

Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών

*28 Νοεμβρίου 2025
Εργαστήριο 1^ο*

*Ομάδα 1
Καμπάνης Σταυριανός 9141
Παπαδόπουλος Παναγιώτης 10697*

1. Εισαγωγή

Σκοπός του παρόντος εργαστηρίου ήταν η εξοικείωση με τη ροή σχεδίασης (design flow) του εργαλείου Vitis HLS της Xilinx. Η διαδικασία περιελάμβανε τη δημιουργία πυρήνων (kernels) χρησιμοποιώντας γλώσσα υψηλού επιπέδου (C/C++), την προσομοίωση (C Simulation), τη σύνθεση (C Synthesis) και την επαλήθευση μέσω συν-προσομοίωσης (C/RTL Cosimulation). Αρχικά μελετήθηκε ένα απλό κύκλωμα ALU και στη συνέχεια σχεδιάστηκε και βελτιστοποιήθηκε ένας επιταχυντής επεξεργασίας εικόνας.

2. Μέρος A: Εισαγωγή στο Vitis HLS (simpleALU)

Στο πρώτο μέρος, υλοποιήθηκε η συνάρτηση `simpleALU`, η οποία εκτελεί τις τέσσερις βασικές αριθμητικές πράξεις (+, -, *, /) βάσει ενός σήματος ελέγχου `op`.

2.1 Αποτελέσματα C Synthesis

Μετά την εκτέλεση της σύνθεσης, παρατηρήθηκαν τα εξής χαρακτηριστικά στο Synthesis Report:

- **Στόχος Ρολογιού (Target Clock):** 10ns
- **Εκτιμώμενο Ρολόι (Estimated Clock):** 3.170 ns
- **Latency (Κύκλοι):**
 - Ελάχιστο (Min): 1 κύκλος
 - Μέγιστο (Max): 35 κύκλοι (Οφείλεται στην πράξη της διαίρεσης)
- **Interval:**
 - Min: 10ns
 - Max: 0.35μs

Παρατήρηση για το Pipelining: Όπως φαίνεται από το Interval, η σχεδίαση δεν είναι pipelined, διότι σε ορισμένες περιπτώσεις φαίνεται ότι χρειάζεται πάνω από ένας κύκλος ρολογιού για να ξεκινήσει η εκτέλεση της επόμενης εντολής.

2.2 Εκτίμηση Πόρων (Resource Utilization)

Η αρχική εκτίμηση χρήσης πόρων για το FPGA Alveo U200 είναι:

Πόρος	Χρήση
BRAM	0
DSP	3
FF	430
LUT	528

3. Μέρος B: Σχεδίαση H/W Accelerator (IMAGE_DIFF_POSTERIZE)

Στο δεύτερο μέρος, σχεδιάστηκε ο επιταχυντής IMAGE_DIFF_POSTERIZE. Ο επιταχυντής δέχεται δύο εικόνες, υπολογίζει την απόλυτη διαφορά τους και εφαρμόζει κατώφλια (thresholds) T1=32 και T2=96 για να παράξει μια posterized εικόνα εξόδου.

Ερώτημα 1: Κώδικας και Testbench

Στην πλατφόρμα του e-Learning έχουν αναρτηθεί ο πηγαίος κώδικας (Source Code) για τον επιταχυντή IMAGE_DIFF_POSTERIZE καθώς και το αρχείο ελέγχου (Testbench) που αναπτύχθηκαν στο Vitis HLS.

Αρχείο Source: image_diff_posterize.c: Ο κώδικας υλοποιεί την αφαίρεση, την απόλυτη τιμή και την εφαρμογή των κατωφλίων (thresholding). Έχουν συμπεριληφθεί οι απαραίτητες οδηγίες INTERFACE για τη διασύνδεση με το σύστημα (AXI4-Master για τη μνήμη και AXI4-Lite για τον έλεγχο).

Αρχείο Testbench: image_diff_posterize_tb.c: Το testbench αρχικοποιεί δύο εικόνες (με μοτίβα gradient και τυχαίες τιμές), εκτελεί τη software reference υλοποίηση και την hardware υλοποίηση, και συγκρίνει τα αποτελέσματα για επαλήθευση.

Σύμφωνα με τις προδιαγραφές της άσκησης, ο κώδικας υλοποιεί τον επιταχυντή υλικού IMAGE_DIFF_POSTERIZE. Συνοπτικά συμβαίνουν τα εξής:

1. **Διεπαφή (I/O):** Η συνάρτηση δέχεται ως είσοδο δύο πίνακες εικόνας 8-bit (A και B) και παράγει έναν πίνακα εξόδου (C) ίδιων διαστάσεων (HEIGHT x WIDTH), χρησιμοποιώντας το πρωτόκολλο AXI για την επικοινωνία με τη μνήμη.
2. **Επεξεργασία Pixel:** Σαρώνει τις εικόνες pixel προς pixel και υπολογίζει την απόλυτη διαφορά φωτεινότητας μεταξύ τους: $D = |A[i][j] - B[i][j]|$.
3. **Posterization:** Εφαρμόζει τη μη-γραμμική μετασχημάτιση βάσει των κατωφλίων T1=32 και T2=96:
 - ο Για μικρές διαφορές ($D < 32$), το pixel εξόδου γίνεται **μαύρο (0)**.
 - ο Για μεσαίες διαφορές ($32 \leq D < 96$), το pixel εξόδου γίνεται **γκρι (128)**.
 - ο Για μεγάλες διαφορές ($D \geq 96$), το pixel εξόδου γίνεται **λευκό (255)**.

Ερώτημα 2: Αρχική Σύνθεση (Baseline Synthesis)

Χωρίς τη χρήση directives, και για μέγεθος εικόνας **HEIGHT = [256]**, **WIDTH = [256]**, τα αποτελέσματα της σύνθεσης είναι:

Estimated clock period: 7.30 ns

Worst case latency: 0.656 ms / 65554 cycles

Number of DSP48E used: 0

Number of BRAMs used: 0

Number of FFs used: 2458

Number of LUTs used: 3565

Σε αυτό το στάδιο προστέθηκαν μόνο pragmas που αφορούν το interface και default ρυθμίσεις. Είναι εμφανές ότι οι αριθμοί των κύκλων είναι πολύ κοντά στον αριθμό πίξελ που επεξεργάστηκαν ($256 \times 256 = 65536$) και γι' αυτόν τον λόγο υποψιαζόμαστε ότι συμβαίνει Pipelining αυτόματα μέσω του εργαλείου. Πράγματι, μόλις προστεθεί το pragma *HLS PIPELINE off*, τα αποτελέσματα έχουν ως εξής:

Estimated clock period: 7.30 ns

Worst case latency: 2.627 ms / 262669 cycles

Number of DSP48E used: 0

Number of BRAMs used: 0

Number of FFs used: 2472

Number of LUTs used: 3417

Εδώ φαίνεται ότι το κάθε πίξελ επεξεργάζεται για 4 κύκλους ρολογιού περίπου.

Ερώτημα 3: C/RTL Cosimulation

Η επαλήθευση της σχεδίασης μέσω Cosimulation (για τις ίδιες διαστάσεις εικόνας) έδωσε τα παρακάτω χρονικά αποτελέσματα:

Total Execution Time: Latency (Cycles) x Clock Period (s) = $65554 \times 10 \times 10^{-9} = 0.655\text{ms}$

Min latency: 65578 cycles

Avg. latency: 65578 cycles

Max latency: 65578 cycles

Η σχεδίαση πέρασε επιτυχώς το testbench ("Test Passed"). Προστίθενται τα αποτελέσματα με απενεργοποιημένο το pipelining.

Total Execution Time: Latency (Cycles) x Clock Period (s) = $262693 \times 10 \times 10^{-9} = 2.63\text{ms}$

Min latency: 262693 cycles

Avg. latency: 262693 cycles

Max latency: 262693 cycles

Ερώτημα 4: Βελτιστοποίηση (Optimization & Directives)

i) Ανάλυση Κλιμάκωσης (Scaling Analysis)

Με την απλή υλοποίηση του επιταχυντή η προφανής βελτιστοποίηση όπως αναλύθηκε παραπάνω ήταν η εισαγωγή Pipeline με Initiation Interval = 1 στο εσωτερικό loop σάρωσης των στοιχείων. Με αυτή την τεχνική δεν αναμένεται η ολοκλήρωση των υπολογισμών των πράξεων κάθε φορά για να ξεκινήσει η επεξεργασία του επόμενου πίξελ, αλλά ξεκινά σε κάθε κύκλο ρολογιού η επεξεργασία νέου πίξελ.

Επιχειρήθηκε επίσης η εφαρμογή Loop Unrolling (με factor 2, 4 και 8) για περαιτέρω παραλληλοποίηση. Ωστόσο, κατά τη σύνθεση παρατηρήθηκε ότι το εργαλείο Vitis HLS δεν πραγματοποίησε αυτόματη διεύρυνση του διαύλου μνήμης (Memory Coalescing/Vectorization) για τους συγκεκριμένους pointers. Αυτό οδήγησε σε σφάλμα εξάντλησης των διαθέσιμων θυρών (Port Resource Limit), καθώς το hardware προσπάθησε να πραγματοποιήσει πολλαπλές ανεξάρτητες προσπελάσεις ταυτόχρονα, υπερβαίνοντας τις 2 φυσικές θύρες που παρέχει το interface της μνήμης.

Ακόμη, στην προκειμένη υλοποίηση δεν έχει νόημα η εφαρμογή Array Partition, εφόσον δεν χρησιμοποιείται η BRAM παρά μόνο η DRAM.

Επομένως τροποποιήθηκε η υλοποίηση για την εισαγωγή ενός BRAM buffer προκειμένου να εξεταστεί η χρήση των Loop Unrolling και Array Partition.

Εξετάστηκαν οι παρακάτω περιπτώσεις με συνδυασμούς παραμέτρων και διαστάσεων των πινάκων (με σταθερή τη μία διάσταση HEIGHT στα 64 πίξελ) όπως φαίνεται στον πίνακα. Τα παρακάτω πειράματα χρησιμοποιούν pipelining στην nested loop.

Φάση Πειράματος	Διάσταση Εικόνας	Τιμή Παραμέτρων	Συνολικό Latency (cycles)	Χρόνος Εκτέλεσης	Παρατήρηση
Pipelining	64x64	$\Pi=1$	4444	0.044ms	Baseline (Best Latency)
Unroll & Array Partition Factor	64x64	factor=2	11149	0.111 ms	Memory Bound
	64x64	factor=4	10125	0.101 ms	Overhead Dominant
	64x64	factor=8	9613	0.096 ms	
	64x128	factor=8	18317	0.183 ms	
	64x256	factor=8	35725	0.357 ms	
	64x512	factor=8	70541	0.705 ms	
Unroll & Array Partition Factor With DATAFLOW	64x64	factor=8	5016	0.05 ms	~2x Speedup vs No-Dataflow
	64x128	factor=8	9184	0.09 ms	
	64x256	factor=8	17520	0.175 ms	
	64x512	factor=8	34192	0.342 ms	

Αρχικά, η απουσία του DATAFLOW οδηγούσε σε σειριακή εκτέλεση των σταδίων (Read-Compute-Write), με αποτέλεσμα το συνολικό latency να ισούται με το άθροισμα των χρόνων τους. Η ενεργοποίηση του directive επέτρεψε την παράλληλη εκτέλεση, μειώνοντας τον συνολικό χρόνο στη διάρκεια του πιο αργού σταδίου και επαναφέροντας το throughput στο βέλτιστο 1 cycle/pixel.

Παρατηρούμε ότι η απλή τεχνική Pipeline (χωρίς buffers) πέτυχε τον καλύτερο απόλυτο χρόνο (4444 cycles). Αυτό συμβαίνει διότι ο αλγόριθμος είναι Memory Bound (περιορίζεται από το εύρος ζώνης της μνήμης) και όχι Compute Bound.

Η "απλή" τεχνική Pipeline ήδη προκαλεί κορεσμό (saturate) στον δίαυλο μνήμης (1 pixel/cycle). Η προσθήκη της σύνθετης αρχιτεκτονικής (Buffers + Unroll + Partition) πρόσθεσε πολυπλοκότητα και μικρό latency (overhead) χωρίς να μπορεί να ξεπεράσει το φυσικό όριο του διαύλου μνήμης.

Ωστόσο, η αρχιτεκτονική με buffers είναι απαραίτητη σε πιο σύνθετους αλγορίθμους όπου απαιτείται επαναχρησιμοποίηση δεδομένων, κάτι που δεν ισχύει στην απλή αφαίρεση εικόνων.

ii) Βέλτιστη Υλοποίηση (Optimized Implementation)

Για την περίπτωση **256x256**, επιλέχθηκε η βέλτιστη από τις παραπάνω λύσεις ως προς το latency, δηλαδή η χρήση μόνο του Directive Pipeline. Παρακάτω παρατίθενται τα αποτελέσματα που ζητούνται:

Estimated clock period: 7.30ns

Number of DSP48E used: 0

Number of BRAMs used: 0

Number of FFs used: 2458

Number of LUTs used: 3565

Total Execution Time: 0.65 ms

Min latency: 65578 cycles

Avg. latency: 65578 cycles

Max latency: 65578 cycles

iii) Υπολογισμός Επιτάχυνσης (Speed-up)

Υπολογίζοντας την επιτάχυνση (speed-up) σε σχέση με την αρχική σειριακή υλοποίηση (χωρίς directives, 262593 cycles), η βέλτιστη σχεδίαση πέτυχε:

$$Speedup = \frac{T_{initial}}{T_{best}} = \frac{262593}{65578} = 4.004$$

Η επιτάχυνση αυτή οφείλεται στην ικανότητα του Pipelining (και του Dataflow στην περίπτωση των Buffers) να κρύβει το latency της μνήμης και να επεξεργάζεται 1 pixel ανά κύκλο ρολογιού, αντί για 4 κύκλους ανά pixel.

Δηλαδή, η βελτιστοποιημένη σχεδίαση είναι περίπου **4 φορές** γρηγορότερη.

4. Συμπεράσματα

Στο παρόν εργαστήριο μελετήθηκε η ροή σχεδίασης του Vitis HLS και η επιτάχυνση αλγορίθμων C. Βασικό συμπέρασμα αποτέλεσε το γεγονός ότι ο αλγόριθμος είναι **Memory Bound**, δηλαδή η ταχύτητά του περιορίζεται από το εύρος ζώνης της εξωτερικής μνήμης και όχι από την υπολογιστική ισχύ.

Η βέλτιστη υλοποίηση επιτεύχθηκε με την απλή χρήση του directive **PIPELINE**, η οποία κατάφερε να αξιοποιήσει πλήρως τον δίαυλο μνήμης (throughput 1 pixel/cycle). Αντιθέτως, η χρήση Buffers και Unrolling εισήγαγε πολυπλοκότητα και καθυστέρηση (overhead) χωρίς να ξεπερνά τα φυσικά όρια της DRAM. Συνολικά, η βέλτιστη σχεδίαση πέτυχε **επιτάχυνση (speedup) 4x** συγκριτικά με την αρχική σειριακή εκτέλεση.