ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο Εξάμηνο ΗΜΜΥ

5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Εργασία για High Level Synthesis σε FPGA

Ο σκοπός της άσκησης είναι να γίνει μελέτη γύρω από τον προγραμματισμό FPGA με High Level Synthesis (HLS). Πρόκειται για optimization και επιτάχυνση C κώδικα για να τρέξει τελικώς στο hardware του Xilinx Zybo FPGA. Η εφαρμογή που θα μελετηθεί για επιτάχυνση θα σχετίζεται με νευρωνικά δίκτυα και ειδικότερα τα Generative Adversarial Networks (GANs). Συγκεκριμένα, η εφαρμογή του GAN που δίνεται έτοιμη αφορά την ανακατασκευή ημιτελών εικόνων από χειρόγραφα ψηφία. Ο τελικός στόχος είναι να επιταχυνθεί ο αλγόριθμος αυτός σε σχέση με την SW υλοποίηση αλλά και να μετρηθεί η ποιότητα ανακατασκευής των εικόνων.

Link εφαρμογής: [Google Drive link]

Εργαλεία:

Xilinx SDSoC 2016.4 [link]: Περιβάλλον για να αναπτύξετε την εφαρμογή ώστε να τρέξει στο Zybo. Οδηγίες για την εγκατάσταση του Xilinx SDSoC 2016.4 υπάρχουν στη σελίδα του μαθήματος στον σύνδεσμο Έγγραφα → Lab_material → Lab_4 → SDSoC_installation_guide.pdf.

Jupyter Notebook: Περιβάλλον όπου θα εμφανίζετε το output του FPGA που έχει τις ανακατασκευασμένες εικόνες. Σας δίνεται ο κώδικας *plot_output.ipynb*. Μπορείτε να τον τρέξετε τοπικά σε linux αφού κάνετε install το jupyter (pip install notebook). Επίσης μπορείτε μέσω google collab online από οποιοδήποτε pc: αφού δημιουργήσετε ένα copy του αρχείου στο drive σας (μαζί με το output.txt και data.txt) μπορείτε να κάνετε διπλό κλικ στο notebook και open with Google Collab.

Εφαρμογή: Ανακατασκευή εικόνας με GAN

Η εφαρμογή αφορά την ανακατασκευή εικόνων από το MNIST dataset (28x28 χειρόγραφοι grayscale αριθμοί). Συγκεκριμένα, έχετε το generator νευρωνικό μοντέλο από το GAN όπου σας δίνεται η υλοποίηση του σε C μαζί με τα trained βάρη. Η εφαρμογή δέχεται ως input το πάνω μισό των αριθμών και

καλεί το νευρωνικό για να κάνει generate/predict το υπόλοιπο μισό της εικόνας. Μετά την εκτέλεση θα φανεί ο μέσος χρόνος εκτέλεσης ανά εικόνα σε κύκλους για SW και HW και το speed-up. Σκοπός είναι να μεγιστοποιήσετε το speed-up με HLS optimizations στην HW function. Το *output.txt* που παράγει η εφαρμογή μπορείτε να το επεξεργαστείτε έπειτα μέσω Jupyter για να εμφανίσετε τους αριθμούς.

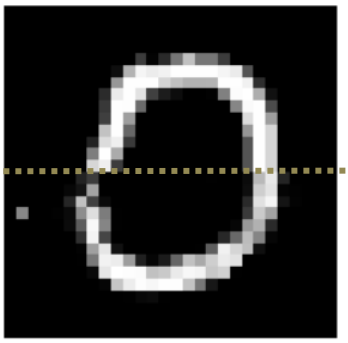
Ενδεικτικό παράδειγμα εκτέλεσης της εφαρμογής στο Zybo (το .elf είναι το ARM εκτελέσιμο)

```
$ /mnt# ./embedded.elf
Starting dataset parsing...
Parsing finished...
Starting hardware calculations...
Hardware calculations finished.
Starting software calculations...
Software calculations finished.
Hardware cycles : 683780
Software cycles : 1448258
Speed-Up        : 2.1180174
Saving results to output.txt...
$/mnt#
```

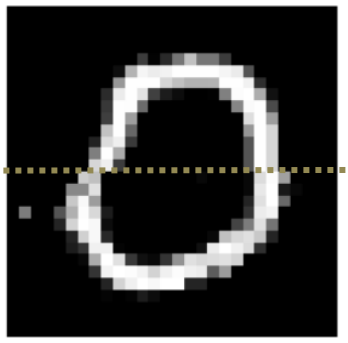
Μπορείτε έπειτα να δείτε το αρχείο output με την εντολή vi output.txt. Θα περιέχει τις τιμές για κάθε pixel (-1 ως 1) από κάθε αριθμό (1000 συνολικά) τη μία μετά την άλλη από την έξοδο σε SW και HW. Επίσης μπορείτε να το μεταφέρετε από την SD card στο pc σας για να το επεξεργαστείτε μέσω του jupyter notebook *plot_output.ipynb* που σας παρέχεται.

Ενδεικτικό παράδειγμα εξόδου του *plot_output.ipynb*. Εμφανίζουμε τον 10^{ο} αριθμό από το output (idx=10)

Software output



Hardware output



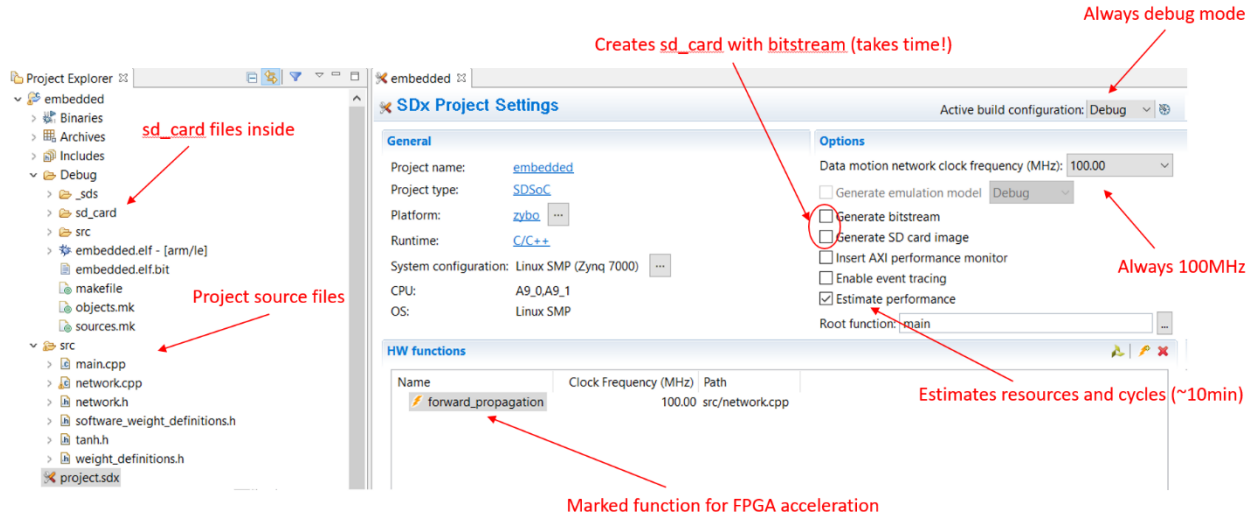
Source files Εφαρμογής:

main.cpp → Τρέχει σε ARM CPU. Καλεί το μοντέλο του Generator σε SW and HW και μετράει την επίδοση.
network.cpp → Είναι ο core κώδικας της εφαρμογής δηλαδή του Generator. Αυτός θα πρέπει να γίνει optimized με HLS για να τρέξει στο HW (μόνο η συνάρτηση *forward_propagation* θελει optimizations).
weight_definitions.h → Περιέχει τα trained βάρη του νευρωνικού του generator.
tanh.h → Αποθηκεύει τις pre-computed τιμές της μαθηματικής συνάρτησης Tanh για το HW.
network.h → Περιέχει διάφορα ορίσματα του Generator (datatypes, loop limits, etc.).
data.txt → Περιέχει το input dataset (το πάνω μισό των εικόνων). Απαιτείται να βρίσκεται στο φάκελο του εκτελέσιμου όταν τρέχει στον ARM αλλά και στο plot_output.ipynb.

Χρήση του εργαλείου SDSoC 2016.4

Μέσω του εργαλείου SDSoC θα μπορείτε να αναπτύξετε την εφαρμογή για το embedded Zybo FPGA. Συγκεκριμένα το SDSoC παρέχει ένα Eclipse τύπου περιβάλλον όπου μπορείτε να γράφετε C/C++ κώδικα και να αναπτύσσετε έναν accelerator που θα τρέξει σε FPGA με χρήση HLS optimizations. Επίσης θα μπορείτε να κάνετε performance και resource estimation, να φτιάξετε το bitstream του υλικού αλλά και τελικώς την sd boot card όπου θα "bootάρει" το σύστημα (τα παραγόμενα αρχεία μεσα στην sd_card περιλαμβάνουν το εκτελέσιμο (.elf), το bitstream και το λειτουργικό petalinux του συστήματος).

SDSoC project



Αφού φτιάξετε ένα Xilinx SDx project σύμφωνα με το πάνω configuration (zybo, linux, debug mode, 100Mhz) και εισάγετε τα source files που σας δίνονται, μαρκάρετε την συνάρτηση που θέλετε να υλοποιηθεί στο HW. Στη συγκεκριμένη περίπτωση θα είναι η *forward_propagation* η οποία στο τέλος θα έχει όλα τα HLS optimizations που θα πρέπει να προσθέσετε. Τέλος, κάνετε build το project κάθε φορά για να υλοποιηθεί αυτό που επιθυμείτε.

Σημαντικές παρατηρήσεις:

1. Δε θα χρειαστεί να κάνετε κάποια αλλαγή στο εκτελέσιμο που θα τρέξει στον ARM (main.cpp) αλλά ούτε και στα βάρη του νευρωνικού.
2. Πριν φτιάξετε bitstream θα πρέπει να κάνετε estimate performance ώστε να δείτε τα resources αν χωράνε στο zybo αλλά και τον αριθμό κύκλων του HW για να αποφανθείτε για το estimated performance.
3. Όταν θα θέλετε να φτιάξετε νέο bitstream καλό θα είναι να κάνετε clean ή ακόμα και να σβήνετε το προηγούμενο debug folder πριν κάνετε νέο build.
4. Για να φτιάξετε sd με bitstream θα τικάρετε μαζί τις 2 επιλογές όπως φαίνεται στη φωτογραφία. Να γνωρίζετε ότι ο χρόνος εξαγωγής bitstream μπορεί να φτάσει τα 45min.
5. Τέλος, όταν είναι έτοιμα τα αρχεία της sd_card θα πρέπει να μεταφέρετε ό,τι έχει μέσα στην sd_card του zybo σας και να αντικαθιστάτε τα παλιά αρχεία με τα νέα. Έπειτα αρκεί να κάνετε reboot το board για να φορτώσει το νέο σύστημα. Μόνο τα αρχεία στην sd παραμένουν μετά από κάθε reboot, τα αρχεία στο Petalinux OS δεν είναι persistent! (πχ. home/)

Ασκήσεις

Άσκηση 1. Performance and resources measurement

Η συνάρτηση forward_propagation είναι ο κώδικας C που τρέχει το νευρωνικό, συγκεκριμένα ο generator. Είναι διάφορα layers που τρέχουν το ένα μετά το άλλο (dense→relu→dense→relu→dense→tanh) με σκοπό να παράξουν το κάτω μισό της εικόνας. Ουσιαστικά πρόκειται για πολ/σμούς matrix-vector το οποίο είναι μια διαδικασία με πολλές πράξεις, αργή για CPU.

Α) Δοκιμάστε να την θέσετε ως HW function και κάντε estimate performance χωρίς να κάνετε κάποιο optimization. Καταγράψτε το report που δείχνει τα estimated resources και cycles της hardware function.

Β) Δοκιμάστε τώρα να φτιάξετε την sd_card με το bitstream και να τα περάσετε στην sd του zybo. (Αγνοείστε πιθανά warnings κατά το compilation (unused labels, κτλ)). Μη ξεχάσετε να περάσετε και το *data.txt*. Κάντε reboot και τρέξτε την εφαρμογή στο board. Σε πόσους κύκλους τρέχει? Συμφωνεί με το estimation? Ποιο είναι το speed-up σε σχέση με την SW εκτέλεση στον ARM?

Γ) Σε αυτό το σημείο καλείσθε να κάνετε design space exploration για να βρείτε τα optimizations ώστε να επιταχυνθεί σημαντικά ο αλγόριθμος. Δοκιμάστε διάφορα HLS pragmas και δείτε τι βγάζει το estimation. Αν σας ικανοποιεί παράξτε την sd με το bitstream και τρέξτε την εφαρμογή στο board. Καταγράψτε ξανά κύκλους και speed-up από το board. Συγκρίνετε αυτή την υλοποίηση με την un-optimized προηγούμενη.

Δ) Επίσης δείτε για την optimized υλοποίηση σας στο SDSoC το HLS report που παράγεται. Καταγράψτε τα latency details για κάθε loop (Latency → Detail → Loop). Ποιό layer έχει το πιο μεγάλο latency? Είναι fully pipelined το design σας? Τώρα, κατευθυνθείτε στην καρτέλα *Resource profile* του HLS report. Καταγράψτε τα είδη των μαθηματικών εκφράσεων (expressions) του design. Ποια έκφραση χρειάζεται τα πιο πολλά DSP?

Άσκηση 2. Quality measurement

Σε αυτό το σημείο καλείσθε να μετρήσετε τη ποιότητα ανακατασκευής των εικόνων μέσω του jupyter notebook που σας δίνεται. Μέσω αυτού θα κάνετε combine τις μισές εικόνες που δόθηκαν σαν input στο *data.txt* με τις μισές εικόνες από το *output.txt* που παρήγαγε το SW αλλά και το HW.

Α) Τα source files που σας δόθηκαν για το SDSoC τρέχουν τον αλγόριθμο του generator επιτυχώς. Μεταφέρετε το output.txt που παρήγαγε το εκτελέσιμο από το board στο pc σας (δεν απαιτείται να είναι optimized το design). Τρέξτε το *plot_output.ipynb* είτε από Linux locally είτε από google collab online. Εμφανίστε τις combined εικόνες από SW και HW για διάφορα νούμερα, συγκεκριμένα για idx: 10, 11, 12.

Β) Η υλοποίηση που σας δόθηκε χρησιμοποιεί datatypes που έχουν 8-bit δεκαδική ακρίβεια. Για να την αλλάξετε αρχικά αλλάξτε τα ορίσματα *BITS* και *BITS_EXP* στο αρχείο *network.h* όπου *BITS_EXP*=$2^{(BITS+2)}$ (πχ. Αν θέλετε 6 bits θέτετε BITS=6, BITS_EXP=256). Έπειτα αλλάξτε τις pre-computed τιμές της Tanh στο αρχείο *tanh.h.* Σας δίνονται οι υπολογισμένες τιμές για διάφορα bits στο φάκελο *tanh*. (πχ. για 6 bits αντικαθιστάτε το tanh.h με το tanh_6.h). Δοκιμάστε να φτιάξετε νέα designs με 4 και 10 bit και να τα τρέξετε. Εμφανίστε τώρα τις combined εικόνες που προκύπτουν από το output πάλι για idx: 10,11,12. Τι παρατηρείτε και γιατί?

Γ) Στο jupyter notebook μετριέται η ποιότητα ανακατασκευής των εικόνων του HW σε σχέση με το SW. Για τα διάφορα bits που δοκιμάσετε (4,8,10) ποιό είναι το max pixel error και ποιό το Peak Signal-to-Noise Ratio (psnr)? Ποιά τεχνική μέτρησης προτιμάτε? Ποιά ακρίβεια bit (από 3 ως 10) είναι ιδανική για εσάς και γιατί ?