Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών Lab 1 Report

Δάιος Γρηγόριος - **ΑΕΜ** 10334
Παπαδάκης Κωνσταντίνος Φώτιος - **ΑΕΜ** 10371

December 7, 2025

Contents

Chapter 1

Exercise 1

1.1 Core concepts

- LUTs: **Look Up Tables** are programmable truth tables inside an FPGA that implements logic operations.
- DSPs: **Digital Signal Processing** units are specialized hardware whose purpose is to do mathematical operations (mainly multiplication and division) really fast.
- BRAM: **Block Random Access Memory** is internal temporary memory smaller but faster than DRAM.
- DRAM: **Dynamic Random Access Memory** is external temporary memory slower but bigger than BRAM.
- FFs: **Flip-Flips** are simple elements that store 1 bit.

1.2 Core concepts applied on our code

In our implementation we create two instances of the A, B and C 2D arrays. One which holds the data stored inside DRAM and one which holds the data inside BRAM. The reason why we decided to split the data is so that we could implement array_partition later on, which needs our matrices to reside in BRAM. When it comes to FFs and LUTs they are directly proportional to the size of our matrices because enlarging the matrices' dimensions leads to more hardware needed to translate the software. Additionally, it turned out that since our math operations are simple subtractions between low bit unsigned integers, DSPs were not utilized.

1.3 Interface

1.3.1 m_axi

Connects

```
#pragma HLS INTERFACE m_axi port=A depth=1024 offset=slave
#pragma HLS INTERFACE m_axi port=B depth=1024 offset=slave
#pragma HLS INTERFACE m_axi port=C depth=1024 offset=slave
```

1.3.2 s_axilite

Connects

```
#pragma HLS INTERFACE s_axilite port=A bundle=control
#pragma HLS INTERFACE s_axilite port=B bundle=control
#pragma HLS INTERFACE s_axilite port=C bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control
```

1.4 Pragmas

1.4.1 bind_storage

We bind our local variants of the A, B, C matrices as bram using 2 ports.

```
#pragma HLS bind_storage variable=A_local type=ram_2p impl=bram
#pragma HLS bind_storage variable=B_local type=ram_2p impl=bram
#pragma HLS bind_storage variable=C_local type=ram_2p impl=bram
```

1.4.2 array_partition

We partition the second dimension of our matrices so that we can increase our data throughput. Since we use cyclic partitioning the data is split like so:

Bank 1	Bank 2	Bank 3	Bank 4
1st element	2nd element	3rd element	4th element
5th element	6th element	7th element	8th element
...	...	...	...

This permits us to fetch x elements per cycle where x the number of banks. Here follows

```
#pragma HLS array_partition variable=A_local cyclic factor=8 dim=2
#pragma HLS array_partition variable=B_local cyclic factor=8 dim=2
#pragma HLS array_partition variable=C_local cyclic factor=8 dim=2
```

1.4.3 unroll

Loop unrolling takes a number of individual loops from a for, depending on the unroll factor, and effectively stacks them together allowing their parallel execution. When a for loop is not unrolled the next loop's operations can't begin , and thus can't be parallelized, since we are not yet sure of the branch destination.

Since, as we were taught, pipelining flattens the loops automatically, we avoided using loop unrolling but this is the notation we used on our experiments:

```
#pragma HLS unroll factor=4
```

1.4.4 pipeline

Pipelining allows identical assembly operations to be parallelized by executing them with 1 clock cycle time delay. This is possible due to the fact that we utilize different hardware on each of an operation's cycles. Expanding on that notion we can also pipeline different operations as long as they don't utilize the same hardware at the same cycle. Here we attempt to pipeline our code every 1 cycle:

```
#pragma HLS pipeline II=1
```

Chapter 2

Exercise 2

Name/Loop	Latency (cycles)	Latency (ns)	Interval	Pipelined	BRAM	DSP	FF	LUT	URAM
Top function	262295	2.623E6	262296	no	12	0	46751	12036	0
Loop 1	131145	1.311E6	131145	no	0	0	40567	1764	0
Loop 2	65538	6.550E5	65538	no	0	0	55	255	0
Loop 3	65539	6.550E5	65539	no	0	0	1061	1055	0

For an array of 256×256 we have the following results:

Estimated clock period	7.300ns
Worst case latency	262295 cycles
Number of DSP48E used	0
Number of BRAMs used	12
Number of FFs used	46751
Number of LUTs used	12036

Chapter 3

Exercise 3

Total Execution Time	2,675,185.0 ns
Min Latency	267446 cycles
Avg Latency	267446 cycles
Max Latency	267446 cycles

Chapter 4

Exercise 4

4.1 Part 1

4.2 Part 2

The optimal, between our tested configurations, setup has the following attributes:

Parameter	Value
Estimated Clock period	0
Number of DSPs	0
Number of BRAMs	0
Number od FFs	0
Number of LUTs	0
Total Execution Time	0
Min Latency	0
Avg Latency	0
Max Latency	0