REPORT ΕΡΓΑΣΤΗΡΙΟΥ 1

Δημήτριος Διακολουκάς 10642

Γεώργιος Βαβούρας 10623

Ερώτημα 1

Αρχικά για το ερώτημα αυτό ορίσαμε τις διαστάσεις των πινάκων ως σταθερές χρησιμοποιώντας τη δήλωση #define σε ξεχωριστό header file και υπολογίσαμε τις τιμές τους ως δυνάμεις του 2 με χρήση της εντολής shift left. Δημιουργήσαμε τη συνάρτηση matrix_mul για τον πολλαπλασιασμό των πινάκων A και B, όπου κάθε στοιχείο του αποτελέσματος υπολογίζεται με το άθροισμα των γινομένων των αντίστοιχων στοιχείων από τους πίνακες εισόδου. Στη συνέχεια, υλοποιήσαμε ένα testbench σε C για να αρχικοποιήσουμε τους πίνακες με ψευδό-τυχαίες τιμές στο εύρος 0-255 και να επαληθεύσουμε τη σωστή λειτουργία της υλοποίησης μέσω σύγκρισης των αποτελεσμάτων S/W και H/W όπου η H/W υλοποίηση είναι η υλοποίηση matrix multiplication στο source ενώ η S/W είναι η ίδια αλλά ορισμένη στο testbench. Τέλος, εκτυπώσαμε τα αποτελέσματα και επιβεβαιώσαμε την επιτυχία του τεστ με το μήνυμα "Test Passed".

Ερώτημα 2

Στο ερώτημα αυτό κληθήκαμε να εφαρμόσουμε C Synthesis με default settings για lm=ln=lp=6. Παρακάτω εξηγούμε τι είναι το κάθε ζητούμενο της άσκησης και έπειτα δείχνουμε τα αποτελέσματά μας από τον κώδικα δίχως την χρήση pragmas όπως ακριβώς τον συντάξαμε για το Ερώτημα 1.

Estimated Clock Period: Είναι ο εκτιμώμενος χρονικός κύκλος του ρολογιού, ο οποίος εμφανίζεται στο synthesis report και δίνεται σε nanoseconds.

Worst Case Latency: Αναφέρεται στην καθυστέρηση για την ολοκλήρωση της λειτουργίας, εκφρασμένη σε κύκλους ρολογιού.

Number of DSP48E Used: Ο αριθμός των DSP48E μπλοκ που χρησιμοποιούνται για αριθμητικές λειτουργίες (π.χ., πολλαπλασιασμοί).

Number of BRAMs Used: Ο αριθμός των ενσωματωμένων μνημών μπλοκ (Block RAMs) που χρησιμοποιούνται.

Number of FFs Used: Ο αριθμός των flip-flops που χρησιμοποιούνται για τον σχεδιασμό.

Number of LUTs Used: Ο αριθμός των λογικών στοιχείων (Lookup Tables) που χρησιμοποιούνται.

Έτσι τα αποτελέσματα που βγάλαμε είναι τα παρακάτω και βρέθηκαν στο detailed report του C synthesis:

Estimated clock period	6.304ns
Worst case latency	131091 cycles
Number of DSP48E used	32
Number of BRAMs used	0
Number of FFs used	1491
Number of LUTs used	4420

Ερώτημα 3

Σε αυτό το ερώτημα έπρεπε να τρέξουμε C/RTL cosimulation και βεβαιωθούμε ότι η σχεδίασή μας περνάει το test επιτυχώς πράγμα που σε matrix multiplication με ψευδοτυχαίους αριθμούς συμβαίνει και στην περίπτωσή μας. Έτσι τα αποτελέσματα που βγάλαμε είναι τα παρακάτω και βρέθηκαν στο report του C/RTL cosimulation:

Total Execution Time	1311095 ns
Min latency	131089 cycles
Avg. latency	131089 cycles
Max latency	131089 cycles

Ερώτημα 4

Στο παρόν ερώτημα, στόχος μας είναι να βελτιστοποιήσουμε τον χρόνο εκτέλεσης της παραπάνω εφαρμογής πολλαπλασιασμού matrixes χρησιμοποιώντας το Vivado High-Level Synthesis (HLS). Για την επίτευξη αυτού του στόχου, εφαρμόζουμε διάφορες directives όπως ARRAY_PARTITION, PIPELINE, και UNROLL. Αυτές οι directives επιτρέπουν την παράλληλη επεξεργασία δεδομένων και την αύξηση της αποδοτικότητας του κώδικα, οδηγώντας σε ταχύτερη εκτέλεση του αλγορίθμου. Μέσα από πειραματισμούς με διάφορες διαστάσεις των πινάκων, αξιολογήσαμε την επίδραση στον χρόνο εκτέλεσης και επιτύχαμε τη βέλτιστη βελτιστοποίηση.

Εξήγηση βασικών pragmas που χρησιμοποιήσαμε ή δοκιμάσαμε για να βελτιστοποιήσουμε τον source κώδικα:

1) **ARRAY_PARTITION:** Αυτά τα pragmas χρησιμοποιούνται για να χωρίσουν τους πίνακες A και B σε μικρότερα τμήματα, επιτρέποντας την παράλληλη πρόσβαση σε πολλαπλές εγγραφές ταυτόχρονα. Συγκεκριμένα ορίσαμε τις εξής παραμέτρους:

- **variable=A (variable=B αντίστοιχα)**: Ορίζει τον πίνακα που θα κατατμηθεί αναλόγως.
- **type=cyclic**: Είδος κατατμησης που επαναλαμβάνεται κυκλικά.
- **factor=64**: Καθορίζει τον αριθμό των τμημάτων.
- **dim=1 (dim=2 αντίστοιχα)**: Ορίζει τη διάσταση του πίνακα που θα κατατμηθεί (1η ή 2η διάσταση). Στην περίπτωση μας, το A κατατμήθηκε στη 2η διάσταση και το B στην 1η, επιτρέποντας έτσι γρηγορότερη πρόσβαση στα δεδομένα τους κατά τον πολλαπλασιασμό.

2) **PIPELINE**: Με αυτή την τεχνική κάθε στάδιο εκτελείται σε ξεχωριστό στάδιο του pipeline, επιτρέποντας πολλαπλές εντολές να βρίσκονται σε διαφορετικά στάδια εκτέλεσης ταυτόχρονα. Ο βρόχος διαχωρίζεται σε πολλαπλά stages. Κάθε stage επεξεργάζεται μια διαφορετική επανάληψη του βρόχου. Όταν το πρώτο στάδιο ολοκληρωθεί, περνάει τα αποτελέσματα στο επόμενο στάδιο κλπ. Εμείς λοιπόν στον κώδικα χρησιμοποιήσαμε και την έκφραση II=1 (Initiation Interval = 1) για το pipelining που πρακτικά εκφράζει την επιθυμία για μέγιστη παράλληλη εκτέλεση του βρόχου, ξεκινώντας νέα επανάληψη σε κάθε κύκλο ρολογιού.

3) **LOOP UNROLLING:** Είναι η διαδικασία κατά την οποία οι επαναλήψεις ενός βρόχου αντιγράφονται για να μειωθεί ο αριθμός των επαναλήψεων που εκτελούνται κατά τη διάρκεια της εκτέλεσης. Αντί να εκτελείται ο βρόχος πολλές φορές, οι εντολές του βρόχου αντιγράφονται πολλές φορές μέσα στο σώμα του βρόχου. Αυτό μειώνει τον αριθμό των επαναλήψεων και επιτρέπει την παράλληλη εκτέλεση πολλών εντολών.
***Για παράδειγμα*:**
➔ Απλή υλοποίηση

```
for (int i = 0; i < 4; i++) {
  C[i] = A[i] + B[i];
}
```

➔ Με unrolling factor 4

```
C[0] = A[0] + B[0];
C[1] = A[1] + B[1];
C[2] = A[2] + B[2];
C[3] = A[3] + B[3];
```

4) **LOOP_TRIPCOUNT:** Αυτό το pragma χρησιμοποιείται για να δείξει στον compiler τον αριθμό των επαναλήψεων που θα εκτελεστούν μέσα στον βρόχο. Αυτό βοηθά στο να γίνουν καλύτερες βελτιστοποιήσεις.

Παρακάτω παρατίθενται τα αποτελέσματά μας για τα υποερωτήματα του θέματος αυτού.

i) Σε αυτό το υποερώτημα παρατηρούμε ότι με την χρήση διαφορετικών μεταβλητών In και Ip όσο εκείνα μειώνονται μειώνεται η πολυπλοκότητα του κώδικα και άρα μειώνονται τα Resources και οι κύκλοι ρολογιού αντίθετα όσο αυξάνονται άρα και η πολυπλοκότητα του κώδικα (περισσότερες επαναλήψεις loops) αυξάνονται τα Resources και οι κύκλοι ρολογιού.

Performance Estimates

Timing

Clock		solution8_no-unroll_n4_p4	solution6_no_unroll	solution7_no-unroll_n7_p7
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	4.236 ns	6.304 ns	6.728 ns

Latency

		solution8_no-unroll_n4_p4	solution6_no_unroll	solution7_no-unroll_n7_p7
Latency (cycles)	min	1029	4101	8197
	max	1029	4101	8197
Latency (absolute)	min	10.290 us	41.010 us	81.970 us
	max	10.290 us	41.010 us	81.970 us
Interval (cycles)	min	1030	4102	8198
	max	1030	4102	8198

Utilization Estimates

	solution8_no-unroll_n4_p4	solution6_no_unroll	solution7_no-unroll_n7_p7
BRAM_18K	0	0	0
DSP	8	32	64
FF	130	140	171
LUT	702	2293	4438
URAM	0	0	0

ii) Παρακάτω παρατίθενται και τα αποτελέσματα μας στην περίπτωση που χρησιμοποιούμε Im = In = Ip = 6 έχοντας χρησιμοποιήσει τα παρακάτω directives όπως φαίνονται στον κώδικα. Έπειτα από πειραματισμό προέκυψε ότι τα βέλτιστα pragmas για εμάς ήταν τα εξής:

```
void matrix_mul(uint8_t A[m][n], uint8_t B[n][p], uint32_t C[m][p]) {
    #pragma HLS ARRAY_PARTITION variable=A type=cyclic factor=64 dim=2
    #pragma HLS ARRAY_PARTITION variable=B type=cyclic factor=64 dim=1

    const int c_size1 = m;
    const int c_size2 = p;
    const int c_size3 = n;

    int i;
    int j;
    int k;
    uint32_t sum;
    for (i = 0; i < m; i++) {
    #pragma HLS LOOP_TRIPCOUNT min=c_size1 max=c_size1
        for (j = 0; j < p; j++) {
        #pragma HLS LOOP_TRIPCOUNT min=c_size2 max=c_size2
        #pragma HLS PIPELINE
            sum = 0;
            for (k = 0; k < n; k++) {
            #pragma HLS LOOP_TRIPCOUNT min=c_size2 max=c_size3
            //#pragma HLS UNROLL factor=16
                sum += (uint32_t)A[i][k] * (uint32_t)B[k][j];
            }
            C[i][j] = sum;
        }
    }
}
```

Παρατηρήσαμε ότι η χρήση LOOP UNROLL αύξανε και τους κύκλους ρολογιού (διπλασιασμός περίπου), τα Resources (FF, LUTS) αλλά και τον χρόνο speed-up ταυτόχρονα. Αυτό μπορεί να συμβαίνει καθώς συνήθως κάνουμε είτε unroll είτε pipeline και ο συνδυασμός τους δεν μας βοήθησε στην επιτάχυνση αλλά και στους πόρους.

Estimated clock period	6.304 ns
Number of DSP48E used	32
Number of BRAMs used	0
Number of FFs used	140
Number of LUTs used	2293
Total Execution Time	41195 ns
Min latency	4099 cycles
Avg. latency	4099 cycles
Max latency	4099 cycles

iii) Σε αυτό το υποερώτημα κληθήκαμε να βρούμε την βέλτιστη hardware υλοποίηση ώστε να πετύχουμε το μεγαλύτερο speed-up συγκριτικά με την αρχική υλοποίηση χωρίς directives.

Και τα αποτελέσματα ήταν τα εξής:

- Speedup = (αρχική σχεδίαση σε hardware (χωρίς directives)) / (χρόνος βέλτιστης hardware υλοποίησης)
- Speedup = 1311095 / 41195
- Speedup = 31.8265566209