# HIGH LEVEL SYNTHESIS

Το high-level synthesis (HLS) αποτελεί μια επέκταση του λογισμικού Vivado το οποίο κατασκευάστηκε από την εταιρεία xillinx τον Απρίλιο του 2012. Το Vivado αποτελεί ένα λογισμικό για τη σχεδίαση και ανάλυση HDL (Hardware Description Language) κυκλωμάτων, αντικαθιστώντας το προκάτοχο λογισμικό που είχε φτιάξει το οποίο είναι το Xilinx ISΕ (Integrated Synthesis Enviroment). Τα πρόσθετα χαρακτηριστικά του vivado είναι η δυνατότητα ανάπτυξης chip καθώς και το εργαλείο high-level synthesis που θα μελετήσουμε περαιτέρω. Το Vivado αντιπροσωπεύει μια επαναγραφή και επαναξέταση της συνολικής ροής σχεδιασμού ( σε σύγκριση με το ISE) και έχει περιγραφεί από πολλούς ως «καλά σχεδιασμό, γρήγορο και διατηρήσιμο εργαλείο».

Οι επεκτάσεις που έχει το Vivado Design Suite είναι οι παρακάτω:

* Το Vivado TCL Store αποτελεί ένα σύστημα scripting για την ανάπτυξη πρόσθετων επεκτάσεων για το vivado. Μπορεί να χρησιμοποιηθεί για να προσθέσει και να τροποποιήσει τις δυνατότητες του Vivado. To TCL που προέρχεται από τις λέξεις Tool Command Language, αποτελεί μια scripting γλώσσα στην οποία βασίζεται το ίδιο το vivado. Όλες οι λειτουργίες που προσφέρει το εργαλείο μπορούν να χρησιμοποιηθούν και να ελεγχθούν από TCL scripts.
* To Vivado Simulator αποτελεί έναν προσομοιωτή το οποίο υποστηρίζει διάφορες γλώσσες προγραμματισμού, TCL scripts, κρυπτογραφημένες IP καθώς και βελτιωμένες επαληθεύσεις για το τελικό αποτέλεσμα μετά την σύνθεση.
* Το Vivado IP integrator επιτρέπει στους μηχανικούς να ενσωματώσουν και να ρυθμίσουν γρήγορα τα IP με την βοήθεια της χρήσης της μεγάλης βιβλιοθήκη IP που υποστηρίζει το εργαλείο. Το integrator είναι επίσης σχεδιασμένο ώστε να δουλεύει κυκλώματα MathWorks Simulink τα οποία όμως έχουν κατασκευαστεί μέσω των εργαλείων Xilinx System Generator και Vivado High-Level Synthesis.
* Το εργαλείο Vivado High-Level Synthesis αποτελεί έναν μεταγλωττιστή υψηλού επιπέδου που επιτρέπει στα προγράμματα που έχουν γραφτεί σε γλώσσα C, C++ και SystemC να λειτουργήσουν άμεσα στις συσκευές FPGA (Field-programmable gate array) χωρίς την χειροκίνητη δημιουργία RTL( Register Transfer Level). Το vivado HLS αυξάνει την παραγωγικότητα των προγραμματιστών, καθώς έχει επιβεβαιωθεί ότι υποστηρίζει κλάσσεις, πρότυπα, συναρτήσεις καθώς και διάφορους τελεστές της γλώσσας C++.

**Κατανοώντας το HLS**

Αυτό που κάνει το εργαλείο Xilinx Vivado HLS είναι ότι κάνει σύνθεση συναρτήσεις που είναι γραμμένες σε γλώσσα υψηλού επιπέδου σε IP Block, τα οποία μπορούν να ενσωματωθούν στο υλικό. Επειδή είναι ενσωματωμένο στο λογισμικό αυτό , μπορεί να χρησιμοποιηθεί το hls και από άλλα εργαλεία σχεδιασμού της Xilinx καθώς υποστηρίζει πολλά εργαλεία για την δημιουργία του βέλτιστης υλοποίησης του αλγορίθμου που θα γραφτεί σε γλώσσα υψηλού επιπέδου όπως C. Η διαδικασία εκτέλεσης του εργαλείου είναι οι παρακάτω:

1. Μεταγλώττιση, εκτέλεση (προσομοίωση) και αποσφαλμάτωση του αλγορίθμου C.
2. Σύνθεση του αλγορίθμου C σε υλοποίηση RTL , μπορούν να χρησιμοποιηθούν διάφορα directives για βελτιστοποίηση.
3. Δημιουργία αναλυτικών reports και ανάλυση του κυκλώματος μετά την σύνθεση.
4. Επαλήθευση της RTL υλοποίησης με την χρήση μιας ροής που θα δημιουργηθεί.
5. Μετατροπή των RTL υλοποιήσεων σε μια εύρος από IP formats.

Το Vivado HLS μπορεί να πάρει ως εισόδους τα παρακάτω:

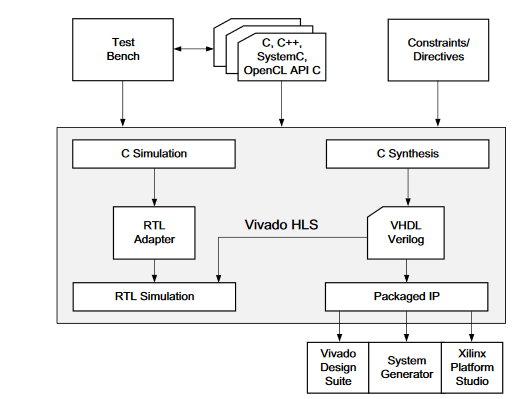
* Συναρτήσεις γραμμένες σε γλώσσα υψηλού επιπέδου όπως C, C++, SystemC και ακόμη OpenCL API C kernel. Aποτελούν την βασική είσοδο για το εργαλείο Vivado HLS. Eπιπλέον, στις συναρτήσεις μπορούν να εμπεριέχουν και άλλες υποσυναρτήσεις.
* Ορίσματα. Τα ορίσματα είναι υποχρεωτικά και χρειάζονται για να συμπεριλάβουμε στο κύκλωμα την περίοδο του ρολογιού, την απόκλιση που μπορεί να έχει το ρολόι καθώς και την πλακέτα FPGA που μπορούμε να χρησιμοποιήσουμε για να τρέξουμε τον αλγόριθμο. Η απόκλιση του ρολογιού είναι περίπου ως προεπιλογή στο 12.5 % της περίοδος του ρολογιού.
* Directives. Τα Directives αποτελούν προαιρετικά εργαλεία που μπορούν να χρησιμοποιηθούν στην σύνθεση του κυκλώματος ώστε να υλοποιήσουν μια συγκεκριμένη συμπεριφορά ή και ακόμη για να βελτιστοποιήσουν το κύκλωμα.
* Test bench και άλλα αρχεία. Το Vivado HLS χρησιμοποιεί C test bench ώστε να προσομοίωση τον αλγόριθμο πριν την σύνθεση του. Και ακόμη να επαληθεύσει τα αποτελέσματα που θα βγάλει το κύκλωμα που έχει μετατραπεί σε RTL. Για αυτήν την λειτουργία χρησιμοποιείται το C/RTL συν-προσομοίωση.

Όλα αυτά μπορούν εισαχθούν με την χρήση του Vivado HLS graphical user interface (GUI) ή και ακόμη με την χρήση tcl εντολών στο command prompt.

Οι εξόδους που παράγει το Vivado HLS είναι τα παρακάτω:

* Αρχεία υλοποίησης RTL που είναι γραμμένα σε μορφή hardware description language (HDL). Αποτελεί την βασική έξοδο του vivado HLS. Μέσω του vivado synthesis, μπορούνε να κάνουμε σύνθεση το RTL σε υλοποίηση επιπέδου πυλών (gate - level implementation) και να παράξουμε αρχείο FPGA bitstream που θα χρησιμοποιηθεί ώστε να δουλέψει το παραγόμενο κύκλωμα στην πλακέτα FPGA. Το RTL που θα παραχθεί μπορεί είναι διαθέσιμο είτε σε μορφή VHDL είτε σε μορφή Verilog.
* Αρχεία ανάλυσης. Αποτελεί την έξοδο που θα βγάλει το κύκλωμα μετά την σύνθεση, C/RTL συν-προσομοίωση και IP packaging.

Στην παρακάτω εικόνα βλέπουμε μια επισκόπηση των αρχείων εισόδων και εξόδων του εργαλείου Vivado HLS.



**Test Bench, Υποστήριξη γλωσσών, βιβλιοθήκες της C.**

Σε κάθε πρόγραμμα που είναι γραμμένο σε C, η top-level function ονομάζεται main(). Στο Vivado HLS μπορούμε να ορίσουμε οποιαδήποτε υποσυνάρτηση που είναι κάτω από την main ως top-level function για σύνθεση. Δεν μπορούμε να κάνουμε σύνθεση όμως την top-level function main(). Και επιπλέον πρέπει να ισχύουν όλα τα παρακάτω:

* Μόνο μια συνάρτηση επιτρέπεται να είναι top-level function για σύνθεση.
* Όποια συνάρτηση περιλαμβάνεται, καλείται στην top-level function για σύνθεση τότε και η ίδια θα μετατραπεί.
* Αν θέλουμε να κάνουμε σύνθεση συναρτήσεις τα οποία δεν ανήκουν στην ιεραρχία, δεν είναι κάτω από την top-level function που θα γίνει η σύνθεση, τότε πρέπει να ενώσουμε αυτές τις συναρτήσεις για να μετατραπούν και αυτές.

**Τest Bench**

Όταν χρησιμοποιούμε το Vivado HLS, είναι πολύ χρονοβόρο να κάνουμε σύνθεση μια συνάρτηση C που δεν λειτουργεί σωστά και μετά να αναλύσουμε τις αναφορές που βγάζει ώστε να ξεκαθαρίσουμε γιατί δεν η συνάρτηση δεν δουλεύει σωστά όπως περιμέναμε. Για να αυξήσουμε την παραγωγικότητα στο συγκεκριμένο χρόνο, μπορούμε να χρησιμοποιήσουμε τα test bench ώστε να εξακριβώσουμε αν δουλεύει σωστά η συνάρτηση πριν την κάνουμε σύνθεση.

Το test bench περιέχει την συνάρτηση main() και μπορεί να περιέχει και άλλες συναρτήσεις που είναι μην ανήκουν στην δομή, ιεραρχία της top-level function που πρόκειται να γίνει η σύνθεση. Με αυτές τις συναρτήσεις μπορούμε να ελέγξουμε αν η συνάρτηση που πρόκειται να γίνει η σύνθεση είναι λειτουργικά σωστή, βγάζοντας το ανάλογο output στην οθόνη.

Το Vivado HLS χρησιμοποιεί τα test bench ώστε να μεταγλωττίζει και να εκτελέσει το C- Simulation. Κατά την διάρκεια της μεταγλώττισης μπορούμε να διαλέξουμε την επιλογή **Launch Debugger** για να κάνουμε αποσφαλμάτωση του προγράμματος σε περίπτωση που χρειαστεί.

Γλώσσες που υποστηρίζει:

* ANSI-C
* C++
* OpenCL
* SystemC

To Vivado HLS υποστηρίζει διάφορα εργαλεία της C, C++, και systemC καθώς πολλά είδη τύπων για την κάθε γλώσσα όπως διπλής ακρίβειας και πραγματικούς αριθμούς. Ωστόσο, δεν μπορεί να γίνει σύνθεση σε όλα τα εργαλεία που προσφέρουν οι παραπάνω γλώσσες όπως είναι τα παρακάτω:

* Δυναμική κατανομή μνήμης. Μια πλακέτα FPGA έχει ένα σταθερό σύνολο πόρων και η δυναμική δημιουργία και απελευθέρωση πόρων μνήμης δεν υποστηρίζεται.
* Λειτουργίες λειτουργικού συστήματος (OS). Όλα τα δεδομένα από και προς τις πλακέτες FPGA πρέπει να διαβάζονται από τις πύλες εισόδου ή να γράφονται στις πύλες εξόδου. Δεν υποστηρίζονται λειτουργίες του λειτουργικού συστήματος όπως διάβασμα/δημιουργία αρχείου ή συναρτήσεις που χρησιμοποιούν την ώρα και την ημερομηνία. Αντί αυτού μπορούμε να χρησιμοποιήσουμε τις παραπάνω λειτουργίες και να μεταφέρουμε τα δεδομένα στην συνάρτηση που πρόκειται να γίνει σύνθεση μέσω των test benches.

Εκτός από όλα αυτά, το Vivado HLS υποστηρίζει και παρά πολλές βιβλιοθήκες της γλώσσας C, οι οποίες περιλαμβάνουν συναρτήσεις και άλλα εργαλεία τα οποία είναι χρήσιμα ώστε να γίνει βελτιστοποίηση της υλοποίησης του κυκλώματος σε μια πλακέτα FPGA. Η χρήση αυτών των βιβλιοθηκών συμβάλλει στην εξασφάλιση υψηλής ποιότητας αποτελεσμάτων (QoR) με αποτέλεσμα να παράξουμε ένα κύκλωμα υψηλής απόδοσης εφόσον έχουμε κάνει την βέλτιστη χρήση των πόρων που μας δίνονται. Αυτές οι βιβλιοθήκες μπορούν να χρησιμοποιηθούν για την προσομοίωση του κυκλώματος πριν γίνει η σύνθεση.

Το Vivado HLS υποστηρίζει και άλλες βιβλιοθήκες για να επεκτείνει τις δυνατότητες της γλώσσας C και αυτές είναι οι παρακάτω:

* Τύποι δεδομένων αυθαίρετης ακρίβειας
* Τύποι δεδομένων κινητής υποδιαστολής μισής ακρίβειας των 16 bit
* Μαθηματικές συναρτήσεις
* Συναρτήσεις για Video
* Συναρτήσεις Xilinx IP, συμπεριλαμβάνοντας μεθόδους όπως τον fast fourier transform (FFT) και Finite impulse response (FIR)
* Συναρτήσεις πόρων της FPGA πλακέτας ώστε να μεγιστοποιείται η χρησιμότητα των πόρων shift register LUT ( Look Up tables)

**Σύνθεση, βελτιστοποίηση και ανάλυση**

Το Vivado HLS αποτελεί πρόγραμμα το οποίο βασίζεται πάνω σε project. Κάθε project εμπεριέχει ένα σύνολο από C κώδικα και μπορεί να εμπεριέχει διάφορες λύσεις υλοποιήσεις για το project αυτό. Κάθε υλοποίηση μπορεί να εμπεριέχει διάφορα constraints και directives για βελτιστοποίηση. Μπορούμε να αναλύσουμε και να συγκρίνουμε τα αποτελέσματα κάθε υλοποίησης μέσω του γραφικού περιβάλλον του Vivado HLS.

Τα βήματα για την χρήση του λογισμικού αυτού είναι τα εξής:

1. Δημιουργία project με μια αρχικοποιημένη υλοποίηση.
2. Έλεγχος της προσομοίωσης σε C ώστε να μην παρουσιάσει κανένα σφάλμα.
3. Σύνθεση της υλοποίησης ώστε να λάβουμε τα αντίστοιχα αποτελέσματα
4. Ανάλυση των αποτελεσμάτων.

Μετά την ανάλυση των αποτελεσμάτων, μπορούμε να δημιουργήσουμε καινούρια λύση – υλοποίηση για το project με διαφορετικά constraints και directives ώστε να κάνουμε σύνθεση την νέα υλοποίηση. Επιπλέον ,μπορούμε να επαναλάβουμε αυτήν την διαδικασία μέχρι το κύκλωμα μας έχει την επιθυμητή απόδοση. Με την χρήση των πολλαπλών λύσεων μπορούμε να εύκολα να αξιολογήσουμε τις διαφορετικές υλοποιήσεις καθώς αναπτύσσουμε την βέλτιστή υλοποίηση για το κύκλωμα μας.

**Βελτιστοποίηση**

Με την χρήση του Vivado HLS, μπορούμε να εφαρμόσουμε στις υλοποιήσεις διάφορα directives ώστε να βελτιστοποιήσουμε. Κάποια από αυτά είναι:

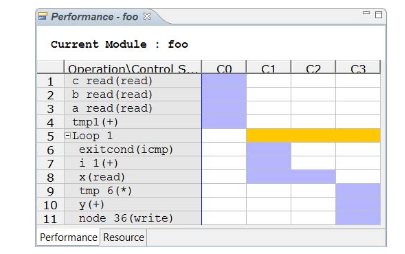
* Μπορούμε να κάνουμε μια εντολή να εκτελεστεί με την χρήση διοχέτευσης (pipeline), επιτρέποντας να εκτελεστεί η επόμενη εντολή πριν ολοκληρωθεί η τρέχουσα εντολή αξιοποιώντας πολύτιμο χρόνο.
* Μπορούμε να καθορίσουμε το χρόνο (latency) που χρειάζεται για την ολοκλήρωση των συναρτήσεων και βρόγχων.
* Μπορούμε να καθορίσουμε το όριο των πόρων που χρησιμοποιούνται στο κύκλωμα.
* Μπορούμε να καταργήσουμε τις εξαρτήσεις των εντολών του κώδικα και να εκτελέσουμε κάποιες συγκεκριμένες λειτουργίες. Για παράδειγμα αν είναι εφικτό να αγνοήσουμε αρχικές τιμές που έχουν κάποιες μεταβλητές σε ένα κώδικα όπως είναι μια ροή video, επιτρέποντας να διαβάσει την μνήμη πριν τα γράψει σε αυτήν αν τα αποτελέσματα που βγάλει είναι αποδοτικότερα.
* Μπορούμε να επιλέξουμε το πρωτόκολλο εισόδου/εξόδου ώστε να σιγουρευτούμε ότι το τελικό σχέδιο μπορεί να συνδεθεί με τα άλλα blocks του hardware με το ίδιο πρωτόκολλο.

Με την χρήση του γραφικού περιβάλλον του Vivado μπορούμε εισάγουμε τα directives που μας ενδιαφέρουν ώστε να βελτιστοποιήσουμε το κύκλωμα.

**Ανάλυση**

Όταν ολοκληρωθεί η σύνθεση, το Vivado HLS αυτομάτως φτιάχνει ένα report για το παραγόμενο ώστε να μας βοηθήσει να καταλάβουμε την απόδοση της υλοποίησης μας. Στο γραφικό περιβάλλον του Vivado HLS η analysis perspective εμπεριέχει το performance tab, στο οποίο μας επιτρέπει να δούμε και να αναλύσουμε με λεπτομέρεια τα αποτελέσματα.

Όπως φαίνεται στο παρακάτω παράδειγμα:



Αυτή η καρτέλα δείχνει:

* C0: Το πρώτο στάδιο που διαβάζει τις μεταβλητές c, b,a καθώς και την πρόσθεση.
* C1 και C2: Το κύκλωμα μπαίνει σε βρόγχο και ελέγχει τον μετρητή και την συνθήκη για να βγει από την βρόγχο. Το κύκλωμα μετά διαβάζει την μεταβλητή x και βλέπουμε ότι χρειάζεται δύο κύκλους ρολογιού. Αυτό χρειάζεται γιατί το κύκλωμα αποκτάει πρόσβαση στην block Ram καθώς καταναλώνει ένα κύκλο ρολογιού για να βρει την διεύθυνση και έναν ακόμη για να διαβάσει τα δεδομένα.
* C3: Το κύκλωμα εκτελεί τις υπολογισμούς καθώς γράφει το output στον y. Έπειτα, ο βρόγχος ξεκινάει από την αρχή.

**Επαλήθευση RTL.**

Με την χρήση ενός test bench που είναι γραμμένο σε γλώσσα C , μπορούμε να επαληθεύσουμε εάν λειτουργικά είναι ίδια το παραγόμενο κύκλωμα RTL με το πρωτότυπο που είναι γραμμένο σε C. To test bench αυτό που κάνει, είναι να επαληθεύει την έξοδο από την top-level function που την κάναμε σύνθεση, επιστρέφοντας μηδέν αν RTL είναι λειτουργικά ίδιο με το πρωτότυπο. Αυτό χρησιμοποιείται και στο C simulation και στο C/RTL co-simulation. Εάν το αποτέλεσμα που βγάλει είναι διάφορο του μηδέν τότε το Vivado HLS εμφανίζει ότι απέτυχε η προσομοίωση.

Όλα αυτά που αναλύσαμε πιο πάνω θα τα χρησιμοποιήσουμε στο δικό μας αλγόριθμο Αυτό που κάνουμε είναι να αναλύσουμε τις διαφορετικές υλοποιήσεις που έχουμε φτιάξει για τον αλγόριθμο αυτόν, καθώς να σχολιάσουμε τους λόγους για τους οποίους είναι αποδοτικότερη η πρώτη υλοποίηση από τις άλλους δύο. Στην συνέχεια, θα χρησιμοποιήσουμε διάφορες τεχνικές ώστε να βελτιστοποιήσουμε την απόδοση της πιο αργής υλοποίησης. Στο τέλος θα μελετήσουμε διάφορες τεχνικές ώστε να διευκολύνουμε αυτούς που πρόκειται να χρησιμοποιήσουν το εργαλείο αυτό , να γράψουν αποδοτικότερες υλοποιήσεις για το κύκλωμα που σκοπεύουν να φτιάξουν.

Ο αλγόριθμός που έχουμε αποφασίσει να εξετάσουμε είναι αλγόριθμός κρυπτογράφησης AES.

**ΑΛΓΟΡΙΘΜΟΣ AES**

Συμβουλές/προτεινόμενοι μέθοδοι:

Όσον αφορά τον κώδικα:

Πρώτο μέλημα μας είναι να λιγοστεύσουμε τα δεδομένα που διαβάζει η κύρια συνάρτηση, δηλαδή τα βασικά inputs που χρειάζεται η συνάρτηση που θα γίνει η σύνθεση για να λειτουργήσει. Αυτά τα δεδομένα θα μεταφραστούν σε I/O Ports στο κύκλωμα. Όταν τα δεδομένα που έχει ο αλγόριθμος που χρησιμοποιούμε διαβαστούν σε ένα block της πλακέτας, μπορούν εύκολα να εύκολα να τα διαχειριστούμε παράλληλα αλλά τα input ports μπορούν να γίνουν «bottleneck» καθώς να καθυστερήσουν το ολικό κύκλωμα με αποτέλεσμα να αυξηθεί το latency. Αυτό που προτείνουμε είναι να διαβαστούν μια φορά τα δεδομένα και με τα να τα χρησιμοποιήσουμε σε μια τοπική μεταβλητή cache αν τα δεδομένα πρόκειται να ξανά χρησιμοποιηθούν.

Εκτός από την μείωση των δεδομένων που χρησιμοποιούνται σαν είσοδο στην κύρια συνάρτηση, θα πρέπει να λιγοστεύσουμε τις προσπελάσεις που γίνονται σε πίνακες , ειδικά στα μεγάλους πίνακες. Οι πίνακες μετά την σύνθεση υλοποιούνται σε Block RAM οι οποίες όπως και τα I/O Ports είναι περιορισμένα σαν αριθμό και μπορεί να καθυστερήσουν το ολικό κύκλωμα, καθώς μπορούν να γίνουν και αυτά η καθυστέρηση, «bottleneck» για το ολικό κύκλωμα. Οι πίνακες μπορούν να κατανεμηθούν σε μικρότερους πίνακες και ακόμη και σε ξεχωριστές καταχωρητές, αλλά η διαμέριση των μεγάλων πινάκων θα χρειαστεί αναγκαία την χρήση ενός πολύ μεγάλου αριθμού καταχωρητών. Αυτό που προτείνουμε είναι η χρήση τοπικών μνήμων cache μικρού μεγέθους για να γίνει η αποθήκευση των αποτελεσμάτων που μπορεί να προκύψει από κάποια πράξη και στην συνέχεια να γραφτεί το τελικό αποτέλεσμα στον πίνακα.

Επιπλέον, πρέπει να επιδιώξουμε να εκτελούμε διακλαδώσεις υπό συνθήκη εντός σε tasks που εκτελούνται με την χρήση διοχέτευσης , αντί να εκτελούμε άλλες εντολές υπό συνθήκη ακόμη και αν εκτελούνται με την χρήση διοχέτευσης. Οι συνθήκες θα υλοποιηθούν σε διαφορετικά μονοπάτια με την χρήση διοχέτευσης. Με το να επιτρέπεται στα δεδομένα από μια εντολή να χρησιμοποιούνται παράλληλα καθώς εκτελείται η επόμενη εντολή το οποίο είναι υπό συνθήκη θα οδηγήσει στην αύξηση της απόδοσης του κυκλώματος μας.

Εκτός από όλα αυτά πρέπει να λάβουμε υπόψιν και τα outputs. Τα δεδομένα που θα γράψει η κύρια συνάρτηση πρέπει να περιοριστούν για τον ίδιο λόγο που θα πρέπει και στα δεδομένα που πρόκειται να διαβάσει. Τα I/O Ports μπορούν να καθυστερήσουν την συνολική απόδοση του κυκλώματος που πρόκειται να δημιουργηθεί μετά την σύνθεση. Η δημιουργία περισσότερων ports τα οποία μπορεί να είναι είτε είσοδο είτε για έξοδο, επιβαρύνει περισσότερο το σύστημα.

Προτείνουμε κατά την δημιουργία του κώδικα που πρόκειται να γίνει η σύνθεση , να χρησιμοποιηθεί η εντολή **hls::streams**. Με την χρήση αυτής της εντολής το εργαλείο HLS μπορεί να δημιουργήσει κυκλώματα για την πλακέτες, που θα έχουν καλή επίδοση καθώς θα χρησιμοποιούν λιγότερο πόρους. Είναι πιο πρακτικό, παραγωγικό ο προγραμματιστής να προσπαθήσει να δημιουργήσει μια υλοποίηση υψηλής απόδοσης για συσκευές FPGA, παρά να προσπαθήσει μετά την σύνθεση να καταλάβει τους λόγους που δεν έχει την απόδοση που πρέπει να έχει.

Όσον αφορά την γλώσσα προγραμματισμού:

Όπως αναφέρουμε πιο πάνω, το εργαλείο Vivado HLS μπορεί να μετατρέψει σε υλοποιήσεις για την πλακέτα, τις γλώσσες C,C++, SystemC και OpenCL.

Η Προτεινομένη γλώσσα αποτελεί η systemC.

Οι βασικές διαφορές που έχει η systemC με τις άλλες γλώσσες είναι ότι η SystemC είναι ειδικά σχεδιασμένη, διαμορφωμένη να σχεδιάζει σε επίπεδο συστήματος. Εκτός από τις βασικές κλάσεις που έχει η C++, παρέχει εργαλεία ώστε να προσομοιώνει διαδικασίες που εκτελούνται παράλληλα με την χρήση ρολογιών τα όποια μπορεί να εισάγει ο προγραμματιστής στην υλοποίηση που πρόκειται να σχεδιάσει. Έκτος από όλα αυτά, παρέχει Ports, Signals, Channels και Events τα οποία υπάρχουν και σε μια γλώσσα περιγραφής υλικού HDL.

Όσον αφορά την υλοποίηση σε RTL.

Για να αυξήσουμε την απόδοση του κυκλώματος, που παράχθηκε μετά την σύνθεση μπορούμε να χρησιμοποιήσουμε διάφορους οδηγούς που μας παρέχει το εργαλείο Vivado HLS. Αυτοί οι οδηγοί λέγονται directives και μπορούν να χρησιμοποιηθούν για την αύξηση της απόδοσης του συστήματος καθώς και για την μείωση των πόρων που χρησιμοποιούνται στο κύκλωμα.

Τα directives που μπορούν να χρησιμοποιηθούν είναι τα παρακάτω:

ALLOCATION: Μπορούμε να ορίσουμε ένα αριθμητικό όριο για εκτέλεση operations, συναρτήσεων καθώς και πυρήνων που χρησιμοποιούνται. Με την χρήση αυτού του directive μπορεί οδηγήσει στην χρήση πιο πολλών πόρων του συστήματος με αποτέλεσμα να μειωθεί η απόδοση του συστήματος.

ARRAY\_MAP: Μπορούμε να συνδυάσουμε μικρότερους πίνακες σε ένα μεγαλύτερο ώστε να μειώσουμε την χρήση των Block RAM.

ARRAY\_PARTITION: Μπορούμε να διαμερίσουμε μεγάλους πίνακες σε πολλά μικρότερα και ακόμη σε καταχωρητές, ώστε να αυξήσουμε την προσπέλαση στα δεδομένα καθώς να μειώσουμε τις block RAM καθυστερήσεις.

ARRAY\_RESHAPE: Μπορούμε να επανασχεδιάσουμε έναν πίνακα που μπορεί είτε να έχει ένα στοιχείο είτε πολλά σε ένα πίνακα μεγαλύτερου μεγέθους. Χρήσιμο για αυξήσουμε την προσπέλαση στην Block RAM χωρίς να χρησιμοποιήσουμε και άλλο τέτοιο πόρο.

CLOCK: Για τις υλοποιήσεις που έχουν γραφτεί σε SystemC, μπορούμε να φτιάξουμε νέα ρολόγια πάνω στα SC\_MODULE χρησιμοποιώντας αυτό το Directive.

DATA\_PACK: Ομαδοποιεί τα ορίσματα σε δομή σε ένα scalar χρησιμοποιώντας μεταβλητή μεγαλύτερου μεγέθους.

DEPENDENCE: Χρησιμοποιείται ώστε να παρέχει επιπλέον πληροφορίες με σκοπό να ξεπεράσει εξαρτήσεις μεταξύ των βρόγχων ώστε να επιτευχθεί η χρήση της μεθόδου διοχέτευσης σε αυτά.

EXPRESSION\_BALANCE: Απενεργοποιεί την αυτόματη εξισορρόπηση των expression.

FUNCTION\_INSTANTIATE: Επιτρέπει διαφορετικά instances της ίδιας συνάρτησης να βελτιστοποιηθούν.

INLINE: Κάνει inline σε μια συνάρτηση, ανεξαρτήτως την ιεραρχία που έχουν μεταξύ τους, οι συναρτήσεις. Χρησιμοποιείται ώστε να αυξήσει το latency, το interval του συστήματος μειώνοντας τα function call overhead.

INTERFACE: Ορίζει πώς τα RTL ports έχουν δημιουργηθεί από την περιγραφή της συνάρτησης.

LATENCY: Μπορούμε να ορίσουμε το ελάχιστο και μέγιστο latency που χρησιμοποιείται στο συγκεκριμένο resource.

LOOP\_FLATEN: Επιτρέπει τους εμφωλευμένους βρόγχους να εκτελεστούν σε ένα βρόγχο αλλά με αυξημένο καθυστέρηση.

LOOP\_MERGE: Ενώνει διαδοχικούς βρόγχους για μειώσει το latency, αυξάνοντας το μοιρασμό των πόρων.

LOOP\_TRIPCOUNT: Χρησιμοποιείται στους βρόγχους, στους οποίους είναι μεταβλητό ο αριθμός των επαναλήψεων. Παρέχει μια εκτίμηση για το αριθμό των επαναλήψεων. Δεν έχει επίδραση στην σύνθεση, αλλά μόνο στην τελική αναφορά.

OCCURRENCE: Χρησιμοποιείται όταν χρησιμοποιούμε την μέθοδο διοχέτευσης σε συναρτήσεις, βρόγχους, για να προσδιορίσουμε ότι ο κώδικας στο συγκεκριμένο πεδίο χρησιμοποιείται σε χαμηλότερο ρυθμό σε σύγκριση με το υπόλοιπο κώδικα.

PIPELINE: Μειώνει το διάστημα έναρξης επιτρέποντας την παράλληλη εκτέλεση εντολών που ανήκουν σε ένα βρόγχο ή σε μια συνάρτηση.

PROTOCOL: Με αυτήν την εντολή, μπορούμε να ορίσουμε ένα μέρος του κώδικα να είναι πρωτόκολλο region. Ένα protocol region μπορεί να χρησιμοποιηθεί ώστε να ορίσουμε ένα interface πρωτόκολλο.

RESET: Με αυτό το directive, μπορούμε να προσθέσουμε ή να αφαιρέσουμε σε μια μεταβλητή το state δηλαδή αν είναι global ή static.

RESOURCE: Μπορούμε να ορίσουμε μια συγκεκριμένη βιβλιοθήκη να χρησιμοποιηθεί ώστε να υλοποιήσουμε μια μεταβλητή που μπορεί να είναι πίνακας, αριθμητικός τελεστής και όρισμα σε μια συνάρτηση, στο RTL κύκλωμα.

STREAM: Mπορούμε να ορίσουμε έναν συγκριμένο πίνακα να υλοποιηθεί με την χρήση FIFO ή RAM για βελτιστοποίηση του κυκλώματος

TOP: Η top-level συνάρτηση για σύνθεση ορίζεται στα project settings. Αυτός ο οδηγός μπορεί να χρησιμοποιηθεί ώστε να ορίσουμε οποιαδήποτε ως top-level. Αυτό μας διευκολύνει ώστε να βρούμε την καλύτερη λύση για το κύκλωμα χωρίς να φτιάξουμε νέο project.

UNROLL: Κάνει Unroll βρόγχους τύπου for ώστε να εκτελέσει τις εντολές που μπορεί να είναι ανεξάρτητες η μία από την άλλη σειριακά αλλά και μπορεί και παράλληλα να χρησιμοποιηθεί μαζί με άλλο directive.