Τεκμηρίωση Πρώτου Θέματος εξαμηνιαίας εργασίας μαθήματος «Προγραμματισμός Ενσωματωμένων Συστημάτων σε Περιβάλλοντα Edge»:

Εαρινό Εξάμηνο 2021 – 2022

Αλέξανδρος Βεντούρας 21106

Χρήστος Τζέλης 21115

1. Υλοποίηση επιταχυντή:

Η υλοποίηση του επιταχυντή για τη λειτουργία της συνέλιξης, έγινε με χρήση του Vitis HLS. Αναλυτικότερα, γράφτηκε ένα πρόγραμμα σε γλώσσα C++, το οποίο δημιουργεί δύο πίνακες, που αναπαριστούν την εικόνα και τον πυρήνα.

Ο πίνακας της εικόνας (διαστάσεων 128x128) δημιουργείται μέσω μίας συνάρτησης παραγωγής ψευδοτυχαίων τιμών, η οποία είναι σχεδιασμένη έτσι ώστε να παράγει ακέραιες τιμές από 0 μέχρι 255, προκειμένου να αναπαρασταθεί η συγκεκριμένη λειτουργία εάν είχαμε εικόνα με ένα κανάλι των 8 bit για κάθε pixel (γραμμές 27 – 35).

Από την άλλη, ο πίνακας του πυρήνα (διαστάσεων 2x2 ή 4x4 ή 8x8), αποτελείται από float τιμές, οι οποίες προκύπτουν εάν διαιρέσουμε τη μονάδα με το γινόμενο των διαστάσεων της εικόνας, σύμφωνα με τις υποδείξεις της εκφώνησης (γραμμές 15 – 23).

Στη συνέχεια, με τη βοήθεια μιας δομής εμφωλευμένων for εντολών επανάληψης, γίνεται η πράξη της συνέλιξης (γραμμές 47 – 63). Το αποτέλεσμα αυτής, που δεν είναι παρά ένας πίνακας διαστάσεων ίσων με αυτών της αρχικής εικόνας, αποθηκεύεται σε ένα νέο πίνακα. Επειδή δε κατά τη διαδικασία της συνέλιξης υπάρχει αριθμητική πράξη μεταξύ float και ακέραιου, προκύπτει ένας νέος float αριθμός για κάθε τιμή του παραγόμενου πίνακα, και προκειμένου να έχουμε ως

αποτέλεσμα έναν πίνακα ο οποίος πάλι θα έχει τιμές από 0 μέχρι 128, γίνονται ορισμένες πράξεις προς αυτήν την κατεύθυνση. (γραμμή 61).

Έγιναν αρκετές προσπάθειες να δοθεί ως είσοδος εικόνα στο πρόγραμμα, αλλά οι διάφορες βιβλιοθήκες που παρέχει το συγκεκριμένο λογισμικό που χρησιμοποιήσαμε (openCV κλπ), εμφάνιζαν ορισμένα σφάλματα, όχι τόσο κατά τη διαδικασία της προσομοίωσης, όσο κατά τη διαδικασία της σύνθεσης. Επομένως, προτιμήθηκε η προσομοίωση της όλης διαδικασίας με τη χρήση των προαναφερθέντων πινάκων. Το ίδιο συνέβαινε και με τη χρήση άλλων βιβλιοθηκών, όπως αυτήν που περιέχει τη γεννήτρια παραγωγής τυχαίων αριθμών της C++. Ευτυχώς η συγκεκριμένη μπορούσε να αντικατασταθεί με μία custom συνάρτηση, η οποία παράγει ψευδοτυχαίες τιμές.

2. Σύνθεση επιταχυντή:

Η σύνθεση του επιταχυντή, επιτευχθείσα με τη βοήθεια του προαναφερθέντος εργαλείου, μας έδωσε διάφορα αποτελέσματα σχετικά με την κατανάλωση υπολογιστικών πόρων του fpga. Στις επόμενες εικόνες, παρατίθενται τα αποτελέσματα των σχετικών πειραμάτων για τα τρία διαφορετικά μεγέθη του πυρήνα, μαζί με τους χρόνους εκτέλεσης.

Resource Usage

	Verilog
SLICE	693
LUT	1487
FF	1961
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	71
CLB	0

2x2

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	6.872
CP achieved post-implementation	7.355

Timing met

Resource Usage

	Verilog
SLICE	1289
LUT	2960
FF	4492
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	183
CLB	0

4x4

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	6.872
CP achieved post-implementation	7.830

Timing met

Resource Usage

	Verilog
SLICE	3581
LUT	8232
FF	12379
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	710
CLB	0

8x8

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	6.872
CP achieved post-implementation	8.766

Timing met

Να σημειωθεί ότι η συσκευή (xc7z007sclg400-2) που επιλέχτηκε είχε τις προδιαγραφές που όριζε η εκφώνηση, εκτός από το speed grade που ήταν -2 αντί -3 (δεν υπήρχε στην έκδοση που χρησιμοποιούσαμε συσκευή με τα συγκεκριμένα χαρακτηριστικά και ταυτόχρονα να έχει speed grade -3).

3. HLS Directives:

Παρακάτω παρουσιάζεται η μετατροπή στον κώδικα που έγινε, προκειμένου να υπάρξει βελτιστοποίηση της υλοποίησης του hardware. Συγκεκριμένα, ακολουθώντας τις οδηγίες που υπάρχουν στις διαφάνειες του εργαστηρίου, δημιουργήθηκε μία ντιρεκτίβα με τη βοήθεια της εντολής pragma HLS, τύπου pipeline, για τον εξωτερικότερο από τα 4 for loop του convolusion. Η χρήση της εντολής αυτής πράγματι, σύμφωνα με τις μετρικές που εξάγει η synthesis του Vitis, μείωσαν σε όλες τις περιπτώσεις το latency και το Interval, όπως φαίνεται και στα screenshots της επόμενης σελίδας.

```
#ifdef loop_optimization
IR: for (int i = 0; i<image_rows; i++) {

#pragma HLS PIPELINE II=1
        for (int j = 0; j<image_cols; j++) {
```

Από την άλλη, ο εκτιμώμενος χρόνος αυξανόταν όποτε εφαρμοζόταν η βελτιστοποίηση, και αυτό εικάζουμε ότι είναι αναμενόμενο επειδή η διαδικασία αυτή καταναλώνει περισσότερο χρόνο. Αυξημένες ήταν και οι χρήσεις των components της συσκευής, και αυτό προφανώς γιατί η βελτιστοποίηση οδηγεί στην μείωση του ποσοστού των idle πόρων, προκειμένου να μειώσει το latency.

Να σημειωθεί ότι έγιναν διάφορα επιπλέον πειράματα, προκειμένου να εξακριβωθεί το είδος καθώς και το ποσοστό της μείωσης του latency, αναλόγως του πόσες directives εισάγουμε στον κώδικα. Έγιναν δοκιμές για εισαγωγή δύο ή τεσσάρων directives, για τα δύο εξωτερικά ή για όλα τα for loop αντιστοίχως. Παραδόξως, σε κάθε μία από αυτές τις δοκιμές δεν παρατηρήθηκε περαιτέρω

μείωση του latency. Προφανώς αυτό έχει να κάνει με την αδυναμία εκτέλεσης το συγκεκριμένων ντιρεκτίβων εμφωλευμένα.

Επίσης, δοκιμάστηκαν διάφορες άλλες παράμετροι, για παράδειγμα η παράμετρος II =2, ή και παραπάνω, καθώς οι παράμετροι που υπάρχουν στα εργαστήρια, ή και άλλες που βρέθηκαν στο Internet. Γενικά, μπορούμε να πούμε ότι δεν παρατηρήθηκε ιδιαίτερη βελτίωση στο latency, σε κάθε μία από αυτές τις περιπτώσεις, για αυτό και τα αποτελέσματα των παραπάνω πειραμάτων θεωρήσαμε ότι δεν έχει νόημα να συγκαταλέγονται ανάμεσα στα παρακάτω στιγμιότυπα οθόνης. Να σημειωθεί επίσης, ότι το Violation που εμφανιζόταν αναφορικά με την παράμετρο II, δεν επηρεαζόταν από τις παραπάνω μετατροπές, και εμφανιζόταν ακόμα και εάν είχε εντελώς εξαφανιστεί η βελτιστοποίηση (με comment out των directives).

α) Με χρήση kernel 2x2:

Χωρίς Directive:

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.142 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-	32808	3.280E5	-	32809	-	no	0	13	2505	3360	0
1_VITIS_LOOP_35_2	II Violation			-	32805	3.280E5	-	32805	-	no	0	13	2308	3313	0
2	II Violation	Resource Limitation		-	32803	3.280E5	38	2	16384	yes	-	-	-	-	-

Με Directive:

Target	Estimated	Uncertainty
10.00 ns	8.868 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

ps	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
on				-1.57	16478	1.650E5	-	16479	-	no	0	23	14090	26131	0
usion_Pipeline_IR	II Violation			-1.57	16475	1.650E5	-	16475	-	no	0	23	13893	26084	0
	II Violation	Resource Limitation		-	16473	1.650E5	218	128	128	yes	-	-	-	-	-

β) Με χρήση kernel 4x4:

Χωρίς Directive:

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.142 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-	131172	1.312E6	-	131173	-	no	0	13	5902	5078	0
1_VITIS_LOOP_35_2	II Violation			-	131163	1.312E6	-	131163	-	no	0	13	4931	4959	0
2	II Violation	Resource Limitation		-	131161	1.312E6	98	8	16384	yes	-	-	-	-	-

Με Directive:

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	9.470 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

ps	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
on				-2.17	32907	3.290E5	-	32908	-	no	0	43	29658	47237	0
usion_Pipeline_IR	II Violation			-2.17	32898	3.290E5	-	32898	-	no	0	43	28687	47118	0
	II Violation	Resource Limitation		-	32896	3.290E5	355	256	128	yes	-	-	-	-	-

γ) Με χρήση kernel 8x8:

Χωρίς Directive:

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.946 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-0.65	524628	5.246E6	-	524629	-	no	0	13	15728	10736	0
VITIS_LOOP_35_2	II Violation			-0.65	524595	5.246E6	-	524595	-	no	0	13	11661	10281	0
	II Violation	Resource Limitation		-	524593	5.246E6	338	32	16384	yes	-	-	-	-	-

Με Directive:

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	7.946 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-0.65	524628	5.246E6	-	524629	-	no	0	13	15728	10736	0
VITIS_LOOP_17_1	II Violation			-0.65	524595	5.246E6	-	524595	-	no	0	13	11661	10281	0
	II Violation	Resource Limitation		-	524593	5.246E6	338	32	16384	yes	-	-	-	-	-

(προστέθηκε η παράμετρος port = 10 στην εντολή pragma προκειμένου να αποφευχθεί η ατέρμονη εκτέλεση του synthesis[1])

[1] Η παράμετρος προστέθηκε βάσει οδηγιών που βρίσκονται παρακάτω σύνδεσμο. Ο σύνδεσμος αυτός προτάθηκε ως πιθανή λύση για το αναφερόμενο πρόβλημα κατά την εκτέλεση του Synthesis, από το command line του εργαλείου.
https://www.xilinx.com/htmldocs/xilinx2021_2/hls-guidance/200-885.html

4.Συγκριτική μελέτη 2 FPGA:

Αναφορικά με τις δοκιμές που πρέπει να γίνουν, διαλέξαμε δύο fpga. Αρχικά, έγιναν τρεις δοκιμές με την συσκευή xc7s6ftgb196-1 τύπου Spartan-7, η οποία έχει τα ακόλουθα χαρακτηριστικά:

Package	Speed	SLICE	LUT	FF	DSP	BRAM
ftgb196	-1	2000	3750	7500	10	5

Για το συγκεκριμένο fpga, τα πρώτα στάδια εκτέλεσης ολοκληρώθηκαν επιτυχώς, αλλά κατά το implementation προέκυψε σφάλμα. Και αυτό, γιατί όπως μας πληροφόρησε το σχετικό error message, είχαμε επιλέξει συσκευή με λιγότερα DSP από όσα απαιτούνταν προκειμένου να τρέξει επιτυχώς το κύκλωμα που έχουμε δημιουργήσει. Συγκεκριμένα, το fpga αυτό είχε 10 DSP και η δική μας υλοποίηση απαιτούσε τουλάχιστον 13.

Έτσι, προκειμένου να ξεπεράσουμε αυτό το πρόβλημα, επιλέξαμε τη συσκευή xc7a15tcsg324-1 (τύπου Artix-7), η οποία ικανοποιούσε την παραπάνω συνθήκη, προκειμένου να τρέξει επιτυχώς το implementation, και η οποία συσκευή έχει τα ακόλουθα χαρακτηριστικά:

Package	Speed	SLICE	LUT	FF	DSP	BRAM
csg324	-1	8150	10400	20800	45	25

Με τη συσκευή αυτή εκτελέστηκε επιτυχώς το implementation, και προέκυψαν τα παρακάτω αποτελέσματα για τα διάφορα μεγέθη του πυρήνα:

Σχολιάζοντας αυτά τα αποτελέσματα, σχετικά με το χρόνο εκτέλεσης, το δεύτερο μηχάνημα είναι κατά πολύ ταχύτερο από το πρώτο, και αυτό είναι λογικό εφόσον διαθέτει περισσότερους πόρους. Αυτή η χρήση περισσότερων πόρων απεικονίζεται και αριθμητικά στα στοιχεία για τα χρησιμοποιούμενα component κάθε συσκευής.

Παρατηρούμε επίσης, ότι σχεδόν πάντα, όσο αυξάνει το μέγεθος του kernel, τόσο αυξάνουν και οι χρήσεις των διαφόρων πόρων και στα δύο μηχανήματα, κάτι που είναι απόλυτα φυσικό. Αποκλίσεις από το παραπάνω μοτίβο, παρουσιάζει η πρώτη συσκευή χαμηλών προδιαγραφών, η οποία με τη χρήση του kernel 4x4, καταναλώνει περισσότερο χρόνο για τη σύνθεση, αλλά λιγότερο χρόνο για την υλοποίηση, σε σχέση τόσο με τον πυρήνα 2x2 όσο και με τον πυρήνα 8x8.

Resource Usage

	Verilog
SLICE	748
LUT	1596
FF	2300
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	104
CLB	0

2x2

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	5.629
CP achieved post-implementation	8.779

Timing met

Resource Usage

	Verilog
SLICE	1381
LUT	3080
FF	4892
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	224
CLB	0

4x4

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	6.232
CP achieved post-implementation	7.548

Timing met

Resource Usage

	Verilog
SLICE	3816
LUT	8023
FF	12929
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	808
CLB	0

8x8

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	5.680
CP achieved post-implementation	9.386

Timing met

Έπειτα, έγιναν τρεις ακόμα δοκιμές με την συσκευή xczu5cg-fbvb900-2-i (τύπου Zynq UltraScale+ MPSoCs), οποία έχει τα ακόλουθα χαρακτηριστικά:

Package	Speed	SLICE	LUT	FF	DSP	BRAM
-fbvb900	-2	14640	117120	234240	1248	144

Έπειτα από την επιτυχή εκτέλεση του implementation, προέκυψαν τα παρακάτω αποτελέσματα για τα διάφορα μεγέθη του πυρήνα:

Resource Usage

	Verilog
SLICE	0
LUT	1434
FF	1810
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	46
CLB	319

2x2

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	3.810
CP achieved post-implementation	4.925

Timing met

Resource Usage

	Verilog
SLICE	0
LUT	2750
FF	3993
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	161
CLB	643

4x4

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	3.810
CP achieved post-implementation	5.564

Timing met

Resource Usage

	Verilog
SLICE	0
LUT	7436
FF	11936
DSP	13
BRAM	0
URAM	0
LATCH	0
SRL	611
CLB	2085

8x8

Final Timing

	Verilog
CP required	10.000
CP achieved post-synthesis	3.810
CP achieved post-implementation	5.683

Timing met