

**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΗΛΕΚΤΡΟΝΙΚΗΣ

**Σχεδιασμός Ενσωματωμένων Συστημάτων**

9ο εξάμηνο

4η Εργαστηριακή Άσκηση:

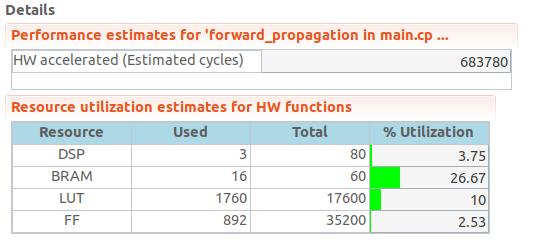
**Εργασία για High Level Synthesis σε FPGA**

ΑΜ: 03116112

Ονοματεπώνυμο: Γεζεκελιάν Βικέν

**Άσκηση 1: Performance and resources measurement**

Α) Μετά την εγκατάσταση του SDSoC και αφού δημιουργήσαμε κατάλληλα το workspace και το project, θέσαμε τη συνάρτηση forward\_propagation εντός του network.cpp ως την Hardware συνάρτηση όπου θέλουμε να υλοποιήσουμε το FPGA acceleration. Έπειτα, κάναμε estimate performance, χωρίς ακόμη να χτίσουμε bitstream ή την sd card που θα χρειαστούμε για να τρέξουμε τη συνάρτηση στο zybo. Τα αποτελέσματα του estimation φαίνονται παρακάτω:



Όπως μπορούμε να δούμε, η χρήση των πόρων είναι αρκετά χαμηλή, γεγονός που σημαίνει πως θα έχουμε περιθώριο για την πραγματοποίηση αρκετών optimizations.

Β) Για το ερώτημα αυτό, χρειάστηκε να περάσουμε τον φάκελο sd\_card που δημιουργήθηκε από το SDSoC(καθώς και το αρχείο του input, data.txt) στο zybo όπου θέλουμε τελικά να τρέξει μέσω του portal. Αφού αντικαταστήσαμε κατάλληλα τα αρχεία και κάναμε reboot το board τα αποτελέσματα του εκτελέσιμου ήταν τα εξής:

Hardware cycles : 683010

Software cycles : 1475360

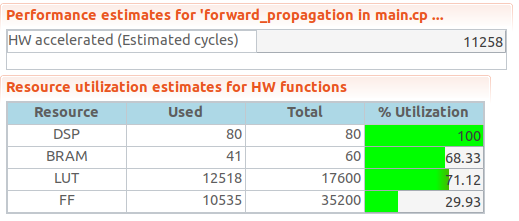
Speed-Up : 2.16009

Βλέπουμε πως ήδη, παρόλο που δεν έχουμε εκτελέσει κάποιο optimization στον κώδικα, η hardware συνάρτηση τρέχει σε λιγότερο από τους μισούς κύκλους σε σχέση με τη software συνάρτηση, με το speed up να είναι ίσο με 2.16009. Επίσης, παρατηρούμε πως το estimation του SDSoC είναι σε πολύ μεγάλο βαθμό ακριβές, αφού η απόκλιση από την πραγματικότητα ήταν περίπου ίση με 0.1%

Γ) Για το ερώτημα αυτό, χρειάστηκε να εισάγουμε διάφορα HLS pragmas στον κώδικα ώστε να τον επιταχύνουμε. Οι ακριβείς εισαγωγές pragmas φαίνονται στο αρχείο network.cpp που υπέβαλλα. Το design που δημιουργήσαμε είναι fully pipelined, με τη μεγαλύτερη μείωση κύκλων να προέρχεται από την αλλαγή αυτή. Το γεγονός αυτό οφείλεται στο ότι ο κώδικας αποτελείται από 5 loops, 2 εκ των οποίων απλώς διατρέχουν έναν μονοδιάστατο πίνακα(στη μία περίπτωση για διάβασμα εισόδου και στη δεύτερη για να κρατηθούν τα θετικά στοιχεία), και 3 εκ των οποίων εκτελούν διαδοχικούς πολλαπλασιασμούς πινάκων εντός φωλιασμένου loop. Σε όλες τις περιπτώσεις, μετατρέποντας το design σε pipelined, επιτρέπουμε την ταυτόχρονη εκτέλεση πολλαπλών πράξεων. Ακόμη, μεγάλη διαφορά είδαμε και κατά το loop unrolling του πρώτου layer. Σε αντίθεση με τα δύο επόμενα, το layer 1 περιείχε και μια ανάθεση στο εσωτερικό loop, στην οποία οφείλεται η μεγαλύτερη βελτίωση κατά το unrolling, αφού κατά το unrolling των υπόλοιπων layers δεν υπήρξε βελτίωση στην εκτέλεση. Επίσης, με πολλαπλές προσπάθειες βρήκαμε πως το βέλτιστο unrolling factor είναι το 9, καθώς το μόνο που γνωρίζαμε αρχικά είναι πως το βέλτιστο factor θα είναι πολλαπλάσιο του 3 και θα εξαρτάται από το pipeline που χρησιμοποιείται. Πέραν των αλλαγών αυτών, δεν μπορούσαμε να προσθέσουμε άλλα pragmas στον κώδικα, λόγω των υλικών περιορισμών.

Σημείωση: θα μπορούσαμε να βελτιστοποιήσουμε τον κώδικα αφαιρώντας το layer\_1\_act loop και κάνοντας κατευθείαν ReLU εντός του layer\_1 loop, όμως ζητήθηκαν αλλαγές μόνο όσον αφορά τη προσθήκη pragmas.

Παρακάτω, φαίνονται τα αποτελέσματα του estimation καθώς και τα αποτελέσματα από το zybo:



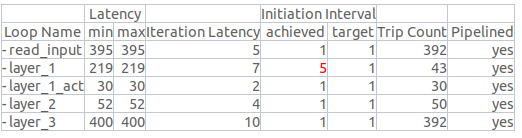
Hardware cycles : 11430

Software cycles : 1475214

Speed-Up : 129.065

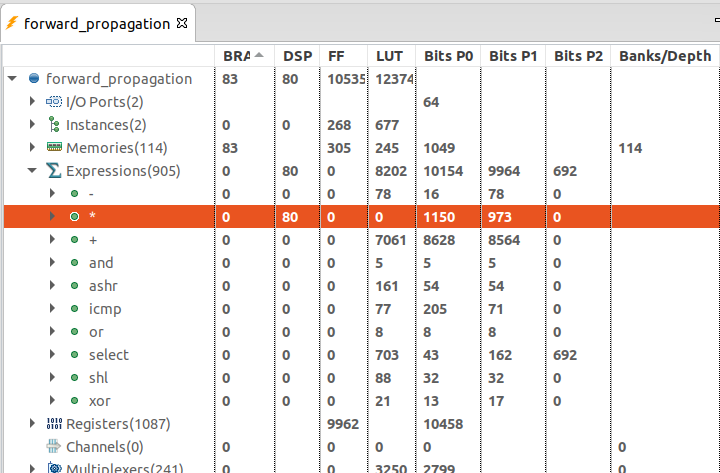
Όπως βλέπουμε, υπάρχει πολύ μεγάλη διαφορά μεταξύ optimized και un-optimized υλοποίησης. Το speed up αυτή τη φορά είναι 129.065, πολύ μεγαλύτερο απ’ ότι προηγουμένως.

Δ) Παρακάτω φαίνονται αναλυτικά τα latency details για κάθε loop, όπως φαίνονται εντός του SDSoC.



Όπως μπορούμε να δούμε από τον πίνακα, το μεγαλύτερο latency βρίσκεται στο layer\_3 ενώ το design μας είναι fully pipelined, όπως αναφέραμε και προηγουμένως.

Ακολουθεί το Resource profile όπως φαίνεται εντός του SDSoC:

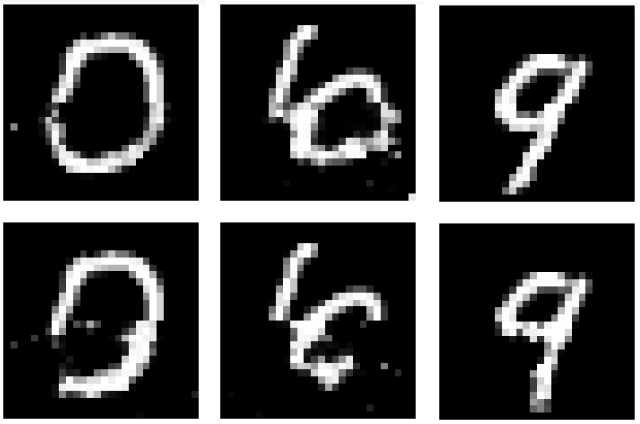


Όπως μπορούμε να δούμε, το σύνολο των DSPs χρησιμοποιούνται για την εκτέλεση πολλαπλασιασμών.

**Άσκηση 2: Quality measurement**

Α) Ο κώδικας που έχει χρησιμοποιηθεί για την υλοποίηση των παρακάτω είναι αυτός που μας δόθηκε αρχικά με την εκφώνηση. Τα αποτελέσματα που παρήχθησαν φαίνονται παρακάτω, πάνω η software και έπειτα η hardware υλοποίηση:



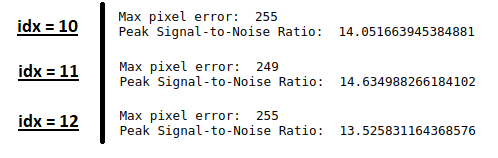
Β) Για το ερώτημα αυτό, κάναμε τις απαραίτητες μετατροπές στον κώδικα, αλλάζοντας τις τιμές των BITS και BITS\_EXP στο αρχείο network.h και αντιγράφοντας τα περιεχόμενα του κατάλληλου tanh\_n.h εντός του tanh.h κάθε φορά, τα αποτελέσματα φαίνονται παρακάτω, όπως και πριν αρχικά η software, ακολουθούμενη από την hardware υλοποίηση. Αρχικά για 4 bits:

Και έπειτα για 10 bits:

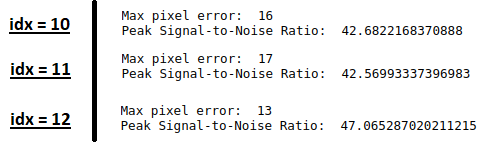


Στις παραπάνω εικόνες, παρατηρούμε πως η υλοποίηση με 4 bits έχει τα χειρότερα αποτελέσματα εκ των τριών, ενώ η αύξηση των bits βελτιώνει την κατασκευή των αριθμών από το GAN. Το γεγονός αυτό είναι αναμενόμενο, αφού μείωση των χρησιμοποιούμενων bits έχει ως αποτέλεσμα μείωση την ακρίβειας του προγράμματος και κατ’ επέκταση μείωση της ποιότητας κατασκευής.

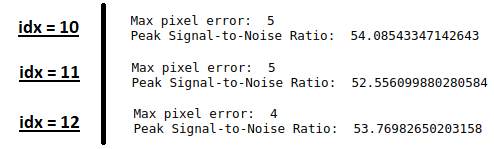
Γ) Στη συγκεκριμένη περίπτωση, τα max pixel error και psnr προκύπτουν από τη σύγκριση της εικόνας που παράγει η software εν σχέσει με την εικόνα που παράγεται από την hardware υλοποίηση. Παρακάτω φαίνονται τα αποτελέσματα, όπως αυτά εμφανίστηκαν εντός του notebook, αρχικά για n=4 bits:



n=8 bits:



n=10 bits:



Βάσει των αποτελεσμάτων, φαίνεται πως και οι δύο τεχνικές μέτρησης λειτουργούν με ικανοποιητικό τρόπο. Παρ’ όλα αυτά, μεταξύ των δύο η εύρεση των διαφορετικών pixel φαίνεται να είναι πιο χρήσιμη στη συγκεκριμένη περίπτωση, καθώς η psnr είναι χρήσιμη κυρίως για τη σύγκριση του θορύβου για διαφορετικά bits. Παρ’ όλα αυτά, δεν προσφέρει κάποια αντικειμενική μετρική ώστε να ξέρουμε τα περιθώρια βελτίωσης που παραμένουν. Π.χ. στη συγκεκριμένη περίπτωση, βλέπουμε πως ακόμη και να αυξήσουμε παραπάνω τα bits, τα περιθώρια βελτίωσης είναι πολύ μικρά, αφού η max pixel error επιστρέφει 4 ή 5 για n=10 bits. Παρ’ όλα αυτά, με τη psnr ενώ βλέπουμε βελτίωση σε σχέση με τα 8 bits, δε ξέρουμε εάν θα άξιζε η όχι μια υλοποίηση με περισσότερα bits. Ακόμη ένα ελάττωμα που παρουσιάζουν και οι 2 τεχνικές είναι η σύγκριση με την παραγόμενη από το software εικόνα, στην οποία επίσης υπάρχουν κάποιες αστοχίες(όπως βλέπουμε στα 0 και 6 υπάρχουν κάποια «ενεργοποιημένα» bits που δεν θα έπρεπε να είναι εκεί.

Τέλος, όσον αφορά την ιδανική ακρίβεια bits, φαίνεται πως για n=8 bits έχουμε πολύ καλά αποτελέσματα, με αποτέλεσμα να μην είναι απαραίτητη η χρήση 10 bits(που έχουν ως αποτέλεσμα τον τετραπλασιασμό του μεγέθους του tanh).