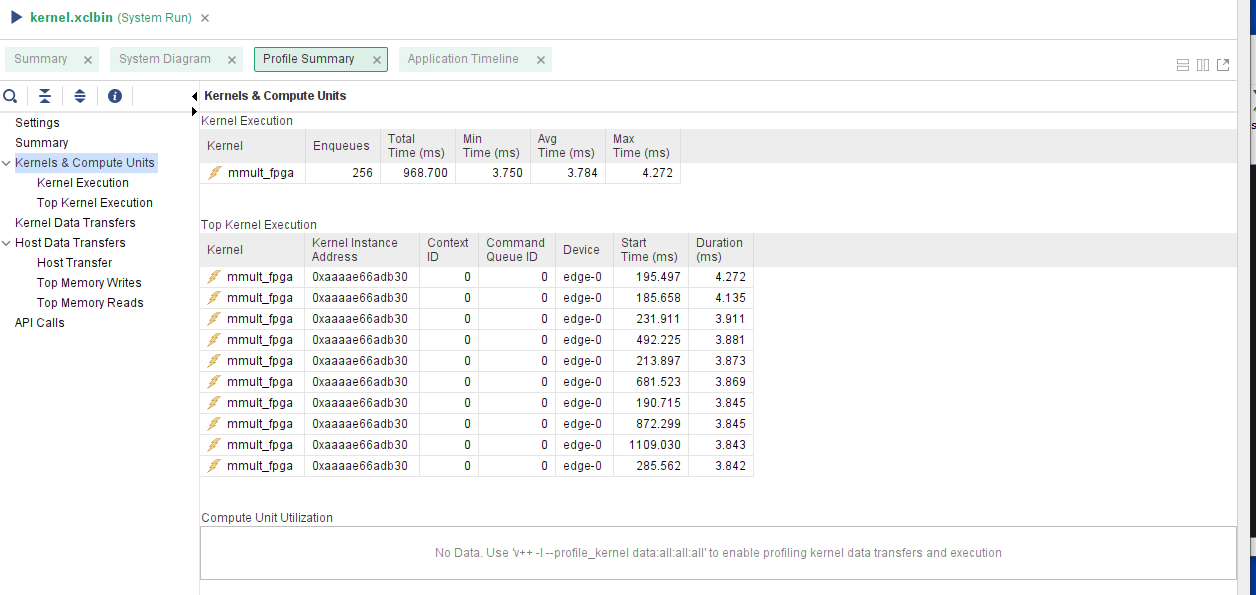
5.2



The latency is 1348.143 ms.

The application latency is 1346.01 ms.

The kernel runtime is 1109.56 ms.

The mmult\_fpga latency is 968.700 ms.



5.3

#define CL\_HPP\_CL\_1\_2\_DEFAULT\_BUILD

#define CL\_HPP\_TARGET\_OPENCL\_VERSION 120

#define CL\_HPP\_MINIMUM\_OPENCL\_VERSION 120

#define CL\_HPP\_ENABLE\_PROGRAM\_CONSTRUCTION\_FROM\_ARRAY\_COMPATIBILITY 1

#define CL\_USE\_DEPRECATED\_OPENCL\_1\_2\_APIS

#include "EventTimer.h"

#include <CL/cl2.hpp>

#include <cstdint>

#include <cstdlib>

#include <fstream>

#include <iostream>

#include <unistd.h>

#include <vector>

#include "MMult.h"

#include "Utilities.h"

static void init\_arrays(float \*A[NUM\_MAT], float \*B[NUM\_MAT])

{

for (int m = 0; m < NUM\_MAT; m++)

{

for (int c = 0; c < CHUNKS; c++)

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

A[m][ c \* N \* N + i \* N + j] = 1+i\*N+j;

B[m][ c \* N \* N + i \* N + j] = rand() % (N \* N);

}

}

}

}

}

int main(int argc, char \*argv[])

{

EventTimer timer1, timer2;

timer1.add("Main program");

std::cout << "Running " << CHUNKS << "x" <<NUM\_TESTS << " iterations of " << N << "x" << N

<< " task pipelined floating point mmult..." << std::endl;

// ------------------------------------------------------------------------------------

// Step 1: Initialize the OpenCL environment

// ------------------------------------------------------------------------------------

timer2.add("OpenCL Initialization");

cl\_int err;

std::string binaryFile = argv[1];

unsigned fileBufSize;

std::vector<cl::Device> devices = get\_xilinx\_devices();

devices.resize(1);

cl::Device device = devices[0];

cl::Context context(device, NULL, NULL, NULL, &err);

char \*fileBuf = read\_binary\_file(binaryFile, fileBufSize);

cl::Program::Binaries bins{{fileBuf, fileBufSize}};

cl::Program program(context, devices, bins, NULL, &err);

cl::CommandQueue q(context, device, CL\_QUEUE\_PROFILING\_ENABLE, &err);

cl::CommandQueue q1(context, device, CL\_QUEUE\_PROFILING\_ENABLE, &err);

cl::CommandQueue q2(context, device, CL\_QUEUE\_PROFILING\_ENABLE, &err);

cl::CommandQueue q3(context, device, CL\_QUEUE\_PROFILING\_ENABLE, &err);

cl::Kernel krnl\_mmult(program, "mmult\_fpga", &err);

// ------------------------------------------------------------------------------------

// Step 2: Create buffers and initialize test values

// ------------------------------------------------------------------------------------

timer2.add("Allocate contiguous OpenCL buffers");

size\_t elements\_per\_iteration = CHUNKS \* N \* N;

size\_t bytes\_per\_iteration = elements\_per\_iteration \* sizeof(float);

cl::Buffer a\_buf[NUM\_MAT];

cl::Buffer b\_buf[NUM\_MAT];

cl::Buffer c\_buf[NUM\_MAT];

for(int i = 0; i < NUM\_MAT; i++)

{

a\_buf[i] = cl::Buffer(context, CL\_MEM\_READ\_ONLY, bytes\_per\_iteration, NULL, &err);

b\_buf[i] = cl::Buffer(context, CL\_MEM\_READ\_ONLY, bytes\_per\_iteration, NULL, &err);

c\_buf[i] = cl::Buffer(context, CL\_MEM\_WRITE\_ONLY, bytes\_per\_iteration, NULL, &err);

}

float \*a[NUM\_MAT];

float \*b[NUM\_MAT];

float \*c[NUM\_MAT];

for(int i = 0; i < NUM\_MAT; i++)

{

a[i] = (float\*)q.enqueueMapBuffer(a\_buf[i], CL\_TRUE, CL\_MAP\_WRITE, 0, bytes\_per\_iteration);

b[i] = (float\*)q.enqueueMapBuffer(b\_buf[i], CL\_TRUE, CL\_MAP\_WRITE, 0, bytes\_per\_iteration);

c[i] = (float\*)q.enqueueMapBuffer(c\_buf[i], CL\_TRUE, CL\_MAP\_READ, 0, bytes\_per\_iteration);

}

timer2.add("Populating buffer inputs");

init\_arrays(a, b);

// ------------------------------------------------------------------------------------

// Step 3: Run the kernel

// ------------------------------------------------------------------------------------

timer2.add("Running kernel");

cl::Event w[NUM\_TESTS], x[NUM\_TESTS], r[NUM\_TESTS];

std::vector<cl::Event> w\_wl[NUM\_TESTS], x\_wl[NUM\_TESTS], r\_wl[NUM\_TESTS];

for (int i = 0; i < NUM\_TESTS; i+=4) {

krnl\_mmult.setArg(0, a\_buf[i%NUM\_MAT]);

krnl\_mmult.setArg(1, b\_buf[i%NUM\_MAT]);

krnl\_mmult.setArg(2, c\_buf[i%NUM\_MAT]);

if (i == 0) {

q.enqueueMigrateMemObjects({a\_buf[i%NUM\_MAT], b\_buf[i%NUM\_MAT]}, 0, NULL, &w[i]);

} else {

q.enqueueMigrateMemObjects({a\_buf[i%NUM\_MAT], b\_buf[i%NUM\_MAT]}, 0, &w\_wl[i], &w[i]);

}

x\_wl[i].push\_back(w[i]);

if (i + 1 < NUM\_TESTS) {

w\_wl[i+1].push\_back(w[i]);

}

q.enqueueTask(krnl\_mmult, &x\_wl[i], &x[i]);

r\_wl[i].push\_back(x[i]);

krnl\_mmult.setArg(0, a\_buf[(i+1)%NUM\_MAT]);

krnl\_mmult.setArg(1, b\_buf[(i+1)%NUM\_MAT]);

krnl\_mmult.setArg(2, c\_buf[(i+1)%NUM\_MAT]);

q1.enqueueMigrateMemObjects({a\_buf[(i+1)%NUM\_MAT], b\_buf[(i+1)%NUM\_MAT]}, 0, &w\_wl[i+1], &w[i+1]);

x\_wl[i+1].push\_back(w[i+1]);

if (i + 2 < NUM\_TESTS) {

w\_wl[i+2].push\_back(w[i+1]);

}

q.enqueueMigrateMemObjects({c\_buf[i%NUM\_MAT]}, CL\_MIGRATE\_MEM\_OBJECT\_HOST, &r\_wl[i], &r[i]);

if (i + 4 < NUM\_TESTS) {

w\_wl[i+4].push\_back(r[i]);

}

if (i + 1 < NUM\_TESTS) {

r\_wl[i+1].push\_back(r[i]);

}

q1.enqueueTask(krnl\_mmult, &x\_wl[i+1], &x[i+1]);

r\_wl[i+1].push\_back(x[i+1]);

krnl\_mmult.setArg(0, a\_buf[(i+2)%NUM\_MAT]);

krnl\_mmult.setArg(1, b\_buf[(i+2)%NUM\_MAT]);

krnl\_mmult.setArg(2, c\_buf[(i+2)%NUM\_MAT]);

q2.enqueueMigrateMemObjects({a\_buf[(i+2)%NUM\_MAT], b\_buf[(i+2)%NUM\_MAT]}, 0, &w\_wl[i+2], &w[i+2]);

x\_wl[i+2].push\_back(w[i+2]);

if (i + 3 < NUM\_TESTS) {

w\_wl[i+3].push\_back(w[i+2]);

}

q1.enqueueMigrateMemObjects({c\_buf[(i+1)%NUM\_MAT]}, CL\_MIGRATE\_MEM\_OBJECT\_HOST, &r\_wl[i+1], &r[i+1]);

if (i + 5 < NUM\_TESTS) {

w\_wl[i+5].push\_back(r[i+1]);

}

if (i + 2 < NUM\_TESTS) {

r\_wl[i+2].push\_back(r[i+1]);

}

q2.enqueueTask(krnl\_mmult, &x\_wl[i+2], &x[i+2]);

r\_wl[i+2].push\_back(x[i+2]);

krnl\_mmult.setArg(0, a\_buf[(i+3)%NUM\_MAT]);

krnl\_mmult.setArg(1, b\_buf[(i+3)%NUM\_MAT]);

krnl\_mmult.setArg(2, c\_buf[(i+3)%NUM\_MAT]);

q3.enqueueMigrateMemObjects({a\_buf[(i+3)%NUM\_MAT], b\_buf[(i+3)%NUM\_MAT]}, 0, &w\_wl[i+3], &w[i+3]);

x\_wl[i+3].push\_back(w[i+3]);

if (i + 4 < NUM\_TESTS) {

w\_wl[i+4].push\_back(w[i+3]);

}

q2.enqueueMigrateMemObjects({c\_buf[(i+2)%NUM\_MAT]}, CL\_MIGRATE\_MEM\_OBJECT\_HOST, &r\_wl[i+2], &r[i+2]);

if (i + 6 < NUM\_TESTS) {

w\_wl[i+6].push\_back(r[i+2]);

}

if (i + 3 < NUM\_TESTS) {

r\_wl[i+3].push\_back(r[i+2]);

}

q3.enqueueTask(krnl\_mmult, &x\_wl[i+3], &x[i+3]);

r\_wl[i+3].push\_back(x[i+3]);

q3.enqueueMigrateMemObjects({c\_buf[(i+3)%NUM\_MAT]}, CL\_MIGRATE\_MEM\_OBJECT\_HOST, &r\_wl[i+3], &r[i+3]);

if (i + 7 < NUM\_TESTS) {

w\_wl[i+7].push\_back(r[i+3]);

}

if (i + 4 < NUM\_TESTS) {

r\_wl[i+4].push\_back(r[i+3]);

}

}

q.finish();

q1.finish();

q2.finish();

q3.finish();

// ------------------------------------------------------------------------------------

// Step 4: Release Allocated Resources

// ------------------------------------------------------------------------------------

timer2.add("Writing output to output\_fpga.bin");

FILE \*file = fopen("output\_fpga.bin", "wb");

for (int i = 0; i < NUM\_MAT; i++)

{

fwrite(c[i], 1, bytes\_per\_iteration, file);

}

fclose(file);

delete[] fileBuf;

timer2.finish();

std::cout << "--------------- Key execution times ---------------"

<< std::endl;

timer2.print();

timer1.finish();

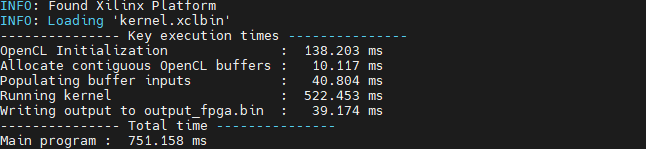
std::cout << "--------------- Total time ---------------"

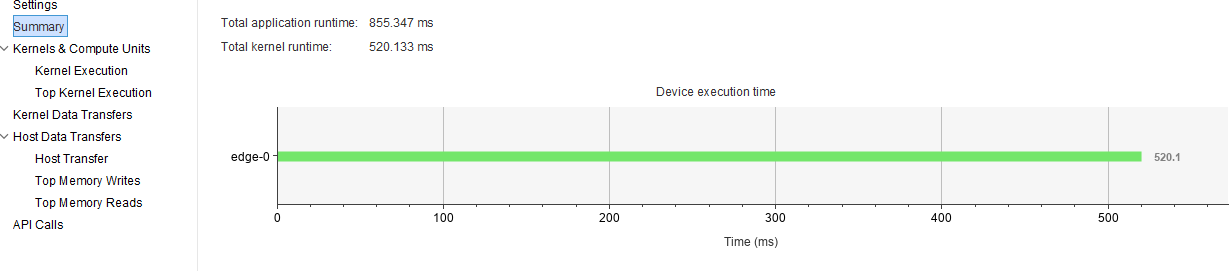
<< std::endl;

timer1.print();

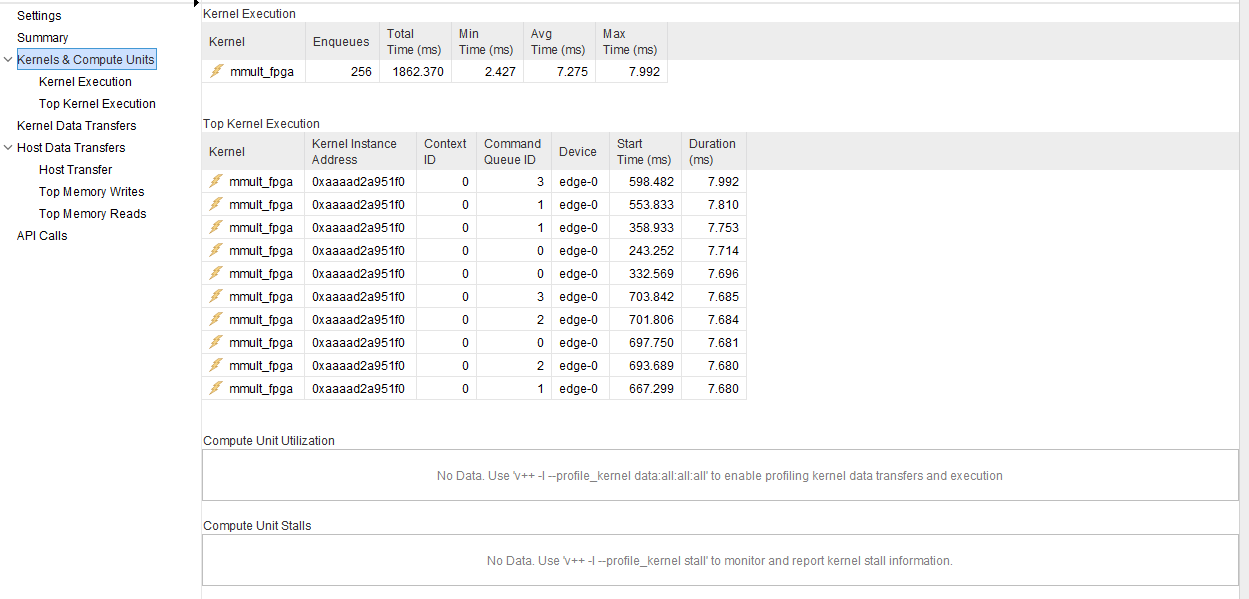
return 0;

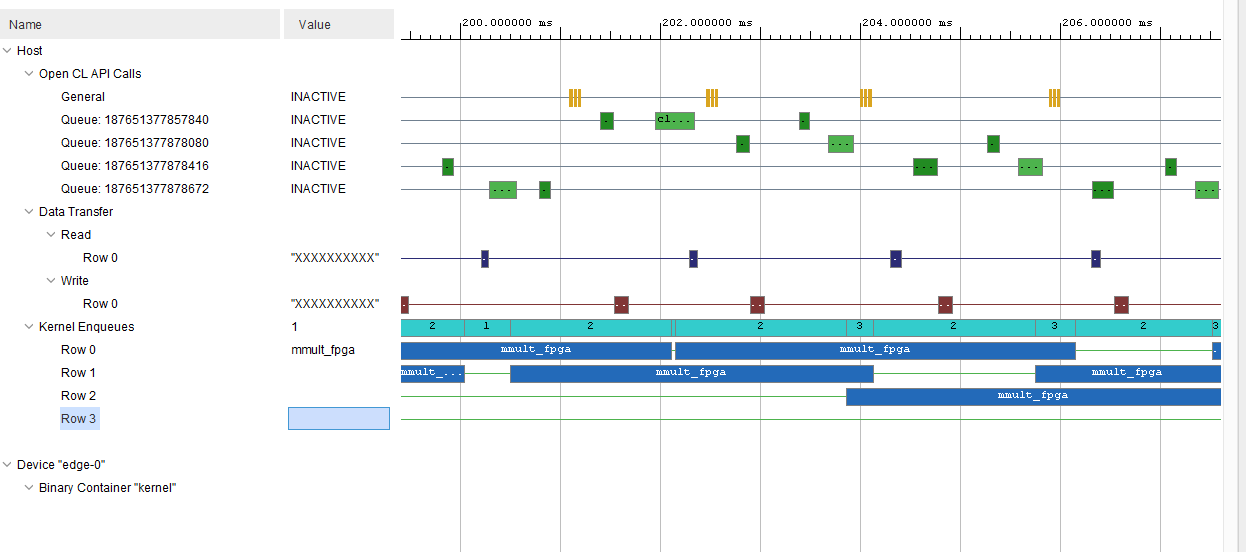
}

The latency is 751.158 ms.

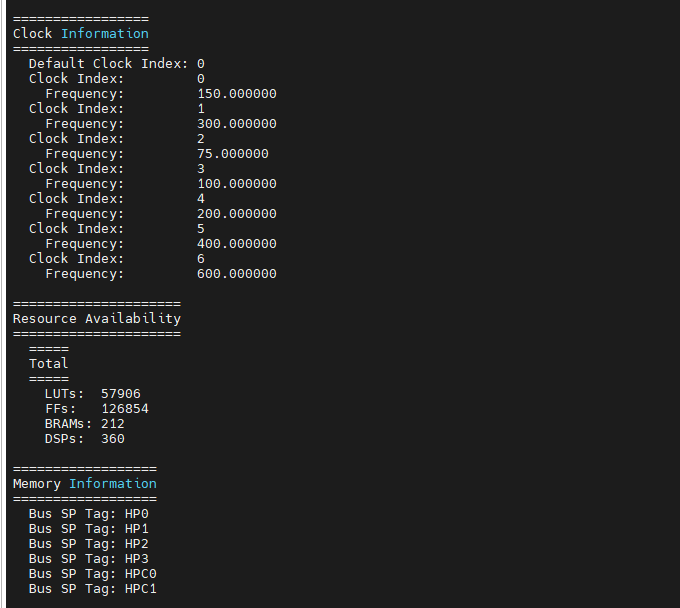
The application latency is 855.347 ms.

The kernel runtime is 520.133 ms.

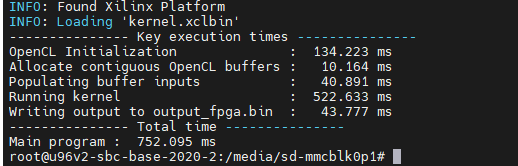
The mmult\_fpga is 1862.370 ms



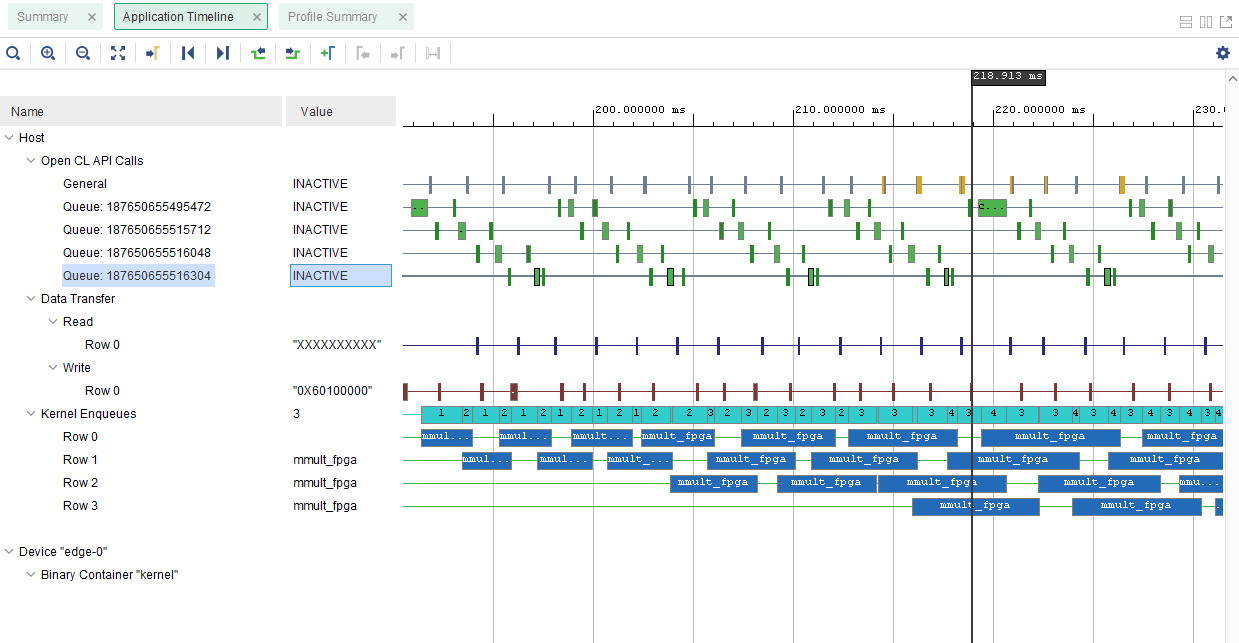
5.4



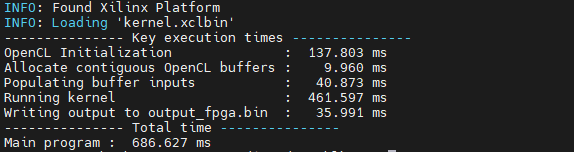
5.5

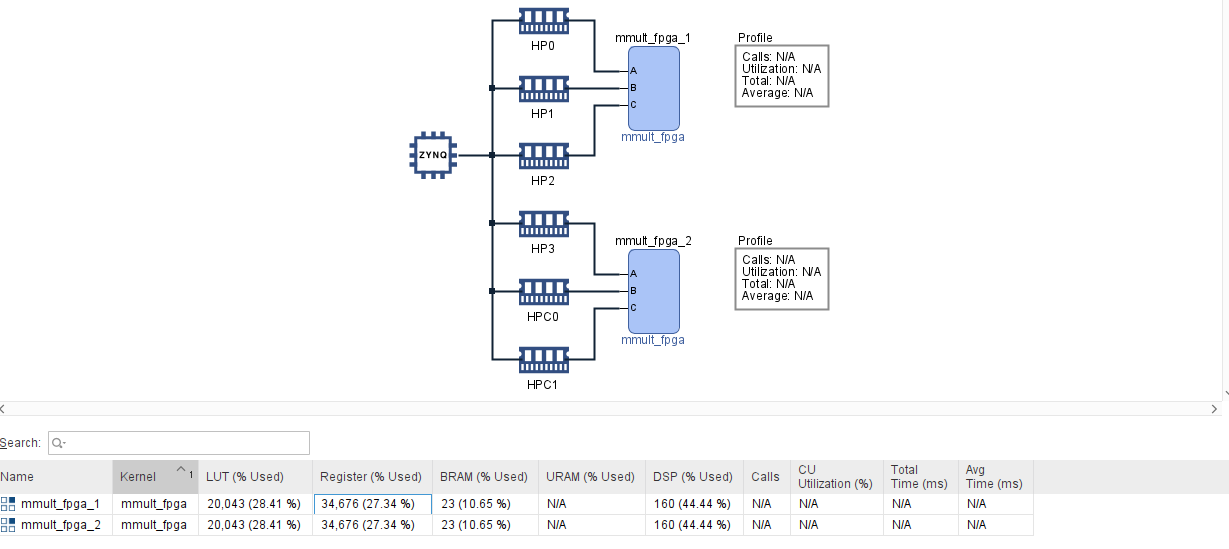


The latency is 752.095 ms.



5.6

The latency is 686.627 ms

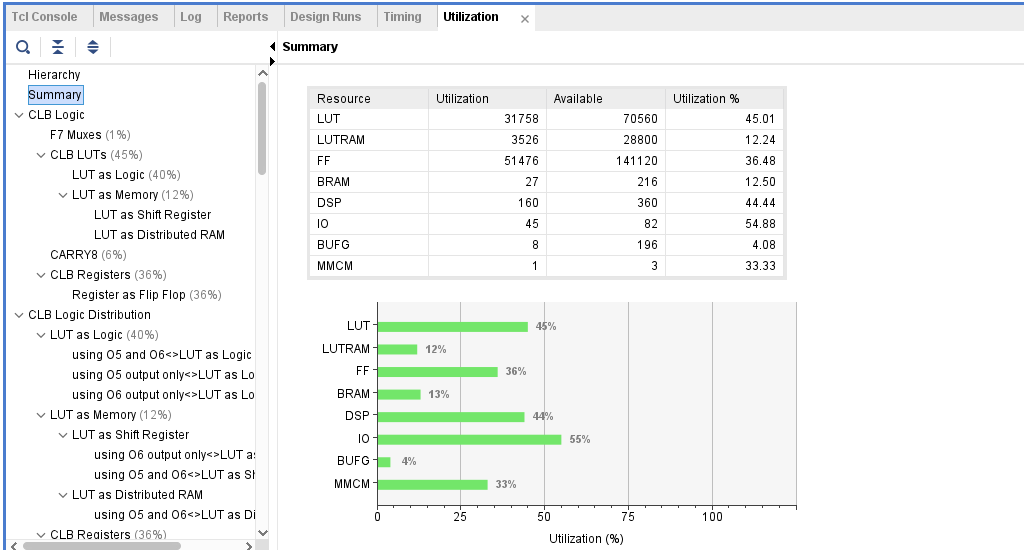


5.7

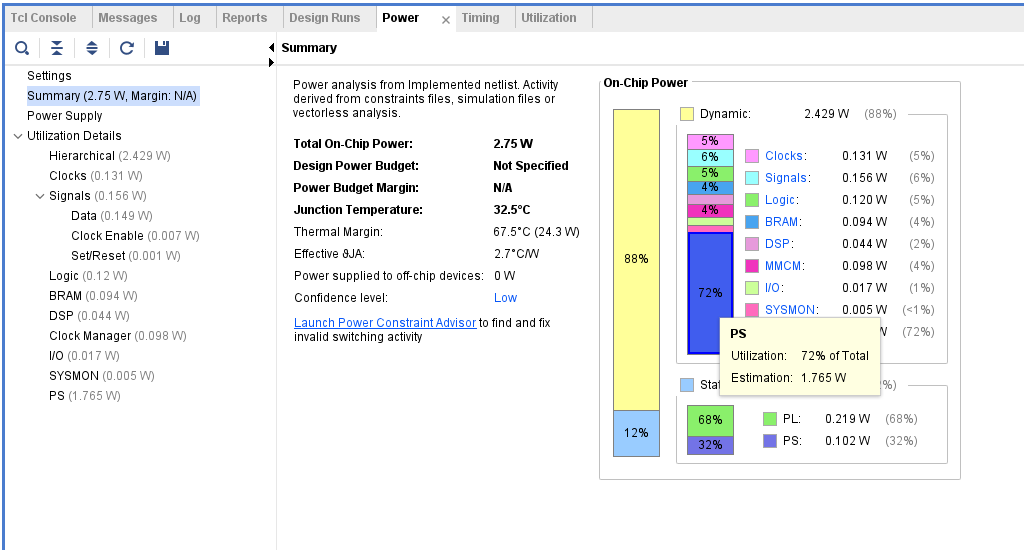
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Cpu | fpga | Host.cpp | Bank | Multicore |
| Latency | 5447.585 | 1609.944 | 751.158 | 752.095 | 686.627 |
| Speed up | 1 | 3.38 | 7.25 | 7.25 | 7.93 |

Q2

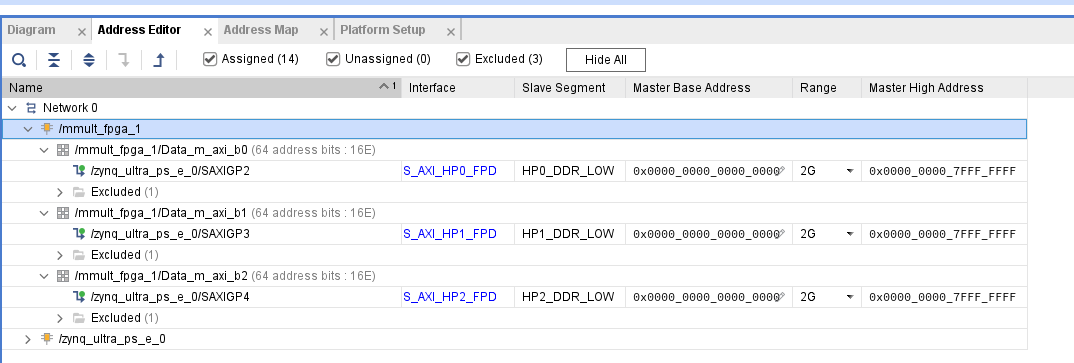
1.



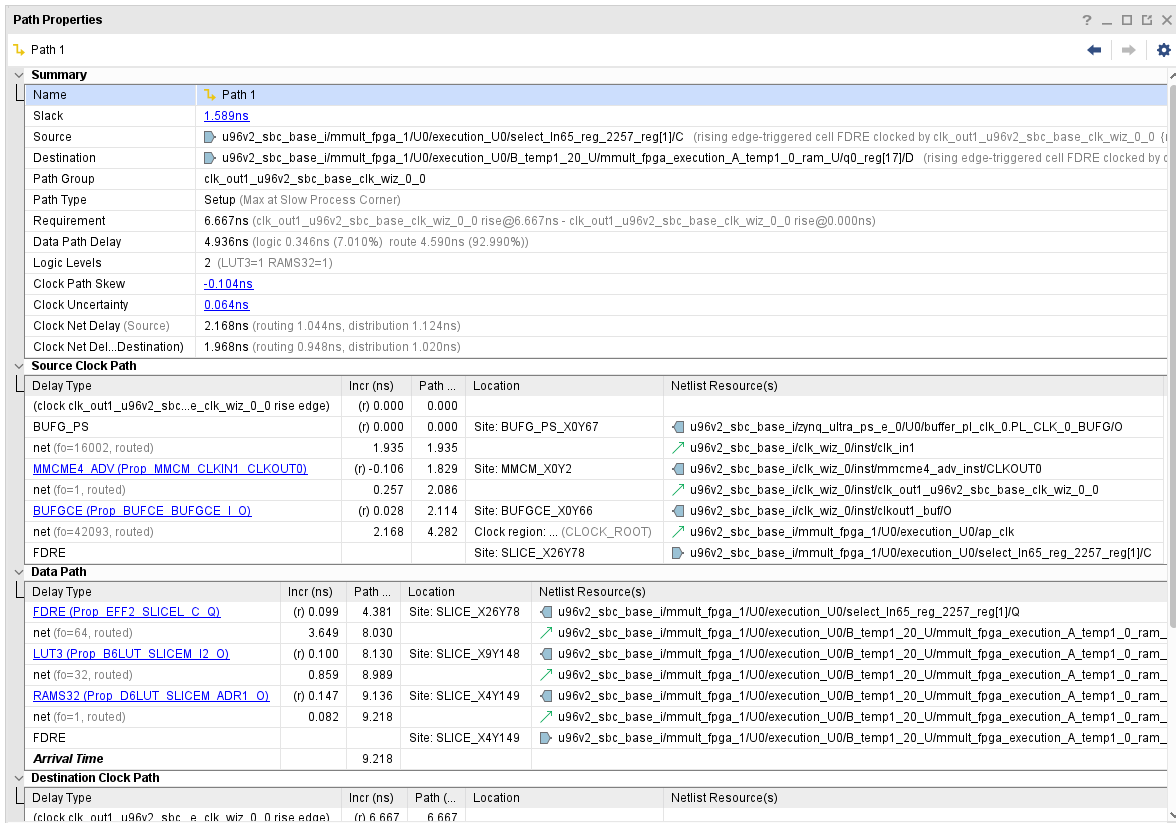
2.



3.

memory region is from 0x0000\_0000\_0000\_0000 to 0x0000\_0000\_7FFF\_FFFF

4.



图示, 示意图

描述已自动生成begin is the select\_In65\_reg2257\_reg[1]

end is the q0\_reg[17]

5.

图表

描述已自动生成

图形用户界面

描述已自动生成

6.

图形用户界面

描述已自动生成

