ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση Συστημάτων Υλικού - Λογισμικού

Εργαστήριο 2
Εξοικείωση με το εργαλείο Vitis

Α. ΑΘΑΝΑΣΙΑΔΗΣ - Δ. ΚΑΡΑΝΑΣΣΟΣ

Διδάσκων: Ιωάννης Παπαευσταθίου

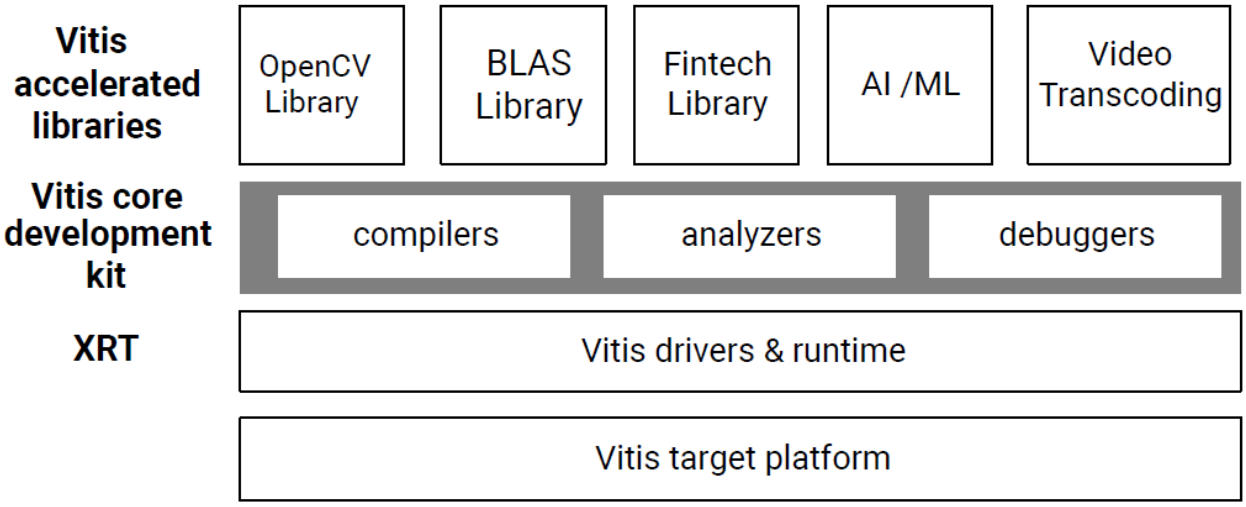
Version 0.5

Δεκέμβριος 2025

1. Εισαγωγή

Στο εργαστήριο 1 ασχοληθήκαμε με τη σχεδίαση του επιταχυντή μας με τη χρήση του εργαλείου Vitis HLS. Στο παρόν εργαστήριο θα εξοικειωθείτε με το εργαλείο Vitis της Xilinx στο οποίο θα τρέξετε εξολοκλήρου τον επιταχυντή που δημιουργήσατε στο 1ο εργαστήριο στη πλακέτα Alveo U200.

Η ενοποιημένη πλατφόρμα λογισμικού Vitis είναι ένα νέο εργαλείο που συνδυάζει όλες τις πτυχές των διαφορετικών λογισμικών της Xilinx σε ένα ενιαίο περιβάλλον. Το Vitis μπορεί να χρησιμοποιηθεί για την επιτάχυνση εφαρμογών ακόμα και από προγραμματιστές λογισμικού χωρίς να έχουν ιδιαίτερες γνώσεις hardware χρησιμοποιώντας το περιβάλλον Xilinx Runtime (XRT).



Όπως φαίνεται στο παραπάνω σχήμα, η πλατφόρμα ενοποιημένου λογισμικού Vitis αποτελείται από τα ακόλουθα:

- Η τεχνολογία Vitis στοχεύει σε πλατφόρμες υλικού επιτάχυνσης, όπως οι κάρτες επιτάχυνσης Alveo™ (οι οποίες βρίσκονται σε Data Centers) ή οι ενσωματωμένες πλατφόρμες επεξεργαστών (π.χ. Zynq® UltraScale+ MPSoC και Zynq-7000 SoC). Στο παρόν εργαστήριο θα ασχοληθούμε με την κάρτα Alveo U200.

- Το XRT παρέχει ένα Application Programming Interface (API) και προγράμματα οδήγησης (drivers) έτσι ώστε να μπορεί απρόσκοπτα να επικοινωνεί και να χειρίζεται μεταφορές δεδομένων μεταξύ του προγράμματος που τρέχει στη CPU (host system) και του FPGA.

- Το Vitis παρέχει μια στοίβα εργαλείων ανάπτυξης λογισμικού, όπως μεταγλωττιστές και cross-compilers, για τη δημιουργία του προγράμματος CPU (host) και του FPGA, αναλυτές (analyzers) που επιτρέπουν στο χρήστη να σχεδιάσει και να αναλύσει την απόδοση της εφαρμογής του και μεθόδους εντοπισμού σφαλμάτων (debuggers) που θα τον βοηθήσουν να εντοπίσει και να διορθώσει προβλήματα στην εφαρμογή του.

- Βιβλιοθήκες επιτάχυνσης οι οποίες παρέχουν επιτάχυνση βελτιστοποιημένη για απόδοση έτσι ώστε με ελάχιστες αλλαγές κώδικα και χωρίς να χρειάζεται να επαναπροσδιορίσει ο χρήστης τους αλγορίθμους του, να αξιοποιούνται τα οφέλη των FPGA της Xilinx. Οι βιβλιοθήκες επιτάχυνσης Vitis είναι διαθέσιμες για κοινές λειτουργίες μαθηματικών, στατιστικής, γραμμικής άλγεβρας, καθώς και για εφαρμογές συγκεκριμένου τομέα, όπως επεξεργασία σήματος και εικόνας, βάση δεδομένων και συμπίεση δεδομένων. (Στα πλαίσια αυτού του εργαστηρίου δε θα χρησιμοποιήσουμε κάποια έτοιμη βιβλιοθήκη καθώς ο στόχος του είναι η κατανόηση βασικών εννοιών του εργαλείου).

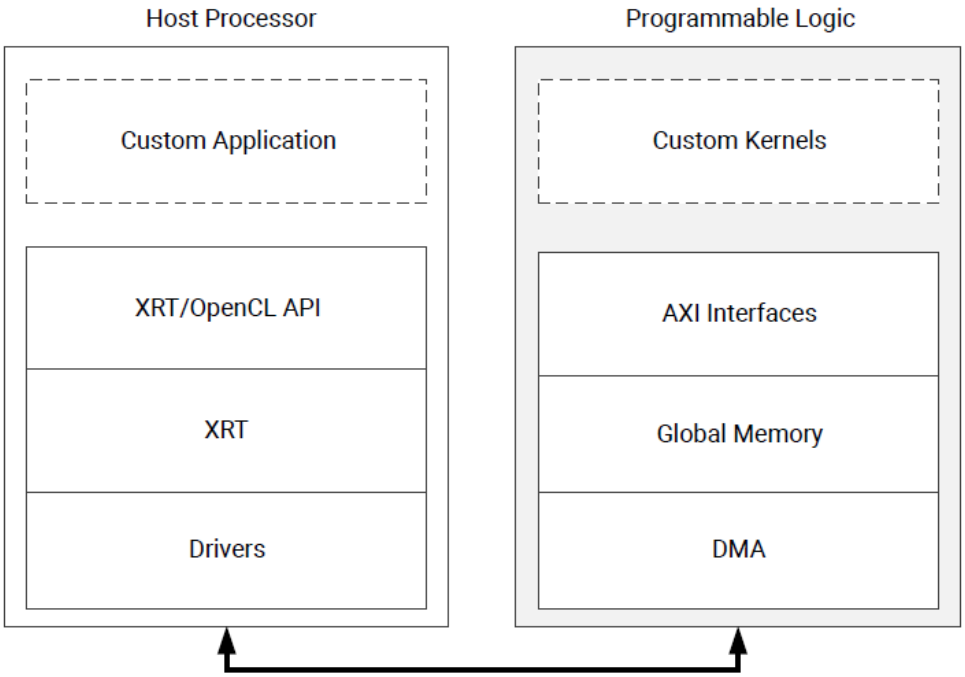
Όπως περιγράψαμε στο 1ο εργαστήριο, στο Vitis ένα πρόγραμμα χωρίζεται μεταξύ μιας εφαρμογής CPU (host) και ενός πυρήνα επιταχυνμένου υλικού συνδεδεμένα με ένα κανάλι επικοινωνίας μεταξύ τους. Το CPU, είναι γραμμένο σε C / C ++ και χρησιμοποιώντας API όπως το OpenCL, εκτελείται σε αρχιτεκτονικής x86 ή ARM για ενσωματωμένες πλατφόρμες, ενώ οι πυρήνες με επιτάχυνση υλικού (hardware accelerated kernels) εκτελούνται εντός της περιοχής προγραμματιζόμενης λογικής (PL) του FPGA.

Οι κλήσεις API, που διαχειρίζεται το XRT, χρησιμοποιούνται για την επεξεργασία συναλλαγών μεταξύ του προγράμματος της CPU και των επιταχυντών υλικού. Η επικοινωνία μεταξύ της CPU και του επιταχυντή, συμπεριλαμβανομένου του ελέγχου και της μεταφοράς δεδομένων, πραγματοποιείται μέσω του διαύλου PCIe ή ενός διαύλου AXI για ενσωματωμένες πλατφόρμες. Ενώ οι πληροφορίες ελέγχου μεταφέρονται μεταξύ συγκεκριμένων τοποθεσιών μνήμης στο υλικό, η καθολική (global) μνήμη χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ του προγράμματος της CPU και των πυρήνων. Η καθολική μνήμη είναι προσβάσιμη τόσο από τη CPU όσο και από τους επιταχυντές υλικού, ενώ η μνήμη RAM του συστήματος της CPU είναι προσβάσιμη μόνο από την εφαρμογή που τρέχει στη CPU.

Για παράδειγμα, μια τυπική εφαρμογή, που εκτελείται στη CPU, μεταφέρει πρώτα δεδομένα από τη μνήμη RAM του συστήματος της CPU στη global μνήμη του FPGA. Ο πυρήνας της FPGA στη συνέχεια επεξεργάζεται τα δεδομένα και αποθηκεύει τα αποτελέσματα πίσω στην global μνήμη. Μετά την ολοκλήρωση του πυρήνα του FPGA, η

global memory

RAM του συστήματος της CPU μεταφέρει τα αποτελέσματα πίσω στη μνήμη RAM του συστήματος της CPU. Οι μεταφορές δεδομένων μεταξύ RAM του συστήματος της CPU και της καθολικής μνήμης εισάγουν καθυστέρηση, η οποία μπορεί να είναι δαπανηρή για τη συνολική εφαρμογή. Για να επιτευχθεί επιτάχυνση σε ένα πραγματικό σύστημα, τα οφέλη που επιτυγχάνονται από τους πυρήνες επιτάχυνσης υλικού πρέπει να υπερτερούν του πρόσθετου χρόνου της μεταφοράς δεδομένων.



Μια FPGA περιέχει τους επιταχυνόμενους πυρήνες, την καθολική μνήμη και την άμεση πρόσβαση στη καθολική μνήμη μέσω της μεθόδου Direct Memory Access (DMA). Οι πυρήνες μπορούν να έχουν μία ή περισσότερες καθολικές διεπαφές μνήμης και είναι προγραμματιζόμενες. Το μοντέλο εκτέλεσης του Vitis μπορεί να αναλυθεί στα ακόλουθα βήματα:

1. Το πρόγραμμα που εκτελείται στη CPU (host) γράφει τα δεδομένα που απαιτούνται από έναν πυρήνα στην καθολική μνήμη της συνδεδεμένης συσκευής μέσω της διεπαφής PCIe σε μια κάρτα επιταχυντή Alveo (Data Center) ή μέσω του διαύλου AXI σε μια ενσωματωμένη πλατφόρμα.

2. Το πρόγραμμα που εκτελείται στη πλευρά του host ρυθμίζει τον πυρήνα με τις παραμέτρους εισόδου του και ενεργοποιεί την εκτέλεση της λειτουργίας πυρήνα στο FPGA.
3. Ο πυρήνας που εκτελείται στη πλευρά του FPGA εκτελεί τον απαιτούμενο υπολογισμό ενώ διαβάζει δεδομένα από την καθολική μνήμη, όπως απαιτείται.

Στη συνέχεια γράφει τα δεδομένα πίσω στη καθολική μνήμη και ειδοποιεί τη CPU ότι έχει ολοκληρώσει την εργασία του.

4. Το πρόγραμμα της CPU μεταφέρει τα δεδομένα από τη καθολική μνήμη στη μνήμη RAM της CPU και συνεχίζει την επεξεργασία όπως απαιτείται.

Ο compiler του Vitis παρέχει τρεις διαφορετικές λειτουργίες, δύο εξομοίωσης που χρησιμοποιούνται για σκοπούς εντοπισμού σφαλμάτων και επικύρωσης και μια που χρησιμοποιείται για τη δημιουργία του πραγματικού πυρήνα FPGA:

- **Software Emulation** (sw_emu): Τόσο ο κώδικας εφαρμογής που προορίζεται για τη CPU όσο και ο κώδικας του πυρήνα της FPGA γίνονται compile στη CPU. Η λειτουργία αυτή είναι χρήσιμη για τον εντοπισμό σφαλμάτων σύνταξης (syntax error), τον εντοπισμό σφαλμάτων του κώδικα του πυρήνα που εκτελείται μαζί με την εφαρμογή και την επαλήθευση της συμπεριφοράς του συστήματος και ολόκληρης της εφαρμογής.
- **Hardware Emulation** (hw_emu): Ο κώδικας του πυρήνα μετατρέπεται σε ένα μοντέλο υλικού (RTL), το οποίο εκτελείται σε έναν αποκλειστικό προσομοιωτή. Η διαδικασία αυτή διαρκεί περισσότερο, αλλά παρέχει μια λεπτομερή, ακριβή προβολή της δραστηριότητας του πυρήνα. Αυτή η λειτουργία είναι χρήσιμη για τον έλεγχο της λειτουργικότητας της λογικής που θα πάει στο FPGA και για τη λήψη αρχικών εκτιμήσεων απόδοσης.
- **Hardware** (hw): Ο κώδικας του πυρήνα μεταγλωττίζεται σε ένα μοντέλο υλικού (RTL) και στη συνέχεια υλοποιείται στο FPGA, με αποτέλεσμα να δημιουργείται ένα binary file που θα τρέχει στο πραγματικό FPGA.

2. Εισαγωγή στο Vitis (20%)

Διαβάστε προσεκτικά το «Introduction to Vitis» που βρίσκεται στη κατηγορία «Εργασίες και Τεστ» του elearning και εκτελέστε όλα τα βήματα του παραδείγματος (vadd) που αναφέρονται περνώντας μόνο από τα **2** στάδια/λειτουργίες του Vitis (**Software Emulation/ Hardware Emulation**). **ΠΡΟΣΟΧΗ**: Παρακαλείστε να **ΜΗΝ** εκτελέσετε τα βήματα που βρίσκονται στη σελίδα **13** (**Hardware**) όπου γίνεται place & route όλος ο κώδικας και παράγεται το bitstream για το Alveo U200 καθώς είναι μια αρκετά απαιτητική υπολογιστική διαδικασία (~3ώρες) και θα έχει συνέπειες στην ανταπόκριση του server. Ωστόσο μπορείτε αν θέλετε να διαβάσετε τη σελίδα αυτή για να διευρύνετε τις γνώσεις σας.

Παρατήρηση: Παρακαλώ πολύ κάθε ομάδα να δημιουργήσει ΕΝΑ MONO project στο server καθώς υπάρχει πρόβλημα χώρου.

3. Υλοποίηση του accelerator που δημιουργήσατε στο 1ο εργαστήριο στο Vitis (80%)

Αφού εκτελέσετε όλα τα βήματα με επιτυχία του παραδείγματος «vadd» που περιγράφεται στο «Introduction to Vitis», θα ενσωματώσετε στο Vitis τον επιταχυντή IMAGE_DIFF_POSTERIZE που αναπτύξατε στο 1ο εργαστήριο, επεκτείνοντάς τον ώστε να εφαρμόζει επιπλέον ένα φίλτρο 3×3 στην εικόνα διαφορών. Όπως είδατε στο «vadd» η πλευρά του host είναι υλοποιημένη με OpenCL. Η εμπειρία μας καθώς και τα παραδείγματα της Xilinx δείχνουν ότι το Vitis λειτουργεί χρησιμοποιώντας OpenCL κώδικα στη πλευρά του host αντί για C/C++. **Hint: Προτείνουμε να χρησιμοποιήσετε ως αναφορά το κώδικα του vadd του παραδείγματος κάνοντας τροποποιήσεις πάνω σε αυτό (όχι να δημιουργήσετε από την αρχή τα αρχεία).**

Για τις ανάγκες των εργαστηρίων, θεωρούμε σταθερές τιμές π.χ. $T_1 = 32$και $T_2 = 96$, όπως ορίστηκαν στο Εργαστήριο 1.

Για λόγους απλότητας και ομοιομορφίας με το παράδειγμα vadd, μπορείτε αν θέλετε, να θεωρήστε ότι τα δεδομένα εικόνας υλοποιούνται με 32-bit ακέραιους τύπους (π.χ. int), παρότι εννοιολογικά οι τιμές των pixels είναι στο [0, 255]. Φροντίστε οι ενδιάμεσοι υπολογισμοί (αφαιρέσεις, πολλαπλασιασμοί, αθροίσματα) να γίνονται σε επαρκές εύρος τιμών και το τελικό αποτέλεσμα να "κόβεται" (clip) στο [0, 255] πριν γραφτεί στην έξοδο.

Για να μην απαιτείται υπερβολικός χρόνος στη Hardware Emulation, επιλέξτε σχετικά μικρές διαστάσεις εικόνας, π.χ. HEIGHT = WIDTH = 64 ή HEIGHT = WIDTH = 128.

Προσθήκη φίλτρου 3×3 (sharpen) στην εικόνα διαφορών

Στο παρόν εργαστήριο θα επεκτείνετε τον επιταχυντή σας έτσι ώστε, αφού υπολογιστεί η C, να εφαρμόζεται πάνω της ένα φίλτρο 3×3 και να παράγεται ένας τελικός πίνακας εξόδου C_filt, ίδιων διαστάσεων με την A, B και C.

Για τις ανάγκες του εργαστηρίου, θεωρούμε ένα σταθερό φίλτρο οξύτητας (sharpen):

$$F = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Για κάθε εσωτερικό pixel (i, j), με $1 \le i \le HEIGHT-2$ και $1 \le j \le WIDTH-2$, ορίζουμε:

$$S(i,j) = \sum_{a=-1}^{1} \sum_{b=-1}^{1} F_{a+1,b+1} \cdot C[i+a][j+b]$$

και στη συνέχεια:

$$C_{\text{filt}}[i][j] = \text{clip}(S(i,j),0,255)$$

όπου η συνάρτηση clip(x, 0, 255):

- επιστρέφει 0, αν x < 0
- επιστρέφει 255, αν x > 255
- αλλιώς επιστρέφει x.

Για τα pixels στα σύνορα της εικόνας (π.χ. i = 0, i = HEIGHT−1, j = 0, j = WIDTH−1) μπορείτε, για λόγους απλότητας, να επιλέξετε μία από τις ακόλουθες πολιτικές:

- C_filt[i][j] = C[i][j] (αντιγραφή της τιμής της εικόνας διαφορών), ή
- C_filt[i][j] = 0.

Η επιλογή που θα κάνετε πρέπει:

- να είναι ίδια στη S/W reference υλοποίηση και στον HLS κώδικα,
- να αναφέρεται ρητά στο κείμενο της αναφοράς σας.

Πρακτικά, το φίλτρο αυτό:

- ενισχύει το κεντρικό pixel σε σχέση με τους γείτονές του,
- τονίζει τις απότομες αλλαγές (άκρα) στην εικόνα C,
- κάνει τις περιοχές με σημαντικές διαφορές μεταξύ A και B πιο "κοφτές" και ευδιάκριτες στην τελική εικόνα C_filt.

Ο κώδικας host θα πρέπει επίσης να υπολογίζει τα αποτελέσματα και με μια καθαρά S/W υλοποίηση (στον CPU) και να ελέγχει ότι ο πίνακας εξόδου C_filt που επιστρέφει ο επιταχυντής στο FPGA συμφωνεί στοιχειομετρικά με τη S/W υλοποίηση (π.χ. να εκτυπώνεται «Test Passed» όταν δεν υπάρχουν διαφορές).

ΠΡΟΣΟΧΗ: Παρακαλείστε και εδώ να **σταματήσετε στα 2** στάδια/λειτουργίες του Vitis (**Software Emulation/ Hardware Emulation**) για τους λόγους που αναφέραμε παραπάνω.

Περισσότερες πληροφορίες για το Vitis μπορείτε να βρείτε εδώ[1]

Αφού ολοκληρώσετε τα παραπάνω βήματα, επιλέξτε Hardware Emulation, τρέξτε το project σας και ανοίξτε το Vitis Analyzer (vadd_system > vadd > Emulation-HW > SystemDebugger_vadd_system_vadd > Run Summary (xclbin)).

Καταγράψτε τις τιμές που πήρατε από το Profile Summary για τα ακόλουθα:

i) Kernels & Compute Units → Kernel Execution
ii) Kernel Data Transfers → Top Kernel Data Transfer
iii) Host Data Transfer → Host Transfer

[1] https://docs.xilinx.com/r/2022.2-English/ug1393-vitis-application-acceleration/Getting-Started-with-Vitis