Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών – Αναφορά 1ου εργαστηρίου

Δημήτριος Ορέστης Βαγενάς, 10595

Αγγελική Στρατάκη, 10523

Ερώτημα 1:

Για το πρώτο εργαστήριο χρησιμοποιήσαμε το εργαλείο Vivado HLS με σκοπό την διαδικασία επιτάχυνσης του αλγορίθμου για πολλαπλασιασμό πινάκων MATRIX_MUL, σε γλώσσα C. Σε ένα testbench έχουμε αρχικοποιήσει τους πίνακες A και B με τυχαία ψευδό-τυχαίες τιμές στο εύρος 0-255 για την S/W λύση φτιάξαμε την συνάρτηση matrix_multiply_sw και την H/W λύση την συνάρτηση matrix_multiply_hw, με σκοπό την διασφάλιση της σωστής εκτέλεσης της hardware accelerator λειτουργικότητας, που θα τροποποιήσουμε.

- *Testbench:*

```
#include <iostream>
#include <cstdlib>  // For rand() and srand()
#include <ctime>    // For time()
#include <iomanip>  // For std::setw

#define LM 6
#define LN 6
#define LP 6

#define M (1 << LM)
#define N (1 << LN)
#define P (1 << LP)

void matrix_multiply_hw(const uint8_t A[N][M], const uint8_t B[M][P], uint32_t C[N][P]);

// Software matrix multiplication for verification
void matrix_multiply_sw(const uint8_t A[N][M], const uint8_t B[M][P], uint32_t C[N][P]) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < P; ++j) {
            C[i][j] = 0;  // Initialize element C[i][j]
            for (int k = 0; k < M; ++k) {
                C[i][j] += (uint32_t)A[i][k] * (uint32_t)B[k][j];
            }
        }
    }
}
```

```
int main() {
    // Initialize random seed
    std::srand(std::time(0));

    // Declare matrices A, B, and C for both HW and SW computation
    uint8_t A[N][M];
    uint8_t B[M][P];
    uint32_t C_hw[N][P];
    uint32_t C_sw[N][P];

    // Initialize matrices A and B with random values in the range 0-255
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            A[i][j] = rand() % 256;
        }
    }
    for (int i = 0; i < M; ++i) {
        for (int j = 0; j < P; ++j) {
            B[i][j] = rand() % 256;
        }
    }

    // Perform matrix multiplication using both hardware and software functions
    matrix_multiply_hw(A, B, C_hw);
    matrix_multiply_sw(A, B, C_sw);
```

```
    // Verify the results
    bool success = true;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < P; ++j) {
            if (C_hw[i][j] != C_sw[i][j]) {
                success = false;
                break;
            }
        }
        if (!success) break;
    }

    // Print matrices and results
    std::cout << "Matrix A:" << std::endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            std::cout << std::setw(4) << static_cast<int>(A[i][j]) << " ";
        }
        std::cout << std::endl;
    }

    std::cout << "Matrix B:" << std::endl;
    for (int i = 0; i < M; ++i) {
        for (int j = 0; j < P; ++j) {
            std::cout << std::setw(4) << static_cast<int>(B[i][j]) << " ";
```

```
    std::cout << "Matrix C (Hardware Result):" << std::endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < P; ++j) {
            std::cout << std::setw(6) << C_hw[i][j] << " ";
        }
        std::cout << std::endl;
    }

    std::cout << "Matrix C (Software Result):" << std::endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < P; ++j) {
            std::cout << std::setw(6) << C_sw[i][j] << " ";
        }
        std::cout << std::endl;
    }

    // Print test result
    if (success) {
        std::cout << "Test Passed" << std::endl;
    } else {
        std::cout << "Test Failed" << std::endl;
    }

    return 0;
}
```

Έχουμε ορίσει lm=ln=lp=6 για το επόμενο ερώτημα.

Ερώτημα 2:

- Η συνάρτηση matrix_multiply_hw ορίζεται αρχικά ως εξής:

```c
#include <stdint.h>

#define LM 6    // Define lm (for example, lm = 6 -> m = 2^3 = 64)
#define LN 6    // Define ln (for example, ln = 6 -> n = 2^4 = 64)
#define LP 6    // Define lp (for example, lp = 6 -> p = 2^5 = 64)

#define M (1 << LM)  // m = 2^lm
#define N (1 << LN)  // n = 2^ln
#define P (1 << LP)  // p = 2^lp

void matrix_multiply_hw(
    const uint8_t A[N][M],
    const uint8_t B[M][P],
    uint32_t C[N][P]
) {
    // Perform matrix multiplication: C = A * B
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < P; ++j) {
            C[i][j] = 0;  // Initialize element C[i][j]
            for (int k = 0; k < M; ++k) {
                C[i][j] += (uint32_t)A[i][k] * (uint32_t)B[k][j];
            }
        }
    }
}
```

Κάνοντας σύνθεση τη παραπάνω σχεδίαση (C Synthesis) με default settings έχουμε:

Target	Estimated	Uncertainty	
10.00 ns	6.304 ns	2.70 ns	

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
131091	131091	1.311 ms	1.311 ms	131092	131092	no

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	32	-	-	-
Expression	-	-	0	2080	-
FIFO	-	-	-	-	-
Instance	-	0	0	1280	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	1060	-
Register	-	-	1491	-	-
Total	0	32	1491	4420	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	~0	~0	~0	0
Utilization SLR (%)	0	1	~0	1	0

Ερώτημα 3:

Για το ερώτημα αυτό τρέξαμε C/RTL cosimulation

```
$finish called at time : 1311095 ns :
977397 1013998
Test Passed
INFO: [SIM 1]
```

Τις πληροφορίες για το latency τις βλέπουμε στο προηγούμενο ερώτημα.

Ερώτημα 4:

Για την βελτιστοποίηση της matrix_multiply_hw χρησιμοποιήσαμε την εντολή #pragma ARRAY_PARTITION pragmas διαμερίζοντας τους πίνακες A και B κατά συγκεκριμένες διαστάσεις. Ο πλήρης διαμερισμός χωρίζει αυτούς τους πίνακες σε μεμονωμένα registers, κάτι που επιταχύνει τους υπολογισμούς επιτρέποντας παραλληλία. Επιπλέον, την εντολή #pragma HLS ARRAY_PARTITION variable=C complete dim=2: διαμερίζει μερικώς τον πίνακα C για να επιτρέψει παράλληλη πρόσβαση στις στήλες. Τέλος, την εντολή #pragma HLS UNROLL μέσα στον εσωτερικό βρόχο ξετυλίγει τον βρόχο για το k, αυξάνοντας την παραλληλία, καθώς επιτρέπει ταυτόχρονους υπολογισμούς για στοιχεία των A και B.

```c
#include <stdint.h>

#define LM 6    // Define lm (for example, lm = 6 -> m = 2^6 = 64)
#define LN 6    // Define ln (for example, ln = 6 -> n = 2^6 = 64)
#define LP 6    // Define lp (for example, lp = 6 -> p = 2^6 = 64)

#define M (1 << LM)  // m = 2^lm
#define N (1 << LN)  // n = 2^ln
#define P (1 << LP)  // p = 2^lp

void matrix_multiply_hw(
    const uint8_t A[N][M],
    const uint8_t B[M][P],
    uint32_t C[N][P]
) {
    #pragma HLS ARRAY_PARTITION variable=A complete dim=2
    #pragma HLS ARRAY_PARTITION variable=B complete dim=1
    #pragma HLS ARRAY_PARTITION variable=C complete dim=2

    // Perform matrix multiplication: C = A * B
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < P; ++j) {
            C[i][j] = 0;  // Initialize element C[i][j]
            for (int k = 0; k < M; ++k) {
                #pragma HLS UNROLL
                C[i][j] += (uint32_t)A[i][k] * (uint32_t)B[k][j];
            }
        }
    }
}
```

i)Η αύξηση του In ή του Ip αυξάνει την ποσότητα των πράξεων που απαιτούνται, και επομένως αυξάνεται ο συνολικός χρόνος, καθώς εκτελούνται περισσότερες πράξεις για κάθε στοιχείο του πίνακα αποτελέσματος C. Η διατήρηση του Im σταθερού μειώνει τις αλλαγές στην αναλογία των στοιχείων που μπορούν να επεξεργαστούν παράλληλα, ενώ η αλλαγή στις διαστάσεις των άλλων δύο πινάκων επηρεάζει τον συνολικό αριθμό των στοιχείων προς υπολογισμό.

ii) Για την βέλτιστη λύση έχουμε:

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	-	2048	33367	133279	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	19615	-
Register	-	-	32803	-	-
Total	0	2048	66170	152894	0
Available	4320	6840	2364480	1182240	960
Available SLR	1440	2280	788160	394080	320
Utilization (%)	0	29	2	12	0
Utilization SLR (%)	0	89	8	38	0

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	5.358 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
102	102	1.020 us	1.020 us	103	103	no

iii) Για την επιτάχυνση έχουμε:

```
$finish called at time : 1205 ns
```