Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Αναδιατασσόμενα Ψηφιακά Συστήματα

Υλοποίηση Αλγόριθμου σε Hardware με τη χρήση τεχνικών high-level synthesis και του περιβάλλοντος SDSoC.

Ομάδα LAB59142109

Τρίμας Χρήστος	***2016030054***
Πίσκοπος Διονύσης	***2015030115***

Σκοπός εργαστηριακής άσκησης:

Στην συγκεκριμένη άσκηση κληθήκαμε να υλοποιήσουμε έναν αλγόριθμο σε αναδιατασσόμενη λογική με χρήση τεχνικών high level synthesis, και του περιβάλλοντος SDSoC της Xilinx. Με τον επιτυχή σχεδιασμό του επιταχυντή, έγινε δοκιμή του κώδικα πάνω στο Zedboard, μια πλακέτα της οικογένειας Zynq-7000 APSoC.

Γενικά:

Το πρότζεκτ κατανεμήθηκε σε 3 φάσεις. Η 1η αφορούσε την κατανόηση του κώδικα C και τροποποιήσεις αυτού για καλύτερη αποτύπωση του κώδικα σε αναδιατασσόμενη λογική. Η 2η αφορούσε τον εσωτερικό σχεδιασμό του επιταχυντή και την εφαρμογή directives μέσω του Vivado HLS. Η 3η και τελευταία, αφορούσε την δημιουργία ενός πραγαμτικού περιβάλλοντος εκτέλεσης μέσω του SDSoC και την χρήση SDS directives για μεταφορά δεδομένων.

Εκτέλεση:

Αφού πραγματοποιήθηκαν οι κατάλληλες προσθήκες στο header file που δόθηκε σαν reference code από τον διδάσκοντα(ορισμός της συνάρτησης myFunc_Accel, ονομασία της βιβλιοθήκης), πραγματοποιήθηκαν αλλαγές στον κώδικα της συνάρτησης main, με σκοπό την μέτρηση του χρόνου εκτέλεσης της συνάρτησης myFunc, και της myFunc_Accel, καθώς επίσης έγινε και έλεγχος ορθότητας των αποτελεσμάτων μεταξύ των δύο συναρτήσεων. Σε πρώτη φάση, η συνάρτηση myFunc_Accel, αποτελεί αντιγραφή της myFunc και κατά την διάρκεια των διαφόρων φάσεων της παρούσας εργασίας, τροποποιήθηκε με σκοπό την επιτάχυνση της.

Περιγραφή της λειτουργίας της συνάρτησης myFunc():

Η συνάρτηση myFunc, χρησιμοποιεί ως ορίσματα τις μεταβλητές size,threshold,dim,*data0,*data1,*data2 και σκοπός της είναι να εκχωρεί σε dim στοιχεία του πίνακα data2, το γινόμενο ενός συγκεκριμένου συνδυασμού στοιχείων των πινάκων data0 &data1, **αν και μόνο αν** προκύψει ένα στοιχείο του data2 ανάdim-αδα, τέτοιο ώστε να είναι μικρότερο του threshold, αλλιώς η dim-άδα παίρνει την τιμή 0. Η προαναφερόμενη διαδικασία, εκτελείται size φορές, δημιουργώντας ένα πίνακα data2 με size*dim στοιχεία.

Η κατανόηση του αλγορίθμου λειτουργίας της συνάρτησης myfunc() θα αποτελέσει και το βασικό αντικείμενο μελέτης στο συγκεκριμένο πρότζεκτ, καθώς σκοπός μας είναι να τροποποιήσουμε το αλγόριθμο έτσι ώστε να προσδίδουμε την βέλτιστη εκτέλεση του προγράμματος και κατανομή πόρων στην FPGA πλακέτα.

Περιγραφή της αρχιτεκτονικής του επιταχυντή:

Η αρχιτεκτονική του επιταχυντή, χωρίζεται σε 4 κομμάτια:

-Αρχικοποίηση των dim στοιχείων του data2.
-Εκχώρηση στα στοιχεία του data2 το γινόμενο των data0-data1.
-Έλεγχος αν κάποιο από τα στοιχεία του data2 είναι μικρότερο από το threshold.
-Εάν δεν ισχύει η συνθήκη, μηδενισμός όλων των στοιχείων της dim-αδαςdata2.

Επιπλέον, χρειάζεται να αναλυθούν οι εξαρτήσεις στην συγκεκριμένη αρχιτεκτονική μεταξύ των μεταβλητών της, καθώς έτσι θα προβούμε στις απαραίτητες αλλαγές που καθιστούν τον επιταχυντή ταχύτερο και λιγότερο σπάταλο σε πόρους. Συγκεκριμένα, παρατηρούμε πως κάθε στοιχείο της dim-αδαςdata2 εξαρτάται από το αποτέλεσμα των data1*data0 (συγκεκριμένα στοιχεία πινάκων ανά loop), συνεπώς χρειάζεται να ολοκληρωθούν όλες οι εκχωρήσεις από την επανάληψη για να πάμε στο βήμα ελέγχου συνθήκης, που βλέπουμε αν κάποιο στοιχείο είναι μικρότερο του threshold. Στο προτελευταίο βήμα, χρειάζεται να ολοκληρωθεί πλήρως και πάλι η λούπα (στην χειρότερη περίπτωση), έτσι ώστε να προχωρήσουμε στον μηδενισμό των στοιχείων ή όχι.

Από την προηγούμενη ανάλυση καταλαβαίνουμε πως η υπάρχουσα ροή του προγράμματος δεν επιτρέπει την εύκολη παράλληλη εκτέλεση του, καθώς επίσης και ότι χρησιμοποιούνται πολλοί πόροι (μεταβλητές και επαναλήψεις) που καθυστερούν σημαντικά το throughput του επιταχυντή.

Τέλος, πάρθηκε η σχεδιαστική απόφαση ο επιταχυντής για dim=4 και για dim=16, να διαχωριστεί, καθώς αν ο σχεδιασμός γινόταν από κοινού θα προκύπταν προβλήματα, τόσο ως προς την απαίτηση πόρων (ακόμα και για πράξεις με dim=4, το latency θα ήταν ίδιο με αυτό για dim=16, άρα μεγαλύτερη κατανάλωση πόρων), όσο και προς την εφαρμογή directives από το HLS (προβλήματα με το pipelining και το loopunrolling των βρόγχων).

Δικαιολόγηση σχεδιαστικών αποφάσεων και τροποποιήσεων του κώδικα:

Αρχικά χρειάζεται να τονιστεί πως δοκιμάστηκαν αρκετές προσεγγίσεις ώστε να καταλήξουμε σε ποιο design του επιταχυντή θα είναι καλύτερα εφαρμόσιμο στην πλακέτα FPGA. Για να επιτευχθεί παραπάνω πρόκληση, ο κώδικας τροποποιήθηκε κατάλληλα σε software λογική καθώς επίσης και σε HLS&SDSoC περιβάλλον (τα τελευταία 2 θα αναλυθούν στα επόμενα μέρη). Οι αρχικές τροποποιήσεις στον κώδικα της myFunc() είχαν ως βασικό κίνητρο την μείωση χρόνου εκτέλεσης και πόρων του επιταχυντή. Πιο συγκεκριμένα, αρχικά καταργήθηκε η αρχικοποίηση των στοιχείων του data2 σε 0, καθώς στο τέλος της συνάρτησης καταχωρούμε ένα αποτέλεσμα ανεξάρτητα προηγούμενης τιμής (0 ή γινόμενο data1-data0). Στην συνέχεια, συνδυάζουμε τις 2 πρώτες επαναλήψεις όπου επιτυγχάνεται η εκχώρηση των στοιχείων του data2 (έχουν τις ίδιες επαναληπτικές συνθήκες), και δημιουργούμε μόνο 2 επαναλήψεις αντί για 3, όπως φαίνεται στην εικόνα παρακάτω. Επιπλέον, καταργούμε την επανάληψη όπου καταχωρούμε την τιμή στην σταθερά r, που ανιχνεύει αν υπάρχει στοιχείο της dim-αδαςdata2 που είναι μεγαλύτερο του threshold. Στην θέση του τοποθετήθηκε μία μεταβλητή flag , μέσα στην λούπα εκχώρησης

τιμών στο πίνακα data2, που παίρνει την τιμή 1 εάν ανιχνευθεί η παραπάνω συνθήκη. Τέλος, έπειτα από διερεύνηση των αναγκών του κώδικα σε περιβάλλον HLS&HLx, που θα αναλυθεί εκτενώς στα επόμενα κομμάτια, δημιουργήθηκαν δύο local πίνακες floatd0[16] &floatd2[4] με σκοπό την μείωση χρόνου εκτέλεσης του αλγορίθμου. Το τελικό αποτέλεσμα του επιταχυντή χωρίς HLS&SDSoC pragmas παρατίθεται στην ακόλουθη εικόνα:

```
void myFunc_Accel_4 (unsigned int size, unsigned int dim, dataType_t threshold, dataType_t * data0, dataType_t * data1, dataType_t * data2)
{
    float d0[16];
    for (  k = 0 ; k < 4 ; k ++ )
        {
        d0[k*4+0]=data0[ k * 4 + 0 ];
        d0[k*4+1]=data0[ k * 4 + 1 ];
        d0[k*4+2]=data0[ k * 4 + 2 ];
        d0[k*4+3]=data0[ k * 4 + 3 ];
        }

    for (  i = 0 ; i < size ; i ++ )
    {
        int flag=0;
        float d2[4];

        for (  k = 0 ; k < 4 ; k ++ )
        {

            d2[k]=0;
            for(l=0; l<4 ; l++)
                d2[k] += d0[k*4+l]*data1[i*4+l];

            if(d2[k]<threshold)
                flag=1;
        }
        for (  k = 0 ; k < 4 ; k ++ )
            data2 [i*4+k] = d2[k]*flag;
     }
}
```

Εικόνα 1*: **Τελικός Κώδικας χωρίς HLS & SDSoC pragmas.***

Περιγραφή Αρχιτεκτονικής Zynq-7000 AP-SoC:

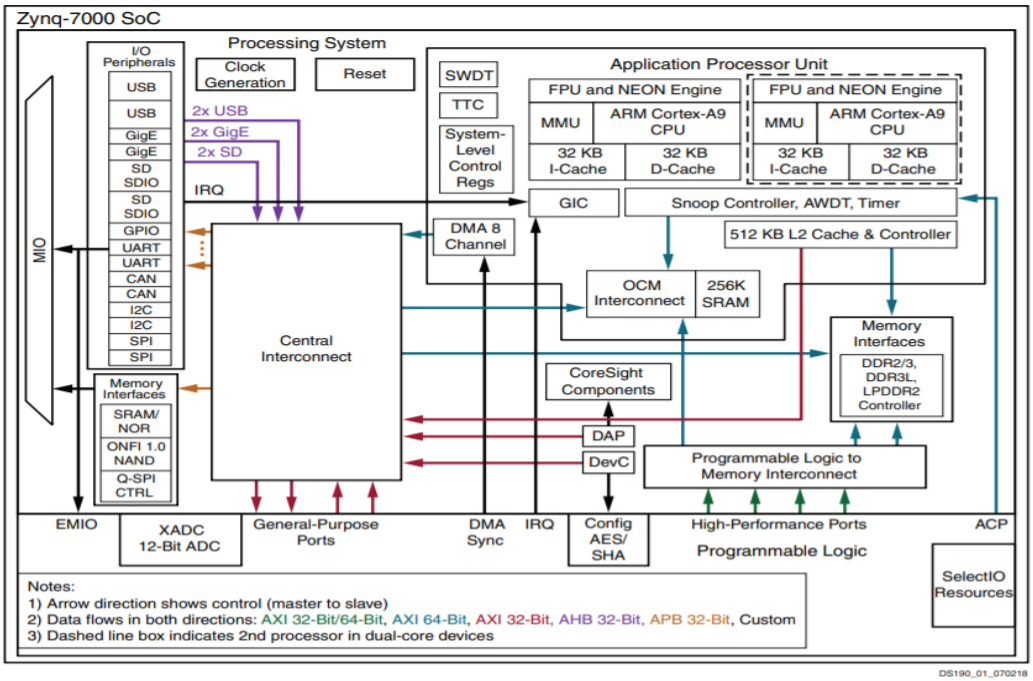
Η οικογένεια των development boards Zynq-7000, βασίζεται στην SystemonChip αρχιτεκτονική της Xilinx και υποστηρίζει τόσο software, όσο και hardware προγραμματισμό. Για την ακρίβεια, περιλαμβάνει ετερογενείς υπολογιστικές πλατφόρμες που συνδυάζουν τα χαρακτηριστικά ενός ARM επεξεργαστή και μιας FPGA. Συγκεκριμένα, «πάνω» στο board, είναι ενσωματωμένος ένας διπύρηνος ARM(Cortex-A9) επεξεργαστήςκαι αποτελεί την καρδιά του επεξεργαστικού συστήματος (το processingsystem ή αλλιώς PS του board) και 28 nanometersεπαναπρογραμαματιζόμενης λογικής (programmablelogic ή PL του board).

Αναλυτικότερα, το processingsystem(softwareprogrammability της συσκευής), αποτελείται από τον προαναφερόμενο διπύρηνο επεξεργαστή, ενώ επίσης περιλαμβάνει on-chip μνήμη, καθώς επίσης και διεπαφές για επικοινωνία με περιφερειακά ή εξωτερική μνήμη. Εκτός αυτών, περιλαμβάνει και κρυφές μνήμες cache. Δυοcache επιπέδου 1, ξεχωριστά ή κάθε μια για την κάθε cpu, και μια επιπέδου 2, την οποία μοιράζονται οι επεξεργαστές. Τέλος, αξιοσημείωτο είναι το γεγονός ότι το processingsystem είναι εξολοκλήρου αυτόνομο από το PL, δηλαδή μια συσκευή Zynq-7000 μπορεί να λειτουργήσει σαν ένας οποιοσδήποτε επεξεργαστής και δεν είναι αναγκαία η διαμόρφωση του PL.

Ο προγραμματισμός του Hardware, γίνεται μέσο της επαναπρογραμματιζόμενης λογικής, η οποία χρησιμοποιείται για να επεκτείνει τις δυνατότητες του ARM επεξεργαστή προσθέτοντας περιφερειακές συσκευές ή/και επιταχυντές. Με αυτό τον τρόπο δίνεται η δυνατότητα στον σχεδιαστή να «δημιουργήσει» τον δικό του επεξεργαστή, ο οποίος θα κάνει target αποκλειστικά την ζητούμενη εφαρμογή του.

Τέλος, παρέχονται αρκετοί τρόποι για την σύνδεση των διαφόρων components εντός ή εκτός του board. Για παράδειγμα, το PS περιέχει μονάδες, αφιερωμένες για είσοδο/έξοδο οι οποίες μοιράζονται μεταξύ της στατικής μνήμης και των διαφόρων περιφερειακών του

processingsystem. Ωστόσο, καθώς ο αριθμός είναι fixed και μπορεί μια εφαρμογή να απαιτεί όλες τις περιφερειακέςμονάδες που είναι διαθέσιμες, υπάρχει μια «γέφυρα» μεταξύ των περιφερειακών και του PL, έτσι ώστε να είναι δυνατή η χρήση του programmable logic I/Os, για την σύνδεση όλων των περιφερειακών με τον έξω κόσμο.



Εικόνα 2: ***Προεπισκόπηση της Αρχιτεκτονικής***

HighLevelSynthesis(HLS) directives:

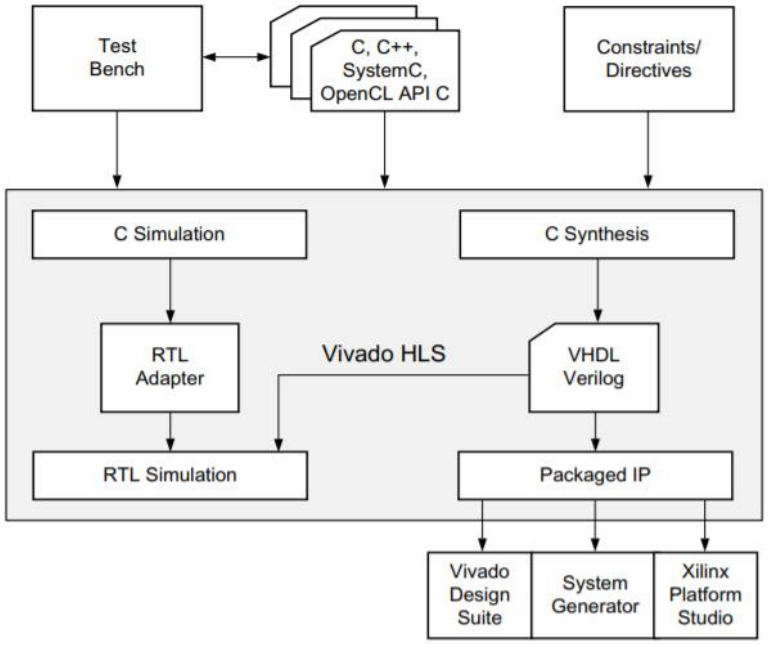
Η σύνθεση υψηλού επιπέδου, είναι μια αυτοματοποιημένη διαδικασία σχεδιασμού, που μεταφράζει τον αλγόριθμο μας από γλώσσα C σε γλώσσα περιγραφής υλικού RegisterTransferLevel(RTL) και συντίθεται στην FPGA. Στην ουσία πρόκειται για τον εσωτερικό σχεδιασμό του επιταχυντή μας και πραγματοποιείται με το εργαλείο της Xilinx,VivadoHLS.

Καθώς η αρχιτεκτονική του επεξεργαστή είναι προκαθορισμένη, το VivadoHLS, έχει ως σκοπό, με την βοήθεια των #pragma, να κατευθύνει τον compiler σχετικά με την αρχιτεκτονική της FPGA και να παρέχει μια σειρά από οδηγίες για την βελτιστοποίηση της σύνθεσης και διαμόρφωσης προς το επιθυμητό αποτέλεσμα.

Ειδικότερα, αφούέγινε απλοποίηση του αρχικού κώδικα C και επαλήθευση του μέσω Csimulation στο VivadoHLS, πραγματοποιήθηκε ο καθορισμός των διεπαφών. Συγκεκριμένα, μέσω του pragma HLS interface, καθορίστηκε ο τρόπος δημιουργίας των θυρώνRTL της συνάρτησης. Για τον επιταχυντή μας επιλέχτηκεap_busmode, το οποίο υλοποιεί δείκτες ή pass-by-referenceθύρες, σαν διεπαφή διαύλου για την ανταλλαγή των δεδομένων μεταξύ των components(π.χ. dataType_t* data0, είναι δείκτης για αυτό και η διεπαφή του είναι τύπου ap_bus).

Το επόμενο στάδιο της διαδικασίας της βελτιστοποίησης είναι να αξιοποιήσουμε όσο το δυνατόν περισσότερο παραλληλισμό γίνεται. Αυτό επιτυγχάνεται μέσω της

«εντολής»PIPELINE σε βρόγχους. Στην πραγματικότητα, αυτό που πραγματοποιείται είναι να μειωθεί το διάστημα έναρξης (InitiationInterval), επιτρέποντας την παράλληλη εκτέλεση λειτουργιών εντός ενός βρόγχου. Είναι αναγκαίο να τονιστεί, ότι το HLS από μόνο του κάνει PIPELINE στους βρόγχους του κώδικα και δεν χρειάζεται επανάληψη της εντολής για κάθε βρόγχο. Στην ουσία, από μόνο του βρίσκει τον ιδανικό τρόπο για διοχετεύση.



Εικόνα 3: ***VivadoHLS***

Για την σχεδίαση της αρχιτεκτονικής του επιταχυντή, μόνο PIPELINE και INTERFACEpragmasχρησιμοποιήθηκαν, ωστόσο, κατά την διάρκεια του synthesis, το VivadoHLS μόνο του πραγματοποιεί loopunrolling στους βρόγχους του κώδικα C.

SDSystem on Chip (SDSoC) directives:

Αφού έχει ολοκληρωθεί ο εσωτερικός σχεδιασμός του επιταχυντή με την χρήση του VivadoHLS, πρέπει να καθοριστούν οι τρόποι μεταφοράς των δεδομένων μέσα στην FPGA, αλλά και το που θα αποθηκευτούν για βελτίωση της απόδοσης.

Η Xilinx για την μεταφορά δεδομένων μεταξύ του PS-PL καθώς και με προσυσκευασμένα τμήματα VHDL κώδικα (IPcores), χρησιμοποιεί το πρωτόκολλο επικοινωνίας AXI (AdvancedeXtensibleInterface). Το συγκεκριμένο πρωτόκολλο είναι μέρος της ARMAdvancedMicrocontrollerBusArchitecture (AMBA), που είναι μια οικογένεια μικροελεγκτών διαύλων.

Για την μεταφορά λοιπόν των δεδομένων έγινε χρήση SDSdirectives (#pragmas) και συγκεκριμένα χρησιμοποιήθηκαν:

1) **#pragma SDS data_access_pattern.** Με την χρήση του, δηλώθηκε ο τρόπος με τον οποίο προσπελάζονται τα δεδομένα και με την επιλογή SEQUENTIAL, γίνεται διαδοχικά για τις «δομές» μας.

2) **#pragma SDS data copy.** Με την χρήση του, αντιγράφουμε τα data0, data1, data2 στην BRAM. Το datacopy στην προκειμένη περίπτωση είναι αποτελεσματικό, σε άλλες περιπτώσεις όμως, όπου τα στοιχεία είναι αρκετά περισσότερα (>16384) η ντριρεκτίβαdatacopy δεν είναι αποτελεσματική.
3) **#pragma SDS data mem_attribute.** Με την χρήση του δηλώθηκε ο τρόπος αποθήκευσης των δεδομένων στο σύστημα επεξεργασίας. Το CACHEABLE, σημαίνει πως ο compiler πρέπει να διατηρεί την συνοχή της κρυφής μνήμης μεταξύ CPU και Accelerator, για την μνήμη που αντιστοιχεί στον πίνακα. Το PHYSICAL_CONTIGUOUS, σημαίνει ότι η μνήμη που αντιστοιχεί στο σχετικό πίνακα(data2) έχει κατανεμηθεί χρησιμοποιώντας το sds_alloc (αντίστοιχη εντολή με την malloc για δέσμευση χώρου για χρήση από το SDSoC), και είναι φυσική συνεχής μνήμη.
4) **#pragma SDS data_mover.** Με την χρήση του, επιλέχτηκε ο μεταφορέας των δεδομένων και συγκεκριμένα από το πρωτόκολλο AXI, επιλέχτηκε AXIDMA_SIMPLE, το οποίο παρέχει άμεση πρόσβαση μνήμης υψηλής ταχύτητας, μεταξύ μνήμης και περιφερειακών.

Σε αυτό το σημείο να σημειωθεί ότι, για να γίνει επιτυχές build από το SDSoC, χρειάστηκε να γίνει import στο my_lib.h η βιβλιοθήκη sds_lib.h, η οποία περιλαμβάνει τις συναρτήσεις sds_alloc και sds_free, αντίστοιχες των malloc και free της στάνταρ βιβλιοθήκης της C.

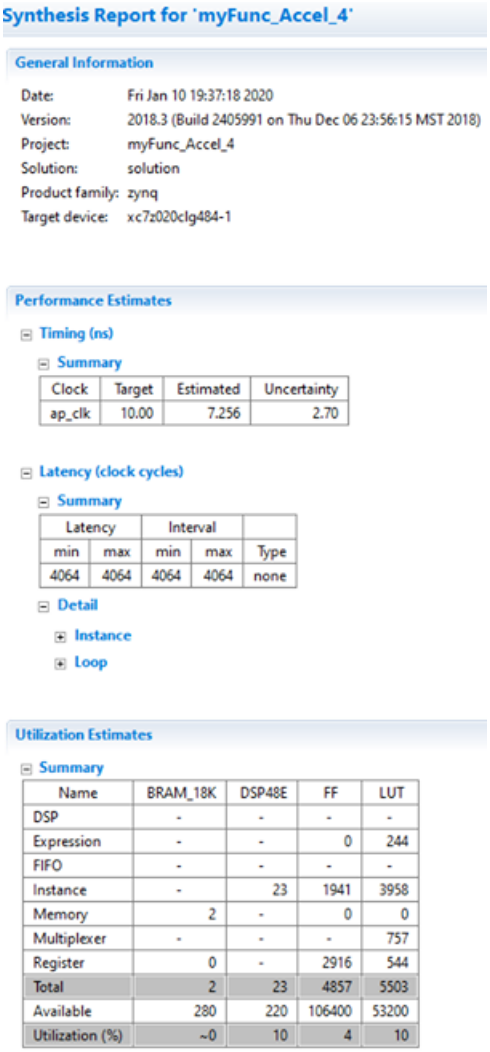
Αξιολόγηση απόδοσης:

Για την επιβεβαίωση της ορθότητας της σχεδίασης της αρχιτεκτονικής του επιταχυντή, κατά την διάρκεια του synthesis, το VivadoHLS, παρέχει πληροφορίες σχετικά με το latency, το initiation interval και τους πόρους που απαιτεί ο επιταχυντής.

Latency, είναι η καθυστέρηση δράσης, δηλαδή ο αριθμός των κύκλων ρολογιού που χρειάζεται για να ολοκληρωθεί ένα σετ εντολών και να παραχθεί μια τιμή αποτελέσματος της εφαρμογής. Το latency αποτελεί βασική μέτρηση της απόδοσης τόσους στους επεξεργαστές, όσο και στις FPGA.

Το θεωρητικό ελάχιστο II, είναι για dim=4, ίσο με 4000 + x, με x<100. Από την διαδικασία της σύνθεσης, παρατηρείται ότι αυτό το φράγμα επιτυγχάνεται που σημαίνει ότι η σχεδίαση του επιταχυντή μας είναι αρκετά ικανοποιητική και εντός των σχεδιαστικών προδιαγραφών.

Παρακάτω παρατίθεται μια εικόνα τόσο από το II, όσο και από τους πόρους τους οποίους απαιτεί η σχεδίαση μας.



Synthesis Report for 'myFunc_Accel_4'

General Information

Date: Fri Jan 10 19:37:18 2020
Version: 2018.3 (Build 2405991 on Thu Dec 06 23:56:15 MST 2018)
Project: myFunc_Accel_4
Solution: solution
Product family: zynq
Target device: xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	7.256	2.70

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
4064	4064	4064	4064	none

Detail

Instance

Loop

Utilization Estimates

Summary

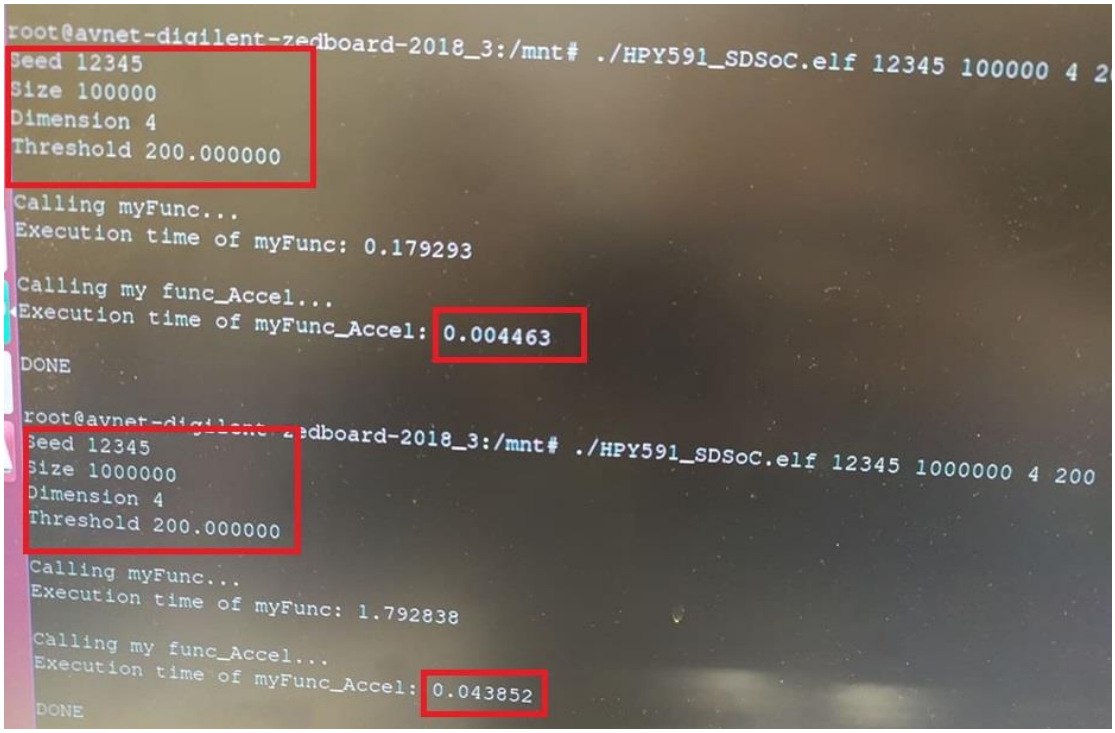
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	244
FIFO	-	-	-	-
Instance	-	23	1941	3958
Memory	2	-	0	0
Multiplexer	-	-	-	757
Register	0	-	2916	544
Total	2	23	4857	5503
Available	280	220	106400	53200
Utilization (%)	~0	10	4	10

Εικόνα 4: ***HLS Synthesis Performance Estimation***

Τέλος, αφού έγινε επιτυχώς build το image από το SDSoC, τεστάραμε τον επιταχυντή στο Zedboard. Συγκεκριμένα, ως μέτρο απόδοσης θεωρήθηκε το throughput, το οποίο ισούται με τον αριθμό των κύκλων ρολογιού που χρειάζεται η προγραμματιζόμενη λογική για να δεχτεί κάθε επόμενο δείγμα δεδομένων εισόδου. Άρα, σημαντικό παράγοντα σε αυτή την τιμή αποτελεί η συχνότητα ρολογιού. Για την προσομοίωση του περιβάλλοντος χρησιμοποιήθηκε ρολόι 100MHz και για dimension ίσο με 4, το θεωρητικό maximumthroughput είναι ισο με 25 εκατ.

Στις δικές μας δοκιμή, για size=100,000, ο χρόνος εκτέλεσης της myFunc_Accel είναι ίσος με 0.004463 και το throughput είναι ίσο με 22,406,453, ενώ για size=1,000,000, ο χρόνος εκτέλεσης είναι ίσος με 0.043852 και το throughputείναι ίσο με 22,803,997.

Και στις δυο περιπτώσεις, παρατηρείται ότι ο επιταχυντής μας είναι σημαντικά ταχύτερος από την myFunc και μάλιστα προσεγγίζει αρκετά την μάξιμουμ θεωρητική τιμή των 25 εκατομμυρίων .



Εικόνα 5: ***Testing Accelerator on Zedboard.***

<u>Συμπεράσματα:</u>

Βασικό συμπέρασμα από το συγκεκριμένο πρότζεκτ, αποτελούν οι δυνατότητες που προσφέρει μία FPGA στην υλοποίηση ενός αλγόριθμου, από άποψης ταχύτητας και πόρων, που αποδεικνύεται από τις παραπάνω μελέτες. Ωστόσο, μίζονος σημασίας αποτελεί η ορθή αλγοριθμική υλοποίηση του προγράμματος, καθώς επίσης και η σωστή χρήση των συναρτήσεων των περιβάλλοντων προγραμματισμού του αναδιατασσόμενου κομματιού της FPGA, όπως στην περίπτωσή μας του vivado HLS & HLx. Τέλος, ως κυριότερο συμπέρασμα εξάγουμε ότι η τεχνολογία των FPGA ήρθε για να μείνει, καθώς ανταποκρίνεται σε όλο το φάσμα πολυπλοκότητας των σημερινών διεργασιών, είναι ευέλικτη, φθηνή και επαναπρογραμματίζεται με μεγάλη ευκολία, καθιστώντας την εύκολα ανταγωνίσιμη στην σημερινή αγορά ένταντι σε υπάρχουσες τεχνολογίες, όπως ASICs και GPUs.

Βιβλιογραφία:

Πηγές στις οποίες βασιστήκαμε τόσο για την κατανόηση του πρότζεκτ και των εργαλείων, όσο και για τον σχεδιασμό και την βελτιστοποίηση του επιταχυντή.

<u>Αρχιτεκτονική και προδιαγραφές του Zedboard:</u>

1) https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf
2) https://www.xilinx.com/video/soc/zynq-hardware-architecture-highlights.html

Vivado High Level Synthesis και directives αυτού:

1) https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug902-vivado-high-level-synthesis.pdf
2) https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/okr1504034364623.html

Software Defined System on Chip και directives αυτού:

1) https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/nmc1504034362475.html
2) https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1027-sdsoc-user-guide.pdf
3) https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1235-sdsoc-optimization-guide.pdf