Διεθνές Πανεπιστήμιο της Ελλάδος
Τμήμα Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών

Διπλωματική Εργασία Πρόγραμμα Μεταπτυχιακών Σπουδών στη Ρομποτική

Μελέτη του Συστήματος σε Προγραμματιζόμενη Ψηφίδα (SoC) ZYNQ και υλοποίηση εφαρμογών με τη βοήθεια του αναπτυξιακού συστήματος PYNQ.

Βασιλείου Ιωάννης
Διεθνές Πανεπιστήμιο της Ελλάδος

Σέρρες, 21-9-2020

Εργασία που υποβλήθηκε στο
Πρόγραμμα Μεταπτυχιακών Σπουδών στη Ρομποτική,
του Διεθνούς Πανεπιστημίου της Ελλάδος,
για τη μερική εκπλήρωση υποχρεώσεων για το Δίπλωμα Ειδίκευσης στη Ρομποτική

Επιβλέπων Καθηγητής: Ιωάννης Καλόμοιρος

Περίληψη

Σκοπός αυτής της εργασίας είναι η ανάπτυξη μίας εφαρμογής ανίχνευσης ακμών με φίλτρο Sobel στην προγραμματιζόμενη λογική ενός επιταχυντή, επιτυγχάνοντας εξαγωγή των ακμών σε πραγματικό χρόνο αφού η επεξεργασία των δεδομένων γίνεται παράλληλα.

Η εφαρμογή αποτυπώνεται στην ετερογενή διάταξη Zynq που ενσωματώνει επεξεργαστή υλικού σε συνδυασμό με το λογικό υλικό και η πλακέτα που χρησιμοποιείται είναι η Pynq - Z2 που διαθέτει το τσιπ Zynq και διάφορα άλλα περιφερειακά. Για την υλοποίηση της εφαρμογής χρησιμοποιούμε από την πλακέτα μία θύρα HDMI για λήψη δεδομένων και μία θύρα HDMI για την προβολή των επεξεργασμένων δεδομένων, δηλαδή των ακμών. Βασικό πλεονέκτημα του περιβάλλοντος Pynq αποτελεί η χρήση της γλώσσας υψηλού επιπέδου Python και της βιβλιοθήκης Overlay, όπου δίνει την δυνατότητα στον χρήστη να αναπτύξει μία εφαρμογή χωρίς την ανάγκη σχεδίασης της.

Το εργαλείο που χρησιμοποιείται για την σχεδίαση, σύνθεση και τοποθέτηση της εφαρμογής στην προγραμματιζόμενη λογική είναι το Vivado Design Suite όπου εφαρμόζουμε έτοιμα μπλοκ που εκτελούν συγκεκριμένες λειτουργίες και επικοινωνούν μεταξύ τους καθώς και με τα στοιχεία της πλακέτας όπως οι θύρες HDMI.

Abstract

The purpose of this work is to develop an edge detection application with Sobel filter in the programmable logic of an accelerator, achieving real - time edge extraction since the data is being processed parallel.

The application is implemented in the heterogeneous Zynq device that incorporates a hardware processor in combination with the logic fabric and the board used is the Pynq - Z2 which has the Zynq chip and various other peripherals. To implement the application we use from the board an HDMI port for receiving data and an HDMI port for viewing the processed data, ie the edges. A key advantage of the Pynq environment is the use of the high - level Python language and it’s Overlay library, which allows the user to develop an application without the need to design it.

The tool used to design, synthesize and implement the application in programmable logic is the Vivado Design Suite where we apply ready - made blocks that perform specific functions and communicate with each other as well as with the components of the board such as HDMI ports.

Περιεχόμενα

Λίστα Εικόνων

1. Επιτάχυνση εφαρμογών ρομποτικής όρασης

Η εξέλιξη της τεχνολογίας βοήθησε την ανάπτυξη μίας ακόμα επιστήμης, αυτής της ρομποτικής όρασης. Με τον όρο ρομποτική όραση εννοούμε την ανάπτυξη μεθόδων και αλγορίθμων για την εξαγωγή χρήσιμης πληροφορίας από εικόνες και βίντεο με σκοπό να ληφθεί μία απόφαση σχετικά με τον κόσμο. Ιδανικά, η ρομποτική όραση απαιτεί την ικανότητα ενός συστήματος να αντιλαμβάνεται τις εικόνες ή τα βίντεο με τον ίδιο τρόπο που τα αντιλαμβάνεται ο άνθρωπος.

Μερικά παραδείγματα τεχνολογιών ακμής που βασίζονται στην ρομποτική όραση είναι τα παρακάτω:

- Αναγνώριση αντικειμένων
- Έξυπνα αυτοκίνητα
- Ιατρική απεικόνιση
- Κινούμενα ρομπότ
- Ειδικά εφέ

Επειδή όμως τα δεδομένα που συλλέγονται από τις εικόνες και τα βίντεο έχουν μεγάλο όγκο και οι εφαρμογές χρειάζονται απόκριση σε πραγματικό χρόνο, υπάρχει η ανάγκη ύπαρξης ενός επιταχυντή (FPGA: Field Programmable Gate Array) που είναι ικανός να εκτελέσει απαιτητικές εργασίες αφού το βασικό χαρακτηριστικό τους είναι η παράλληλη επεξεργασία δεδομένων.

Ένα FPGA είναι μία συσκευή η οποία αποτελείται από λογικές πύλες και άλλα λογικά στοιχεία, που με τη χρήση κατάλληλου λογισμικού μπορεί να διαμορφωθεί για να εκτελέσει μία συγκεκριμένη εργασία.

Τα πλεονεκτήματα μίας τέτοιας συσκευής είναι:

- Δυνατότητα επαναδιαμόρφωσης
- Καλύτερη απόδοση αφού επιτυγχάνει παράλληλη επεξεργασία δεδομένων
- Χαμηλό κόστος σε μακροχρόνια χρήση αφού με μία διάταξη μπορούν να πραγματοποιηθούν πολλές εφαρμογές

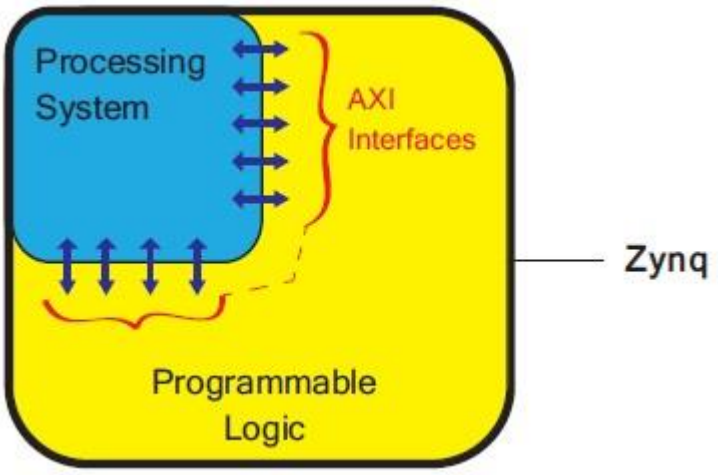
Στην προγραμματιζόμενη λογική του FPGA μπορεί να δημιουργηθεί και επεξεργαστής λογισμικού (soft processor) για την συνεργασία με το υπόλοιπο υλικό. Υπάρχουν και ετερογενείς διατάξεις που ενσωματώνουν επεξεργαστή υλικού (hard processor) μαζί με την προγραμματιζόμενη λογική ενός FPGA.

Το πρόβλημα με το οποίο ασχολείται η εργασία είναι η επιτάχυνση εφαρμογών ρομποτικής όρασης με ετερογενείς διατάξεις και συγκεκριμένα με την διάταξη του τσιπ Zynq της εταιρείας Xillinx και της αναπτυξιακής πλακέτας Pynq - Z2.

1.1 Το Zynq SoC και η αρχιτεκτονική του

Το Zynq είναι ένα τσιπ (Εικόνα 1.1) που ενσωματώνει την προγραμματιζόμενη λογική (PL: Programmable Logic), όπως αυτή σε ένα FPGA και το σύστημα επεξεργασίας (PS: Processing System) που αποτελείται από έναν επεξεργαστή ARM Cortex - A9 δύο πυρήνων. Επίσης, ενσωματώνει μνήμη, ποικιλία περιφερειακών και διεπαφές επικοινωνίας υψηλής ταχύτητας.

Κάθε ένα από τα δύο τμήματα (PS – PL) μπορεί να χρησιμοποιηθεί ανεξάρτητα ή να συνδυαστούν μέσω του εξελιγμένου πρωτόκολλου επικοινωνίας (AXI: Advanced eXtensive Interface).

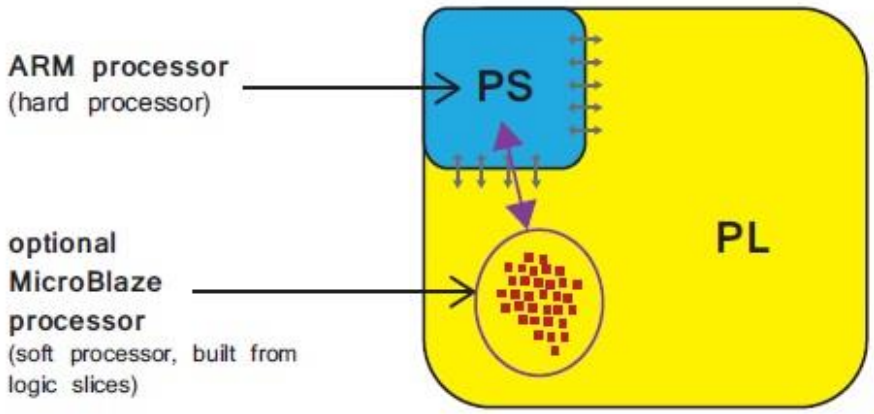


Εικόνα 1.1: Το μοντέλο αρχιτεκτονικής του Zynq [1]

1.1.1 Το σύστημα επεξεργασίας (PS)

Το επεξεργαστικό σύστημα του Zynq αποτελείται κυρίως από έναν επεξεργαστή υλικού (hard processor) ARM Cortex – A9 δύο πυρήνων, δηλαδή αποτελεί ανεξάρτητο στοιχείο στο τσιπ. Επίσης, μπορεί να χρησιμοποιηθεί και επεξεργαστής λογισμικού (soft processor) στην προγραμματιζόμενη λογική, κερδίζοντας σε ευελιξία αλλά χάνοντας σε απόδοση σε σχέση με τον επεξεργαστή υλικού.

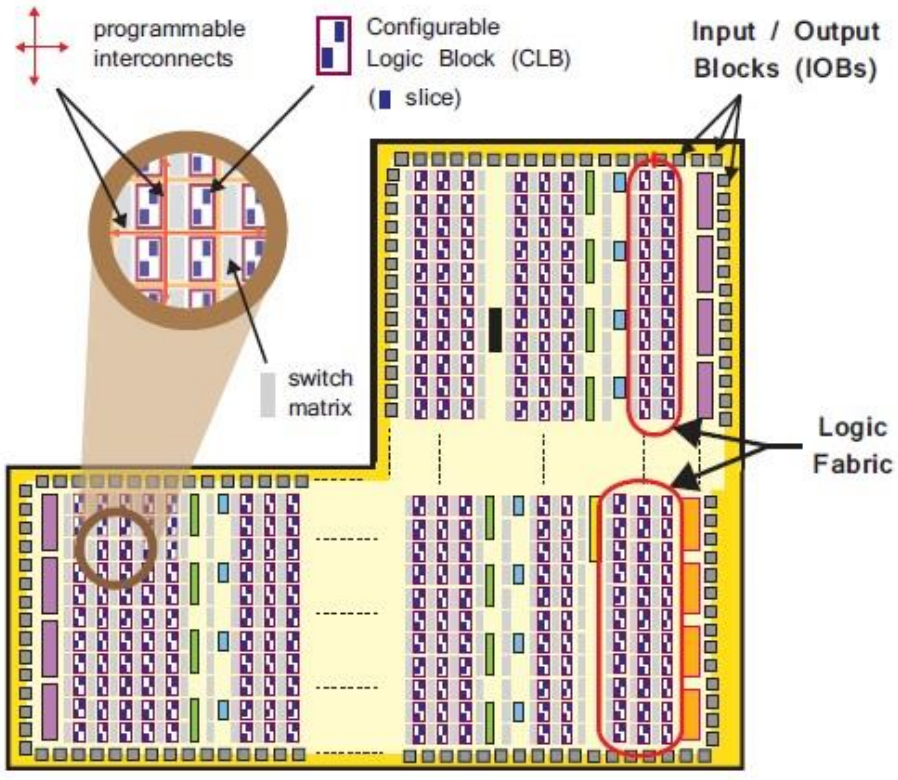
Οι επεξεργαστές λογισμικού που έχουν αναπτυχθεί από την εταιρεία Xilinx είναι ο PicoBlaze (8 - bit) και ο MicroBlaze (32 - bit). Σε μία διάταξη FPGA μπορούν να χρησιμοποιηθούν περισσότεροι από έναν επεξεργαστή λογισμικού, λειτουργώντας έτσι σε συνδυασμό με τον ARM επεξεργαστή. Στην Εικόνα 1.2 φαίνεται ο τρόπος που είναι τοποθετημένος ο επεξεργαστής υλικού και ο επεξεργαστής λογισμικού πάνω στο τσιπ Zynq.



Εικόνα 1.2: Η τοποθεσία του ARM και του MicroBlaze στο τσιπ Zynq [1]

1.1.2 Η προγραμματιζόμενη λογική (PL)

Το PL αποτελείται κυρίως από 'λογικό ύφασμα' (logic fabric) γενικής χρήσης FPGA, το οποίο αποτελείται από φέτες και διαμορφωμένα λογικά μπλοκ (CLB: Configurable Logic Block) και υπάρχουν επίσης μπλοκ εισόδου / εξόδου (IOB: Input / Output Block) για διασύνδεση με περιφερειακά (Εικόνα 1.3).



Εικόνα 1.3: Το λογικό υλικό και τα επιμέρους στοιχεία του [1]

- *Διαμορφωμένα λογικά μπλοκ (CLBs):*
  Είναι μικρές ομάδες λογικών στοιχείων που συνδέονται με άλλους παρόμοιους πόρους μέσω προγραμματιζόμενων διασυνδέσεων. Κάθε CLB είναι τοποθετημένο δίπλα σε έναν πίνακα εναλλαγής (Switch Matrix) και περιέχει δύο λογικές φέτες.
- *Φέτες (Slice):*
  Μια υπομονάδα μέσα στο CLB, που χρησιμοποιείται για την υλοποίηση συνδυαστικών και διαδοχικών λογικών κυκλωμάτων. Αποτελούνται από 4 πίνακες αναζήτησης (LUT: Lookup Table), 8Flip - Flop και άλλα.
- *Πίνακας αναζήτησης (LUT):*
  Ένας ευέλικτος πόρος ικανός να υλοποιήσει (i) μια λογική λειτουργία έως έξι εισόδους, (ii) μια μικρή μνήμη μόνο για ανάγνωση (ROM: Read Only Memory), (iii) μια μικρή μνήμη τυχαίας προσπέλασης (RAM: Read Access Memory), (iv) έναν καταχωρητή ολίσθησης (Shift register). Τα LUTs μπορούν να συνδυαστούν για να σχηματίσουν μεγαλύτερες λογικές λειτουργίες, μνήμες ή καταχωρητές ολίσθησης.
- *Flip – Flop (FF):*
  Ένα διαδοχικό στοιχείο κυκλώματος που εφαρμόζει ένα καταχωρητή 1-bit, με λειτουργία επαναφοράς.
- *Πίνακας εναλλαγής (SwitchMatrix):*
  Ένας πίνακας εναλλαγής τοποθετείται δίπλα σε κάθε CLB και κάνει δυνατή την δρομολόγηση για συνδέσεις (i) μεταξύ στοιχείων εντός ενός CLB και (ii) από έναν CLB σε άλλους πόρους της PL.
- *Λογική μεταφοράς (Carry Logic):*
  Περιλαμβάνει μία αλυσίδα διαδρομών και πολυπλέκτες για την μετάδοση ενδιάμεσων σημάτων μεταξύ παρακείμενων φετών.
- *Μπλοκ εισόδου / εξόδου (IOB):*
  Διασυνδέουν περιφερειακές συσκευές με το PL. Κάθε IOB μπορεί να χειριστεί ένα σήμα εισόδου ή εξόδου 1 bit.

1.2 Το πρότυπο AXI

Το AXI είναι μέρος του ανοιχτού προτύπου ARM AMBA (Advanced Microcontroller Bus Architecture) που αναπτύχθηκε αρχικά από τον ARM για χρήση σε μικροελεγκτές, με την πρώτη έκδοση να κυκλοφορεί το 1996 και τώρα αποτελεί το επίσημο πρότυπο για on - chip επικοινωνία. Η πρώτη έκδοση του AXI συμπεριλήφθηκε για πρώτη φορά στο AMBA 3.0 που κυκλοφόρησε το 2003. Το AMBA 4.0 κυκλοφόρησε το 2010 και περιλαμβάνει τη δεύτερη έκδοση του AXI, το AXI4.

Οι κατηγορίες AXI4 ανάλογα με τις ιδιότητες της σύνδεσης:

- AXI4: Σύνδεση υψηλής απόδοσης με χαρτογράφηση μνήμης (memory mapped) που πραγματοποιεί μεταφορά ριπής δεδομένων (data burst) έως 256 λέξεις.
- AXI4 – Lite: Απλοποιημένη σύνδεση χαρτογραφημένης μνήμης (memory mapped) που υποστηρίζει μονή μεταφορά δεδομένων (single data).
- AXI4 – Stream: Σύνδεση χωρίς χαρτογραφημένη μνήμη (non memory mapped) με ροή δεδομένων (streaming data) υψηλής ταχύτητας, που υποστηρίζει μεταφορά ριπής δεδομένων (data burst) απεριόριστου μεγέθους.

1.2.1 Πώς λειτουργεί

Περιγράφει μια διεπαφή (interface) μεταξύ ενός μόνο «κύριου» (master) AXI και ενός μόνο «υποτελούς» (slave) AXI, που αντιπροσωπεύει IP blocks που ανταλλάσσουν πληροφορίες μεταξύ τους. Οι AXI masters και slaves με χαρτογραφημένη μνήμη μπορούν να συνδεθούν μαζί χρησιμοποιώντας μια δομή που ονομάζεται interconnect. Τα Xilinx AXI Interconnect IP περιέχουν διεπαφές master και slave που είναι συμβατές με AXI και μπορούν να χρησιμοποιηθούν για τη δρομολόγηση συναλλαγών μεταξύ ενός ή περισσοτέρων AXI masters και slaves.

Και οι δύο διεπαφές AXI4 και AXI4 - Lite αποτελούνται από πέντε διαφορετικά κανάλια:

• Κανάλι ανάγνωσης διεύθυνσης

• Κανάλι εγγραφής διεύθυνσης

• Κανάλι ανάγνωσης δεδομένων

• Κανάλι εγγραφής δεδομένων

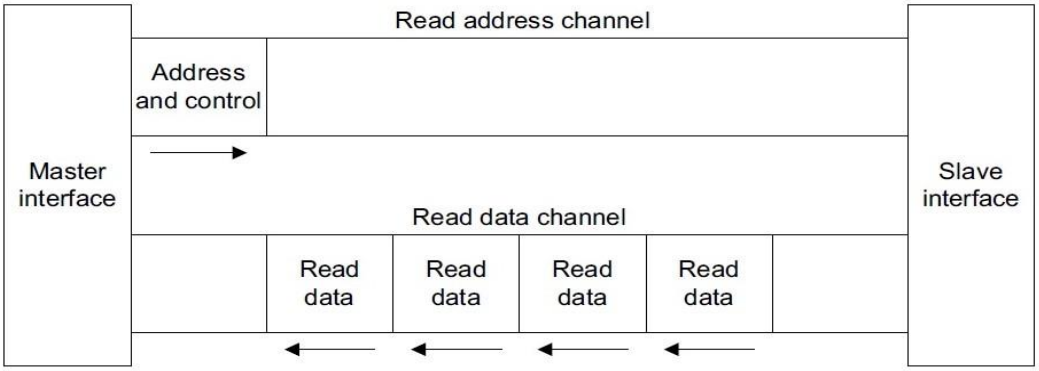
• Κανάλι εγγραφής απόκρισης

Τα δεδομένα μπορούν να κινούνται και προς τις δύο κατευθύνσεις μεταξύ του master και του slave ταυτόχρονα και το μέγεθος μεταφοράς δεδομένων μπορεί να διαφέρει. Το όριο στο AXI4 είναι μια συναλλαγή ‘έκρηξης’ (data burst) έως και 256 μεταφορές δεδομένων. Το AXI4 - Lite επιτρέπει μόνο 1 μεταφορά δεδομένων ανά συναλλαγή.

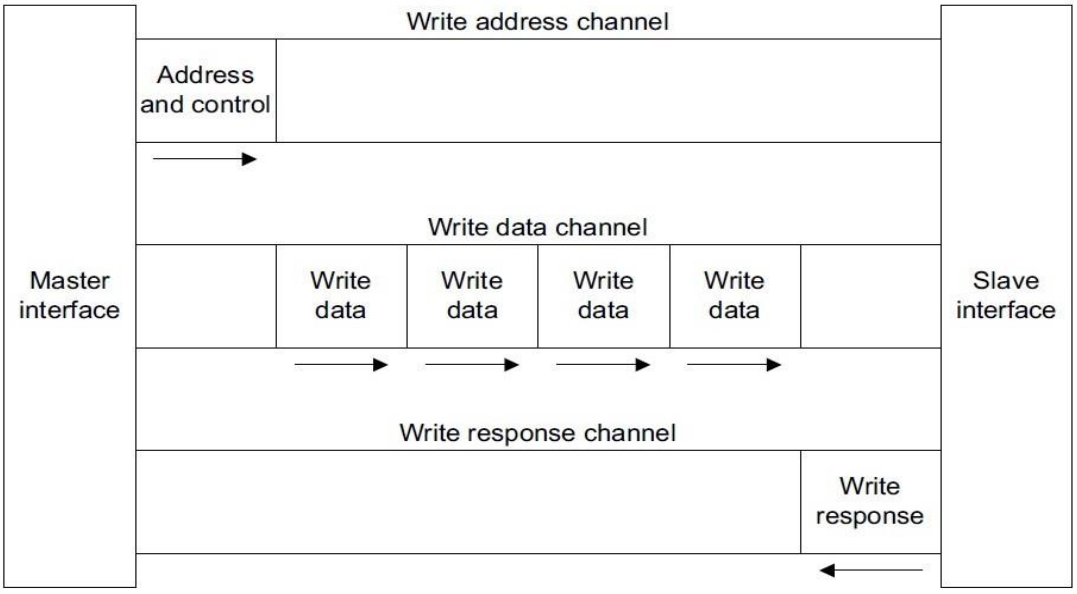
Ένα κανάλι διευθύνσεων μεταφέρει πληροφορίες ελέγχου που περιγράφουν τη φύση των δεδομένων που θα μεταφερθούν. Τα δεδομένα μεταφέρονται μεταξύ master και slave χρησιμοποιώντας είτε:

• Ένα κανάλι ανάγνωσης δεδομένων για τη μεταφορά δεδομένων από τον slave στον master (Εικόνα 1.4).

• Ένα κανάλι εγγραφής δεδομένων για τη μεταφορά δεδομένων από τον master στον slave. Σε μια συναλλαγή εγγραφής, ο slave χρησιμοποιεί το κανάλι εγγραφής της απόκρισης για να σηματοδοτήσει την ολοκλήρωση της μεταφοράς στον master (Εικόνα 1.5).

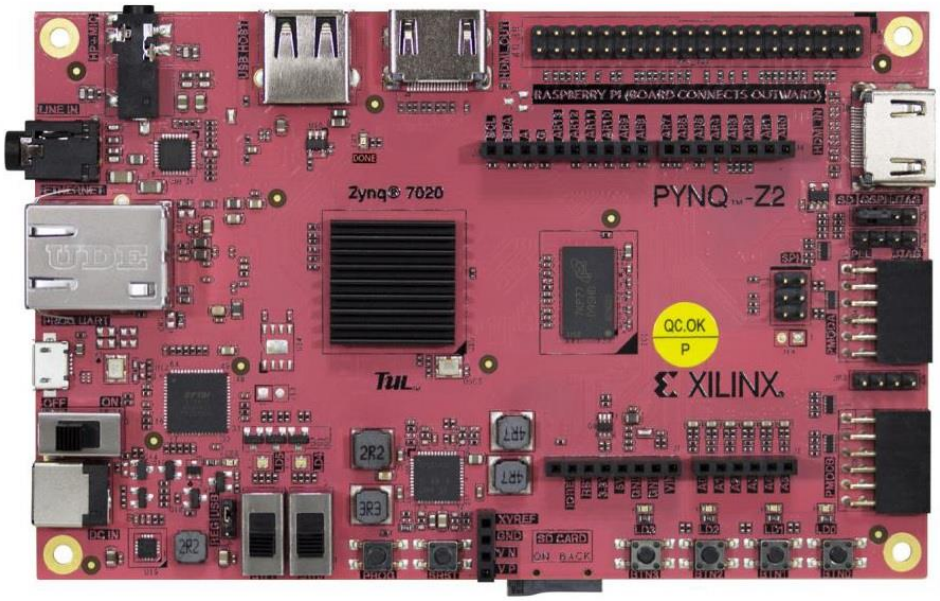


Εικόνα 1.4: Η διαδικασία ανάγνωσης διεύθυνσης [9]



Εικόνα 1.5: Η διαδικασία εγγραφής διεύθυνσης [9]

1.3 Η πλακέτα Pynq – Z2



Εικόνα 1.6: Η αναπτυξιακή πλακέτα Pynq-Z2 [41]

Στην Εικόνα 1.6 φαίνεται η αναπτυξιακή πλακέτα Pynq – Z2, η ονομασία της οποίας προέρχεται από την χρήση της γλώσσας προγραμματισμού Python σε συνδυασμό με το τσιπ Zynq.

Βασίζεται στο FPGA SoC τσιπ της σειράς Zynq 7000 που ενσωματώνει τον επεξεργαστή 2 πυρήνων ARM Cortex - A9. Παρέχει παλμό χρονισμού 50MHz στον επεξεργαστή και η μέγιστη συχνότητα λειτουργίας του είναι 650MHz. Παρέχει και εξωτερικό παλμό χρονισμού στα 125 MHz για να χρονίσει υλικά που αποτυπώνονται στο λογικό υλικό.

Η πλακέτα λειτουργεί με τάση 5Vαπό την θύρα micro - USb (τοποθετούμε το jumper στην θέση UART) από όπου και προγραμματίζεται ή εξωτερική τάση 7V - 15V μέσω αντίστοιχης θύρας (τοποθετούμε το jumper στην θέση REG).

Η αναπτυξιακή πλακέτα έχει μία θύρα για είσοδο εικόνας (HDMI in) και μία θύρα για έξοδο (HDMI out) και έτσι καθιστά δυνατή την χρήση της σε εφαρμογές που απαιτούν δεδομένα εικόνας. Επίσης, παρέχει την δυνατότητα λήψης ήχου μέσα από την θύρα Jack 3.5mm και έξοδο σε διαφορετική θύρα Jack 3.5mm.

Ακόμη, παρέχει υποδοχή για την πλακέτα Arduino και διαθέτει 28 pin γενικής χρήσης όπως αυτά ενός Raspberry Pi για την σύνδεση περιφερειακών εισόδου – εξόδου που μπορεί να είναι είτε αναλογικά είτε ψηφιακά.

Διαθέτει 2 θύρες για υποδοχή περιφερειακών που προσφέρει η εταιρεία Digilent και μπορεί να αξιοποιήσει συσκευές USB μέσω της αντίστοιχης θύρας.

Για την αλληλεπίδραση με την πλακέτα υπάρχουν 4 μπουτόν, 2 διακόπτες, 4 LEDs πράσινα και 2 RGB LEDs.

Τέλος έχει θύρα micro - SD από όπου φορτώνεται το λειτουργικό σύστημα (Ubuntu 18.04).

1.3.1 Σύνδεση με τον υπολογιστή

Η επικοινωνία ανάμεσα στον υπολογιστή και την πλακέτα επιτυγχάνεται μέσω της θύρας micro - USB και την σύνδεση της πλακέτας στο δίκτυο από την θύρα Ethernet.

Υπάρχουν 2 τρόποι για την σύνδεση σε δίκτυο:

- Συνδέουμε την πλακέτα μέσω Ethernet απευθείας στον υπολογιστή (στατική διεύθυνση)
- Συνδέουμε την πλακέτα μέσω Ethernet σε router (αυτόματη διεύθυνση)

1.3.2 Λειτουργία ανάλογα με την χρήση PS-PL

Εδώ περιγράφονται οι 2 τρόποι λειτουργίας της πλακέτας για την περίπτωση που την χρησιμοποιούμε μόνο σαν συσκευή FPGA ή θέλουμε και την χρήση του επεξεργαστή.

Χρήση PL

- Τοποθετούμε το boot jumper στην θέση JTAG (Joint Test Action Group)
- Συνδεόμαστε σε δίκτυο
- Ανοίγουμε τον διακόπτη τροφοδοσίας και η συσκευή είναι έτοιμη όταν ανοίξει το κόκκινο LED

Χρήση PS

- Τοποθετούμε το boot jumper στην θέση SD
- Συνδεόμαστε σε δίκτυο
- Ανοίγουμε τον διακόπτη τροφοδοσίας και περιμένουμε να τελειώσει η παρακάτω ακολουθία:
    - Ανοίγει το κόκκινο LED
    - Μετά από λίγα δευτερόλεπτα ανοίγουν 4 πράσινα LED
    - Μετά από 1 λεπτό αναβοσβήνουν 2 μπλε και τα 4 πράσινα LED

- Όταν μόνο τα πράσινα LED μείνουν αναμμένα σημαίνει ότι το σύστημα μας είναι έτοιμο.

1.3.3 Η γλώσσα Python

Η Python είναι μια γλώσσα προγραμματισμού αντικειμενοστραφής (object - oriented), ερμηνείας (interpreted), αλληλεπίδρασης (interactive) και υψηλού επιπέδου. Δημιουργήθηκε από τον Guido van Rossum κατά την περίοδο 1985 - 1990 και είναι σχεδιασμένη για να είναι ευανάγνωστη. Χρησιμοποιεί συχνά αγγλικές λέξεις - κλειδιά όπου άλλες γλώσσες χρησιμοποιούν σημεία στίξης και έχει λιγότερες συντακτικές κατασκευές από άλλες γλώσσες. Η πιο πρόσφατη έκδοση είναι η 3.8.3 και εκδόθηκε στις 14 Οκτωβρίου του 2019. Η έκδοση που υπάρχει στο λειτουργικό σύστημα είναι η 2.7 ενώ η έκδοση στο πακέτο PYNQ είναι η 3.6.

Τα χαρακτηριστικά της γλώσσας Python

- **Ερμηνευμένη**: Η Python υποβάλλεται σε επεξεργασία κατά το χρόνο εκτέλεσης από τον Python Interpreter.
- **Αντικειμενοστρεφής**: Υποστηρίζει αντικειμενοστραφή χαρακτηριστικά και τεχνικές προγραμματισμού.
- **Διαδραστική**: Οι χρήστες μπορούν να αλληλεπιδράσουν με τον διερμηνέα python απευθείας για τη σύνταξη προγραμμάτων.
- **Εύκολη**: Η Python είναι εύκολο να τη μάθει κάποιος,ειδικά οι αρχάριοι.
- **Απλή στη σύνταξη**: Ο σχηματισμός της σύνταξης python είναι απλός,γεγονός που τη καθιστά επίσης δημοφιλές.
- **Ευανάγνωστη**: Ο πηγαίος κώδικας Python είναι σαφώς καθορισμένος και ορατός στα μάτια.
- **Φορητή**: Οι κωδικοί Python μπορούν να εκτελεστούν σε μια μεγάλη ποικιλία πλατφορμών υλικού που έχουν την ίδια διεπαφή.
- **Επεκτάσιμη**: Οι χρήστες μπορούν να προσθέσουν λειτουργικές μονάδες χαμηλού επιπέδου στο διερμηνέα Python.
- **Κλιμακούμενη**: Η Python παρέχει μια βελτιωμένη δομή για την υποστήριξη μεγάλων προγραμμάτων και σεναρίων.

Υπάρχουν 2 τρόποι για να γράψουμε κώδικα σε γλώσσα Python:

1. Μέσω του nano editor του Linux shell, πραγματοποιώντας SSh (SecureShell) σύνδεση με τον ARM.
2. Μέσω του περιβάλλοντος Jupyter Notebooks από τον περιηγητή.

1.3.4 Η βιβλιοθήκη Overlay

Βασικός σκοπός της είναι η ανάπτυξη εφαρμογών σε συσκευές FPGA χωρίς την ανάγκη δημιουργίας σχεδίων λογικών κυκλωμάτων.

Ένα Overlay είναι μία βιβλιοθήκη υλικού και τυπικά είναι ένα σχέδιο για μία εφαρμογή σε προγραμματιζόμενο υλικό, προσφέροντας την δυνατότητα στον χρήστη να χρησιμοποιήσει το επεξεργαστικό σύστημα σε συνεργασία με το λογικό υλικό για να επιταχύνει την εφαρμογή. Αυτό επιτυγχάνεται μέσω μίας διεπαφής Python που επιτρέπει τον έλεγχο των Overlays στο PL από το PS. Όπως ένας προγραμματιστής λογισμικού χρησιμοποιεί μία βιβλιοθήκη λογισμικού για την ανάπτυξη μίας εφαρμογής, έτσι και ένας προγραμματιστής υλικού χρησιμοποιεί την βιβλιοθήκη Overlay.

Ένα Overlay περιλαμβάνει:

- Το αρχείο bitstream για να διαμορφώσει το λογικό υλικό
- Το αρχείο σχεδίου tcl από το Vivado για να προσδιορίσει τα IP
- Την διεπαφή Python για να προσδιορίσει τα χαρακτηριστικά των IP

Η κλάση Overlay του PYNQ χρησιμοποιείται για να φορτώσει το overlay. Το στιγμιότυπο ενός overlay δημιουργείται με τον καθορισμό της ονομασίας του αρχείου bitstream.

Ο κώδικας σε γλώσσα Python για την δημιουργία του στιγμιότυπου είναι ο παρακάτω:

```
from pynq import Overlay

my_overlay = Overlay("base.bit")
```

1.4 Σχεδιάγραμμα της εργασίας

Το υπόλοιπο της εργασίας οργανώνεται όπως φαίνεται παρακάτω:

Κεφάλαιο 2 Περιγράφει την κατάσταση της τέχνης των FPGAs· γίνεται αναφορά σε τσιπ, αναπτυξιακές πλακέτες και τα χαρακτηριστικά τους, από διάφορες εταιρείες παραγωγής ημιαγωγών.

Κεφάλαιο 3 Αναπτύσσει τα εργαλεία που χρησιμοποιήθηκαν για την δημιουργία των εφαρμογών.

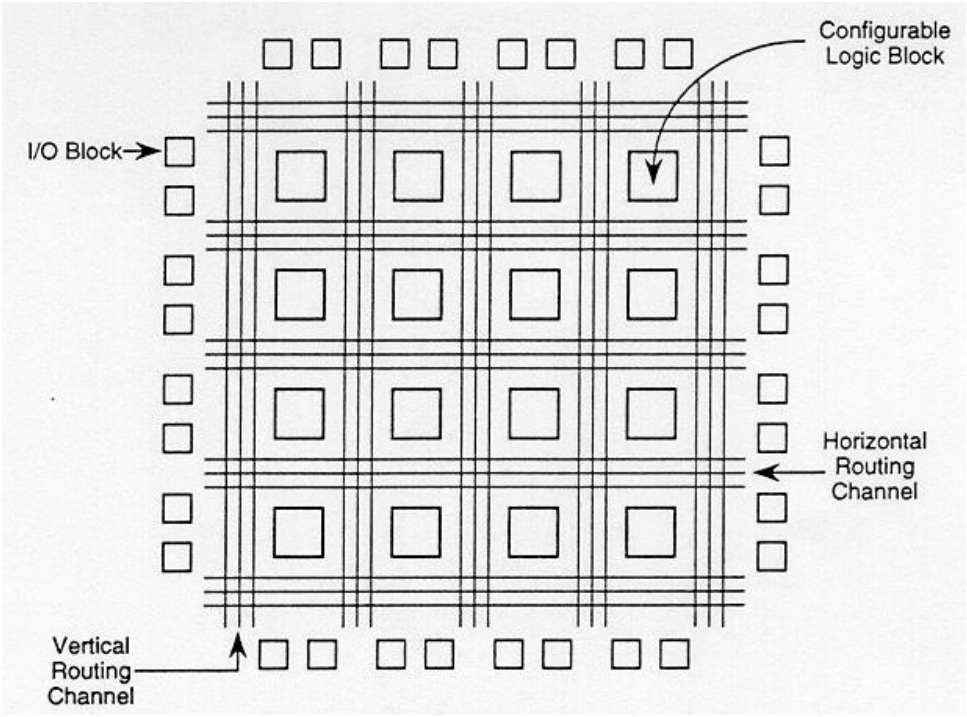
Κεφάλαιο 4 Αναλύει τα βήματα δημιουργίας γενικών εφαρμογών και των αποτελεσμάτων για την κατανόηση της χρήσης του λογισμικού που χρησιμοποιήθηκε και τον τρόπο αλληλεπίδρασης με την συσκευή.

Κεφάλαιο 5 Αναλύει τα βήματα δημιουργίας εφαρμογής για την κατανόηση λειτουργίας του τρόπου λήψης ροής βίντεο και προβολής σε οθόνη και κατόπιν τα βήματα δημιουργίας εφαρμογής ανίχνευσης ακμών Sobel και των αποτελεσμάτων.

Κεφάλαιο 6 Παρουσιάζει τα συμπεράσματα και την μελλοντική επέκταση.

2 Η κατάσταση της τέχνης

Στις αρχές της δεκαετίας του 1980 είχαν δημιουργηθεί τα πρώτα ολοκληρωμένα κυκλώματα για συγκεκριμένες εφαρμογές (ASIC: Application Specific Integrated Circuit). Λόγω υψηλού κόστους παραγωγής αναπτύχθηκε η προγραμματιζόμενη λογική, δηλαδή η δημιουργία συνέσεων μεταξύ λογικών στοιχείων. Έτσι δημιουργήθηκαν οι προγραμματιζόμενοι λογικοί πίνακες (PLA: Programmable Logic Array) που αποτελούνται από λογικές πύλες AND και OR. Στη συνέχεια, η εταιρεία Altera πρωτοπόρησε με την εφαρμογή σύνθετων προγραμματιζόμενων λογικών συσκευών που αποτελούνται από πολλά μπλοκ PAL με μικρότερες διασυνδέσεις. Όμως, επειδή η τεχνολογία απαιτούσε αύξηση στις πύλες με αποτέλεσμα να αυξάνεται και το μέγεθος των συσκευών, εφευρέθηκε η τεχνολογία των FPGAs (Εικόνα 2.1). Η καινοτομία τους ήταν η αφαίρεση των πινάκων AND που παρείχαν τον προγραμματισμό και αντικαταστάθηκαν με ειδικά μπλοκ και πίνακες εναλλαγής για τον έλεγχο των συνδέσεων. Το 1985 η εταιρεία Xilinx έβγαλε το πρώτο FPGA στην αγορά (XC2064) που περιείχε 64 CLB, το καθένα με 2 LUT τριών εισόδων.



Εικόνα 2.1: Το λογικό υλικό του πρώτου FPGA

2.1 Η εξέλιξη της Xilinx

Είναι ο ηγέτης των FPGAs για πολλά χρόνια και έχει μία ποικιλία FPGAs από άποψη κόστους και επίδοσης. Τα προϊόντα της αφορούν FPGAs, SoCs καθώς και την ανάπτυξη λογισμικού.

2.1.1 Τα τσιπ της Xilinx

Μερικά από τα FPGA τσιπ που έχει αναπτύξει η Xilinx είναι τα παρακάτω:

- Σειρά Spartan 7 στα 28nm με λογικά στοιχεία που ξεκινούν από 6000 και φτάνουν μέχρι και 100.000 και χρησιμοποιούν τον MicroBlaze soft processor.
- Σειρά Artix 7 στα 28nm με λογικά στοιχεία που ξεκινούν από 12.000 και φτάνουν μέχρι και 200.000 και χρησιμοποιούν τον MicroBlaze soft processor.
- Σειρά Kintex 7 στα 28nm,20nm και 16nm και έχουν μέχρι και 1.500.000 λογικά στοιχεία
- Σειρά Virtex 7 στα 28nm,20nm και 16nmκαι έχουν μέχρι και 5.500.000 λογικά στοιχεία

Επίσης υπάρχουν και τα SoC FPGA με ARM hard processor:

- Σειρά Zynq – 7000 στα 28nm με λογικό υλικό τύπου Artix ή Kintex και λογικά στοιχεία που ξεκινάνε από 23.000 και φτάνουν τα 444.000
- Σειρά ZynqUltraScale+ MPSoC (MultiProcessor SoC) στα 16nm, ετερογενή συστήματα με επεξεργαστή ARMCortex - A53 για εφαρμογές και το λειτουργικό, επεξεργαστή πραγματικού χρόνου ARMCortex - R5 και επεξεργαστή γραφικών ARM Mali - 400, με λογικά στοιχεία που φτάνουν τα 1.000.000

2.1.2 Τα λογισμικά της Xilinx

Τα λογισμικά που έχει αναπτύξει είναι τα παρακάτω:

Vivado Design Suite: είναι εργαλείο για σχεδίαση, σύνθεση και τοποθέτηση του σχεδίου στο λογικό υλικό

Vivado HLS: αποτελεί εργαλείο για δημιουργία υλικού σε γλώσσα υψηλού επιπέδου

System Generator for DSP: είναι ένα εργαλείο για την δημιουργία κώδικα περιγραφής υλικού από αλγόριθμους ψηφιακής επεξεργασίας σήματος μέσα από το Matlab και το Simulink.

Model Composer: είναι ένα εργαλείο για την δημιουργία υλικού με τη βοήθεια του Matlab και του Simulink.

2.1.3 Αναπτυξιακές πλακέτες με τσιπ της Xilinx

Παρακάτω αναπτύσσονται μερικές αναπτυξιακές πλακέτες που βασίζονται στα τσιπ της Xilinx:

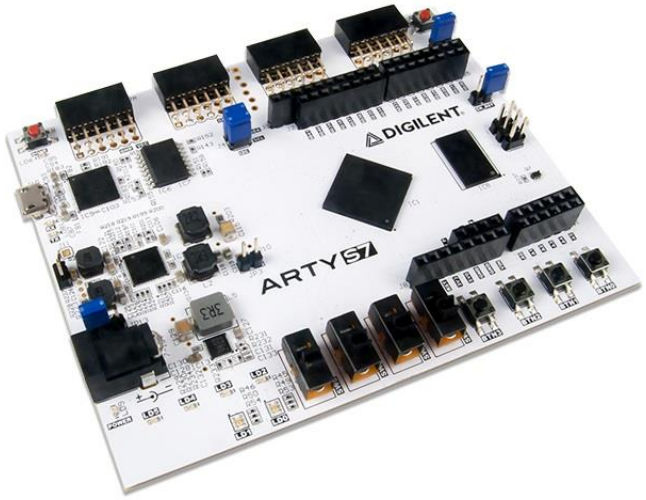
I. **Arty S7-50 Spartan FPGA** (Εικόνα 2.2)

Περιλαμβάνει τσιπ της σειράς Spartan 7 και αποτελεί μία οικονομική λύση.

Λειτουργεί με τάση 5V από την θύρα micro-USB ή εξωτερική τάση 7V - 15V μέσω αντίστοιχης θύρας και η σύνδεση με τον υπολογιστή πραγματοποιείται από την θύρα micro - USB.

Παρέχει εξωτερικό παλμό χρονισμού 100MHz για στοιχεία στο λογικό υλικό και τσιπ για μετατροπή αναλογικού σε ψηφιακού σήματος.

Διαθέτει 4 θύρες επέκτασης, υποδοχή για Arduino, 4 μπουτόν, 4 διακόπτες, 4 πράσινα και 2 κόκκινα LEDs, 1 μπουτόν για reset της συσκευής και 1 για reset του FPGA.



Εικόνα 2.2: Η πλακέτα ArtyS7-50 με το τσιπ Spartan

II. ZYBO Z7-10 (Εικόνα 2.3)

Βασίζεται στο FPGA SoC τσιπ της σειράς Zynq 7000 που ενσωματώνει τον επεξεργαστή 2 πυρήνων ARM Cortex - A9 και μνήμη DDR3 1GB 1066MHz. Παρέχει παλμό χρονισμού 33,33MHz στον επεξεργαστή και η μέγιστη συχνότητα λειτουργίας του είναι 667MHz. Παρέχει και εξωτερικό παλμό χρονισμού στα 125 MHz για να χρονίσει υλικά που αποτυπώνονται στο λογικό υλικό.

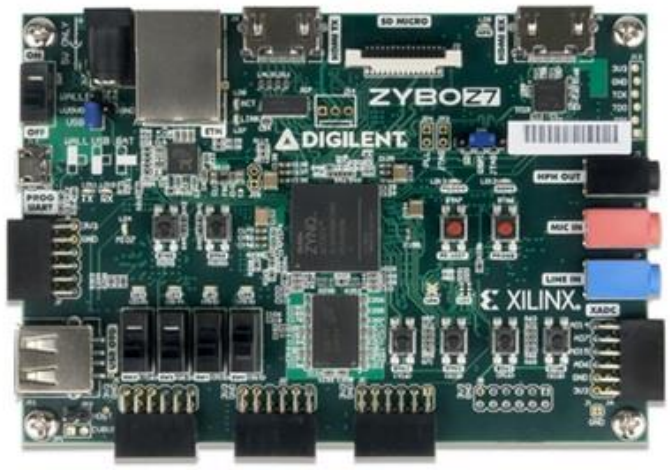
Λειτουργεί μόνο με τάση 5V είτε από την θύρα micro - USB είτε απο εξωτερική τάση 5V μέσω αντίστοιχης θύρας και η σύνδεση με τον υπολογιστή πραγματοποιείται από την θύρα micro - USB.

Το λειτουργικό σύστημα (Ubuntu 18.04) φορτώνεται από την κάρτα micro SD που τοποθετείται στην κατάλληλη θύρα.

Περιλαμβάνει μία θύρα για είσοδο εικόνας (HDMIin) και μία θύρα για έξοδο (HDMIout). Υπάρχει μία θύρα Ethernet απαραίτητη για τον προγραμματισμό του τσιπ. Διαθέτει 5 επεκτάσιμες θύρες για υποδοχή περιφερειακών και θύρα USB για χρήση συσκευών. Παρέχει ειδική θύρα για

σύνδεση κάμερας με διεπαφή MIPI (Mobile Industry Processor Interface) CSI-2 (Camera Serial Interface) που προσφέρει καλύτερη απόδοση σε σχέση με σύνδεση κάμερας στην θύρα USB. Όσον αφορά την επεξεργασία ήχου παρέχει μία είσοδο mono,μία είσοδο stereo και μία έξοδο.

Έχει 6 μπουτόν, 4 διακόπτες, 5 LEDs 1 RGB LED, 1 μπουτόν για επαναφορά του PL και 1 μπουτόν για επεναφορά όλης της συσκευής.



Εικόνα 2.3: Η πλακέτα Zybo Z7-10 με το τσιπ Zynq 7000

III. Virtex-7 VC707 Evaluation Kit (Εικόνα 2.4)

Βασίζεται στι τσιπ Virtex 7 και προορίζεται για πολύπλοκες εφαρμοφές.

Τροφοδοτείται με τάση 12V, έχει οθόνη LCD και συνδέεται σε υπολογιστή μέσω της θύρας PCI. Επιπλέον, διαθέτει μνήμη DDR3 1GB SODIMM 800MHz και παλμό χρονισμού 200MHz.

Διαθέτει 2 θύρες για προσθήκη καρτών FMC (FPGA Mezzanine Card), θύρες SMA (SubMiniature version A) για είσοδο και έξοδο παλμών ρολογιού, θύρα HDMI, θύρα Micro – USB. Ακόμη, διαθέτει θύρα SFP (Small Form - Factor Pluggable) για σύνδεση σε δίκτυο, μπουτόν, διακόπτες και άλλα.

Εικόνα 2.4: Virtex-7 VC707 Evaluation Kit

2.2 Η εξέλιξη της Intel (πρώην Altera)

Η Intel αγόρασε την Altera το 2015. Τα προϊόντα της αφορούν FPGAs, SoCs, CPLDs και ASICs. Καλύπτει τις χαμηλές, μεσαίες και ανώτερες αγορές με τα CPLD και αγορές υψηλών προδιαγραφών με FPGA.

2.2.1 Τα τσιπ της Intel

Μερικά από τα FPGA τσιπ που έχει αναπτύξει η Intel είναι τα παρακάτω:

- Σειρά Max 10 στα 55nm με λογικά στοιχεία που ξεκινούν από 2.000 και φτάνουν τα 50.000 και χρησιμοποιούν τον NIOS II soft processor
- Σειρά Cyclone 10 στα 20nm με λογικά στοιχεία από 85.000 μέχρι 220.000 και χρησιμοποιούν τον NIOS II soft processor
- Σειρά Arria 10 20nm με λογικά στοιχεία από 160.000 - 1.000.000
- Σειρά Stratix10 14nm με λογικά στοιχεία από 378.000 - 10.000.000

Επίσης υπάρχουν και τα SoCFPGA με ARM hard processor:

- Σειρά Arria V SoC στα 28nm με λογικά στοιχεία από 350.000 - 462.000 και επεξεργαστή ARM Cortex - A9
- Σειρά Cyclone V SoC στα 28nm με λογικά στοιχεία από 25.000 - 110.000 και επεξεργαστή ARM Cortex - A9

- Σειρά Arria 10 SoC στα 20nm με λογικά στοιχεία από 160.000 - 600.000 και επεξεργαστή ARM Cortex - A9
- Σειρά Stratix 10 MPSoC στα14nm με λογικά στοιχεία από 378.000 - 10.000.000 και 4 επεξεργαστές ARM Cortex - A53
- Σειρά Agilex MPSoC στα 10nm με λογικά στοιχεία από 390.000 - 2.700.000 και 4 επεξεργαστές ARM Cortex - A53

2.2.2 Τα λογισμικά της Intel

Μερικά από τα λογισμικά που έχει αναπτύξει είναι τα παρακάτω:

Quartus Prime Design Software: είναι εργαλείο για σχεδίαση, σύνθεση και τοποθέτηση του σχεδίου στο λογικό υλικό

HLS compiler: αποτελεί εργαλείο για δημιουργία υλικού σε γλώσσα υψηλού επιπέδου

NIOS II Embedded Design Suite (EDS): περιέχει εργαλεία ανάπτυξης λογισμικών με τον επεξεργαστή λογισμικού NIOS II.

DSP Builder: είναι ένα εργαλείο για την δημιουργία κώδικα περιγραφής υλικού από αλγόριθμους ψηφιακής επεξεργασίας σήματος μέσα από το Matlab και το Simulink.

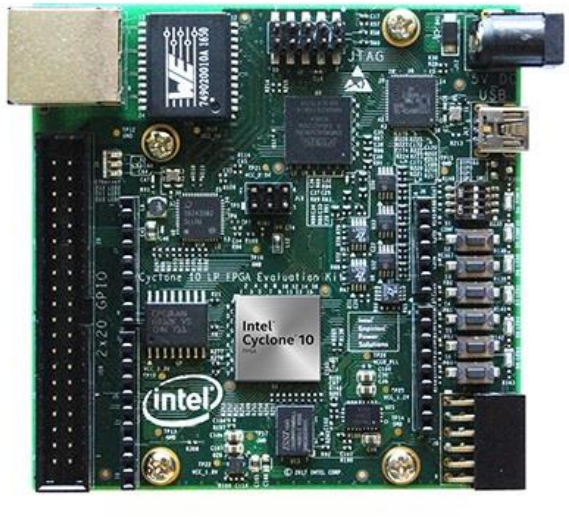
2.2.3 Αναπτυξιακές πλακέτες με τσιπ της Intel

I. Cyclone 10 LP FPGA Evaluation Kit (Εικόνα 2.5)

Βασίζεται στο τσιπ της σειράς Cyclone 10 και αποτελεί μια οικονομική λύση για ανάπτυξη εφαρμογών.

Τροφοδοτείται με τάση 5V είτε από θύρα Mini - USB είτε μέσω αντάπτορα.

Ο μέγιστος παλμός χρονισμού του PL είναι 50MHz και υπάρχει δυνατότητα δημιουργίας επιθυμητού παλμού μέχρι 125 MHz. Διαθέτει θύρα για σύνδεση με Arduino, 1 θύρα για περιφερειακές συσκευές και 40 pins γενικού σκοπού και 1 θύρα Ethernet. Περιλαμβάνει 4 μπουτόν και 4 LEDs γενικού σκοπού, 1 μπουτόν για επαναφορά της συσκευής και 1 μπουτόν για επαναφορά των καταχωρητών του PL.



Εικόνα 2.5: Η πλακέτα Cyclone 10 LP

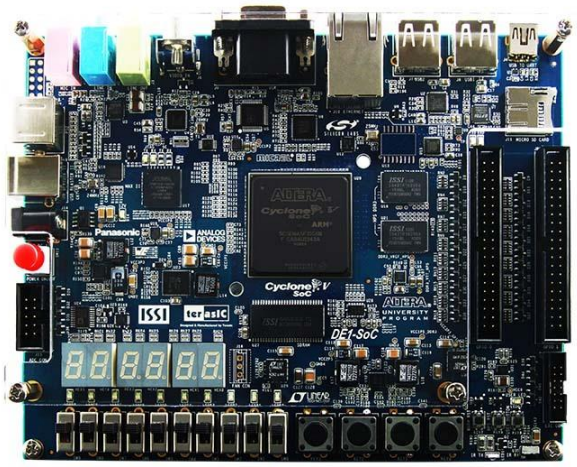
II. Cyclone V DE1 SoC (Εικόνα 2.6)

Βασίζεται στο Cyclone V SoC τσιπ που ενσωματώνει τον επεξεργαστή 2 πυρήνων ARM Cortex - A9 με μέγιστη συχνότητα λειτουργίας 925MHz, μέγιστο παλμό χρονισμού 50MHz και μνήμη DDR3 1GB.

Τροφοδοτείται από τάση 12V DC μέσω αντάπτορα, έχει υποδοχή PS / 2 για σύνδεση πληκτρολογίου / ποντικιού και θύρα Ethernet. Επίσης, μπορεί να γίνει επεξεργασία ήχου αφού διαθέτει κανάλια για είσοδο και έξοδο και εικόνας με είσοδο από θύρα TV - in και έξοδο απο θύρα VGA.

Ακόμη, περιλαμβάνει 2 θύρες USB, 80 pin γενικού σκοπού και 1 θύρα επέκτασης. Διαθέτει 6 7 - segment displays, 4 μπουτόν, 10 διακόπτες και 10 LEDs για αλληλεπίδραση με τον χρήστη.

Το λειτουργικό σύστημα Linux φορτώνεται από την κάρτα SD.

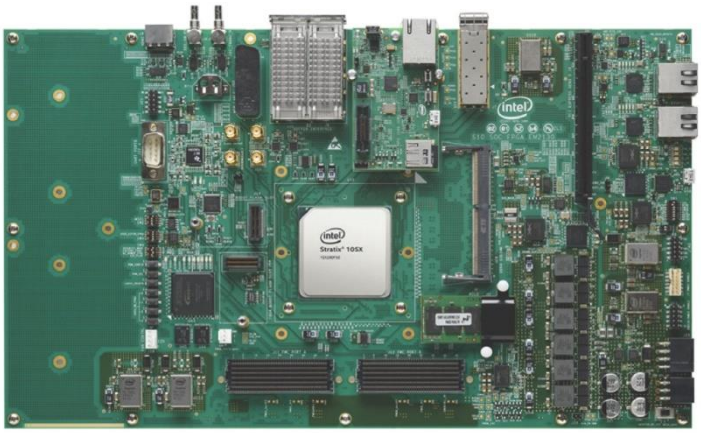


Εικόνα 2.6: Η πλακέτα Cyclone 5 DE1 SoC

III. **Stratix 10 SX SoC Development Kit** (Εικόνα 2.7)

Βασίζεται στο Stratix 10 MPSoC που ενσωματώνει επεξεργαστή 4 πυρήνων ARM Cortex - A53 με μέγιστη συχνότητα λειτουργίας 1.5GHz και μία μνήμη DDR4 SO-DIMM 1200 MHz 16GB. Περιλαμβάνει μία κάρτα μνήμης DDR4 SDRAM 1066 MHz 4 GB, 1 επεκτάσιμη κάρτα με θύρα Ethernet, mini - USB, USB και υποδοχή για κάρτα SD.

Διαθέτει υποδοχές για είσοδο παλμών χρονισμού,υποδοχή PCIe και HDMI. Έχει 2 θύρες για προσθήκη καρτών FMC και θύρες SFP.



Εικόνα 2.7: Η πλακέτα Stratix 10 SX SoC

2.3 Η εξέλιξη της LatticeSemiconductor

Η Lattice Semiconductor παρέχει προϊόντα FPGA και IP χαμηλής ισχύος στους καταναλωτές, τις επικοινωνίες, τη βιομηχανία, τους υπολογιστές και τις αγορές αυτοκινήτων παγκοσμίως. Είναι γνωστή για την παροχή FPGA χαμηλού κόστους με δυνατότητες χαμηλής ισχύος.

2.3.1 Τα τσιπ που ανέπτυξε

Μερικά από τα FPGA τσιπ που έχει αναπτύξει η Lattice είναι τα παρακάτω:

- Σειρά ICE40 LP / HX / LM: στα 40nm με λογικά στοιχεία που ξεκινούν από 384 και φτάνουν τα 7600
- Σειρά ICE40 Ultra / UltraLite: στα 40nm με λογικά στοιχεία που ξεκινούν από 600 και φτάνουν τα 3500
- Σειρά ICE40 UltraPlus: στα 40nm με λογικά στοιχεία που ξεκινούν από 2800 και φτάνουν τα 5200
- Σειρά ECP5: στα 40nmμε λογικά στοιχεία που ξεκινούν από 12.000 και φτάνουν τα 84.000
- Σειρά MachXO3D PLD: στα 65nm με λογικά στοιχεία που ξεκινούν από 4.300 και φτάνουν τα 9.400

Δεν έχει αναπτύξει SoC FPGA,όμως χρησιμοποιεί σαν soft processor τον LatticeMico των 8 ή 32 bit

2.3.2 Τα λογισμικά τηςLatticeSemiconductor

Τα λογισμικά που έχει αναπτύξει είναι τα παρακάτω:

Lattice Diamond Design Software: είναι εργαλείο για σχεδίαση, σύνθεση και τοποθέτηση του σχεδίου στο λογικό υλικό για τα FPGA της σειράς ECP και MachXO

iCEcube2 Design Software: είναι εργαλείο για σχεδίαση, σύνθεση και τοποθέτηση του σχεδίου στο λογικό υλικό για τα FPGA της σειράς ICE40

Radiant Design Software: είναι εργαλείο για σχεδίαση, σύνθεση και τοποθέτηση του σχεδίου στο λογικό υλικό για τα FPGAτης σειράς ICE40 UltraPlus

Neural Network Compiler: χρησιμοποιείται για την εφαρμογή νευρωνικών δικτύων που έχουν αναπτυχθεί με βιβλιοθήκες Tensorflow, Keras και Caffe στην προγραμματιζόμενη λογική.

Lattice Mico System Development Tools: με το εργαλείο αυτό,ο χρήστης μπορεί να διαμορφώσει έναν από τους δύο επεξεργαστές λογισμικού που θα χρησιμοποιήσει στην εφαρμογή του και να υλοποιήσει κώδικα (σε γλώσσα C /C ++) που θα εκτελεί ο επεξεργαστής.

2.3.3 Αναπτυξιακές πλακέτες με τσιπ της Lattice

I. ECP5 Versa Development Kit (Εικόνα 2.8)

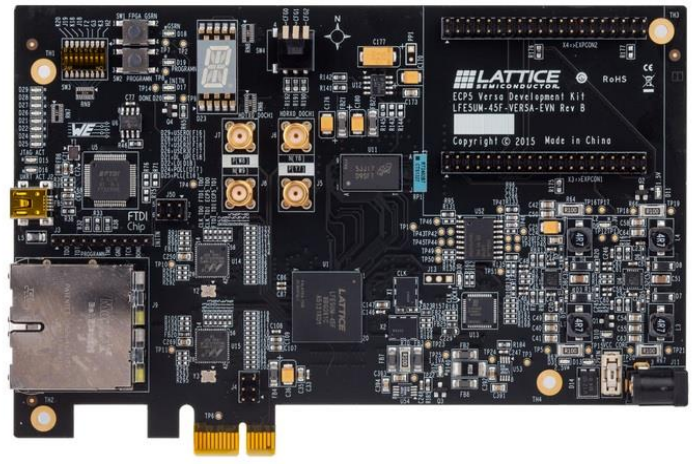
Βασίζεται στο τσιπ της σειράς ECP5 και περιλαμβάνει μνήμη DDR3 που λειτουργεί στα 933MHz.

Κάνει δυνατή την σύνδεση με υπολογιστή μέσω της υποδοχής PCI express από όπου και μπορεί να τροφοδοτηθεί ή με χρήση αντάπτορα με τάση 12V DC.

Ο παλμός χρονισμού μπορεί να προέρχεται από έναν ταλαντωτή που βρίσκεται στην πλακέτα (156,25MHz) ή από την σύνδεση PCI. Και οι 2 παλμοί μπορούν να χρησιμοποιηθούν για το PL ή για τα 2 ζευγάρια SERDES (serializer/deserializer). Υπάρχει και ένας ακόμη ταλαντωτής με παλμό 100MHz γενικής χρήσης.

Ένα κύκλωμα SERDES μετατρέπει πολλαπλές ροές δεδομένων σε μία ροή που εκπέμπεται με υψηλή ταχύτητα σε έναν δέκτη που κάνει την αντίστροφη διαδικασία και γιατό χρειάζεται ένας παλμός χρονισμού για τα δεδομένα. Σκοπός του κυκλώματος είναι η επιτάχυνση της επικοινωνίας χωρίς την αύξηση των pins. Στην πλακέτα υπάρχει ένα ζευγάρι για την μετάδοση και ένα για την λήψη.

Περιλαμβάνει 2 θύρες Ethernet, ένα 14 - segment display, 8 LEDs και 8 dip switches γενικού σκοπού και 80 pin επέκτασης.



Εικόνα 2.8: Η πλακέτα ECP5 Versa

II. MachXO2-4000HC Control Evaluation Board (Εικόνα 2.9)

Βασίζεται στο PLD τσιπ της σειράς MachXO2 4000 που ενσωματώνει μνήμη υλικού RAM, στα 65nm με λογικά στοιχεία που φτάνουν τα 4000.

Τροφοδοτείται με τάση 5V DC μέσω αντάπτορα και προγραμματίζεται από τον υπολογιστή μέσω της θύρας mini - USB.

Υποστηρίζει επεξεργασία εικόνας και ήχου αφού έχει 2 θύρες DVI για έξοδο εικόνας, 2 θύρες DVI για είσοδο εικόνας, 1 θύρα για έξοδο ήχου jack 3.5mm και 1 μικρόφωνο.

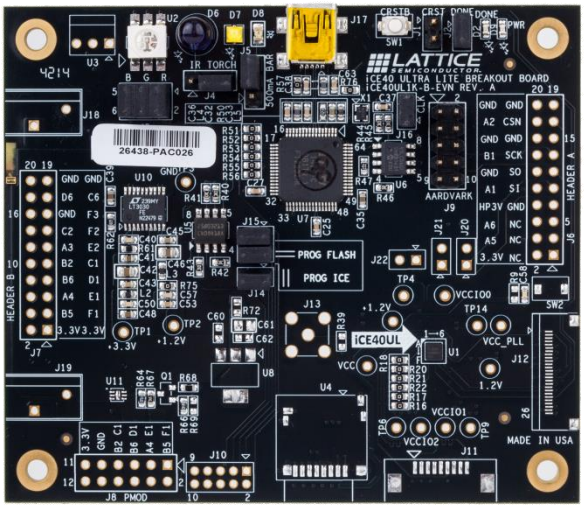
Περιλαμβάνει 4 LEDs, 4 Dip switches και 40 pins γενικού σκοπού.

Εικόνα 2.9: Η πλακέτα MachXO2-4000HC

III. iCE40 UltraLite Breakout Board (Εικόνα 2.10)

Βασίζεται στο πολύ μικρού μεγέθους τσιπ iCE40 UltraLite και έχει παλμό χρονισμού 12MHz.

Διαθέτει θύρα mini - USB για τον προγραμματισμό μέσω υπολογιστή, μπουτόν επαναφοράς και πομπό υπερύθρων. Επίσης, υποστηρίζει 37 pins γενικού σκοπού και και 12 pins για σύνδεση περιφερειακών συσκευών.



Εικόνα 2.10: Η πλακέτα ICE40UltraLite

2.4 Η εξέλιξη της Microsemi

Η Microsemi Corporation, θυγατρική της Microchip Technology Inc. προσφέρει ένα ολοκληρωμένο πακέτο ημιαγωγών για λύσεις συστημάτων επικοινωνίας, ασφάλειας και άμυνας, αεροδιαστημικής και βιομηχανίας. Τα προϊόντα περιλαμβάνουν ενσωματωμένα κυκλώματα αναλογικών μικτών σημάτων υψηλής απόδοσης και ακτινοβολίας, FPGAs, SoCs και ASIC και προϊόντα διαχείρισης ισχύος.

2.4.1 Τα τσιπ της Microsemi

Μερικά από τα τσιπ που ανέπτυξε είναι τα παρακάτω:

- PolarFire στα 28nm με λογικά στοιχεία που ξεκινούν από 100.000 και φτάνουν έως 500.000 και έχουν soft processor αρχιτεκτινικής RISC - V
- Igloo2 με λογικά στοιχεία που ξεκινούν από 6.000 και φτάνουν έως 150.000
- ProASIC3 με λογικά στοιχεία που ξεκινούν από 300 και φτάνουν έως 35.000 και έχουν τον ARM Cortex - M1 soft processor
- Igloo/e με λογικά στοιχεία που ξεκινούν από 300 και φτάνουν έως 35.000 και έχουν τον ARM Cortex - M1 soft processor
- Igloo nano με λογικά στοιχεία που ξεκινούν από 100 και φτάνουν έως 3.000
- Igloo Plus με λογικά στοιχεία που ξεκινούν από 300 και φτάνουν έως 1.500

Κάποια από τα SoC FPGA είναι:

- PolarFire SoC με λογικά στοιχεία που ξεκινούν από 25.000 και φτάνουν έως 460.000 και χρησιμοποιεί τον επεξεργαστή RISC - V 5 πυρήνων
- SmartFusion2 με λογικά στοιχεία που ξεκινούν από 5.000 και φτάνουν έως 150.000 και χρησιμοποιούν τον επεξεργαστή ARM Cortex - M3

2.4.2 Τα λογισμικά της Microsemi

Τα λογισμικά που έχει αναπτύξει είναι τα παρακάτω:

Libero SoC Design Suite: είναι ένα εργαλείο που προσφέρει την σχεδίαση, σύνθεση, αποσφαλμάτωση, ανάλυση και τοποθέτηση του σχεδίου στο λογικό υλικό. Ενσωματώνει το λογισμικό Synopsys Sinplify Pro για την σύνθεση και το Mentor Graphics ModelSim ME για την πιστωποίηση του κώδικα HDL. Υποστηρίζει τα τσιπ της σειράς SmartFusion, PolarFire, IGLOO.

Libero IDE: είναι το ίδιο εργαλείο με το Libero SoC με τη διαφορά ότι υποστηρίζει διαφορετικά τσιπ όπως τσιπ ανεκτικά στην ραδιενέργεια.

Microsemi DSP: δίνει την δυνατότητα στον χρήστη να μετατρέψει κώδικα που υλοποιήθηκε με τη βοήθεια του Matlab και του Simulink σε γλώσσα HDL.

2.4.3 Αναπτυξιακές πλακέτες με τσιπ της Microsemi

I. PolarFire FPGA Video kit (Εικόνα 2.11)

Τροφοδοτείται με τάση 5V DC από την θύρα mini - USB από όπου και προγραμματίζεται ή με εξωτερική τάση 12V DC μέσω αντάπτορα.

Διαθέτει 2 θύρες εξόδου HDMI και 1 εισόδου και 2 αισθητήρες κάμερας. Ακόμη, υποστηρίζει σύνδεση κάμερας με διεπαφή MIPI CSI - 2 και έξοδο μέσω της κατάλληλης θύρας και έχει υποδοχή για προσθήκη καρτών FMC.

Περιλαμβάνει 4 μνήμες DDR4 με συνολική χωρητικότητα 2Gb, 1 ταλαντωτή των 50MHz για τον χρονισμό του λογικού υλικού και έναν των 148.5MHz που χρησιμοποιείται σαν παλμός αναφοράς σε 2 pins.

Στην πλακέτα βρίσκονται 2 μπουτόν, 4 LEDs και 4 Dip switches γενικού σκοπού, 1 μπουτόν επαναφοράς της συσκευής και 1 μπουτόν επαναφοράς του PL.

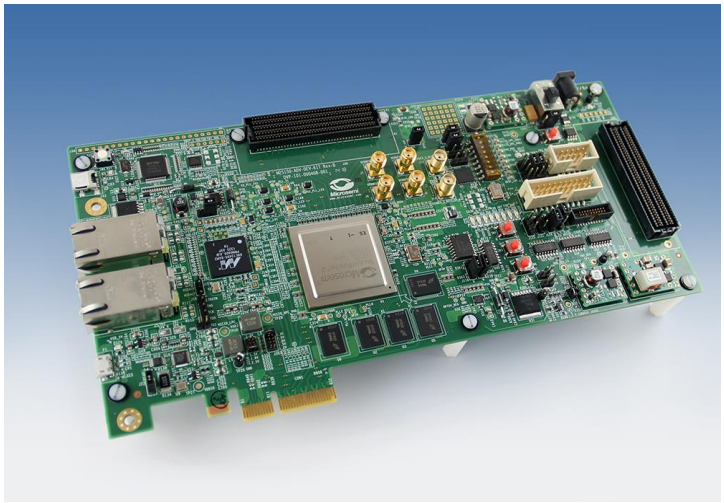
Εικόνα 2.11: PolarFire FPGA Video kit

II. SmartFusion2 Advanced Development Kit (Εικόνα 2.12)

Βασίζεται στο SoC τσιπ SmartFusion2 με τον επεξεργαστή υλικού ARM Cortex - M3 με μέγιστη συχνότητα λειτουργίας 166MHz. Χρησιμοποιεί 4 ταλαντωτές που βγάζουν παλμό χρονισμού 32.7, 50, 100 και 125 MHz αντίστοιχα. Διαθέτει 4 μνήμες DDR3 με συνολοκή χωρητικότητα 1 GB, 4 συνδέσεις SERDES και 2 εισόδους για παλμό ρολογιού αναφοράς.

Περιλαμβάνει 2 θύρες Ethernet, 2 θύρες FMC και PCI express για σύνδεση σε υπολογιστή.

Έχει 4 μπουτόν, 8 LEDs και 8 dip switches γενικού σκοπού καθώς και 1 μπουτόν επαναφοράς του τσιπ.

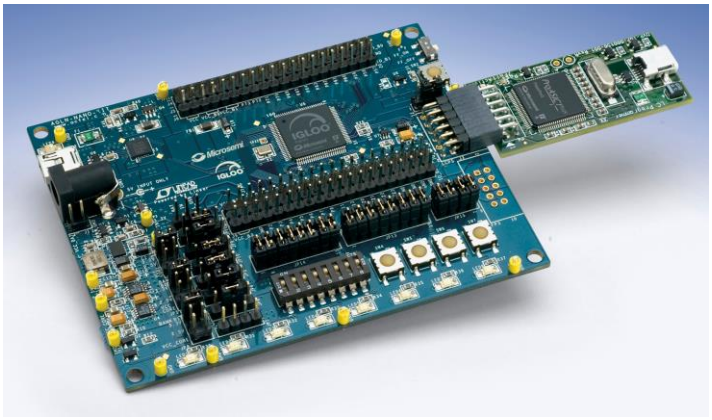


Εικόνα 2.12: SmartFusion2 Advanced Development Kit

III. IGLOO Nano (Εικόνα 2.13)

Τροφοδοτείται με τάση 5V από αντάπτορα και επικοινωνεί με τον υπολογιστή μέσω θύρας mini - USB. Περιλαμβάνει ταλαντωτή των 20MHz και 120 pins γενικού σκοπού.

Διαθέτει 4 μπουτόν, 8 LEDs και 8 dip switches γενικού σκοπού καθώς και 1 μπουτόν επαναφοράς.



Εικόνα 2.13: IGLOO Nano

3 Τα εργαλεία που χρησημοποιήθηκαν

Σε αυτό το κεφάλαιο θα αναπτύξουμε την μεθοδολογία και τα εργαλεία που χρησιμοποιήθηκαν. Το βασικό εργαλείο για την ανάπτυξη των εφαρμογών μας είναι το Vivado Design Tool εκτός από την εφαρμογή του ανιχνευτή ακμών όπου χρησιμοποιήθηκε block που εκτελεί την εξαγωγή ακμών, φτιαγμένο μεσα από το Vivado HLS, χωρίς την παρουσίαση του τρόπου και του κώδικα που χρειάστηκε για την υλοποίηση του.

3.1 Vivado Design Suite

Είναι ένα πρόγραμμα της εταιρείας Xilinx που προσφέρει την δυνατότητα σχεδίασης, σύνθεσης και τοποθέτησης των σχεδίων στο λογικό υλικό των συσκευών της σειράς 7. Επιταχύνει την εφαρμογή σχεδίων και παρέχει εργαλεία τοποθέτησης και δρομολόγησης για την βελτιστοποίηση παραμέτρων όπως ο χρόνος και η κατανάλωση ενέργειας, καθ' όλη τη διάρκεια σχεδίασης.

Το Vivado IDE κάνει δυνατή την οπτικοποίηση και την αλληλεπίδραση με το σχέδιο σε κάθε στάδιο ανάπτυξης, δηλαδή μπορεί κάποιος να ανοίξει ένα σχέδιο και να εμφανιστεί η λίστα με τα στοιχεία που το αποτελούν σε κάθε στάδιο της ανάπτυξης, να δηλώσει τους περιορισμούς και να εφαρμοστούν στην συσκευή. Δίνει την δυνατότητα σύνθεσης, ανάλυσης και εφαρμογής σχεδίων και αναφορά των αποτελεσμάτων.

Ένα σχέδιο μπορεί να αποτελείται από

- ένα υλικό RTL (Register Transfer Level) όπου η περιγραφή του δίνεται σε γλώσσα χαμηλού επιπέδου (VHDL, Verilog και SystemVerilog),
- IP που υπάρχουν στην βιβλιοθήκη του Vivado
- ή IP τρίτων.

Επίσης, παρέχει την δυνατότητα στον χρήστη να δημιουργήσει δικά του IP με το πρωτόκολο AXI, με το κατάλληλο εργαλείο που περιέχεται στο Vivado IDE, με την χρήση του λογισμικού Vivado HLS ή του Model Composer μέσα από το Matlab.

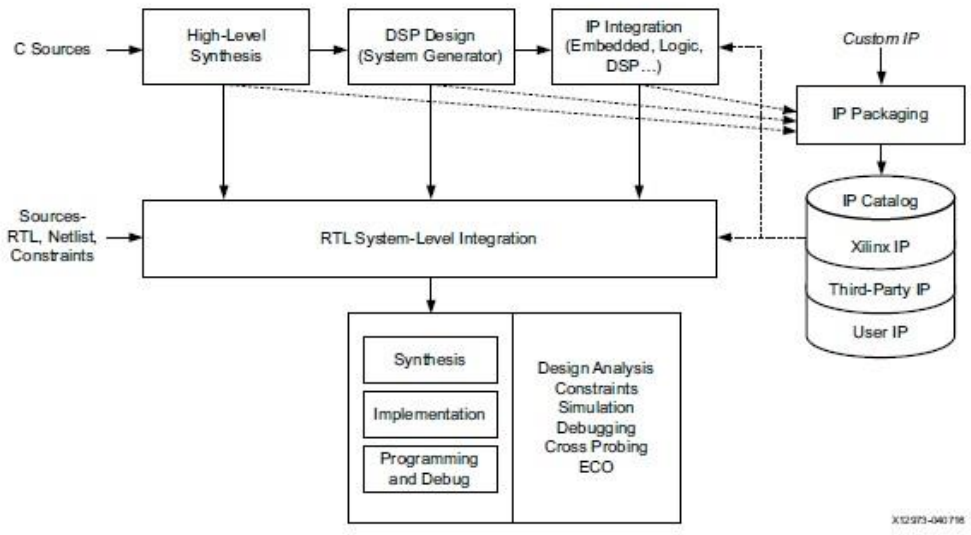
Η σύνθεση είναι η διαδικασία μετατροπής σχεδίου RTL σε απεικόνιση επιπέδου λογικών πυλών που ονομάζεται netlist · είναι χρονικά οδηγούμενη και βελτιστοποιημένη για την χρήση μνήμης και απόδοσης.

Η τοποθέτηση είναι μία διαδικασία που περιλαμβάνει όλα τα βήματα για την δρομολόγηση και τοποθέτηση του netlist στο PL, συμπεριλαμβάνοντας τους λογικούς, φυσικούς και χρονικούς περιορισμούς.

Μερικά χαρακτηριστικά είναι:

- Σχεδίαση RTL σε γλώσσα VHDL,Verilog και SystemVerilog
- Δημιουργία IP
- Σύνθεση
- Υλοποίηση για τοποθέτηση και δρομολόγηση
- Προσθήκη φυσικών και χρονικών περιορισμών
- Συμπεριφορική, λειτουργική και χρονική προσομείωση
- Χρονική και ενεργειακή ανάλυση
- Δημιουργία bitstream
- Λογική ανάλυση για αποσφαλμάτωση
- Καθορισμός συνδέσεων μεταξύ του σχεδίου και των φυσικών στοιχείων της συσκευής

Το διάγραμμα ροής του Vivado Design Tool φαίνεται στην Εικόνα 3.1.



Εικόνα 3.1: Το διάγραμμα ροής του Vivado Design Tool [108]

Στις εφαρμογές της εργασίας που παρουσιάζονται υπάρχουν 5 φάσεις για τον τρόπο υλοποίησης:

i. Πρώτα πραγματοποιείται η σχεδίαση με την χρήση μπλοκ πνευματικής ιδιοκτησίας (IP) που είναι επαναχρησιμοποιήσιμα και τα προσφέρει η εταιρεία ή IP που μπορεί να φτιάξει ο χρήστης
ii. Έπειτα, προστίθενται οι φυσικοί περιορισμοί της αντίστοιχης πλακέτας.
iii. Εκτελείται η σύνθεση δηλαδή η μετατροπή του σχεδίου σε επίπεδο λογικών πυλών RTL

iv. Εκτελείται η εφαρμογή δηλαδή η μετατροπή των λογικών στοιχείων από την διαδικασία της σύνθεσης και των φυσικών περιορισμών, σε σχέδιο έτοιμο να τοποθετηθεί στο λογικό υλικό.

v. Δημιουργείται το αρχείο bitstream.

3.2 Τρόποι περιγραφής υλικού

Ο χρήστης μπορεί να δημιουργήσει το δικό του IP μπλοκ με 2 τρόπους:

1. Με γλώσσα χαμηλού επιπέδου (Verilog / VHDL).
2. Με γλώσσα υψηλού επιπέδου (C, C ++, SystemC)

3.2.1 VHDL

Είναι μια πρότυπη γλώσσα που περιγράφει την λειτουργία ή την δομή ενός ηλεκτρονικού κυκλώματος από το οποίο ένα συμβατό φυσικό κύκλωμα μπορεί να συναχθεί από έναν μεταγλωττιστή. Η πρώτη έκδοση δημιουργήθηκε το 1987 και η πιο πρόσφατη είναι του 2008.

Οι βασικές εφαρμογές της είναι:

- Σύνθεση: μετατροπή του κώδικα σε υλικό που εκτελεί συγκεκριμένη λειτουργία έτοιμο να εφαρμοστεί σε συσκευή CPLD / FPGA
- Προσωμείωση: έλεγχος της λειτουργίας του υλικού σύμφωνα με αυτό που δημιουργήθηκε από την διαδικασία της σύνθεσης

Μία βασική δομή ενός κώδικα VHDL αποτελείται από:

- **Δήλωση βιβλιοθηκών και πακέτων**: είναι απαραίττα κωμάτια κώδικα για την περιγραφή του υλικού, που μπορούν να επαναχρησιμοποιηθούν. Οι συνηθέστερες βιβλιοθήκες είναι η ieee και η std.
    - Τα πακέτα **std**:
        - **standard**: που περιέχει διάφορες δηλώσεις τύπων δεδομένων (bit, integer, Boolean, character και άλλα) και την κατάλληλη λογική (αρίθμηση, σύγκριση, ολίσθηση)
        - **textio**: για αρχεία κειμένου.
    - Κάποια από τα πακέτα **ieee**:
        - **std_logic_1164**: περιέχει τις 9 τιμές των τύπων δεδομένων std_(u)logic, std_(u)logic_vector
        - **numeric_std**: εισάγει τους τύπους signed και unsigned που έχουν σαν βάση τον τύπο std_logic, με τους αντίστοιχους τελεστές

- **ENTITY:** καθορίζει τις κύριες θύρες εισόδου / εξόδου και τις γενικές σταθερές. Ότι βρίσκεται στο πεδίο PORT αποτελούν τα σήματα, δηλαδή ότι μπαίνει και βγαίνει στο κύκλωμα. Στο πεδίο GENERIC, πριν από το BEGIN, εισάγονται οι καθολικές σταθερές. Η οντότητα μπορεί να είναι μόνο μία ή περισσότερες και τότε εκφράζει αυτή του υψηλότερου επιπέδου (top - level) που περιέχει τις υπόλοιπες.
- **ARCHITECTURE:** περιλαμβάνει την λειτουργία του υλικού που τοποθετείται μετά το πεδίο BEGIN, ενώ πρίν δηλώνονται, προαιρετικά, εξαρτήματα καθώς και αντικείμενα όπως αυτά στο ENTITY.

Η VHDL χρησιμοποιεί κάποιους τελεστές όπως οι παρακάτω:

- **Τελεστές ανάθεσης:**
  - ¨< =¨ : αναθέτει μία τιμή σε ένα σήμα
  - **¨:** = ¨ : αναθέτει μία τιμή σε σταθερές και μεταβλητές και αρχικοποιεί τιμές σε σήματα και μεταβλητές.
  - ¨= >¨**:** αναθέτει τιμές σε στοιχεία πινάκων, ξεχωριστά ή μαζί με την λέξη OTHERS
- **Λογικοί τελεστές:**
  - NOT
  - OR
  - AND
  - NAND
  - XOR
  - XNOR
- **Συνδετικός τελεστής:** χρησιμοποιείται για να ενώσει αντικείμενα και τιμές με το σύμβολο ¨&¨. Στο παρακάτω παράδειγμα φαίνεται η λειτουργία του:
  X < = ‘ 0’ & ‘1’ ;
  Το X θα πάρει την τιμή 01

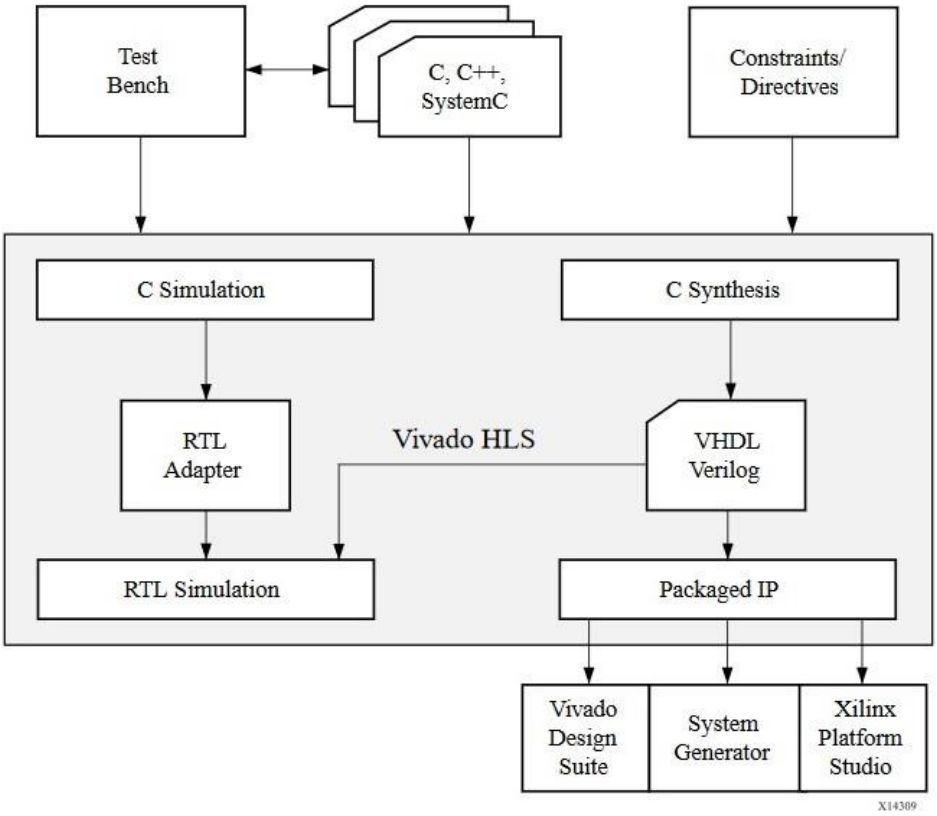
Για τον ιεραρχικό σχεδιασμό, δηλαδή την δημιουργία κυκλώματος υψηλού επιπέδου (top - level) που αποτελείται από άλλα υποκυκλώματα, χρησιμοποιείται η δομή COMPONENT παρέχοντας την δυνατότητα πολλαπλής χρήσης του υποκυκλώματος. Αφού περιγραφεί η οντότητα και η αρχιτεκτονική του υποκυκλώματος, δηλώνεται, πριν από το BEGIN της αρχιτεκτονικής του κυκλώματος υψηλού επιπέδου, το υποκύκλωμα με την δομή COMPONENT (ίδια δομή με αυτή του ENTITY). Στη συνέχεια, μετά από το BEGIN, δηλώνεται ένα ή περισσότερα στιγμιότυπα του υλικού που θέλουμε να τοποθετηθεί στο κύκλωμα υψηλού επιπέδου, δίνοντας μία επιγραφή (label) και με το

PORT MAP αντιστοιχούνται οι θύρες top - level με αυτές του υποκυκλώματος. Αυτός ο τρόπος σχεδιασμού ονομάζεται κατασκευαστικός (structural).

3.2.2 VivadoHLS

Είναι ένα εργαλείο που δίνει την δυνατότητα στον χρήστη να δημιουργήσει ένα IP με τη χρήση γλώσσας υψηλού επιπέδου όπως η C / C++ και η SystemC. Ο χρήστης μπορεί να περιγράψει την λειτουργία του υλικού σε γλώσσα C και το Vivado HLS τη μετατρέπει σε επίπεδο RTL (VHDL / Verilog) και ύστερα εξάγει το IP block.

Στην Εικόνα 3.2 παρουσιάζεται το διάγραμμα ροής του Vivado HLS.



Εικόνα 3.2: Το διάγραμμα ροής του Vivado HLS [100]

Τρόποι φόρτωσης του αρχείου σχεδίασης

Για την υλοποίηση μίας εφαρμογής υπάρχουν 3 τρόποι για να προγραμματίσουμε το σύστημα μας ανάλογα με την χρήση PS - PL

1. Στην περίπτωση που η εφαρμογή μας δεν χρησιμοποιεί επεξεργαστή (δηλαδή αφορά καθαρά PL) τοποθετούμε το κατάλληλο jumper στην θέση JTAG και φορτώνουμε το αρχείο bitstream από το VivadoDesignTool (από το μενού Hardware Manager).
2. Στην περίπτωση που χρησιμοποιήσουμε τον επεξεργαστή τοποθετούμε το jumper στη θέση SD και γράφουμε κώδικα σε Python (διεπαφή LinuxSSH / JupyterNotebook). Εάν θέλουμε να χρησιμοποιήσουμε κάποιο σχέδιο, περνάμε τα αρχεία bitsream και block design στο σύστημα και τα φορτώνουμε σαν Overlay μέσα από την βιβλιοθήκη της Python.

4 Γενικές εφαρμογές

Σκοπός αυτού του κεφαλαίου είναι η διαδικασία υλοποίησης των εφαρμογών ώστε να γίνει πλήρως κατανοητός ο τρόπος χρήσης του λογισμικού.

Πριν ξεκινήσουμε θα πρέπει να υπάρχει στο Vivado Design Tool επιλογή για την δημιουργία εφαρμογών με την πλακέτα Pynq - Z2. Για να γίνει αυτό, κατεβάζουμε τα απαραίτητα αρχεία [41] και τα τοποθετούμε στον φάκελο “board files” που βρίσκεται στην τοποθεσία εγκατάστασης του εργαλείου, στην περίπτωση μας είναι η παρακάτω:

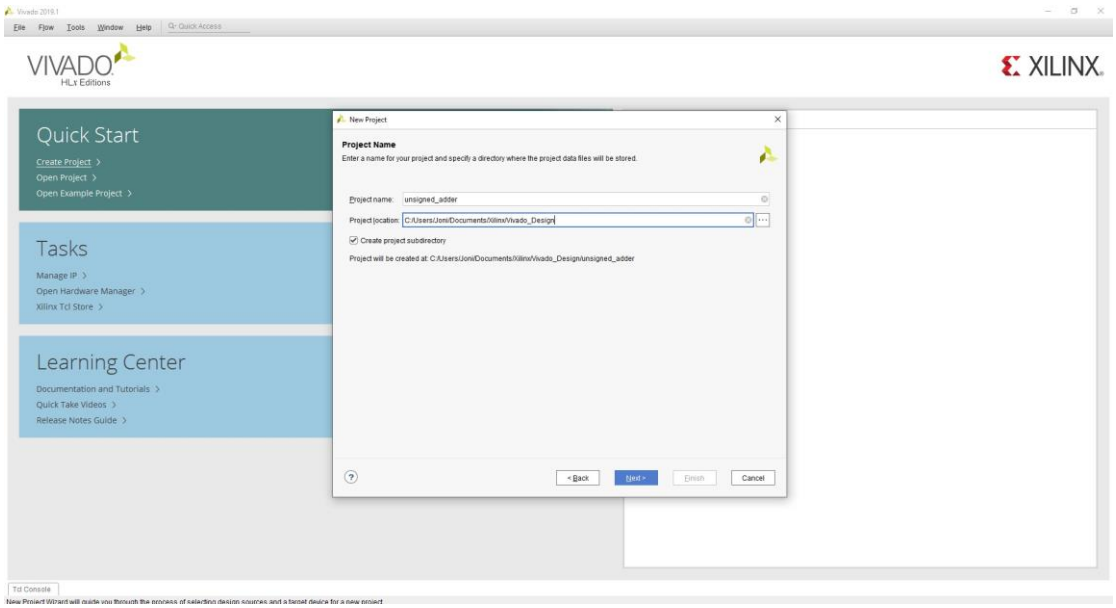
C:\Xilinx\Vivado\2019.1\data\boards\board_files\pynq-z2

4.1 Εφαρμογή μη προσημασμένου αθροιστή (PL)

Σκοπός αυτής της εφαρμογής είναι η υλοποίηση ενός μη προσημασμένου αθροιστή αποκλειστικά στο λογικό υλικό και δοκιμή της λειτουργίας του με τα μπουτόν και τους διακόπτες που έχει η πλακέτα. Ο αθροιστής μας είναι των 4 bit δηλαδή στην έξοδο του μπορεί να εμφανίσει αριθμό των 4 bit. Οι είσοδοι μας είναι των 3 bit ώστε σε περίπτωση υπερχείλισης να χωρέσει το αποτέλεσμα στην έξοδο του συστήματος. Ο πρώτος αριθμός εισάγεται από τα 3 μπουτόν ενώ ο δεύτερος από το τέταρτο μπουτόν και τους 2 διακόπτες.

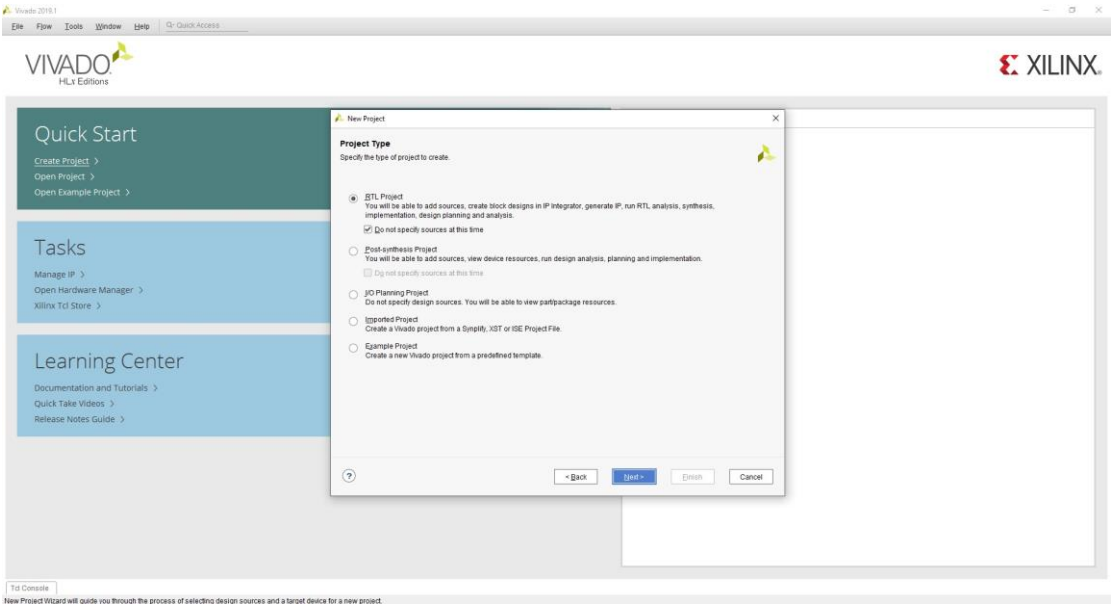
4.1.1 Βήματα για την δημιουργία

Ανοίγουμε το Vivado IDE και δημιουργούμε ένα νέο project (Εικόνα 4.1) δίνοντας την επιθυμητή ονομασία και την θέση στην οποία θα δημιουργηθεί.



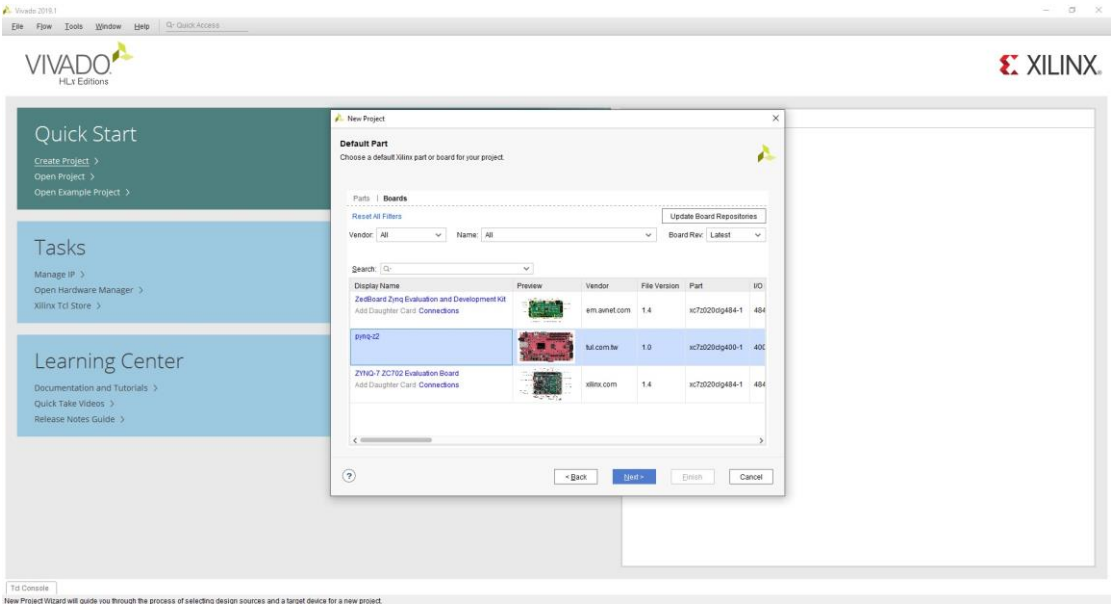
Εικόνα 4.1: Δημιουργία νέου project μη προσημασμένου αθροιστή στο Vivado IDE

Επιλέγουμε τον τύπο του project (Εικόνα 4.2), στην περίπτωση μας RTL project και χωρίς να καθορίσουμε πηγαία αρχεία.



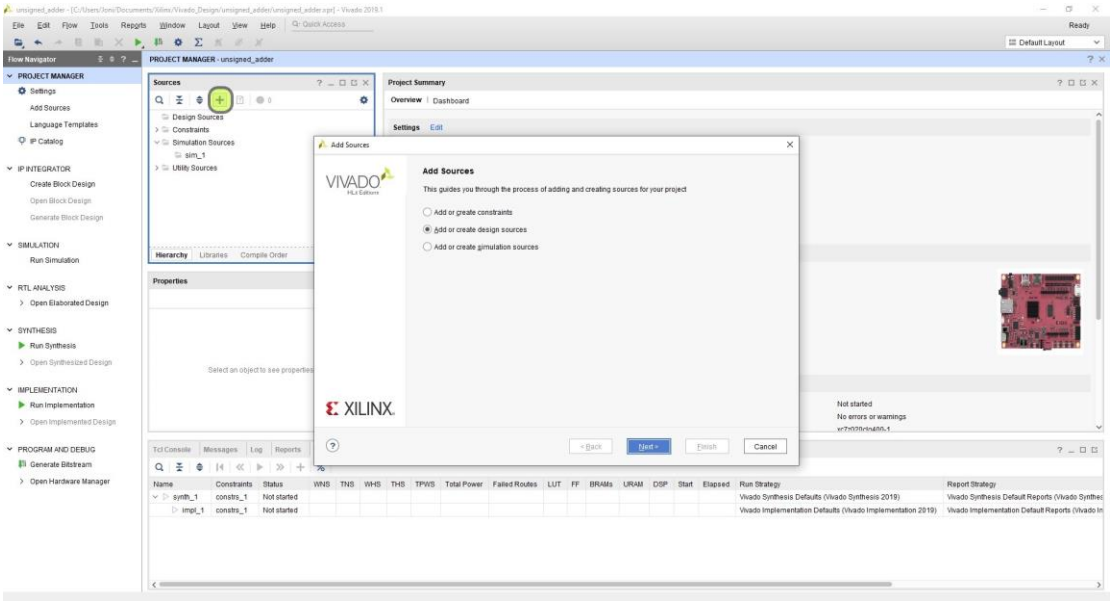
Εικόνα 4.2: Επιλογή τύπου του project για τον μη προσημασμένο αθροιστή

Στη συνέχεια επιλέγουμε την πλακέτα μας (Εικόνα 4.3).

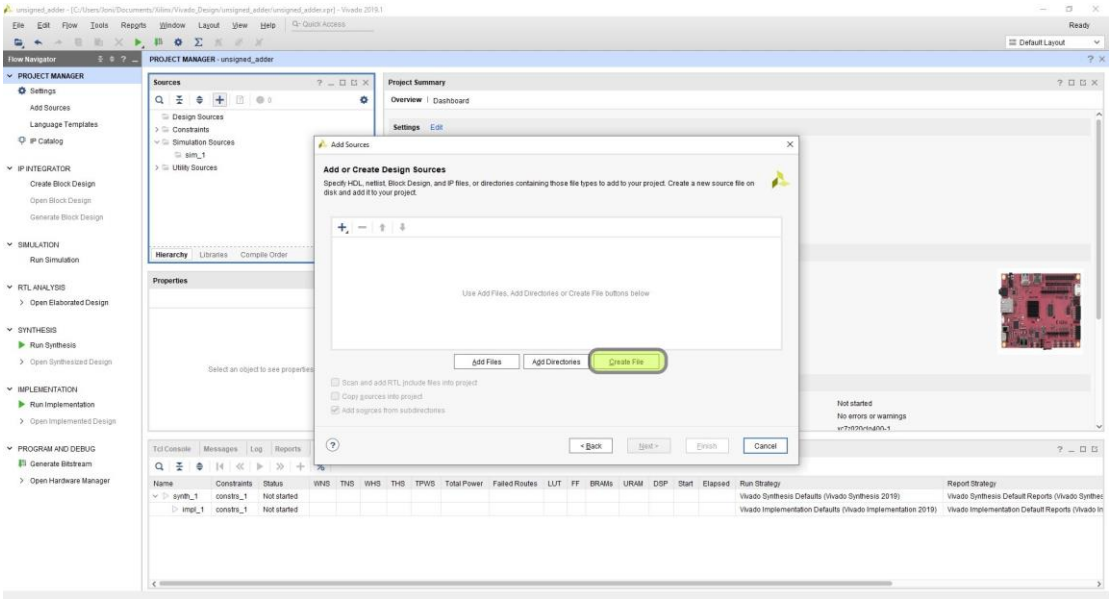


Εικόνα 4.3: Επιλογή της πλακέτας

Αφού ολοκληρωθεί η δημιουργία του και ανοίξει το κεντρικό παράθυρο, προσθέτουμε το αρχείο σχεδίου (Add or create design sources) (Εικόνα 4.4). Για να γίνει αυτό, επιλέγουμε την δημιουργία ενός νέου πηγαίου αρχείου σε γλώσσα VHDL (Εικόνα 4.5).



Εικόνα 4.4: Επιλογή προσθήκης πηγαίου αρχείου σχεδίασης



Εικόνα 4.5: Δημιουργία πηγαίου αρχείου σχεδίασης

Ανοίγοντας το αρχείο γράφουμε τον κώδικα για την περιγραφή του μη προσημασμένου αθροιστή μας.

Κώδικας για την περιγραφή του μη προσημασμένου αθροιστή:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity unsigned_adder  is
-- εισαγωγή των εισόδων και εξόδων
Port ( buttons : in STD_LOGIC_VECTOR (3 downto 0);
   switches : in STD_LOGIC_VECTOR (1 downto 0);
   leds : out STD_LOGIC_VECTOR (3 downto 0));
end unsigned_adder;
architecture Behavioral of unsigned_adder  is
-- εισαγωγή βοηθητικών σημάτων
signal a,b : STD_LOGIC_VECTOR(2 downto 0);
signal a_u,b_u : UNSIGNED(2 downto 0);
signal sum_u : UNSIGNED(3 downto 0);
begin
-- βάζω στο 'a' τα 3 πρώτα μπουτόν απο τα δεξιά
a <= buttons(2 downto 0);
-- σαν πρώτο μπουτόν απο το 'b' βάζω το τέταρτο μπουτόν,
-- ενώ για τα υπόλοιπα bit του 'b' βάζω τους 2 διακόπτες
b(0) <= buttons(3);
b(1) <= switches(0);
b(2) <= switches(1);
```

-- μετατρέπω τα σήματα που εισάγω απο τα μπουτόν και τους διακόπτες,

-- απο την απλή λογική σε μη προσημασμένη λογική

a_u <= UNSIGNED(a);

b_u <= UNSIGNED(b);

-- επειδή η έξοδος μου είναι σήμα 4 bit ενώ οι αριθμοί που εισάγω είναι σήματα 3 bit,

-- χρησιμοποιώ τον συνδετικό τελεστή για να ενώσω στα σήματα μου το '0' ώστε να φτιάξω σήματα των 4 bit

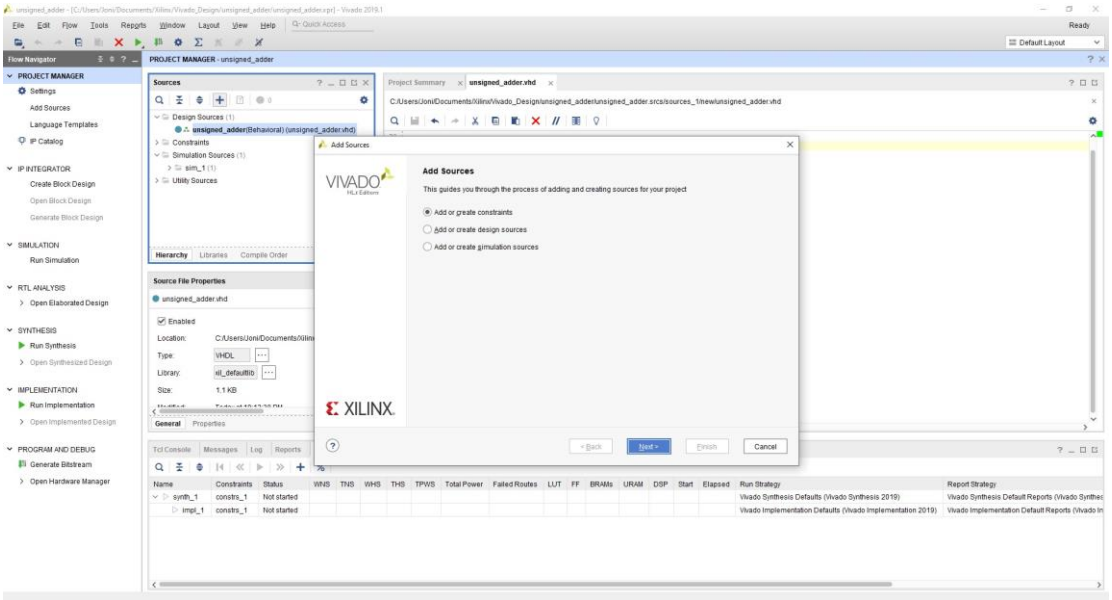
sum_u<=('0' & a_u) + ('0' & b_u) ;

-- βάζω στην έξοδο το αποτέλεσμα της πρόσθεσης αφού το μετατρέψω απο μη προσημασμένη λογική σε απλή λογική

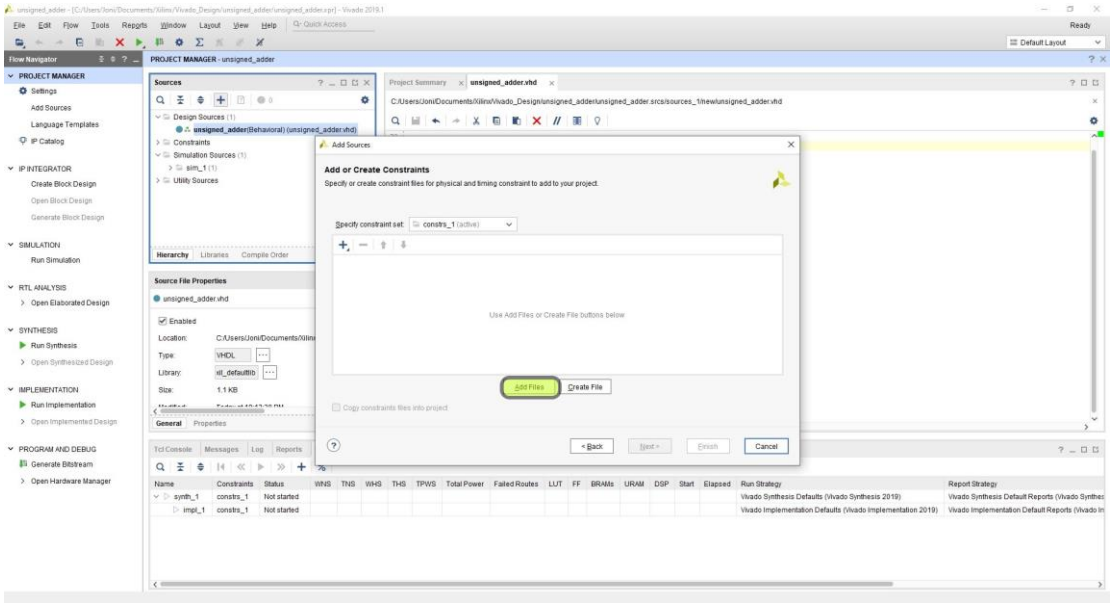
leds <= std_logic_vector(sum_u(3 downto 0));

end Behavioral;

Επόμενο βήμα είναι η προσθήκη του αρχείου με τους φυσικούς περιορισμούς της πλακέτας (Add or create constraints) (Εικόνα 4.6). Σε αυτήν την περίπτωση δεν δημιουργούμε κάποιο αρχείο αλλά προσθέτουμε το αρχείο περιορισμών (Εικόνα 4.7) αφού πρώτα το έχουμε κατεβάσει από την σελίδα του κατασκευαστή (TUL) [41].

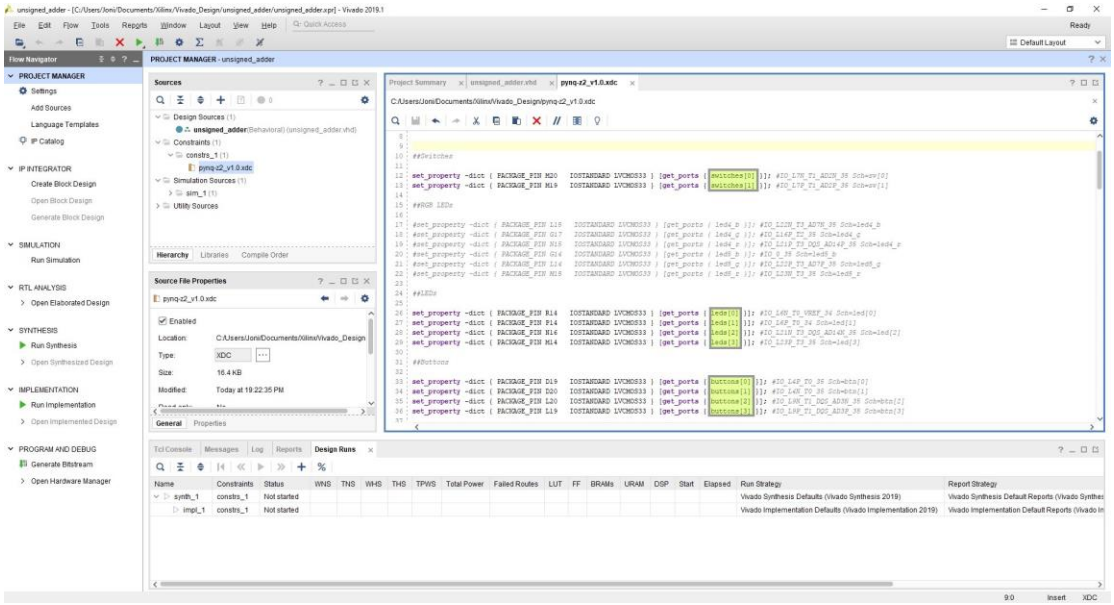


Εικόνα 4.6: Επιλογή προσθήκης αρχείου περιορισμών σχεδίασης



Εικόνα 4.7: Προσθήκη αρχείου περιορισμών σχεδίασης

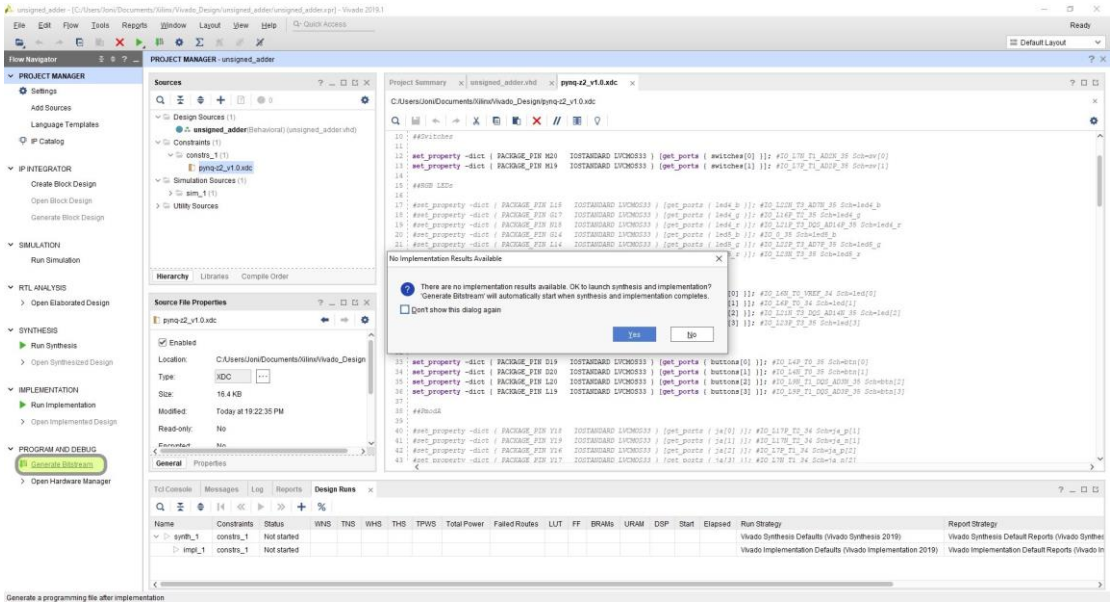
Ανοίγοντας το, παρατηρούμε να υπάρχει σε κάθε γραμμή το σύμβολο της δίεσης (#) υποδηλώνοντας την ύπαρξη σχολίου. Σκοπός μας είναι να αφαιρέσουμε την δίεση από κάθε γραμμή που θέλουμε να χρησιμοποιήσουμε (Εικόνα 4.8) και να υπάρχει η ίδια ονομασία της θύρας με αυτήν στο αρχείο σχεδίου που φτιάξαμε σε γλώσσα VHDL.



Εικόνα 4.8: Τροποποίηση του αρχείου περιορισμών σχεδίασης

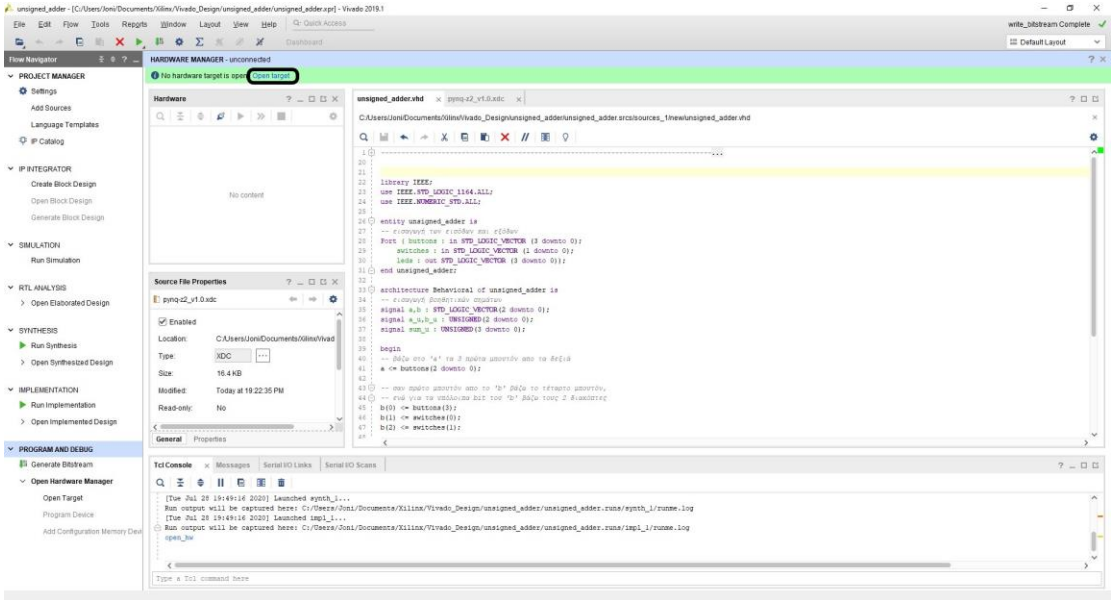
Στη συνέχεια, ξεκινάμε την διαδικασία της παραγωγής του αρχείου bitstream (Εικόνα 4.9) από το μενού PROJECT MANAGER > PROGRAM AND DEBUG > Generate Bitstream, που περιέχει

όλους τους πόρους για την τοποθέτηση της εφαρμογής μας στο λογικό υλικό. Αρχίζει με την λειτουργία της σύνθεσης, συνεχίζει με την λειτουργίας της τοποθέτησης και τέλος παράγεται το αρχείο bitstream.

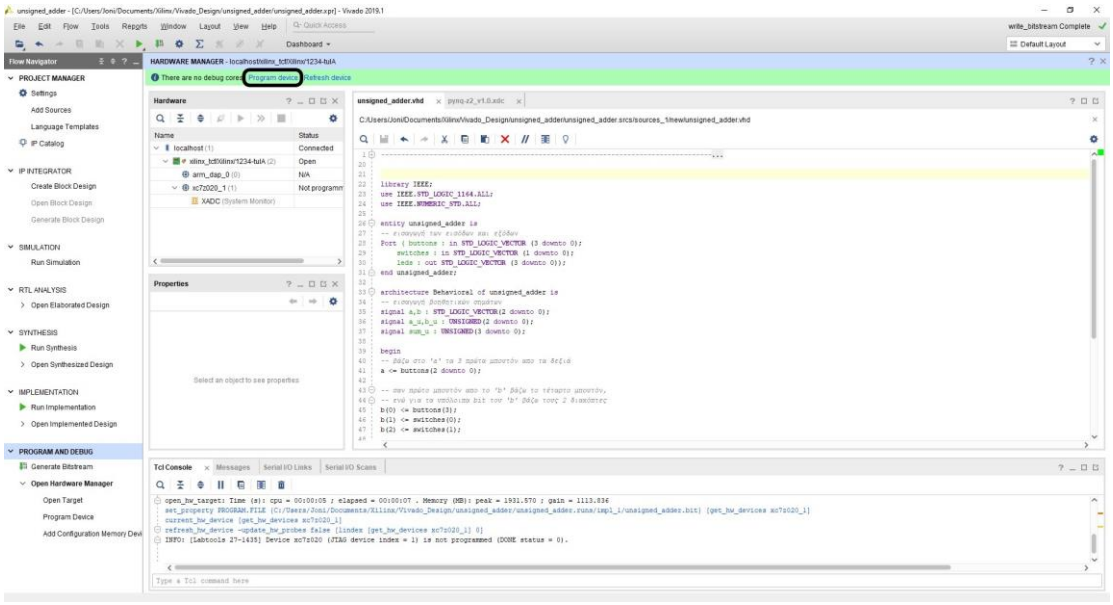


Εικόνα 4.9: Παραγωγή αρχείου bitstream

Για να μεταφέρουμε το αρχείο στην πλακέτα, ανοίγουμε το Hardware Manager (μενού PROGRAM AND DEBUG > Open Hardware Manager), συνδέουμε την συσκευή μας (Open Target > Auto Connect) (Εικόνα 4.10) και επιλέγουμε τον προγραμματισμό της (Program device) (Εικόνα 4.11) ανοίγοντας το αρχείο bitstream που μόλις δημιουργήθηκε.



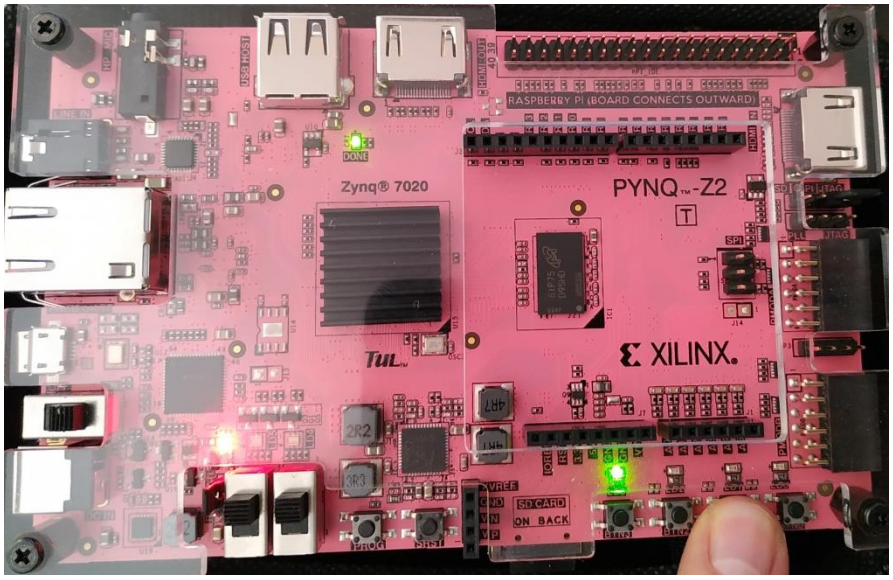
Εικόνα 4.10: Άνοιγμα της συσκευής μας από το Hardware Manager



Εικόνα 4.11: Προγραμματισμός της συσκευής με το αρχείο bitstream που δημιουργήθηκε

Αφού ολοκληρωθεί ο προγραμματισμός της συσκευής μας, είμαστε έτοιμοι για την αλληλεπίδραση με αυτήν.

Σε αυτήν την περίπτωση βλέπουμε το αποτέλεσμα της πρόσθεσης μεταξύ του 2 και του 6 και το αποτέλεσμα φαίνεται στο led (Εικόνα 4.12).



Εικόνα 4.12: Αλληλεπίδραση με την πλακέτα και απεικόνιση του αποτελέσματος της άθροισης 2 αριθμών με τη βοήθεια των LEDs

4.2 Εφαρμογή αριθμομηχανής (PS)

Σκοπός είναι να φτιάξουμε την λειτουργία μιας αριθμομηχανής αποκλειστικά στον επεξεργαστή όπως θα γινόταν και σε έναν υπολογιστή. Χρησιμοποιούμε τον nano editor για την δημιουργία του κώδικα.

4.2.1 Ο κώδικας σε Python

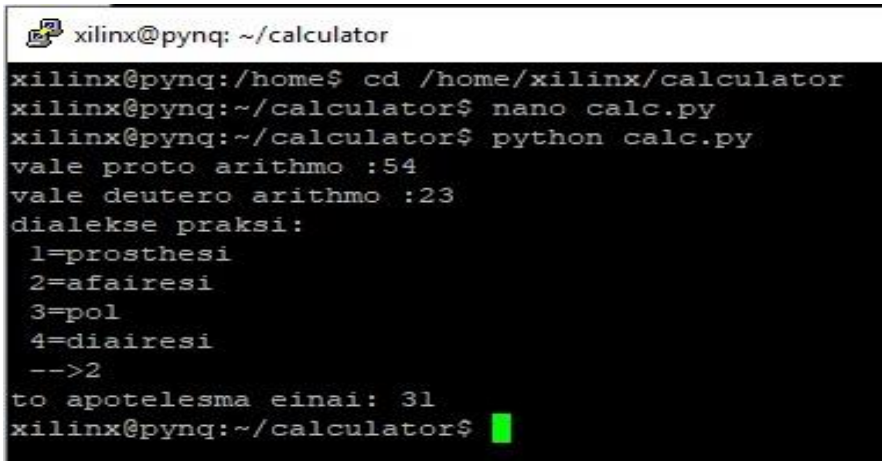
Κώδικας υλοποίησης αριθμομηχανής:

```
number1 = input("vale proto arithmo :")

number2 = input("vale deuteroarithmo :)"

praksi = input("dialekse praksi: \n 1=prosthesi \n 2=afairesi \n 3=pol \n 4=diairesi \n -->")

if praksi==1:

print "to apotelesma einai:", number1 + number2

elif praksi==2:

print "to apotelesma einai:", number1 - number2

elif praksi==3:

print "to apotelesma einai:", number1 * number2

elif praksi==4:

print "to apotelesma einai:", number1 / number2

else:

print "ekanes kati lathos"
```

Εδώ, ανοίγουμε το nano editor και τρέχουμε τον κώδικα Python που ανέπτυξα.

Στην ουσία ο κώδικας τρέχει πάνω στον ARM που έχει το Zynq όπως θα έτρεχε και στον επεξεργαστή ενός υπολογιστή.

Εισάγουμε τους αριθμούς 54 και 23 και επιλέγουμε την πράξη της αφαίρεσης και εμφανίζεται σωστά το αποτέλεσμα 31 (Εικόνα 4.13).



Εικόνα 4.13:Εμφάνιση του αποτελέσματος της εφαρμογής της αριθμομηχανής μέσα από το nano editor

4.3 Εφαρμογή αριθμητικής και λογικής μονάδας (PL – PS)

Στην συγκεκριμένη εφαρμογή χρησιμοποιούμε την προγραμματιζόμενη λογική σε συνεργασία με τον επεξεργαστή για να επιταχύνουμε μία αριθμητική και λογική μονάδα.

Η αριθμητική και λογική μονάδα που φτιάχνουμε μπορεί να εκτελέσει 4 πράξεις:

- Πρόσθεση
- Αφαίρεση
- Λογικό OR
- Λογικό AND

Αποτελείται από 4 καταχωρητές των 32 bit:

- Ο πρώτος καταχωρητής χρησιμοποιείται για την εισαγωγή του πρώτου αριθμού
- Ο επόμενος καταχωρητής χρησιμοποιείται για την εισαγωγή του δεύτερου αριθμού
- Από τον τρίτο καταχωρητή χρησιμοποιούμε μόνο τα 2 bit για την επιλογή της πράξης, αφού έχουμε σύνολο 4 πράξεις
- Ο τελευταίος καταχωρητής χρησιμοποιείται για την εισαγωγή του αποτελέσματος

Η διαδικασία ξεκινάει με την δημιουργία του αρχικού project μέσα από το οποίο φτιάχνουμε ένα νέο project για την δημιουργία του δικού μας IP που υλοποιεί την αριθμητική και λογική μονάδα. Για να

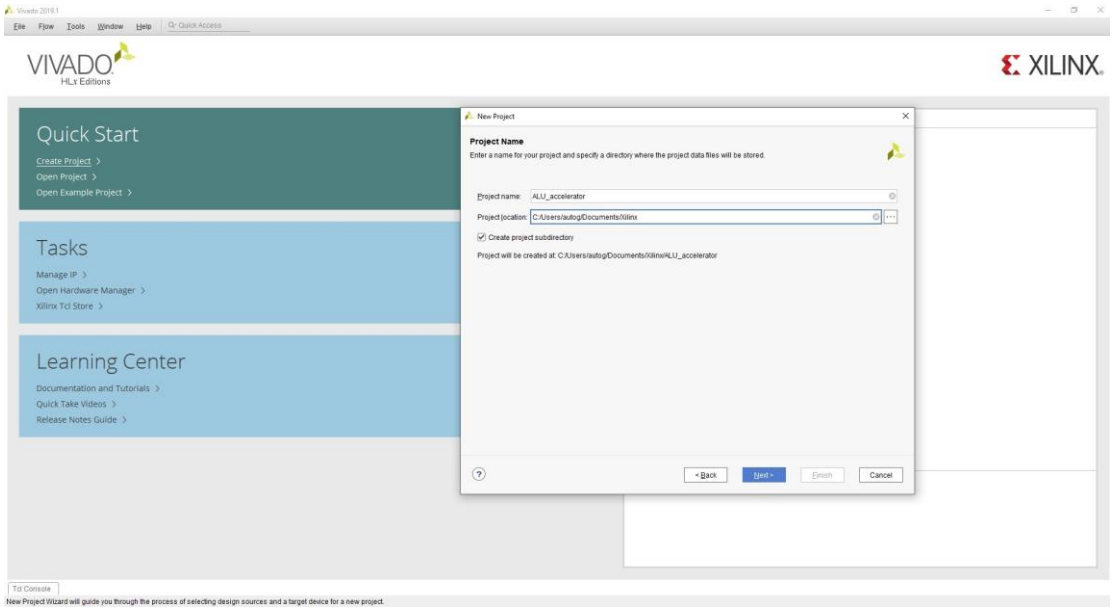
δημιουργήσουμε το IP προσθέτουμε τον κώδικα περιγραφής υλικού σε γώσσα VHDL και κάνουμε κάποιες τροποποιήσεις και προσθήκες σε αρχείο που δημιουργείται από το Vivado.

Μετά την ολοκλήρωση της δημιουργίας του IP, επιστρέφουμε στο αρχικό project και υλοποιούμε την σχεδίαση προσθέτοντας το IP μαζί με άλλα IP blocks.

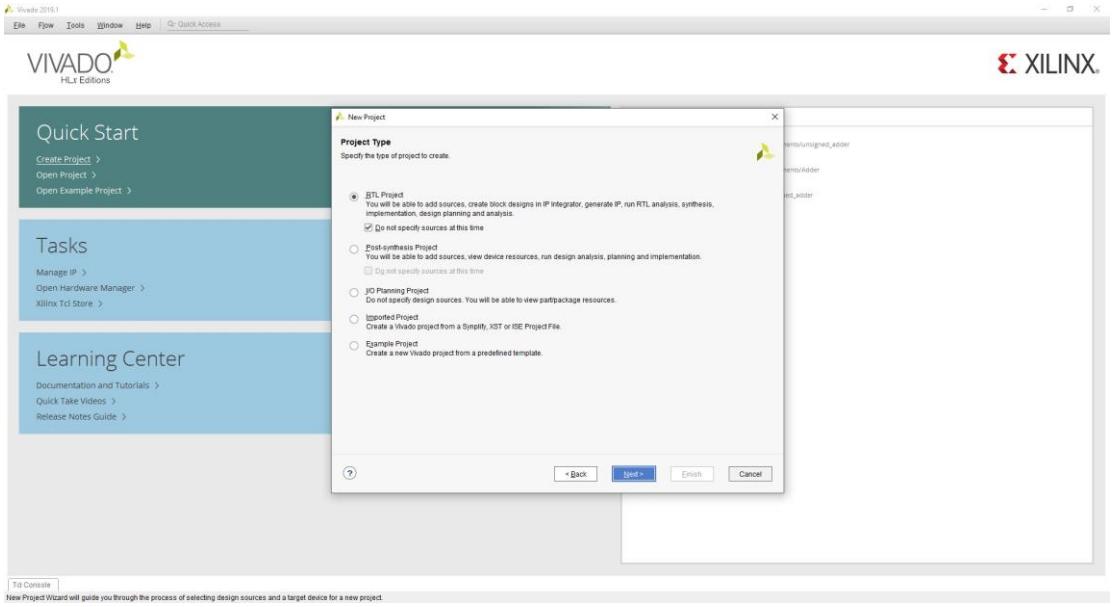
Η ολοκλήρωση της εφαρμογής επιτυγχάνεται με τον έλεγχο της από τον επεξεργαστή μέσω της διεπαφής Python χρησιμοποιώντας το Jupyter Notebook.

4.3.1 Βήματα για την δημιουργία

Ανοίγουμε το Vivado IDE και δημιουργούμε ένα νέο project επιλέγοντας τον τύπο του project,στην περίπτωση μας RTL project και χωρίς να καθορίσουμε πηγαία αρχεία (Εικόνα 4.14, 4.15).

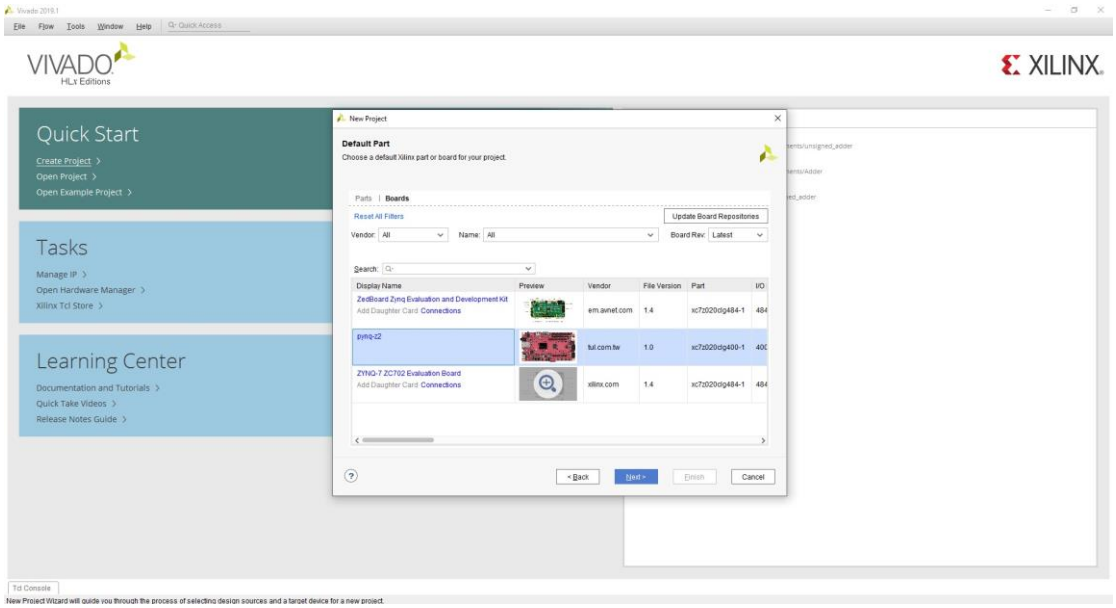


Εικόνα 4.14: Δημιουργία νέου project αριθμητικής και λογικής μονάδας στο Vivado IDE



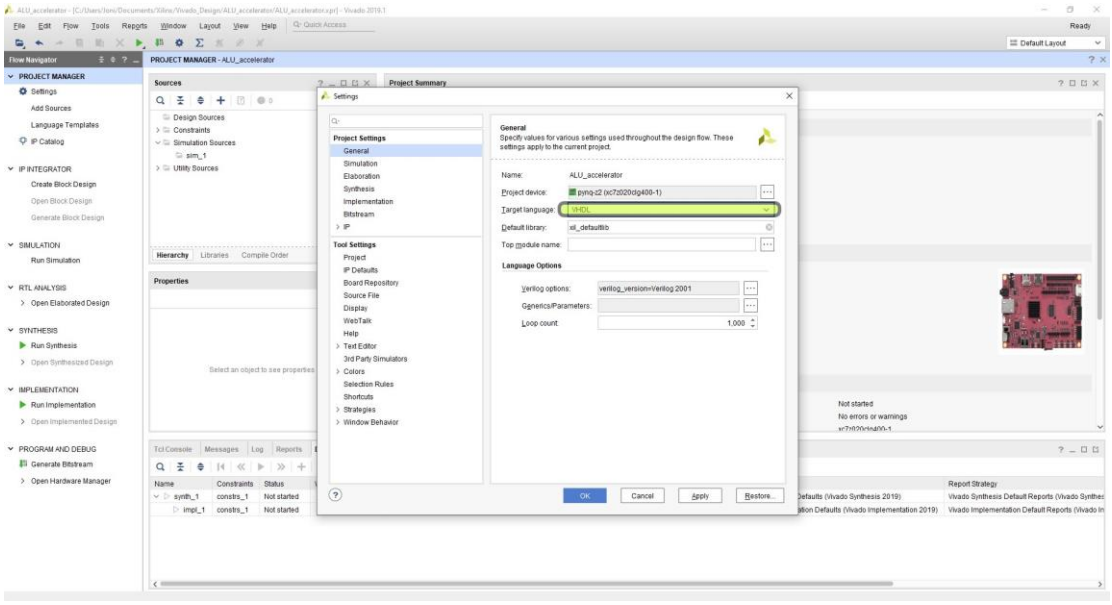
Εικόνα 4.15: Επιλογή τύπου του project για την αρηθμητική και λογική μονάδα

Επιλέγουμε την πλακέτα μας (Εικόνα 4.16).



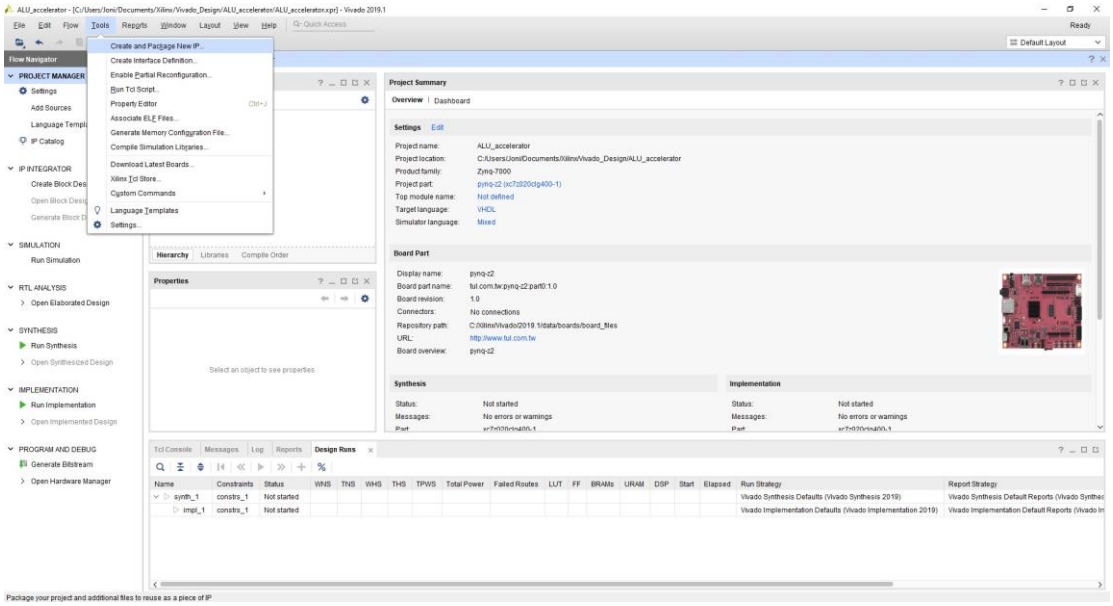
Εικόνα 4.16: Επιλογή της πλακέτας

Αφού ολοκληρωθεί η δημιουργία του και ανοίξει το κεντρικό παράθυρο, αλλάζουμε από τις ρυθμίσεις την γλώσσα σε VHDL (μενού PROJECT MANAGER > Project Settings > General > Target Language) (Εικόνα 4.17).



Εικόνα 4.17: Αλλαγή της γλώσσας περιγραφής υλικού για το project

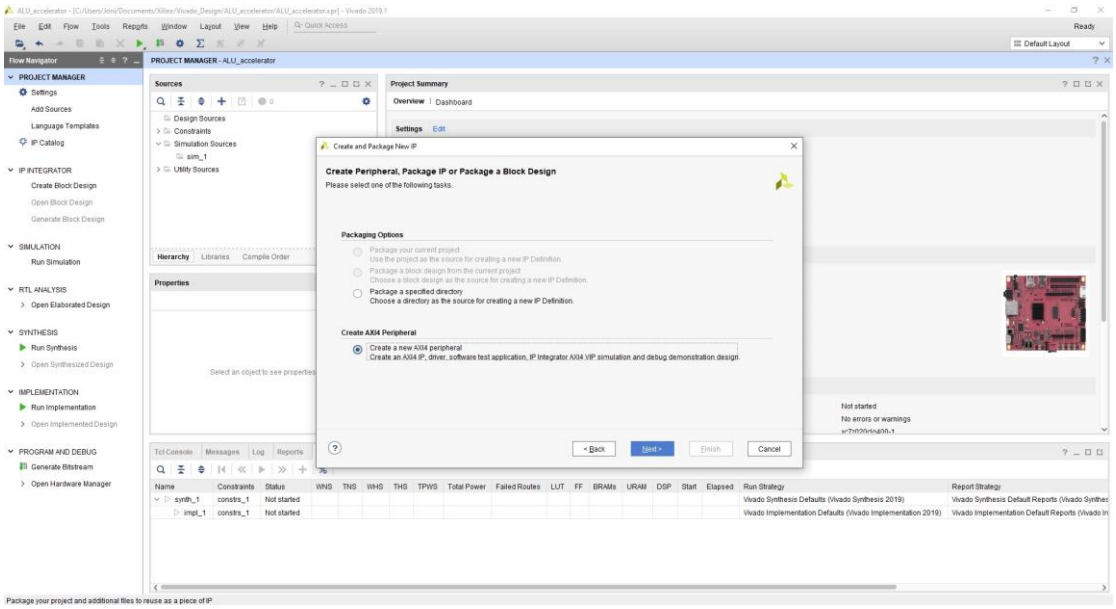
Ύστερα, επιλέγουμε την δημιουργία νέου IP block (μενού Tools > Create and Package New IP) (Εικόνα 4.18).



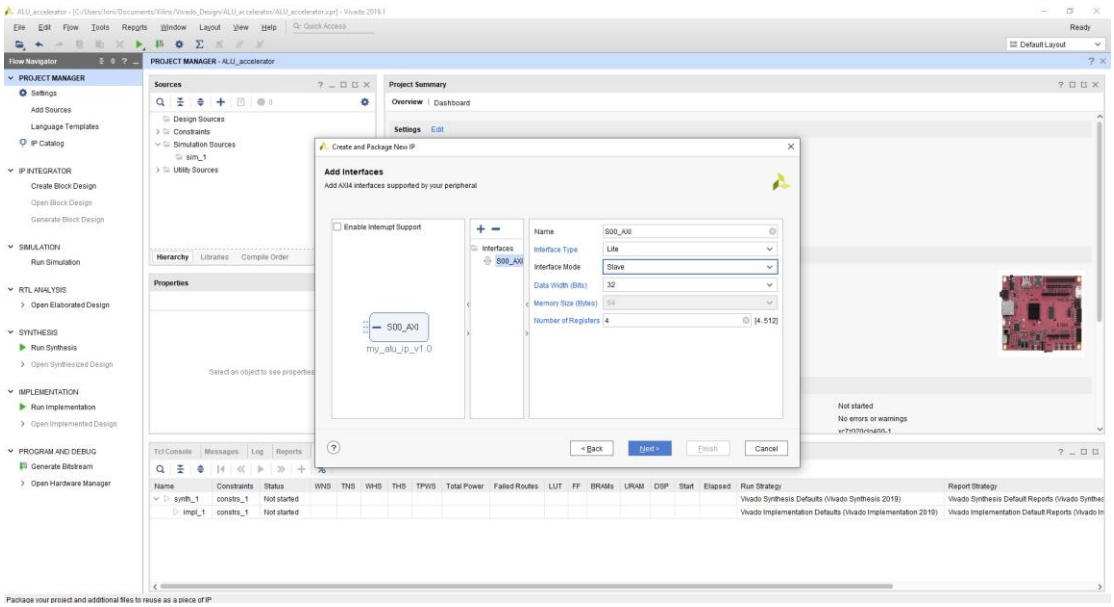
Εικόνα 4.18: Επιλογή δημιουργίας IP

Δημιουργούμε ένα AXI4 περιφερειακό (Εικόνα 4.19) με τις default παραμέτρους (Εικόνα 4.20):

- τύπο διεπαφής: **Lite**
- λειτουργία διεπαφής: **Slave**
- μέγεθος δεδομένων: **32bits**
- μέγεθος μνήμης: **64bytes**
- αριθμό καταχωρητών: **4**

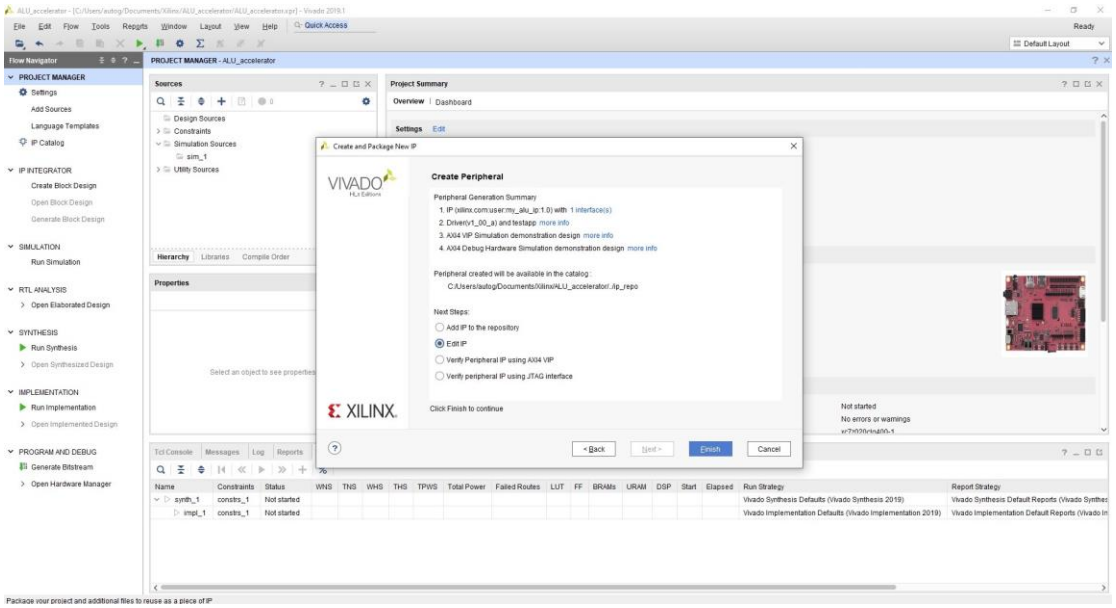


Εικόνα 4.19: Δημιουργία νέου AXI4 IP

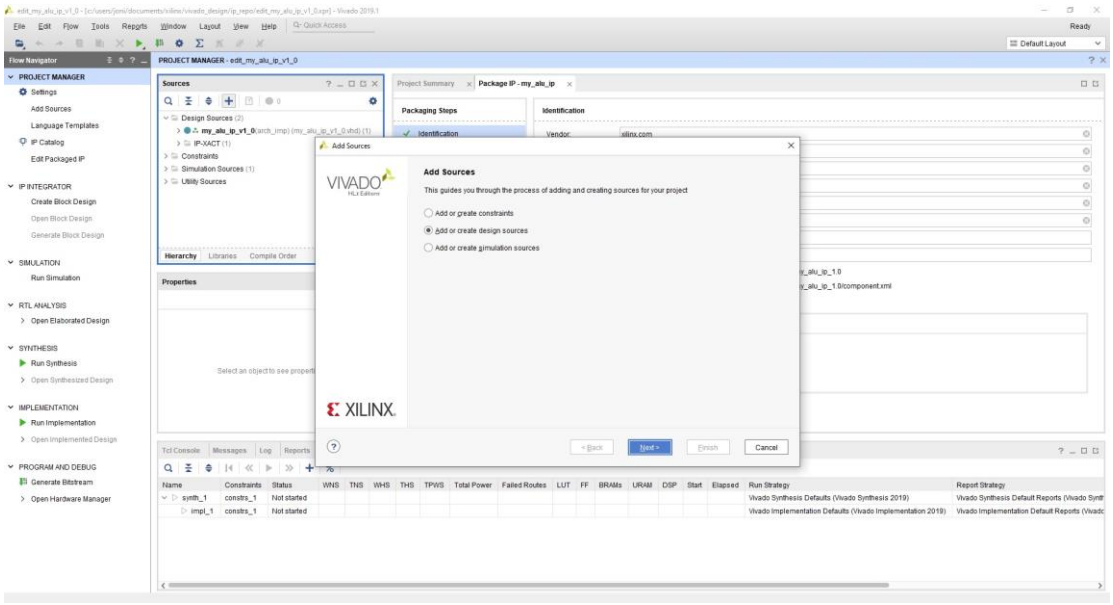


Εικόνα 4.20: Παράμετροι του νέου IP

Στη συνέχεια επιλέγουμε την επεξεργασία του IP (Edit IP) (Εικόνα 4.21) και ανοίγει νέο παράθυρο από όπου προσθέτουμε το αρχείο σχεδίου (Add or create design sources) (Εικόνα 4.22).

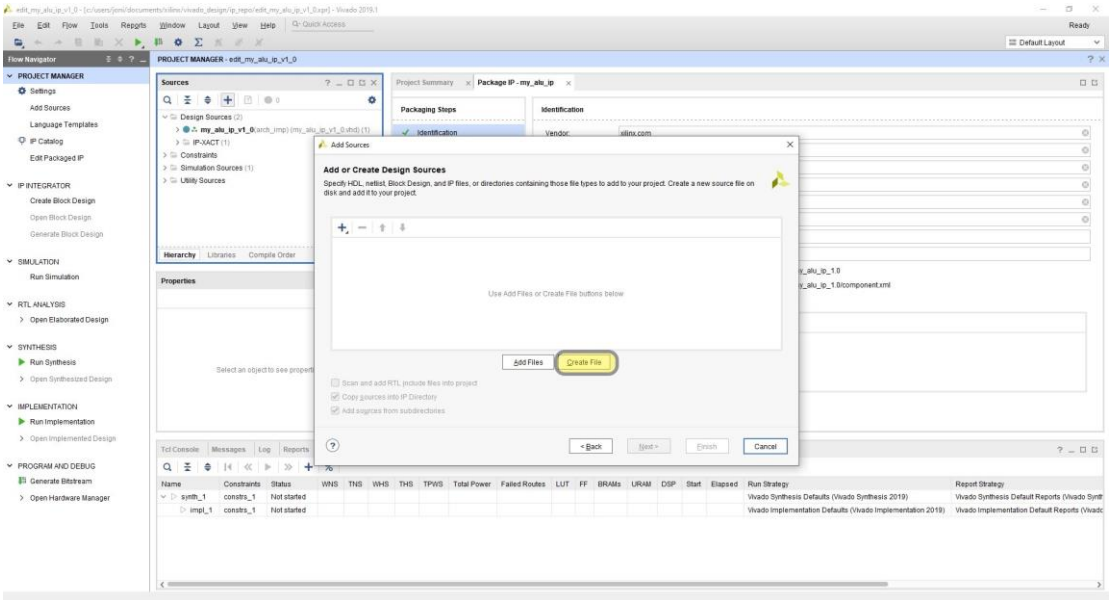


Εικόνα 4.21: Επιλογή Edit IP για την ολοκλήρωση της δημιουργίας



Εικόνα 4.22: Επιλογή προσθήκης πηγαίου αρχείου σχεδίασης

Δημιουργούμε ένα νέο πηγαίο αρχείο σε γλώσσα VHDL (Εικόνα 4.23) και το ανοίγουμε για να γράψουμε τον κώδικα για την περιγραφή της αριθμητικής και λογικής μονάδας.



Εικόνα 4.23: Δημιουργία πηγαίου αρχείου σχεδίασης

Κώδικας για την περιγραφή της αριθμητικής και λογικής μονάδας:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity alu is

-- εισαγωγή γενικής μεταβλητής, εισόδων και εξόδων

GENERIC (N: INTEGER := 31);

Port ( X : in STD_LOGIC_VECTOR (N-1 downto 0);

      Y : in STD_LOGIC_VECTOR (N-1 downto 0);
```

```
      s: in STD_LOGIC_VECTOR (N downto 0);
      F : out STD_LOGIC_VECTOR (N downto 0));
end alu;
architecture Behavioral of alu is
-- εισαγωγή βοηθητικών σημάτων
SIGNAL X_S,Y_S: SIGNED(N-1 downto 0);
SIGNAL sum_s,sub_s: SIGNED(N downto 0);
SIGNAL logic_OR,logic_AND : STD_LOGIC_VECTOR (N downto 0);
begin
-- μετατροπή των σημάτων εισόδου από απλή λογική σε προσημασμένη λογική
X_S <= SIGNED(X);
Y_S <= SIGNED(Y);
-- επειδή η έξοδος μου είναι σήμα 32 bit ενώ οι αριθμοί που εισάγω είναι σήματα των 31 bit,
-- χρησιμοποιώ τον συνδετικό τελεστή για να ενώσω στα σήματα μου το msb που δηλώνει το
-- πρόσημο και για να φτιάξω σήματα των 32 bit για τις πράξεις της πρόσθεσης και αφαίρεσης
sum_s <= (X_S(N-1) & X_S) + (Y_S(N-1) & Y_S);
sub_s <= (X_S(N-1) & X_S) - (Y_S(N-1) & Y_S);
-- το ίδιο κάνω και για τις λογικές πράξεις, μόνο που εδώ ενώνω το ‘0’ για να φτιάξω σήματα των 32
-- bit
logic_OR <= ('0' & X) OR ('0' & Y);
logic_AND <= ('0' & X) AND ('0' & Y);
-- χρησιμοποιώ 2 bit από το σήμα επιλογής αφού έχω 4 πράξεις
with s(1 downto 0) select
   F <= STD_LOGIC_VECTOR(sum_s) when "00",
```

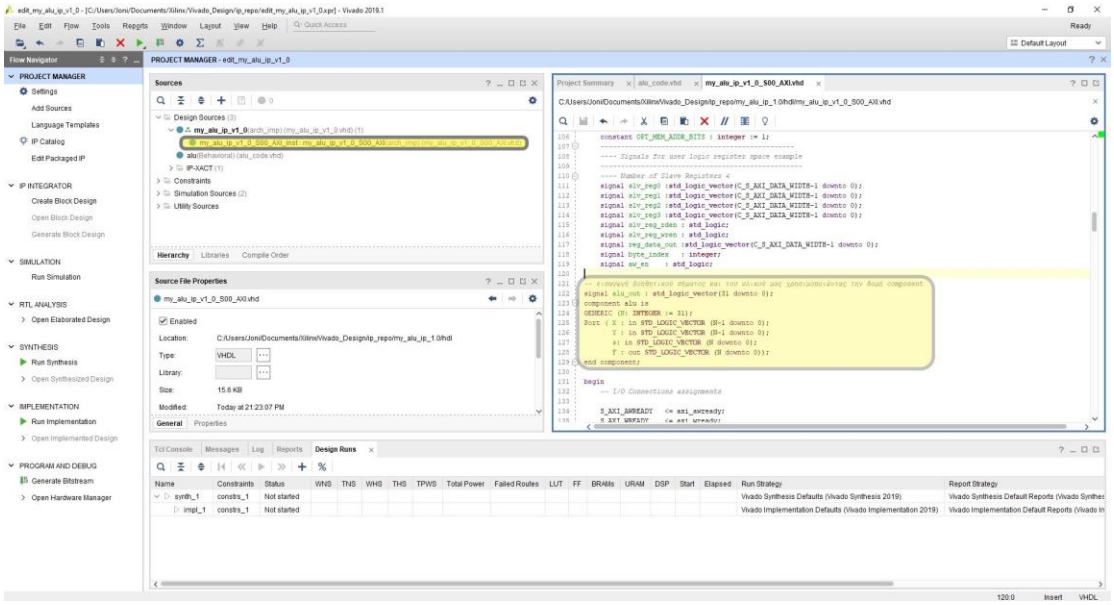
```
    STD_LOGIC_VECTOR(sub_s) when "01",
    logic_OR when "10",
    logic_AND when "11";
end Behavioral;
```

Επόμενο βήμα είναι να ανοίξουμε το στιγμιότυπο του IP block που έχει δημιουργηθεί σύμφωνα με την ονομασία που δώσαμε (στην εφαρμογή μας είναι το my_alu_ip_v1_0_S00_AXI_inst : my_alu_ip_v1_0_S00_AXI) και βρίσκεται στην καρτέλα Sources > Design Sources (Εικόνα 4.24).

Βρίσκουμε την λέξη ‘begin’(γραμμή 131) (Εικόνα 4.24) και προσθέτουμε ένα βοηθητικό σήμα που αντιπροσωπεύει την έξοδο της αριθμητικής και λογικής μονάδας (alu_out). Επίσης, δηλώνουμε το εξάρτημα (component) μα με την ίδια ονομασία που δώσαμε στην οντότητα (entity) στο πηγαίο αρχείο σε γλώσσα VHDL (alu).

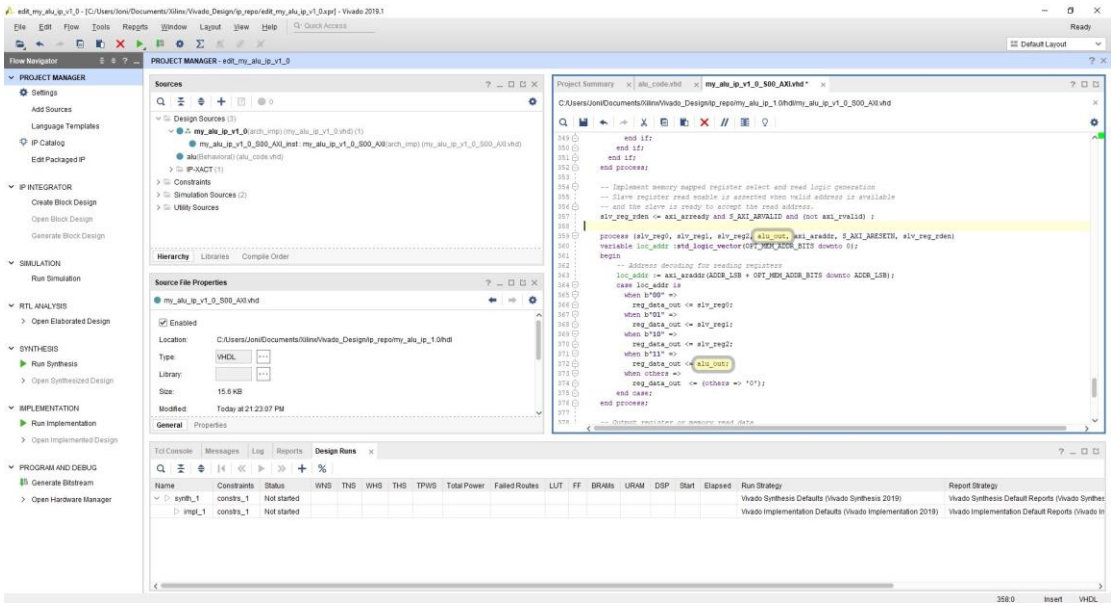
Κώδικας για την εισαγωγή του εξαρτήματος (component):

```
signal alu_out : std_logic_vector(31 downto 0);
component alu is
GENERIC (N: INTEGER := 31);
Port ( X : in STD_LOGIC_VECTOR (N-1 downto 0);
    Y : in STD_LOGIC_VECTOR (N-1 downto 0);
    s: in STD_LOGIC_VECTOR (N downto 0);
    F : out STD_LOGIC_VECTOR (N downto 0));
end component;
```



Εικόνα 4.24: Προσθήκη του εξαρτήματος στο στιγμιότυπο που δημιουργήθηκε

Βρίσκουμε την λέξη ‘process’ (γραμμή 359) και μέσα στην παρένθεση αντικαθιστούμε τον τέταρτο καταχωρητή slv_reg3 με το βοηθητικό σήμα που δηλώσαμε πιο πάνω (alu_out). Το ίδιο κάνουμε και στην συνάρτηση ‘case’, στην εντολή reg_data_out <= slv_reg3 (γραμμή 372) (Εικόνα 4.25).



Εικόνα 4.25: Αντικατάσταση του τέταρτου καταχωρητή στο στιγμιότυπο

Έπειτα, λίγο πριν το τέλος του κώδικα (γραμμή 396) προσθέτουμε το στιγμιότυπο (Εικόνα 4.26).

Κώδικας στιγμιότυπου:

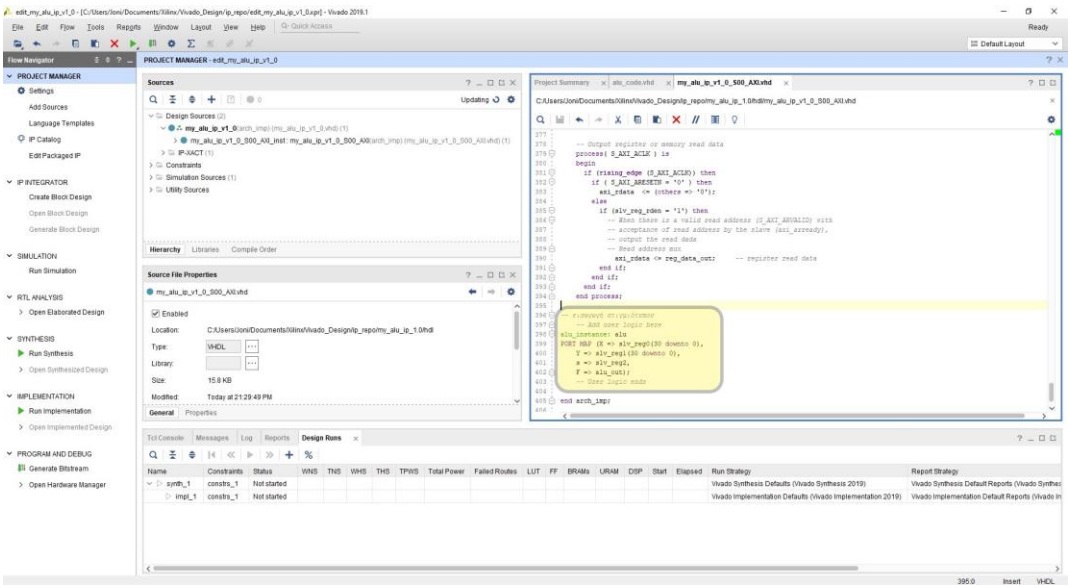
alu_instance: alu

PORT MAP (X => slv_reg0(30 downto 0),

Y => slv_reg1(30 downto 0),

s => slv_reg2,

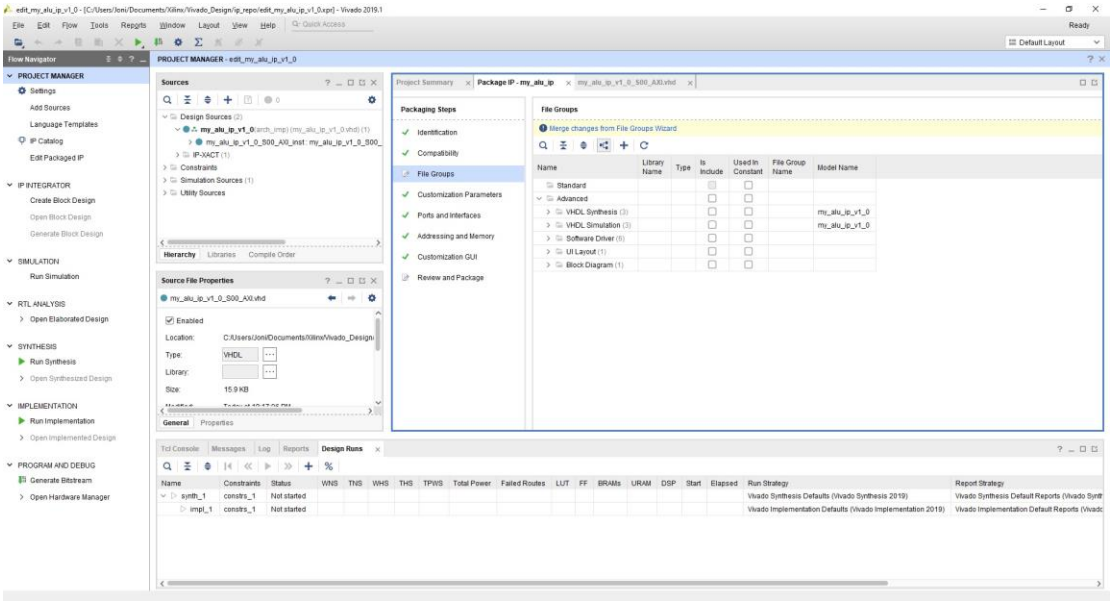
F => alu_out);



Εικόνα 4.26: Προσθήκη στιγμιότυπου

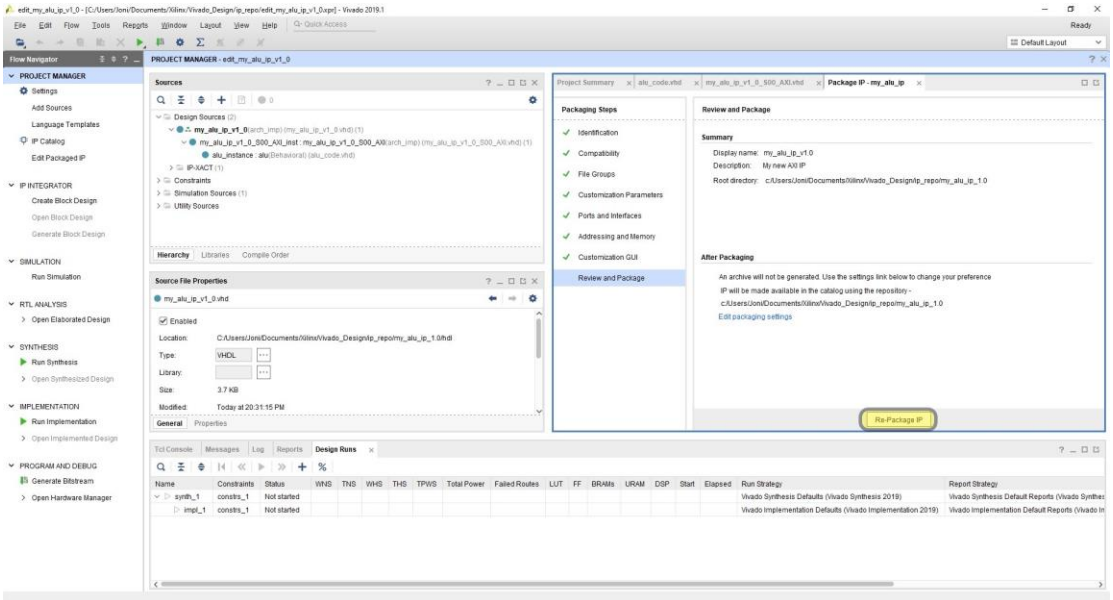
Χρειάζεται πολύ προσοχή διότι πρέπει το όνομα να είναι το ίδιο με αυτό που δηλώσαμε στην οντότητα (entity) στο πηγαίο αρχείο σε γλώσσα VHDL (alu).

Από την καρτέλα Package IP > Packaging Steps > File Groups, πατάμε στο ‘Merge changes from File Groups Wizard’ (Εικόνα 4.27).



Εικόνα 4.27: Επιλογή Merge changes from File Groups Wizard

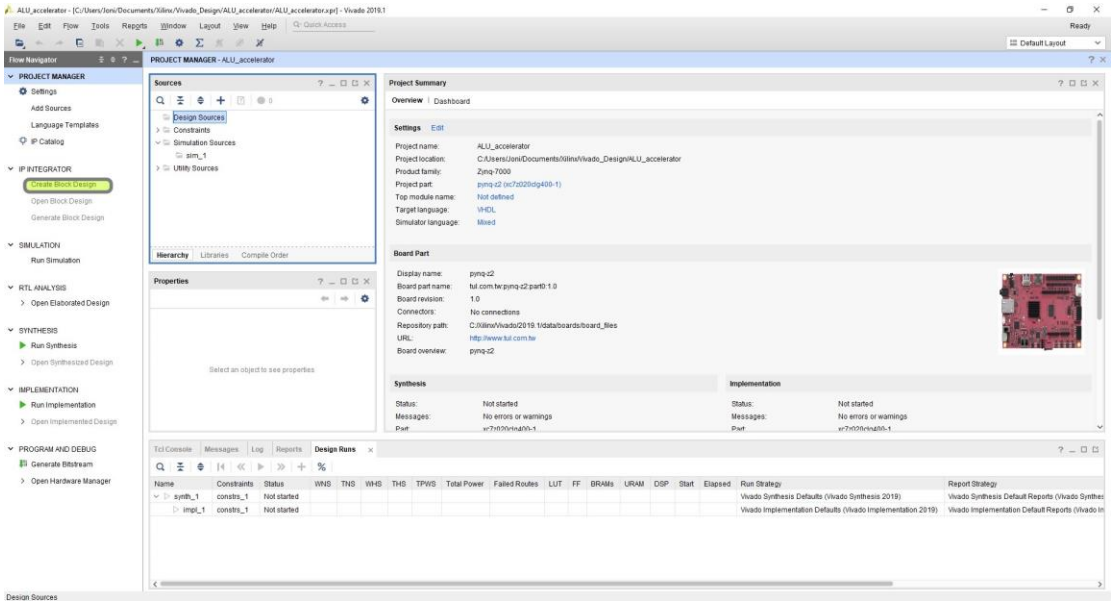
Για να ολοκληρωθεί η δημιουργία του IP, από το Packaging Steps > Review and Package πατάμε στο ‘Re-Package IP’ (Εικόνα 4.28).



Εικόνα 4.28: Επιλογή Re-Package IP’

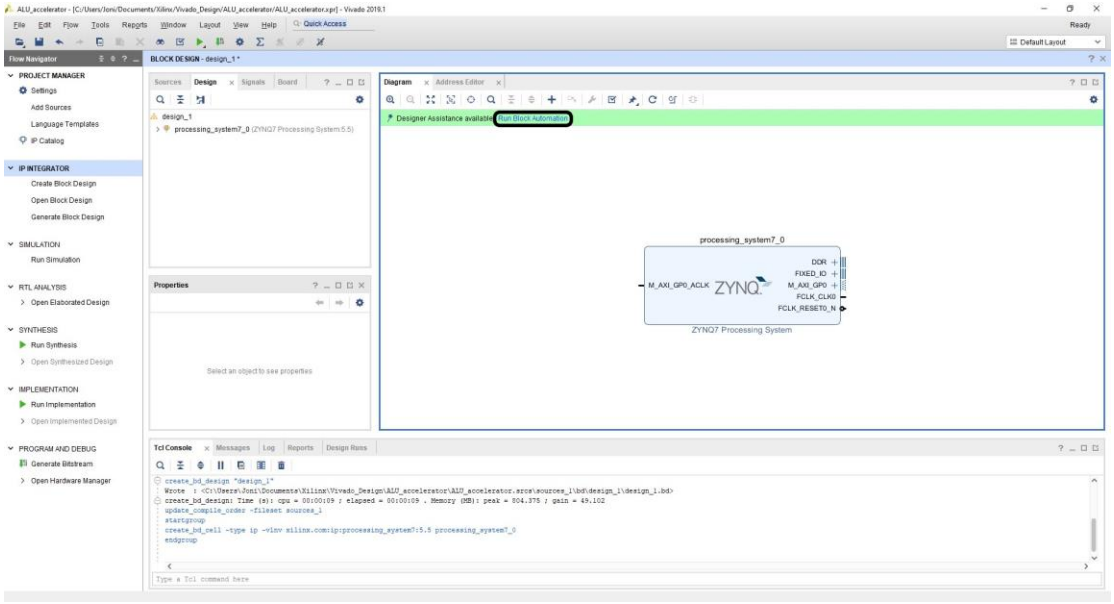
Το IP block έχει προστεθεί τώρα στον κατάλογο του Vivado IDE.

Επιστρέφουμε στο κυρίως project και φτιάχνουμε την επιφάνεια του σχεδίου μας από το μενού PROJECT MANAGER > IP INTEGRATOR > Create Block Design (Εικόνα 4.29).

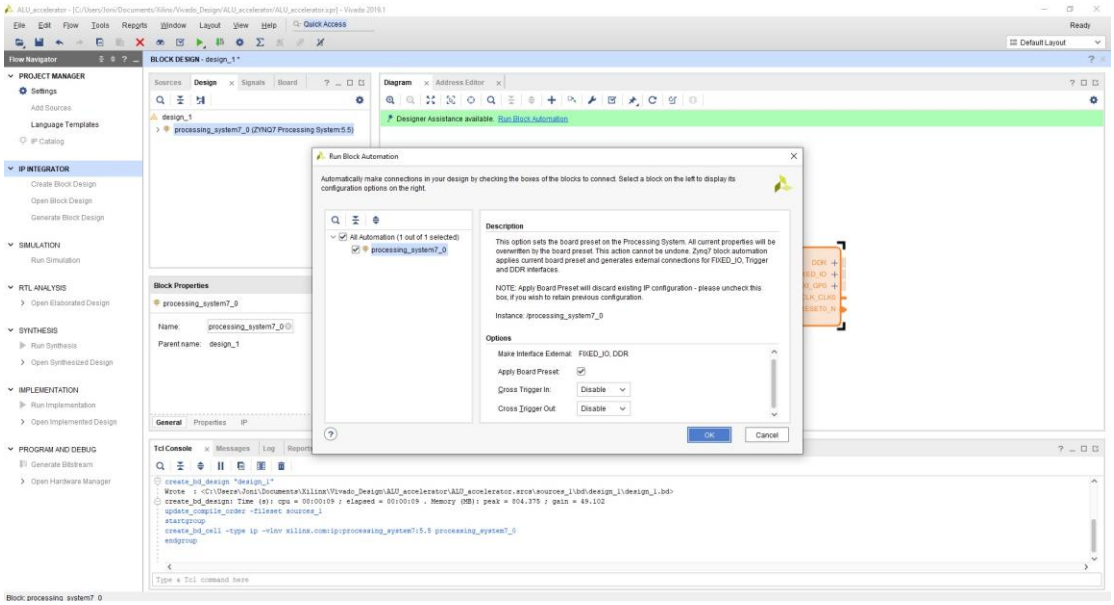


Εικόνα 4.29: Προσθήκη επιφάνειας σχεδίασης

Αρχίζουμε να δημιουργούμε το σχέδιο μας με την προσθήκη του block επεξεργαστή μας (ZYNQ7 Processing System). Επιλέγουμε το μήνυμα που εμφανίζεται ‘Run Block Automation’ (Εικόνα 4.30) για να γίνουν κάποιες αυτόματες συνδέσεις, επιλέγοντας το κουτί ‘processing_system7_0’ και τα υπόλοιπα τα φήνουμε όπως έχει (’apply board reset’, ’Cross Trigger In: Disable’ και ‘Cross Trigger Out: Disable’) (Εικόνα 4.31).



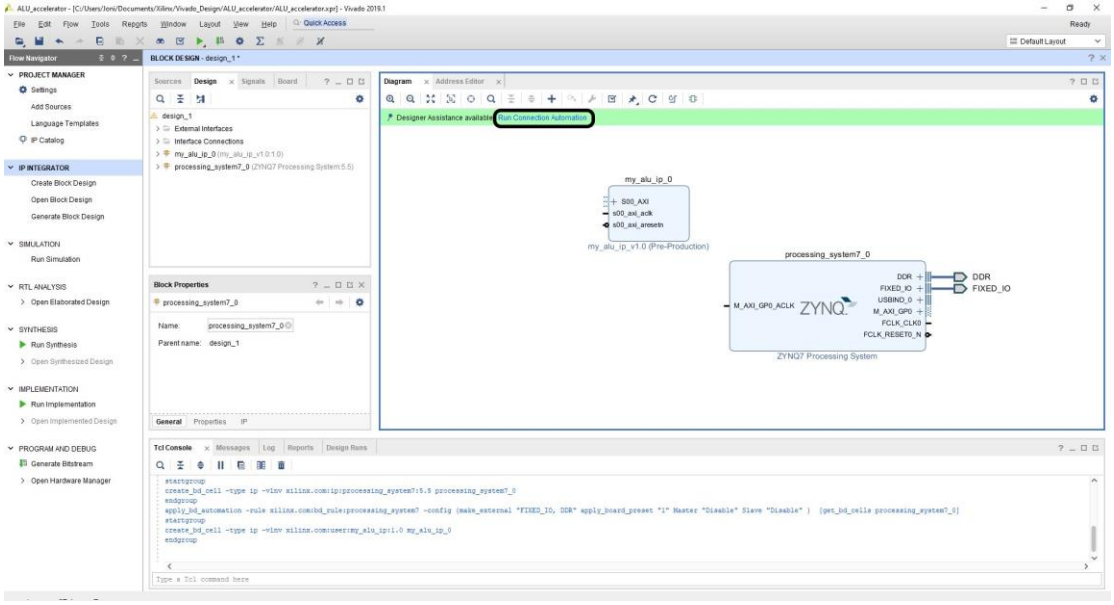
Εικόνα 4.30: Προσθήκη του επεξεργαστή Zynq και επιλογή Run Block Automation



Εικόνα 4.31: Επιλογή των συνδέσεων του επεξεργαστή

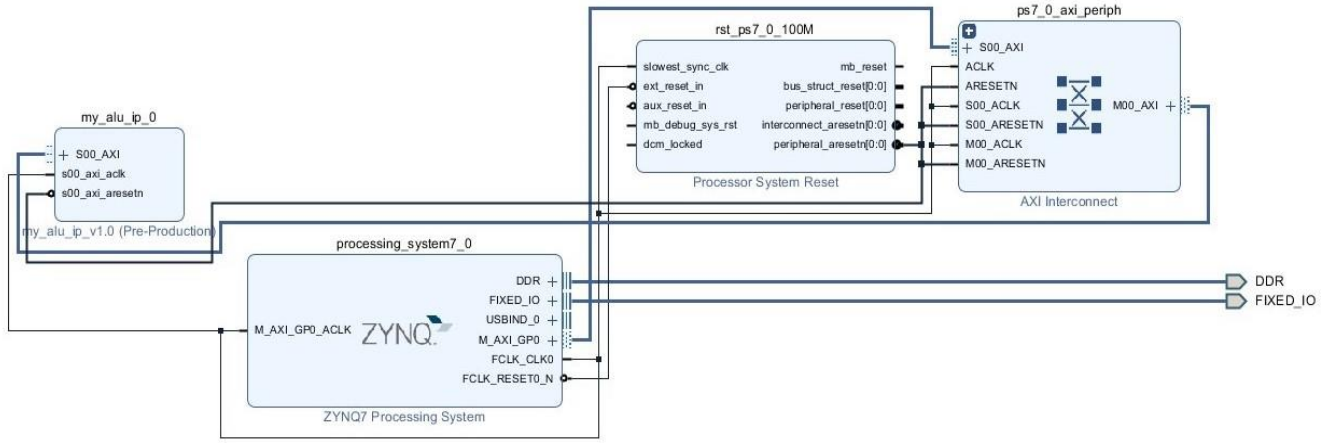
Παρατηρούμε ότι τοποθετήθηκε ένα pin DDR και ένα pin FIXED_IO για να επικοινωνεί ο επεξεργαστής με τη μνήμη και τα περιφερειακά (Εικόνα 4.32).

Έπειτα, προσθέτουμε το ALU IP που δημιουργήσαμε και επιλέγουμε το μήνυμα που εμφανίζεται 'Run Connection Automation' για να γίνουν οι αυτόματες συνδέσεις (Εικόνα 4.32).



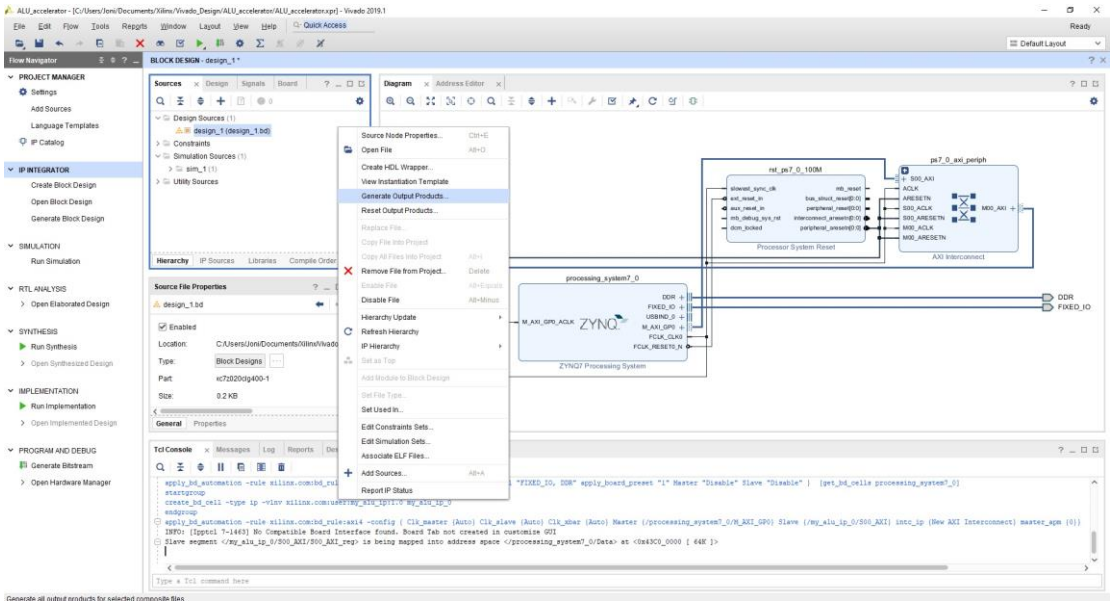
Εικόνα 4.32: Προσθήκη του IP που φτιάξαμε και επιλογή Run Connection Automation

Παρατηρούμε ότι τοποθετήθηκαν άλλα 2 IP blocks, το AXI Interconnect για να μπορεί να ελέγχει ο επεξεργαστής τα περιφερειακά και το Processor System Reset για να επανεκινούνται τα περιφερειακά (Εικόνα 4.33).



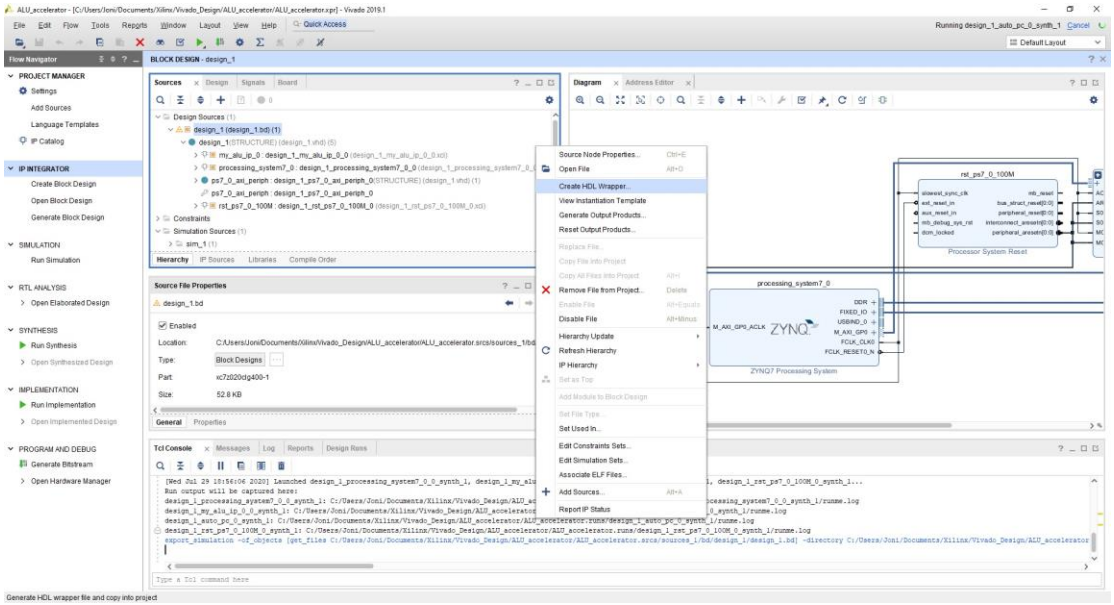
Εικόνα 4.33: Εμφάνιση του ολοκληρωμένου σχεδίου

Στη συνέχεια,από την καρτέλα Sources > Design Sources επιλέγουμε την πάνω οντότητα (top entity) και με δεξί κλίκ επιλέγουμε ‘Generate Output Products’ > ‘Out of context per IP’ (Εικόνα 4.34).



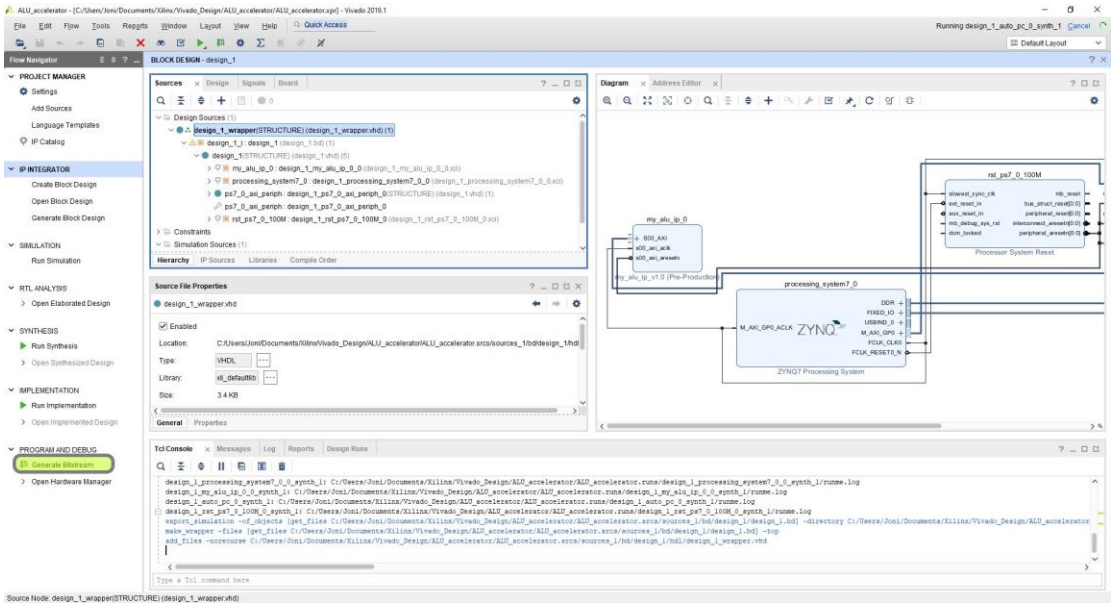
Εικόνα 4.34: Επιλογή Generate Output Products

Ακόμη μία φορά επιλέγουμε την πάνω οντότητα και με δεξί κλικ επιλέγουμε ‘Create HDL Wrapper’ > ‘Let Vivado manage wrapper and auto - update’ ώστε να ενωθούν όλα τα επιμέρους σε ένα (Εικόνα 4.35).



Εικόνα 4.35: Επιλογή Create HDL Wrapper

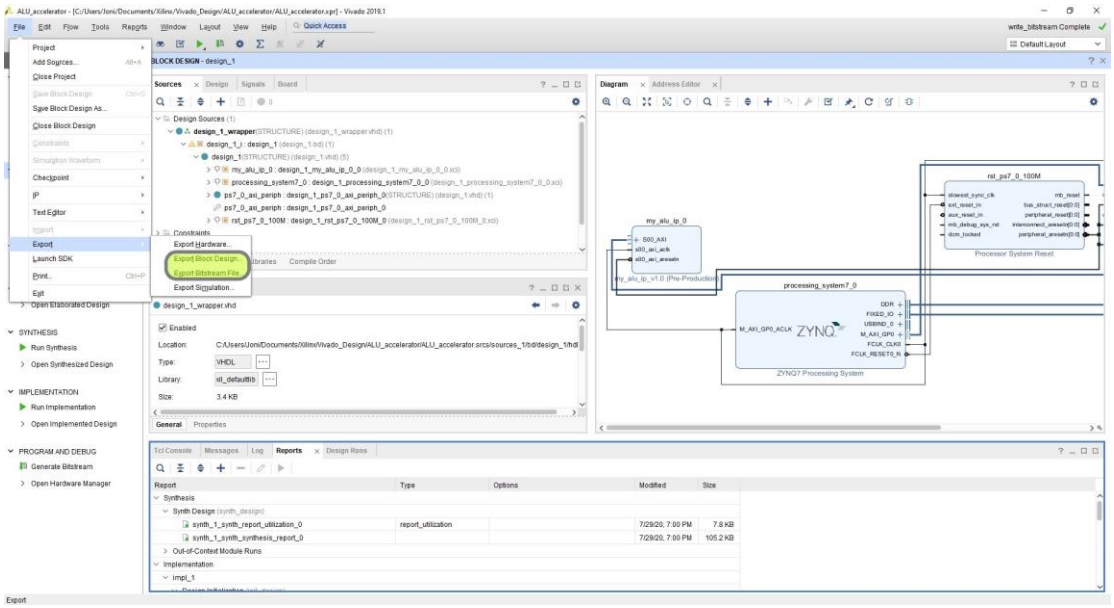
Μετά, ξεκινάμε την διαδικασία για την δημιουργία του αρχείου bitstream από το μενού PROJECT MANAGER > PROGRAM AND DEBUG > Generate Bitstream (Εικόνα 4.36).



Εικόνα 4.36: Παραγωγή αρχείου bitstream

Αφού ολοκληρωθεί με επιτυχία η διαδικασία,από το μενού File > Export, εξάγουμε σε φάκελο της επιλογής μας τα παρακάτω αρχεία (Εικόνα 4.38):

- Block Design με κατάληξη tcl
- Bitsream File με κατάληξη bit



Εικόνα 4.37: Εξαγωγή του αρχείου Block Design και Bitstream σε φάκελο της επιλογής μας

Βρίσκουμε τα αρχεία και αλλάζουμε την ονομασία τους ώστε να είναι ίδια.

Δίνουμε τροφοδοσία στην πλακέτα και μεταβαίνουμε από τον περιηγητή του υπολογιστή μας στην διεύθυνση που πήρε για να ανοίξει το Jupyter Notebook. Δημιουργούμε ένα φάκελο και τοποθετούμε μέσα τα 2 αρχεία (Εικόνα 4.38).

Εικόνα 4.38: Μετάβαση στο Jupyter Notebook,προσθήκη των αρχείων Block Design και Bitstream και δημιουργία νέου αρχείου Python

Δημιουργούμε ένα αρχείο Python 3 και γράφουμε τον κώδικα:

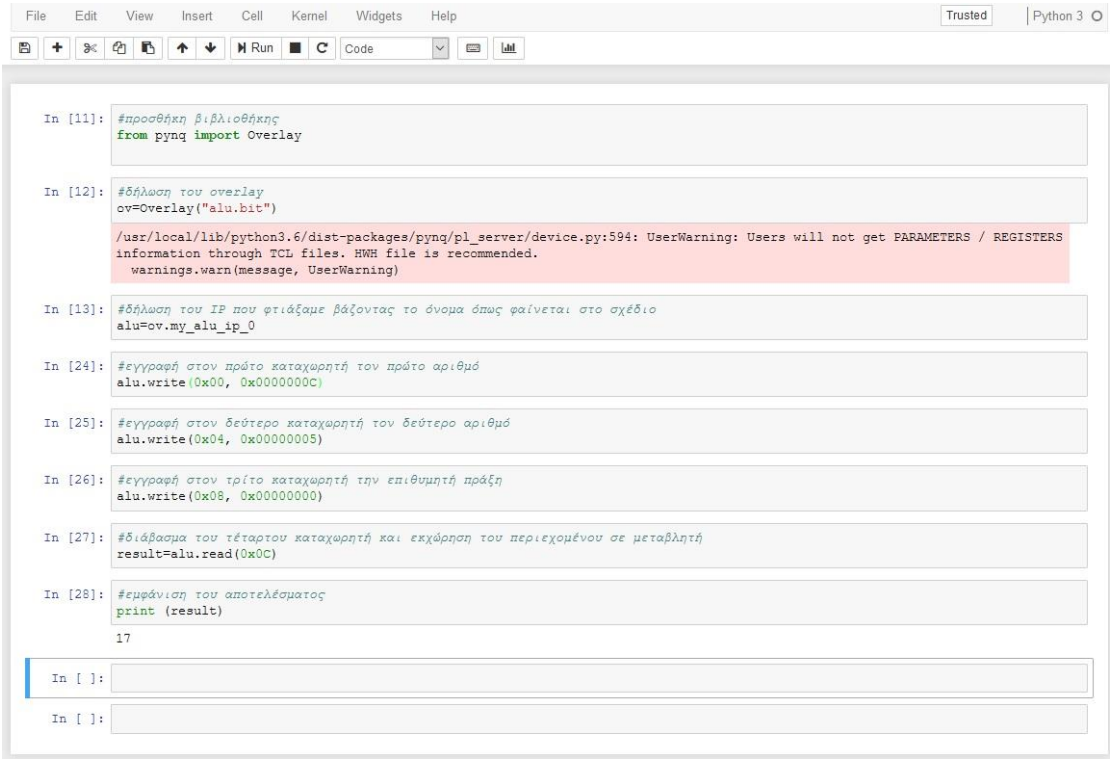
1. Εισάγουμε την βιβλιοθήκη Overlay
2. Δηλώνουμε σαν αρχείο Overlay, το bitstream μέσα από τον φάκελο
3. Δηλώνουμε το IP block που φτιάξαμε και είναι μέρος του Overlay μας
4. Κάνουμε εγγραφή στον πρώτο καταχωρητή με διεύθυνση 0x00, τον πρώτο αριθμό
5. Κάνουμε εγγραφή στον δεύτερο καταχωρητή με διεύθυνση 0x04 (αφού κάθε καταχωρητής έχει μέγεθος 32bit και στο 16δικό σύστημα είναι 4bytes), τον δεύτερο αριθμό
6. Κάνουμε εγγραφή στον τρίτο κατχωρητή με διεύθυνση 0x08, την πράξη που θέλουμε
7. Κάνουμε ανάγνωση του τέταρτου καταχωρητή με διεύθυνση 0x0C, δηλαδή το αποτέλεσμα
8. Τέλος, εμφανίζουμε το αποτέλεσμα

Στην Εικόνα 4.39 και 4.40 παρουσιάζεται η διεπαφή του Jupyter Notebook και ο κώδικας για την πράξη της πρόσθεσης και της λογικής πράξης OR αντίστοιχα.

Χρησιμοποιώ το πλεονέκτημα του αναπτυξιακού περιβάλλοντος του Pynq, δηλαδή την χρήση της βιβλιοθήκης Overlay για το αρχείο bitstream και block design που δημιουργήθηκε από το Vivado Design Tool.

Κώδικας για την πράξη της πρόσθεσης:

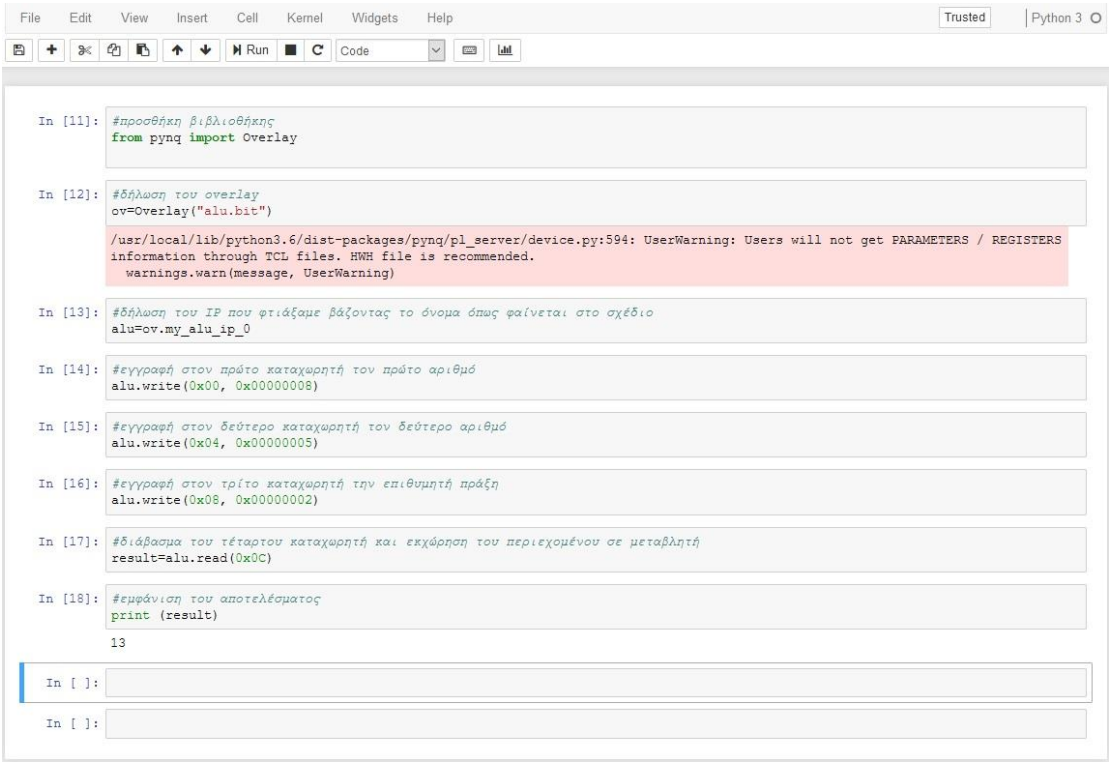
```
from pynq import Overlay

ov = Overlay("alu.bit")

alu = ov.my_alu_ip_0

alu.write(0x00, 0x0000000C)

alu.write(0x04, 0x00000005)

alu.write(0x08, 0x00000000)

result = alu.read(0x0C)

print (result)
```



Εικόνα 4.39: Ο κώδικας για την πράξη της πρόσθεσης

Κώδικας για την λογική πράξη OR:

```
from pynq import Overlay

ov = Overlay(“alu.bit”)

alu = ov.my_alu_ip_0

alu.write(0x00, 0x00000008)

alu.write(0x04, 0x00000005)

alu.write(0x08, 0x00000002)

result = alu.read(0x0C)

print (result)
```



Εικόνα 4.40:Ο κώδικας για την λογική πράξη OR

5 Εφαρμογή ανιχνευτή ακμών Sobel

Στην αρχή αυτού του κεφαλαίου παρουσιάζεται η τεχνική της ανίχνευσης ακμών με φίλτρο Sobel και τα μαθηματικά μοντέλα. Έπειτα, παρουσιάζεται η διαδικασία υλοποίησης εφαρμογής που προβάλει στην θύρα εξόδου HDMI της πλακέτας Pynq - Z2 τα δεδομένα που λαμβάνει από την θύρα εισόδου HDMI χωρίς καμία επεξεργασία με σκοπό να γίνει κατανοητό πώς λειτουργεί η λήψη και η προβολή δεδομένων εικόνας. Στο τέλος του κεφαλαίου προβάλλεται η διαδικασία υλοποίησης της εφαρμογής ανίχνευσης ακμών με φίλτρο Sobel.

5.1 Ανίχνευση ακμών

Η ανίχνευση ακμών είναι μία τεχνική εξαγωγής των ακμών που παρουσιάζονται σε μία εικόνα. Οι ακμές παρουσιάζονται όταν υπάρχουν απότομες αλλαγές στην ένταση της εικόνας και παρέχουν σημαντική πληροφορία για τον περιβάλλοντα χώρο αφού μπορούμε να αναγνωρίσουμε το σχήμα, το μέγεθος και την θέση των αντικειμένων που βρίσκονται στον χώρο.

Για να χρησιμοποιήσουμε μία τέτοια τεχνική χρειάζεται να μετατρέψουμε την εικόνα σε επίπεδο φωτεινότητας του γκρί, ελαχιστοποιώντας έτσι την ποσότητα των δεδομένων που υπόκεινται επεξεργασία. Έτσι, η τιμή από κάθε pixel αντιπροσωπεύει την ένταση, με το μαύρο να αντιστοιχεί στην χαμηλότερη τιμή και το άσπρο στην υψηλότερη τιμή της έντασης.

Πρακτικά, για να βρούμε μία ακμή, αρκεί να βρούμε το μέγεθος και την κατεύθυνση της κατά μήκος του οριζόντιου και κάθετου άξονα σε μία εικόνα.

Το εργαλείο για να βρεθεί το μέγεθος μίας ακμής και η κατεύθυνση της σε μία τοποθεσία (x,y) μίας εικόνας f, είναι η κλίση, συμβολίζεται ως Δf και ορίζεται ως το διάνυσμα (Εικόνα 5.1) :

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Εικόνα 5.1: Το διάνυσμα της κλίσης μίας ακμής σε μία εικόνα

Το μέγεθος (μήκος) του διανύσματος Δf, συμβολίζεται ως $M(x, y)$ όπου (Εικόνα 5.2) :

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

Εικόνα 5.2: Το μέγεθος του διανύσματος της κλίσης μίας ακμής

Η κατεύθυνση του διανύσματος κλίσης δίνεται από την γωνία (Εικόνα 5.3) :

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

Εικόνα 5.3: Η κατεύθυνση του διανύσματος κλίσης

Η απόκτηση της κλίσης μίας εικόνας απαιτεί τον υπολογισμό των μερικών παραγώγων των $\partial f / \partial x$ και $\partial f / \partial y$ σε κάθε pixel της εικόνας (Εικόνα 5.4). Έχουμε να κάνουμε με ψηφιακές ποσότητες οπότε απαιτείται μία ψηφιακή προσέγγιση των μερικών παραγώγων σε μία γειτονική περιοχή γύρω από ένα σημείο (Εικόνα 5.5).

$$\frac{\partial f}{\partial x} = f'(x) = f(x+1) - f(x)$$

$$\frac{\partial f}{\partial y} = f'(y) = f(y+1) - f(y)$$

Εικόνα 5.4: Προσέγγιση της πρώτης παραγώγου

$$g_x = \frac{\partial f(x, y)}{\partial x} = f(x+1, y) - f(x, y)$$

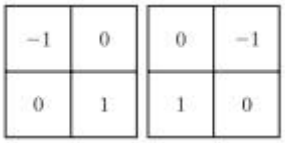
$$g_y = \frac{\partial f(x, y)}{\partial y} = f(x, y+1) - f(x, y)$$

Εικόνα 5.5: Η κλίση μίας εικόνας

Αυτές οι εξισώσεις μπορούν να εφαρμοστούν για όλες τις τιμές των x, y μέσω φιλτραρίσματος του $f(x, y)$ με τις παρακάτω μάσκες μίας διάστασης 1D (Εικόνα 5.6). Πιο συγκεκριμένα, εφαρμόζεται συνέλιξη μεταξύ ενός φίλτρου και της εικόνας, κατά μήκος του οριζόντιου και κατακόρυφου άξονα της εικόνας και το αποτέλεσμα παράγεται από τον συνδυασμό τους.

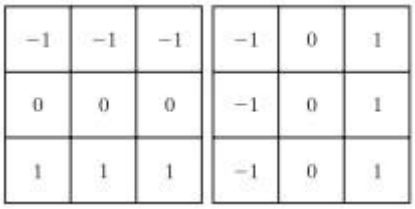
Εικόνα 5.6: Πίνακες μίας διάστασης

Όταν χρειαζόμαστε την κατεύθυνση μίας διαγώνιας ακμής, χρειαζόμαστε μία μάσκα 2 διαστάσεων 2D. Οι μάσκες Roberts είναι από τις πρώτες προσπάθειες διαγώνιας προτίμηση (Εικόνα 5.7).

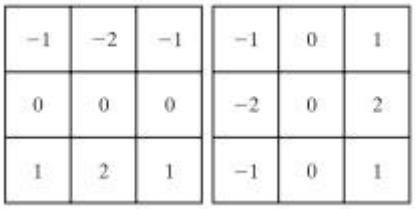


Εικόνα 5.7: Μάσκες Roberts

Οι μάσκες 2x2 είναι απλές αλλά δεν είναι το ίδιο χρήσιμες για τον υπολογισμό της κατεύθυνσης της ακμής όπως οι μάσκες που είναι ασυμετρικές γύρω από το κεντρικό σημείο με τις μικρότερες να έχουν μέγεθος 3x3 (Εικόνα 5.8, 5.9). Αυτές οι μάσκες λαμβάνουν υπόψη την φύση των δεδομένων στις αντίθετες πλευρές του κεντρικού σημείου με αποτέλεσμα να μεταφέρουν περισσότερες πληροφορίες για την κατεύθυνση της ακμής.



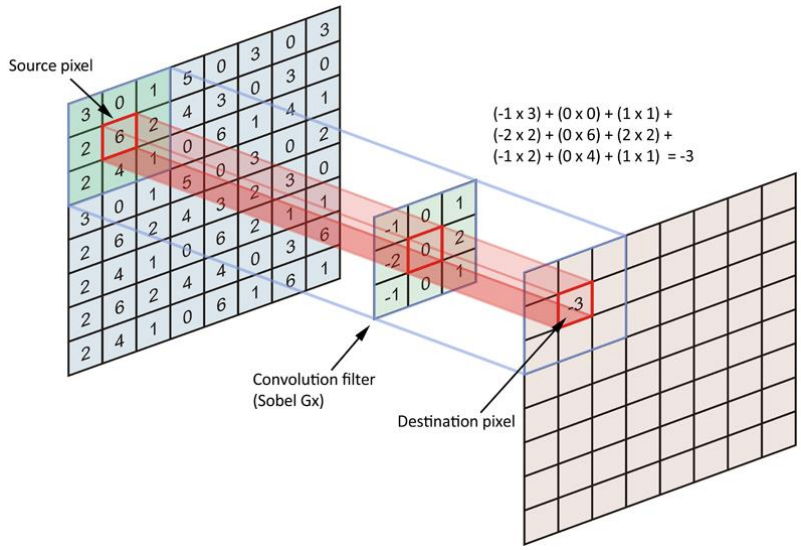
Εικόνα 5.8: Μάσκες Prewitt



Εικόνα 5.9: Μάσκες Sobel

Το πλεονέκτημα των μασκών του Sobel είναι ότι έχουν καλύτερη συμπίεση θορύβου (εξομάλυνση εικόνας στα κεντρικά σημεία μέσω του συντελεστή 2) γι΄ αυτό και προτιμώνται περισσότερο.

Κατά την διαδικασία του φιλτραρίσματος της εικόνας, τοποθετούμε το φίλτρο έτσι ώστε το κέντρο του να συμπίπτει στο pixel που θέλουμε να επεξεργαστούμε και πολλαπλασιάζουμε τις τιμές των pixels της αρχικής εικόνας με αυτές του φίλτρου και ύστερα προσθέτουμε τις τιμές αυτές για να βρούμε την νέα τιμή του pixel (Εικόνα 5.10).



Εικόνα 5.10: Διαδικασία φιλτραρίσματος εικόνας στον οριζόντιο άξονα με μάσκα Sobel

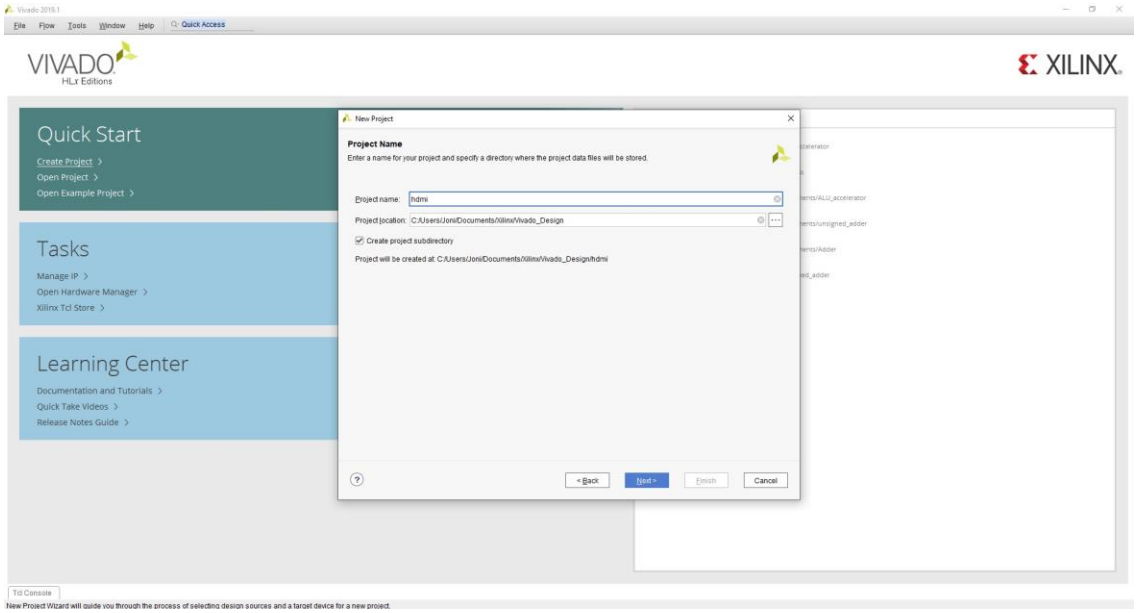
5.2 Προεργασία για την εφαρμογή ανιχνευτή ακμών Sobel

Για να φτιάξουμε τον ανιχνευτή ακμών, πρώτα θα φτιάξουμε μία εφαρμογή που λαμβάνει από μία πηγή HDMI ένα σήμα και το προβάλλει σε μία έξοδο HDMI χωρίς επεξεργασία, ώστε να κατανοήσουμε πώς λειτουργεί η λήψη και η προβολή δεδομένων εικόνας.

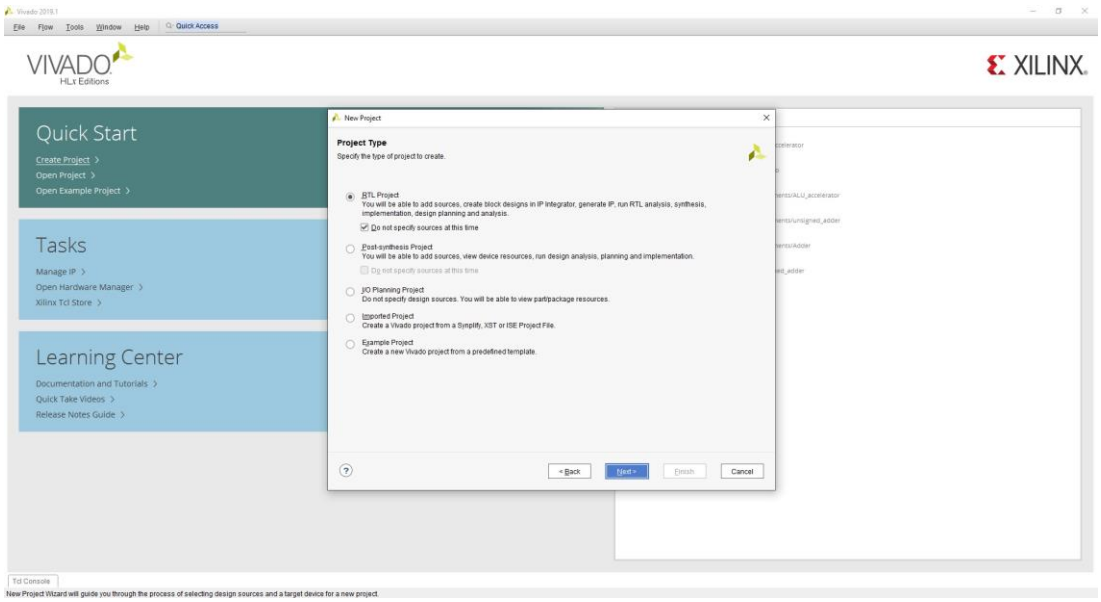
5.2.1 Τα βήματα για την δημιουργία εφαρμογής που προβάλλει ένα σήμα εισόδου HDMI σε έξοδο HDMI

Στην εφαρμογή μας χρησιμοποιούμε τα blocks ‘DVI to RGB Video Decoder’ και ‘RGB to DVI Video Encoder’, τα οποία δεν ανήκουν στην εγκατεστημένη βιβλιοθήκη του Vivado αλλά τα κατεβάζουμε από εδώ [115] .

Ανοίγουμε το Vivado IDE και δημιουργούμε ένα νέο project επιλέγοντας τον τύπο του project, στην περίπτωση μας RTL project και χωρίς να καθορίσουμε πηγαία αρχεία (Εικόνα 5.11, 5.12).

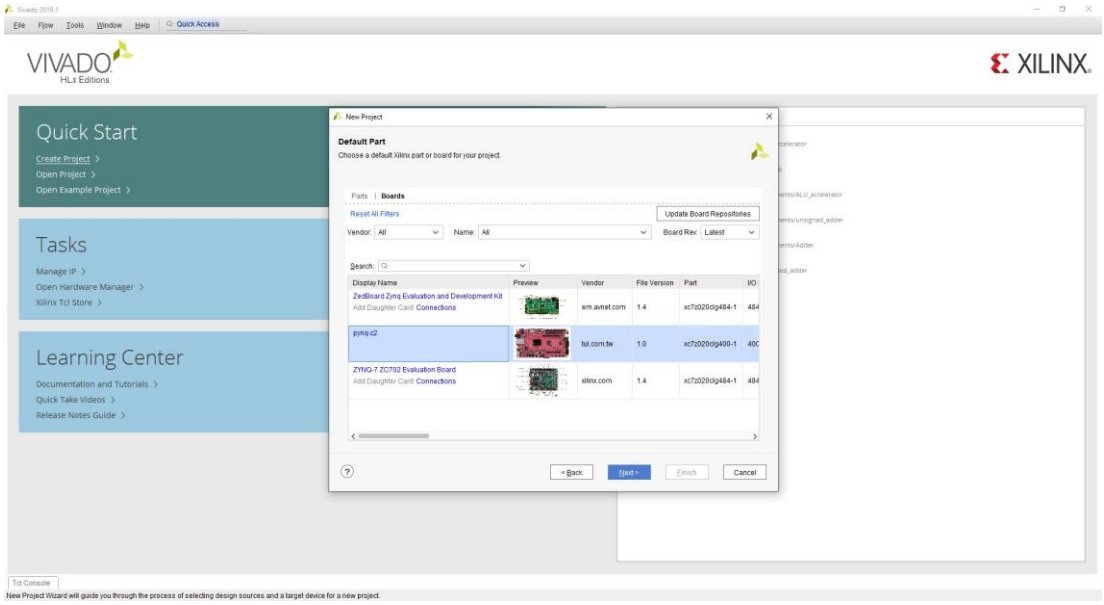


Εικόνα 5.11:Δημιουργία νέου project ροής εικόνας χωρίς επεξεργασία

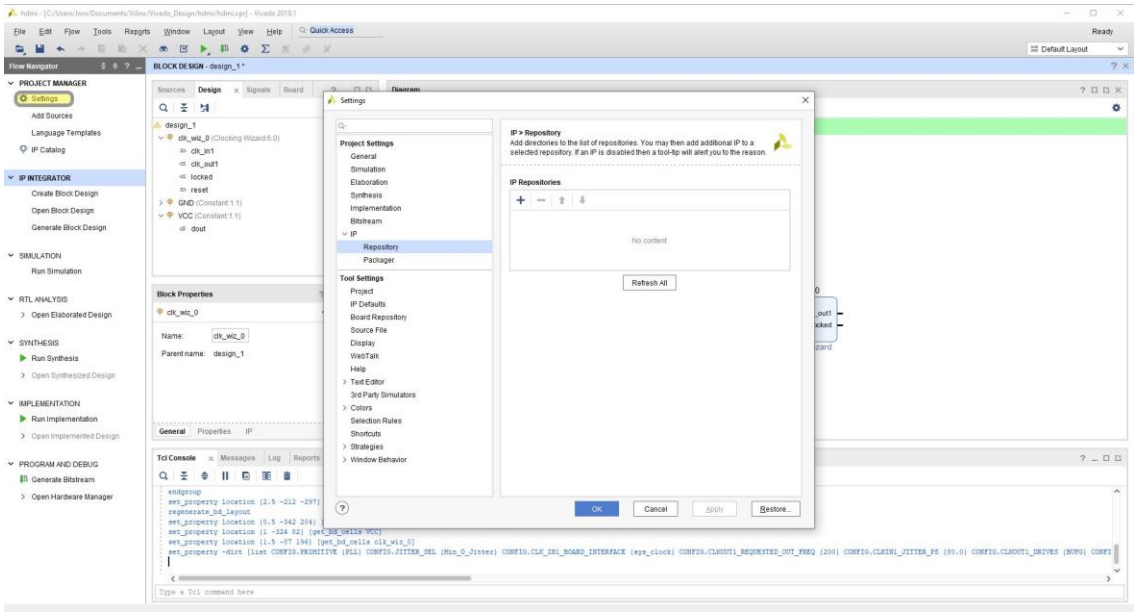


Εικόνα 5.12: Επιλογή τύπου του project ροής εικόνας χωρίς επεξεργασία

Επιλέγουμε την πλακέτα μας και αφού ολοκληρωθεί η δημιουργία του και ανοίξει το κεντρικό παράθυρο, προσθέτουμε την βιβλιοθήκη IP που κατεβάσαμε (Εικόνα 5.13, 5.14).

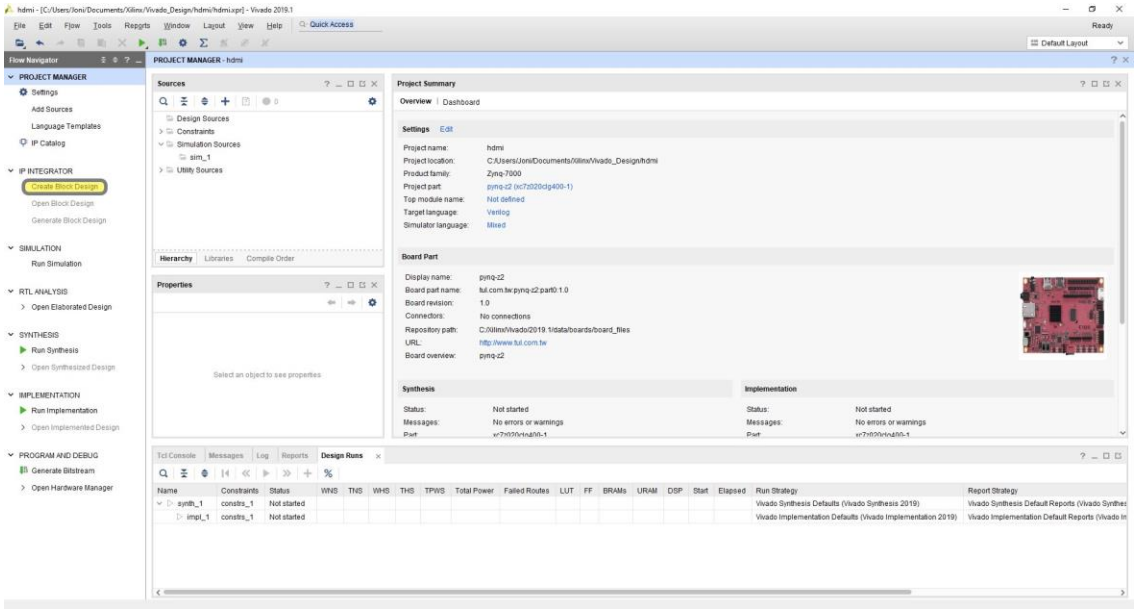


Εικόνα 5.13:Επιλογή πλακέτας



Εικόνα 5.14:Προσθήκης βιβλιοθήκης IP

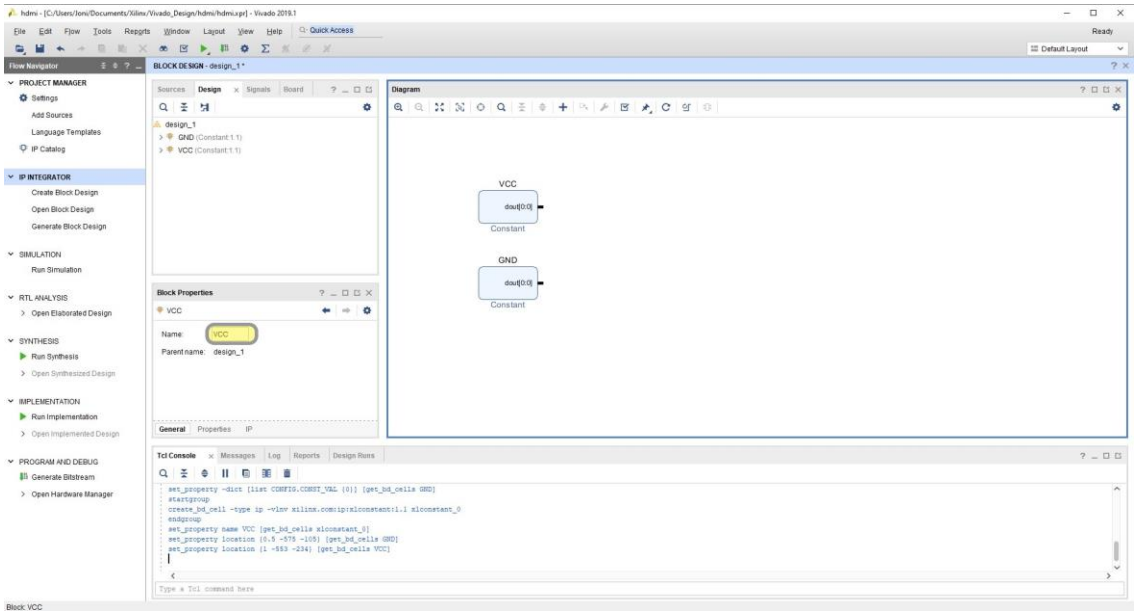
Ύστερα, φτιάχνουμε την επιφάνεια του σχεδίου μας από το μενού PROJECT MANAGER > IP INTEGRATOR > Create Block Design (Εικόνα 5.15).



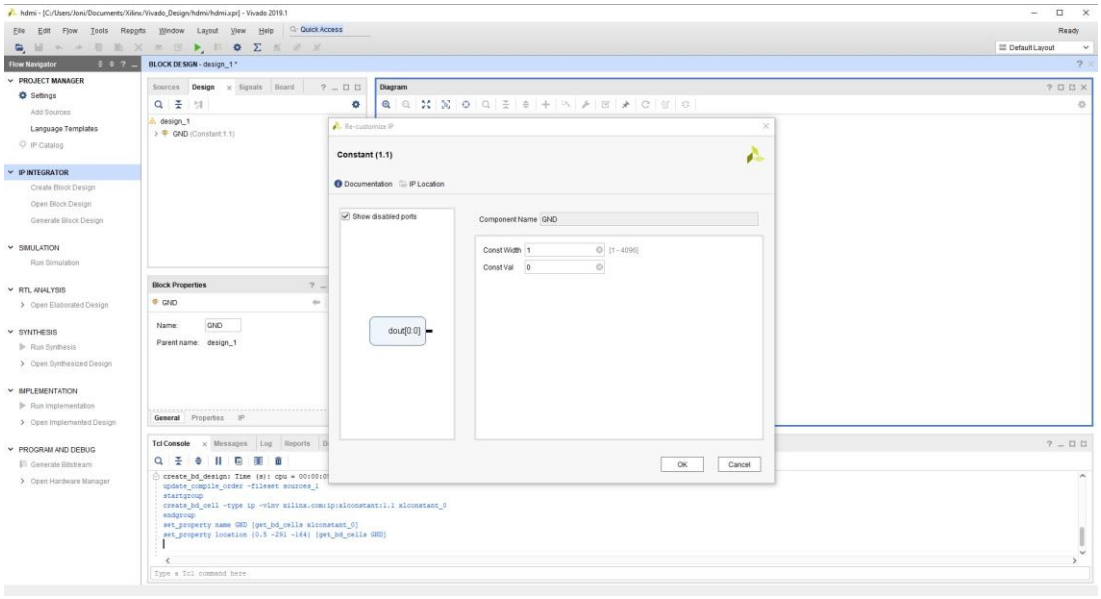
Εικόνα 5.15:Δημιουργία επιφάνεις σχεδίασης

Αρχίζουμε να δημιουργούμε το σχέδιο μας με την προσθήκη των παρακάτω:

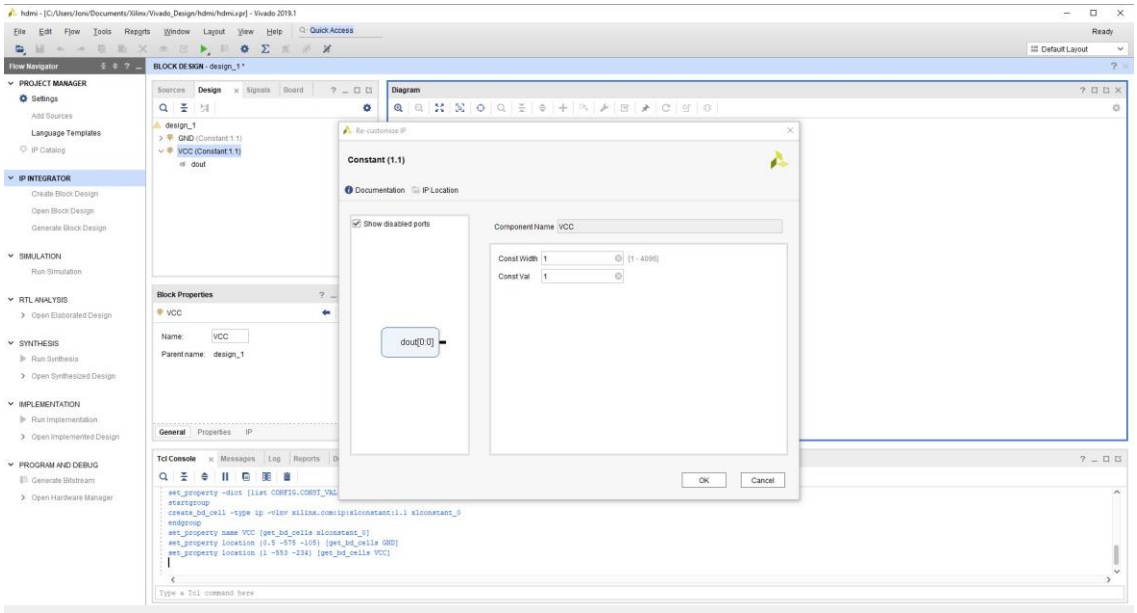
Constant: Χρησιμοποιείται για να δώσει μία σταθερή τιμή. Στην εφαρμογή μας έχουμε 2 blocks (Εικόνα 5.16), το ένα για να δώσουμε τιμή 0 (Const Val = 0) (Εικόνα 5.17) και το άλλο για τιμή 1 (Const Val = 1) (Εικόνα 5.18).



Εικόνα 5.16:Προσθήκη Constant blocks και αλλαγή ονόματος



Εικόνα 5.17:Επεξεργασία πρώτου Constant block (GND)



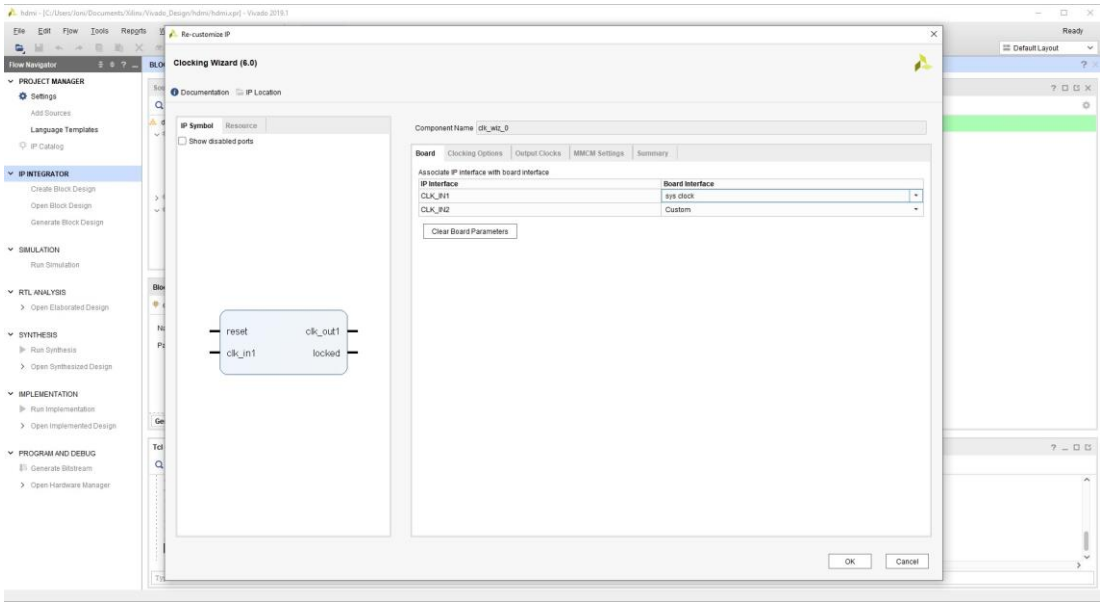
Εικόνα 5.18: Επεξεργασία δεύτερου Constant block (VCC)

Clocking Wizard: Χρησιμοποιείται για να παράγει ένα επιθυμητό παλμό ρολογιού στην έξοδο.

Οι ρυθμίσεις:

Καρτέλα Board (Εικόνα 5.19)

CLK_IN1 = sys clock (δηλώνουμε σαν είσοδο του block τον παλμό ρολογιού της πλακέτας, δηλαδή 125MHz) .



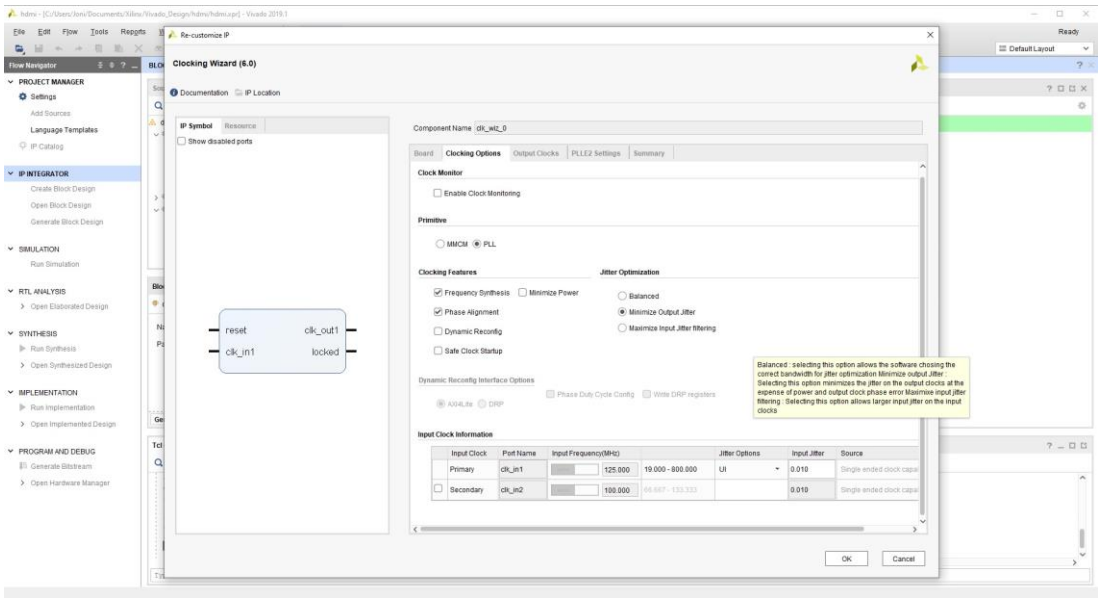
Εικόνα 5.19: Διασύνδεση παλμού εισόδου ρολογιού με την πλακέτα

Καρτέλα Clocking Options (Εικόνα 5.20)

Primitive => PLL

Clocking Features => Frequency Synthesis, Phase Alignment

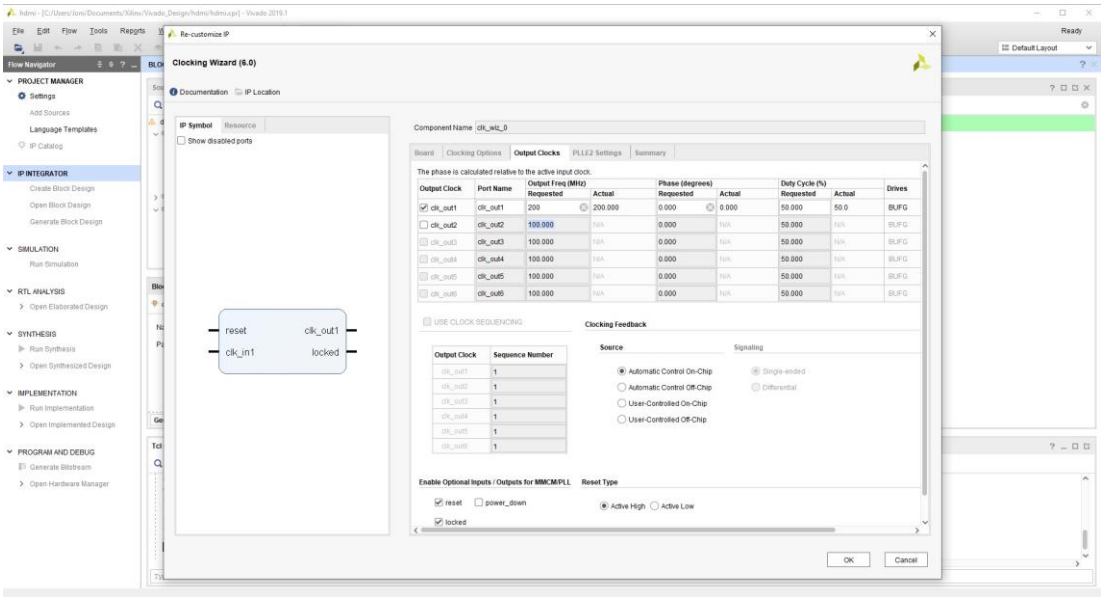
Jitter Optimization: Minimize Output Jitter



Εικόνα 5.20:Ρυθμίσεις χρονισμού

Καρτέλα Output Clocks (Εικόνα 5.21)

Βάζουμε σαν επιθυμητή συχνότητα στην έξοδο **clk_out1** 200MHz.



Εικόνα 5.21:Προσθήκη επιθυμητής τιμής συχνότητας χρονισμού στην έξοδο

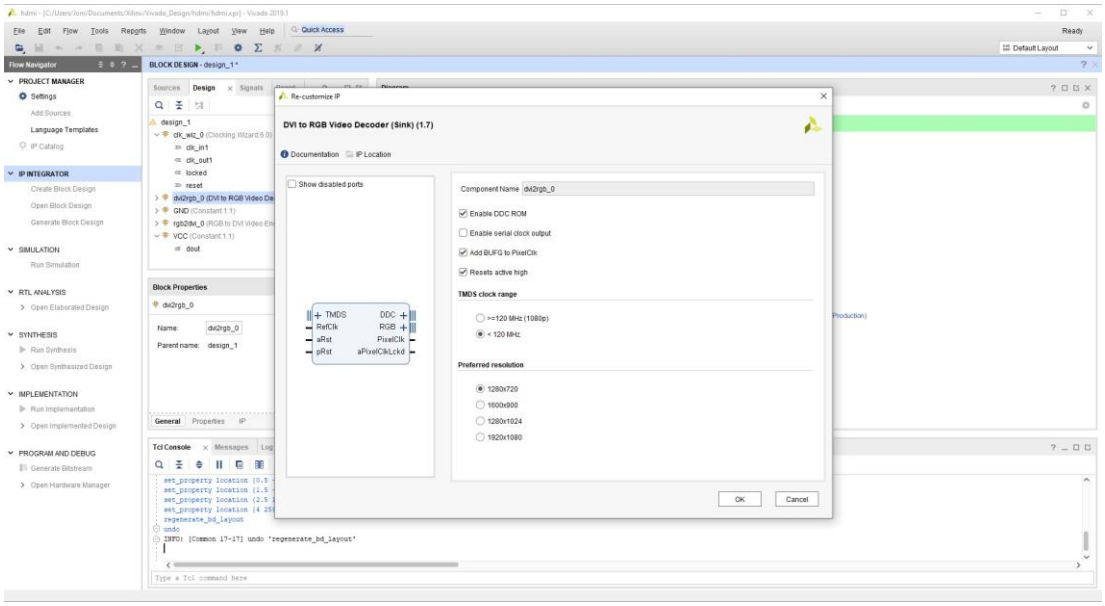
DVI to RGB Video Decoder: Χρησιμοποιείται για να μετατρέψει το σήμα εισόδου (DVI) σε σήμα τριών καναλιών χρωμάτων (RGB).

Οι ρυθμίσεις (Εικόνα 5.22):

Επιλέγουμε **Enable DDC ROM, Add BUFG to PixelClk, Reset active high**

TDMS clock range < 120MHz

Preferred resolution: επιλέγουμε 1280x720



Εικόνα 5.22:Ρυθμίσεις ΄DVI to RGB Video Decoder΄

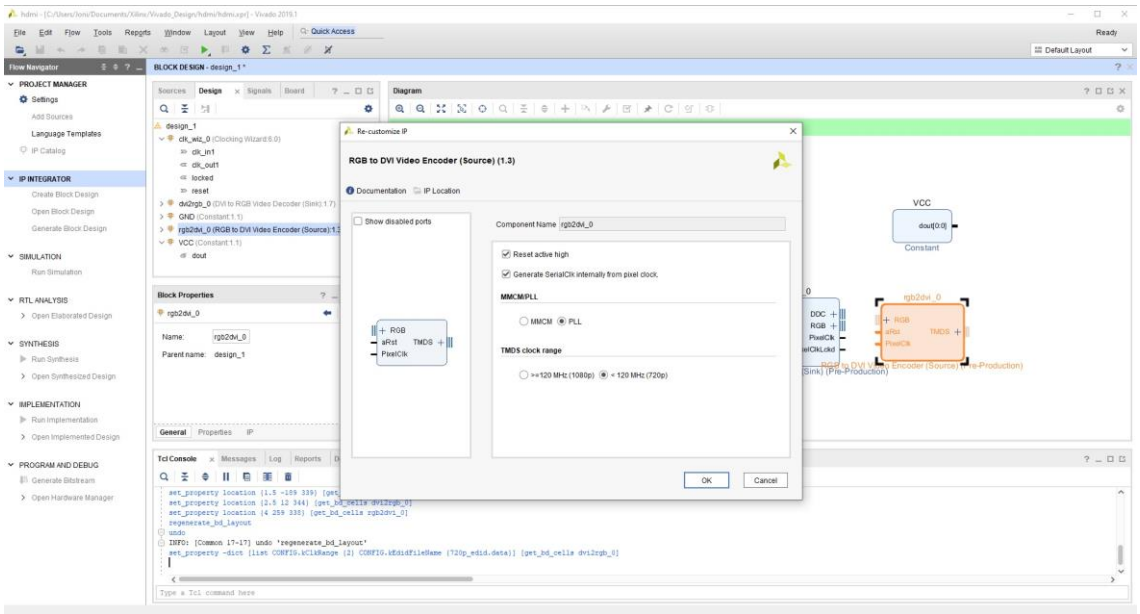
RGB to DVI Video Encoder: Χρησιμοποιείται για να μετατρέψει το σήμα εισόδου RGB σε σήμα DVI.

Οι ρυθμίσεις (Εικόνα 5.23)*:*

Επιλέγουμε **Reset active high, Generate SerialClk internally from pixel clock**.

MMCM/PLL: επιλέγουμε PLL

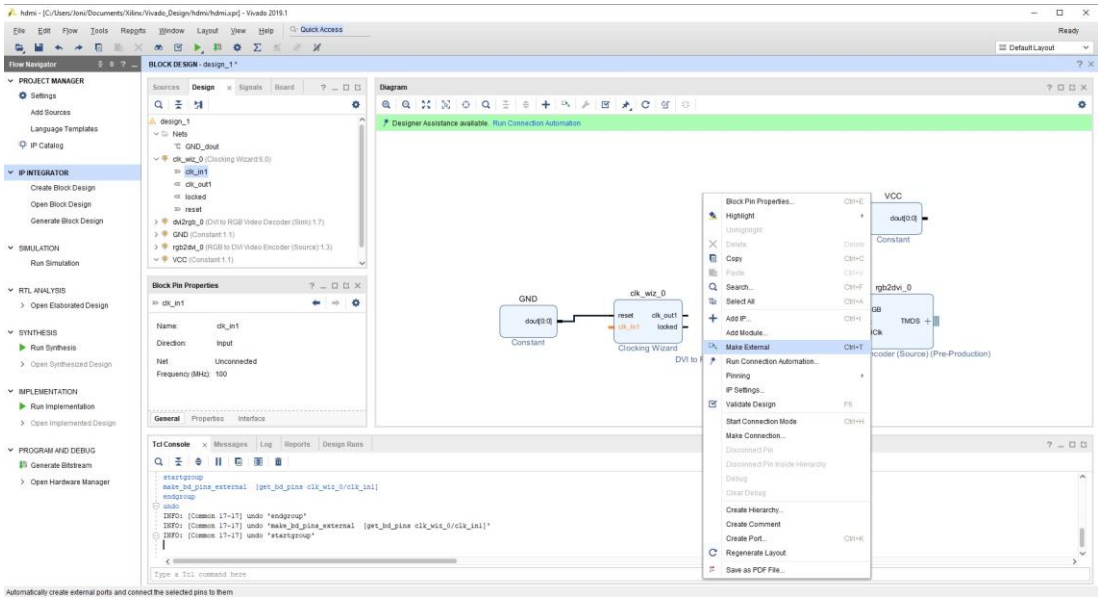
TDMS clock range < 120MHz (720p)



Εικόνα 5.23:Οι ρυθμίσεις ΄RGB to DVI Video Encoder΄

Ακολουθείται η παρακάτω συνδεσμολογία:

Οδηγούμε το σήμα reset του ΄Clocking Wizard΄ σε μόνιμη τιμή 0 (block constant GND), διασυνδέουμε την είσοδο του (clk_in1) με τον παλμό ρολογιού της πλακέτας (Εικόνα 5.24) και την έξοδο του (clk_out1) στο σήμα RefClk από το ΄DVI to RGB Video Decoder΄ για να συγχρονιστεί το block στα 200MHz, σύμφωνα με τον κατασκευαστή.



Εικόνα 5.24:Διασύνδεση παλμού εισόδου με την πλακέτα

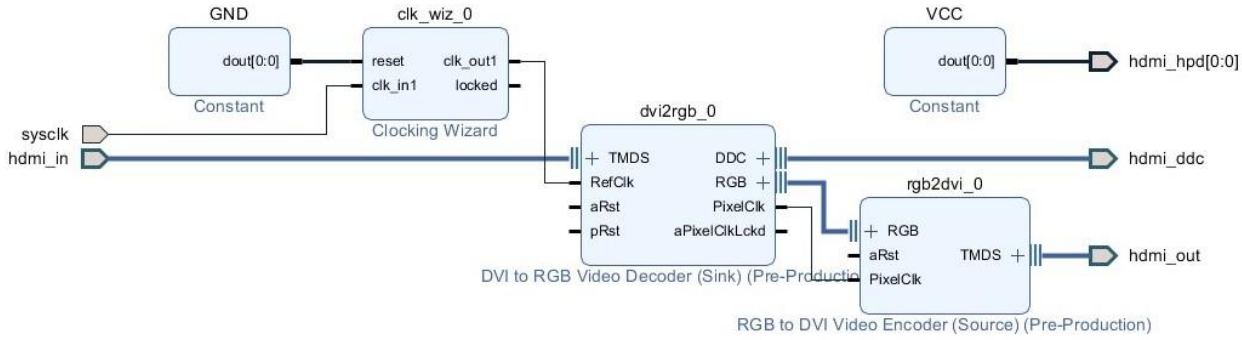
Διασυνδέουμε το σήμα εισόδου TMDS (Transition - Minimized Differential Signaling) με την πλακέτα. Το TMDS είναι ένα πρότυπο για την μετάδοση βίντεο σε DVI και HDMI.

Διασυνδέουμε το σήμα DDC (Display Data Channel) στο αντίστοιχο κανάλι πάνω στην πλακέτα. Το κανάλι DDC αποτελείται από το σήμα SDA (για τα δεδομένα) που είναι αμφίδρομο και το σήμα SCL (για τον παλμό ρολογιού) που είναι μονής κατεύθυνσης.

Ενώνουμε τα σήματα PixelClk και RGB μεταξύ των 2 blocks ΄DVI to RGB Video Decoder΄-΄RGB to DVI Video Encoder΄ και διασυνδέουμε το σήμα εξόδου TMDS με την πλακέτα.

Οδηγούμε το σήμα HPD (Hot Plug Detect) στην τιμή 1 για να δηλώσουμε ότι υπάρχει σήμα στην είσοδο HDMI In της πλακέτας.

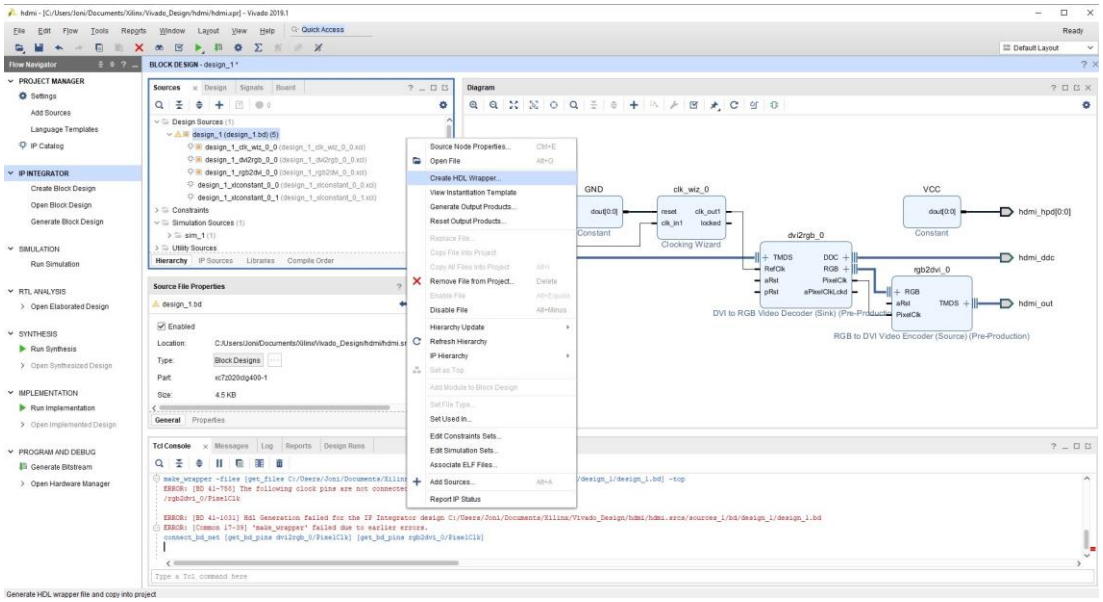
Το τελικό σχέδιο φαίνεται στην Εικόνα 5.25.



Εικόνα 5.25:Ολοκληρωμένη σχεδίαση εφαρμογής ροής εικόνας

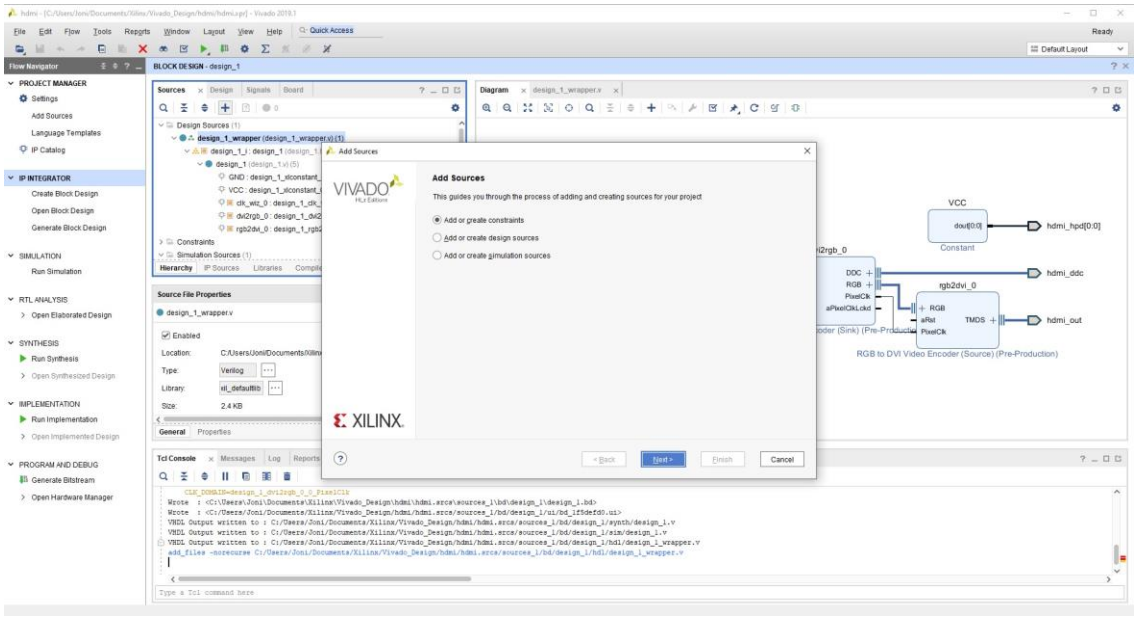
Στη συνέχεια,από την καρτέλα Sources > Design Sources επιλέγουμε την πάνω οντότητα (top entity)και με δεξί κλίκ επιλέγουμε ‘Generate Output Products’ > ‘Out of context per IP’

Ακόμη μία φορά επιλέγουμε την πάνω οντότητα και με δεξί κλικ επιλέγουμε ‘Create HDL Wrapper’ > ‘Let Vivado manage wrapper and auto-update’ ώστε να ενωθούν όλα τα επιμέρους σε ένα (Εικόνα 5.26).

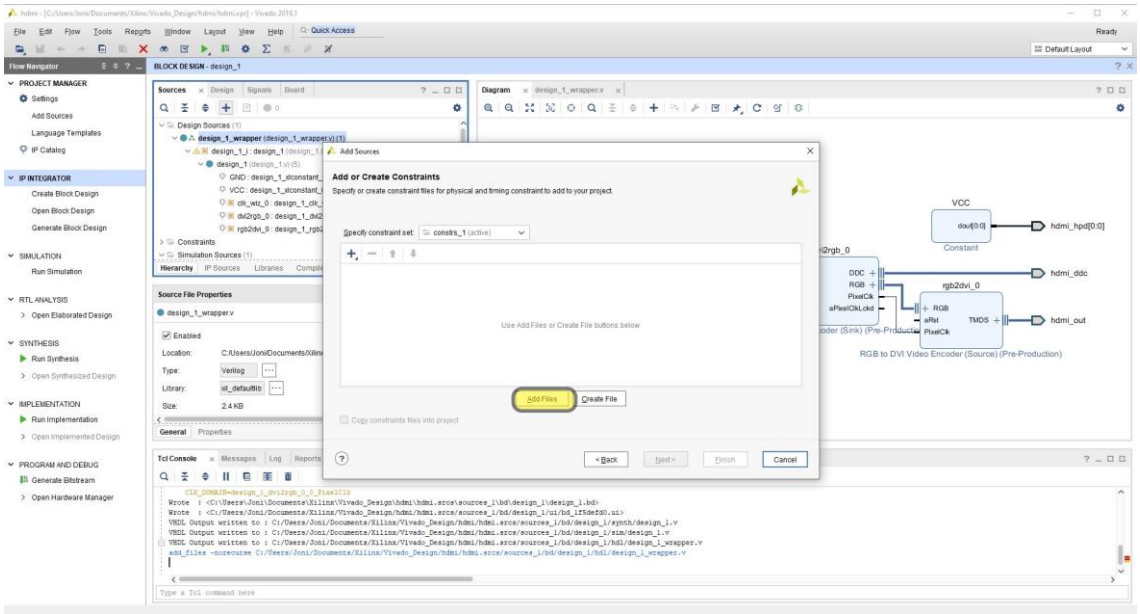


Εικόνα 5.26:Επιλογή ενοποίησης των υλικών ‘Create HDL Wrapper’

Επόμενο βήμα είναι να προσθέσουμε τους φυσικούς περιορισμούς που κατεβάζουμε από εδώ [41] (Εικόνα 5.27, 5.28, 5.29).



Εικόνα 5.27:Επιλογή προσθήκης φυσικών περιορισμών



Εικόνα 5.28:Προσθήκη αρχείου φυσικών περιορισμών

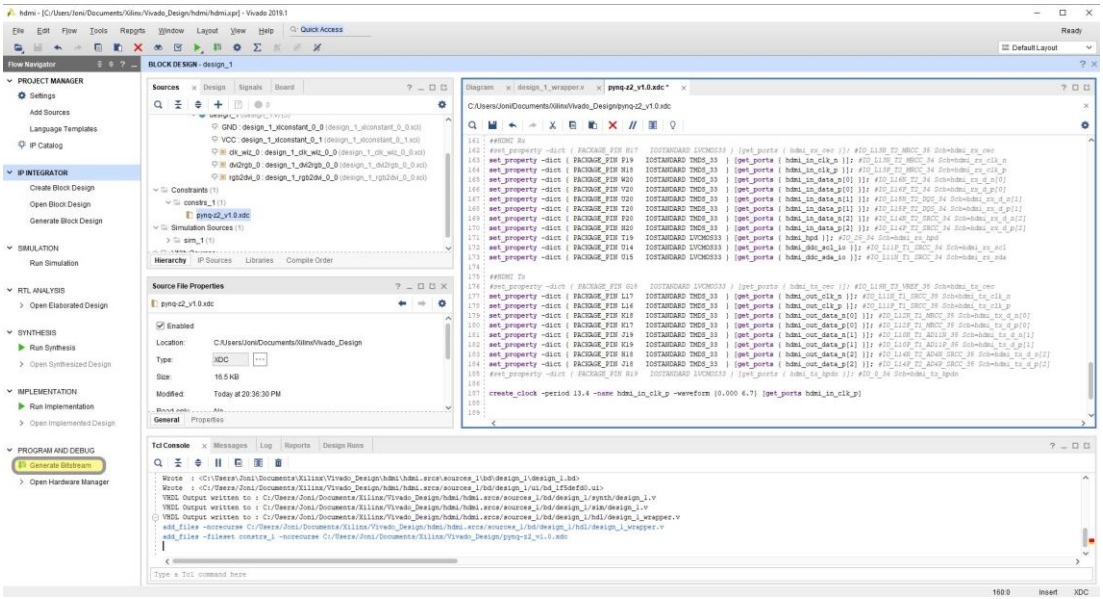
Είναι απαραίτητο να συνδέσουμε ένα παλμό ρολογιού στο σήμα εισόδου TMDS_Clk_p για τον χρονισμό των pixels. Με βάση τους φυσικούς περιορισμούς του block, η μέγιστη συχνότητα που μπορεί να έχει ο παλμός αυτός είναι τα 165MHz, δηλαδή περίοδος 6,06s.

Για ανάλυση 720p αντιστοιχεί παλμός συχνότητας 74,25MHz που σημαίνει περίοδος 13,4s.

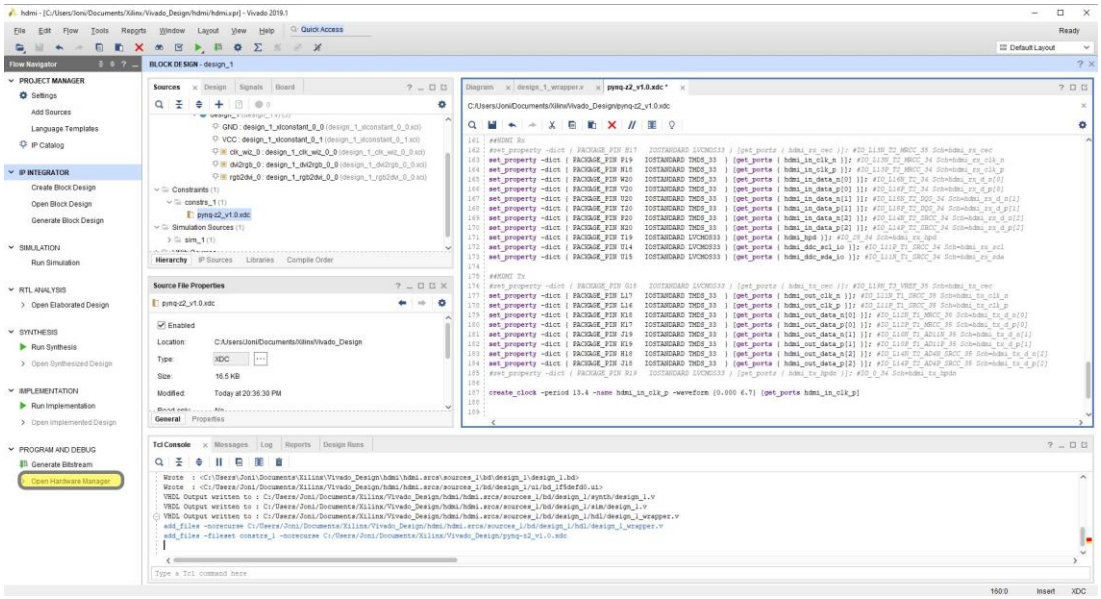
```
##HDMI Rx
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_cec }]; #IO_L13N_T2_MRCC_35 Sch=hdmi_rx_cec
set_property -dict { PACKAGE_PIN P19   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_clk_n }]; #IO_L13N_T2_MRCC_34 Sch=hdmi_rx_clk_n
set_property -dict { PACKAGE_PIN N18   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_clk_p }]; #IO_L13P_T2_MRCC_34 Sch=hdmi_rx_clk_p
set_property -dict { PACKAGE_PIN W20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_n[0] }]; #IO_L16N_T2_34 Sch=hdmi_rx_d_n[0]
set_property -dict { PACKAGE_PIN V20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_p[0] }]; #IO_L16P_T2_34 Sch=hdmi_rx_d_p[0]
set_property -dict { PACKAGE_PIN U20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_n[1] }]; #IO_L15N_T2_DQS_34 Sch=hdmi_rx_d_n[1]
set_property -dict { PACKAGE_PIN T20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_p[1] }]; #IO_L15P_T2_DQS_34 Sch=hdmi_rx_d_p[1]
set_property -dict { PACKAGE_PIN P20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_n[2] }]; #IO_L14N_T2_SRCC_34 Sch=hdmi_rx_d_n[2]
set_property -dict { PACKAGE_PIN N20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_p[2] }]; #IO_L14P_T2_SRCC_34 Sch=hdmi_rx_d_p[2]
set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports { hdmi_hpd }]; #IO_25_34 Sch=hdmi_rx_hpd
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { hdmi_ddc_scl_io }]; #IO_L11P_T1_SRCC_34 Sch=hdmi_rx_scl
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports { hdmi_ddc_sda_io }]; #IO_L11N_T1_SRCC_34 Sch=hdmi_rx_sda

##HDMI Tx
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_cec }]; #IO_L19N_T3_VREF_35 Sch=hdmi_tx_cec
set_property -dict { PACKAGE_PIN L17   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_clk_n }]; #IO_L11N_T1_SRCC_35 Sch=hdmi_tx_clk_n
set_property -dict { PACKAGE_PIN L16   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_clk_p }]; #IO_L11P_T1_SRCC_35 Sch=hdmi_tx_clk_p
set_property -dict { PACKAGE_PIN K18   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_n[0] }]; #IO_L12N_T1_MRCC_35 Sch=hdmi_tx_d_n[0]
set_property -dict { PACKAGE_PIN K17   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_p[0] }]; #IO_L12P_T1_MRCC_35 Sch=hdmi_tx_d_p[0]
set_property -dict { PACKAGE_PIN J19   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_n[1] }]; #IO_L10N_T1_AD11N_35 Sch=hdmi_tx_d_n[1]
set_property -dict { PACKAGE_PIN K19   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_p[1] }]; #IO_L10P_T1_AD11P_35 Sch=hdmi_tx_d_p[1]
set_property -dict { PACKAGE_PIN H18   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_n[2] }]; #IO_L14N_T2_AD4N_SRCC_35 Sch=hdmi_tx_d_n[2]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_p[2] }]; #IO_L14P_T2_AD4P_SRCC_35 Sch=hdmi_tx_d_p[2]
#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_hpdn }]; #IO_0_34 Sch=hdmi_tx_hpdn

create_clock -period 13.4 -name hdmi_in_clk_p -waveform {0.000 6.7} [get_ports hdmi_in_clk_p]


```

Εικόνα 5.29:Οι φυσικοί περιορισμοί

Ακολουθούμε την διαδικασία για την δημιουργίας του αρχείου bitstream και στη συνέχεια ανοίγουμε τον Hardware Manager για να το μεταφέρουμε στο λογικό υλικό της πλακέτας (Εικόνα 5.30, 5.31).

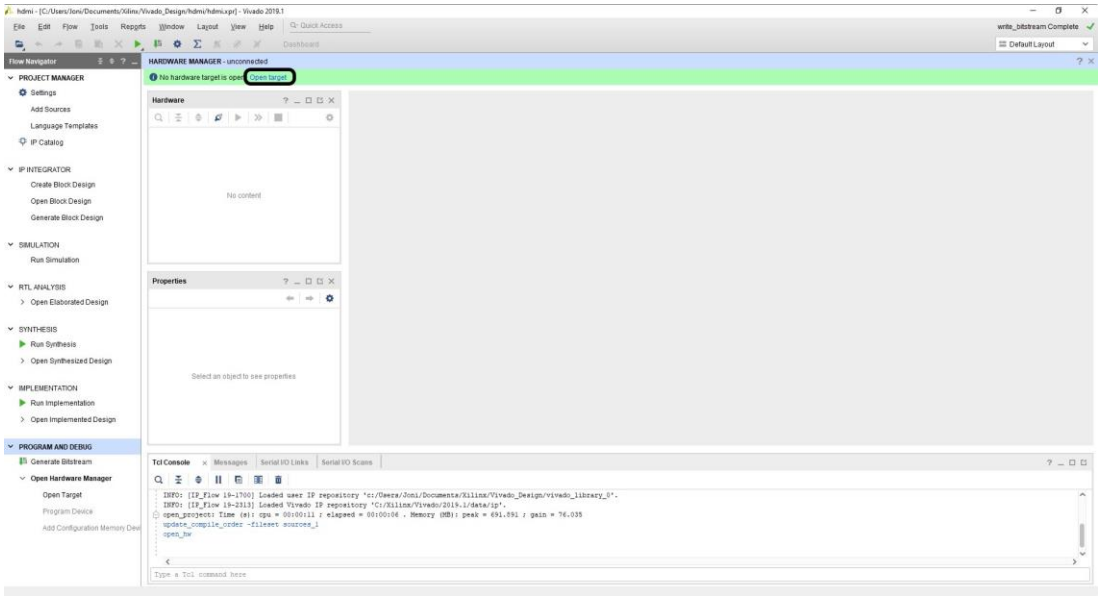


Εικόνα 5.30:Δημιουργία αρχείου Bitstream

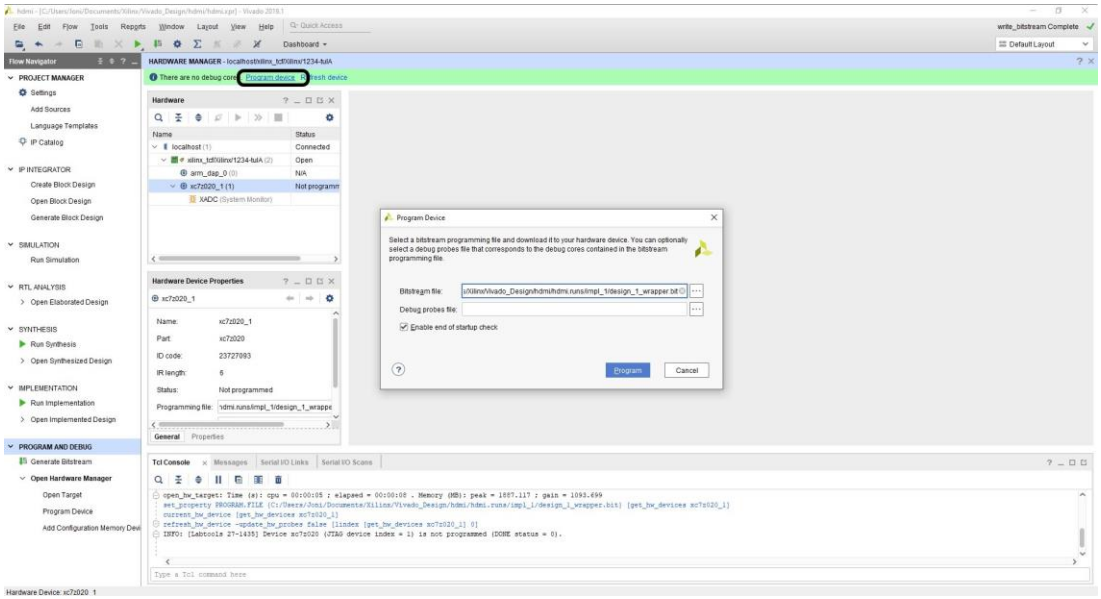


Εικόνα 5.31:Άνοιγμα του Hardware Manager

Στη συνέχεια ανοίγουμε την συσκευή και την προγραμματίζουμε (Εικόνα 5.32, 5.33).



Εικόνα 5.32:Άνοιγμα της συσκευής

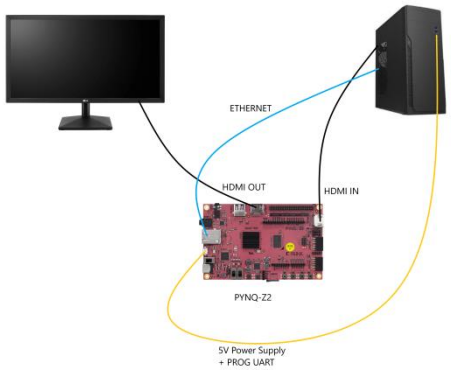


Εικόνα 5.33:Προγαμματισμός της συσκευής

Η εφαρμογή αυτή δοκιμάστηκε χρησιμοποιώντας σαν πηγή εισόδου εικόνας:

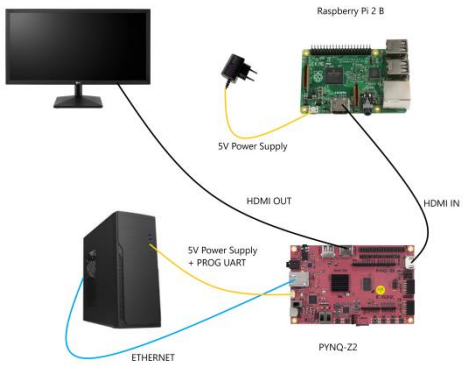
- Την έξοδο της κάρτας γραφικών του υπολογιστή
- Την έξοδο της αναπτυξιακής πλακέτας Rasberry Pi 2 B

Παρακάτω φαίνεται πώς μπορεί να συνδέσουμε σαν είσοδο στη συσκευή PYNQ - Z2 την έξοδο HDMI της κάρτας γραφικών του υπολογιστή και σαν έξοδο μία οθόνη (Εικόνα 5.34).



Εικόνα 5.34: Σύνδεση πηγής εικόνας την έξοδο της κάρτας γραφικών του υπολογιστή

Παρακάτω φαίνεται πώς μπορεί να συνδέσουμε σαν είσοδο στη συσκευή PYNQ - Z2 την έξοδο HDMI της συσκευής Raspberry Pi 2 B και σαν έξοδο μία οθόνη (Εικόνα 5.35).



Εικόνα 5.35: Σύνδεση σαν πηγή εικόνας την έξοδο της αναπτυξιακής πλακέτας Rasberry Pi 2 B

5.3 Εφαρμογή ανιχνευτή ακμών Sobel (PL)

Σκοπός αυτού του project είναι η δημιουργία ενός επιταχυντή για ανίχνευση ακμών με φίλτρο Sobel σε πραγματικό χρόνο. Είναι μία εφαρμογή που απαιτεί επεξεργασία πολλών δεδομένων εικόνας και εδώ εφαρμόζεται μόνο στο λογικό υλικό για να επιτευχθεί.

5.3.1 Τα βήματα για την δημιουργία του ανιχνευτή ακμών Sobel

Εκτός από τα τα blocks ‘DVI to RGB Video Decoder’ και ‘RGB to DVI Video Encoder’, τα οποία δεν ανήκουν στην εγκατεστημένη βιβλιοθήκη του Vivado αλλά τα κατεβάζουμε από εδώ [115],χρησιμοποιούμε και το block ανίχνευσης ακμών Sobel ‘Sobel_edge_detect’ που το κατεβάζουμε από εδώ [116]

Ακολουθούμε την παραπάνω προεργασία και παρεμβάλουμε ανάμεσα στα IP blocks ΄DVIv to RGB Video Decoder΄ και ΄RGB to DVI Video Encoder΄ τα παρακάτω:

Video In to AXI-4 Stream: χρησιμοποιείται για την μετατροπή του σήματος της μορφής RGB σε ροή AXI για το IP ανίχνευσης ακμών.

Οι ρυθμίσεις (Εικόνα 5.36)*:*

Pixels Per Clock = 1

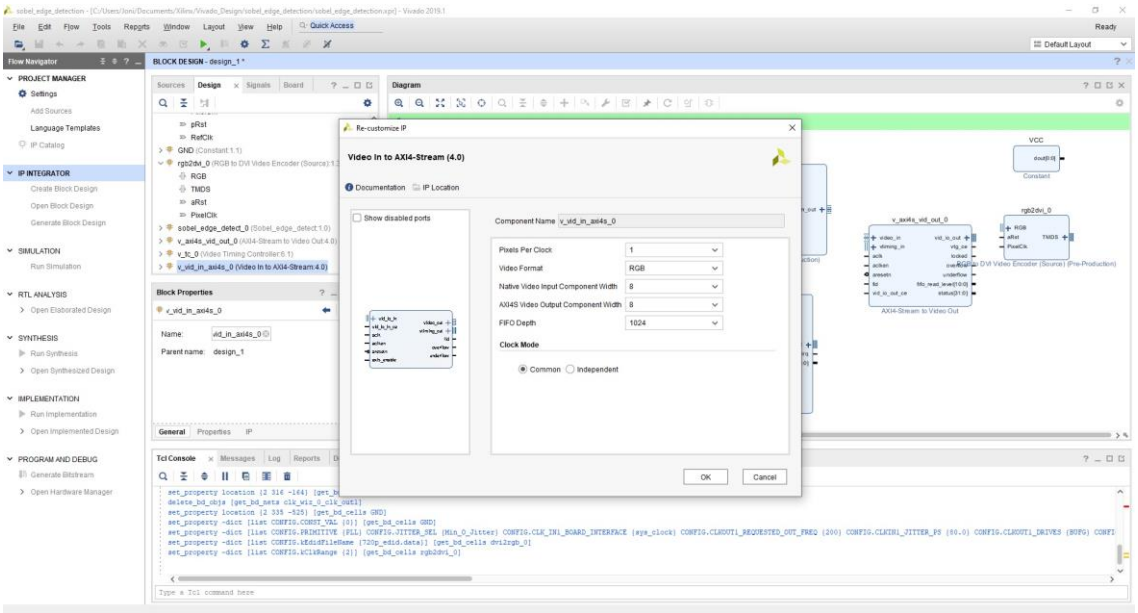
Video Format = RGB

Native Video Input Component Width = 8

AXI4S Video Output Component Width = 8

FIFO Depth = 1024

Clock Mode: επιλέγουμε Common



Εικόνα 5.36:Οι ρυθμίσεις του block Video In to AXI-4 Stream

AXI4-Stream to Video Out: χρησιμοποιείται για την μετατροπή της ροής AXI από το IP ανίχνευσης ακμών σε σήμα της μορφής RGB.

Οι ρυθμίσεις (Εικόνα 5.37)*:*

Pixels Per Clock = 1

Video Format = RGB

Native Video Input Component Width = 8

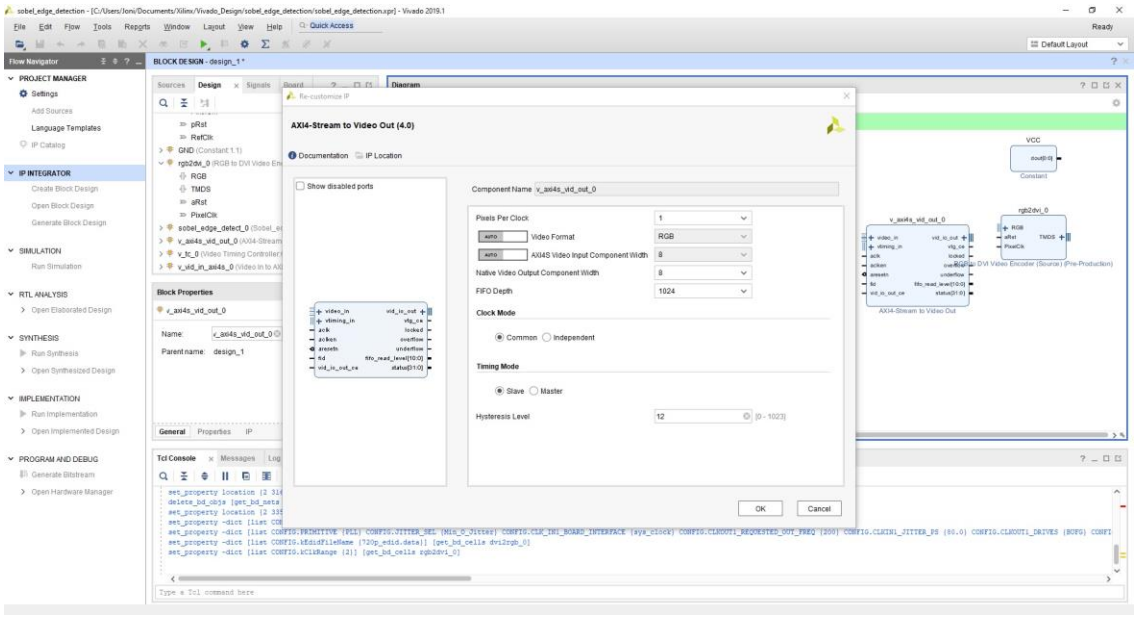
AXI4S Video Output Component Width = 8

FIFO Depth = 1024

Clock Mode: επιλέγουμε Common

Timing Mode: επιλέγουμε Slave

Hysteresis Level = 12



Εικόνα 5.37: Οι ρυθμίσεις του block AXI4-Stream to Video Out

Video Timing Controller: χρησιμοποιείται για να συγχρονίσει την ροή εισόδου με την ροή εξόδου.

Οι ρυθμίσεις:

Καρτέλα Detection/Generation (Εικόνα 5.38):

Αφαιρούμε την επιλογή **Include AXI4-Lite Interface**

Max Clocks Per Line = 4096

Max Lines Per Frame = 4096

Frame Syncs = 1

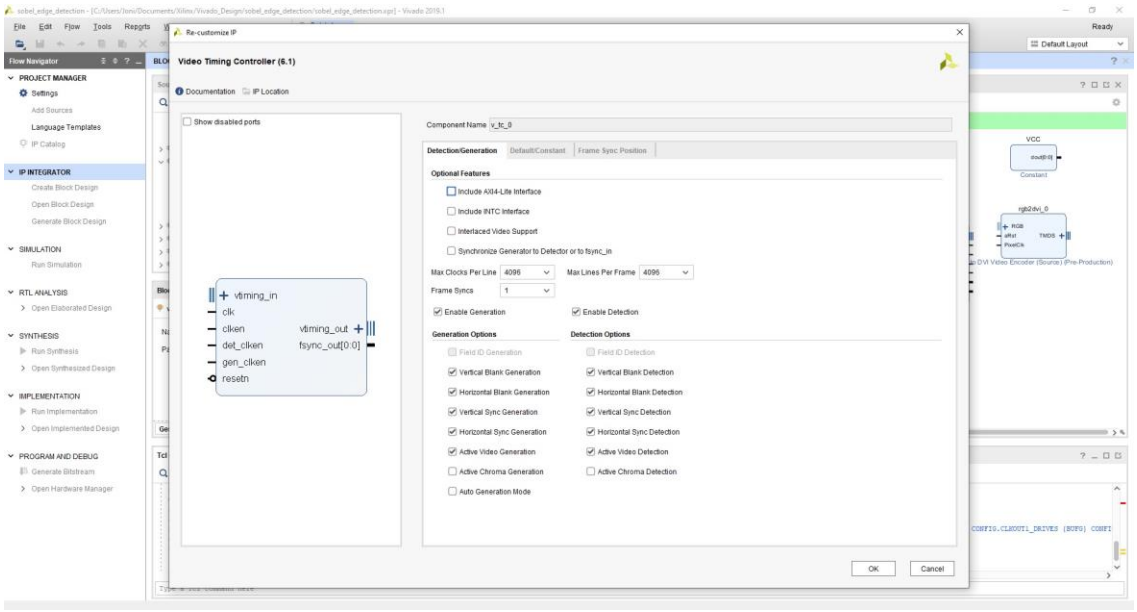
Επιλέγουμε **Enable Generation, Enable Detection**

Generation Options

Επιλέγουμε **Vertical Blank Generation, Horizontal Blank Generation, Vertical Sync Generation, Horizontal Sync Generation, Active Video Generation**

Detection Options

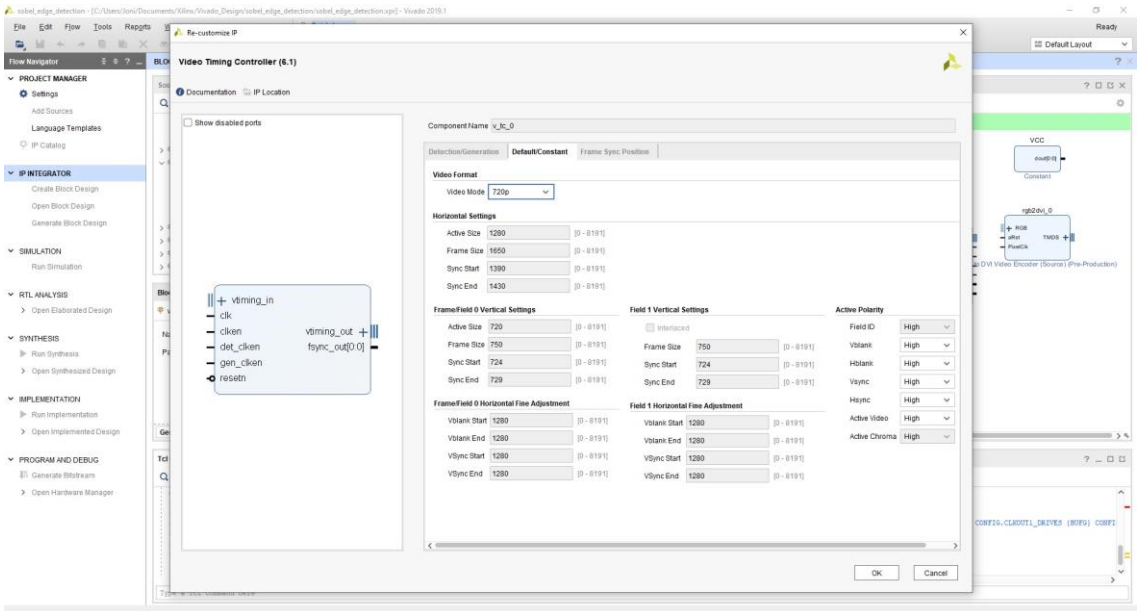
Επιλέγουμε **Vertical Blank Detection, Horizontal Blank Detection, Vertical Sync Generation, Detection Sync Detection, Active Video Detection**



Εικόνα 5.38:Οι ρυθμίσεις του block Video Timing Controller

Καρτέλα Default / Constant (Εικόνα 5.39):

Video Mode = 720p



Εικόνα 5.39:Επιλογή λειτουργίας βίντεο

Προσθέτουμε το Sobel_edge_detect IP block και κάνουμε τις ακόλουθες συνδέσεις:

Σήματα σε κατάσταση λογικού ΄0΄:

Συνδέουμε τα ΄**aRst**΄, ΄**pRst**΄ (΄DVI to RGB Video Decoder΄) και το ΄**aRst**΄ (΄RGB to DVI Video Encoder΄) με το ΄**0**΄ (block constant GND).

Σήματα σε κατάσταση λογικού ΄1΄:

Συνδέουμε τα ΄**vid_io_in_ce**΄, ΄**aclken**΄, ΄**aresetn**΄, ΄**axis_enable**΄ (΄Video In to AXI-4 Stream΄), ΄**ap_start**΄, ΄**ap_rst_n**΄ (΄Sobel_edge_detect΄), ΄**clken**΄, ΄**det_clken**΄, ΄**resetn**΄ (΄Video Timing Controller΄), ΄**aclken**΄, ΄**aresetn**΄, ΄**vid_io_out_ce**΄ (΄AXI-4 Stream to Video Out΄) με το ΄1΄ (block constant VCC).

Σήματα Clock:

Συνδέουμε τα ΄**PixelClk**΄ (΄DVI to RGB Video Decoder΄), ΄**aclk**΄ (΄Video In to AXI-4 Stream΄), ΄**ap_clk**΄ (΄Sobel_edge_detect΄), ΄**clk**΄ (΄Video Timing Controller΄), ΄**aclk**΄ (΄AXI-4 Stream to Video Out΄) με το ΄**PixelClk**΄ (΄RGB to DVI Video Encoder΄).

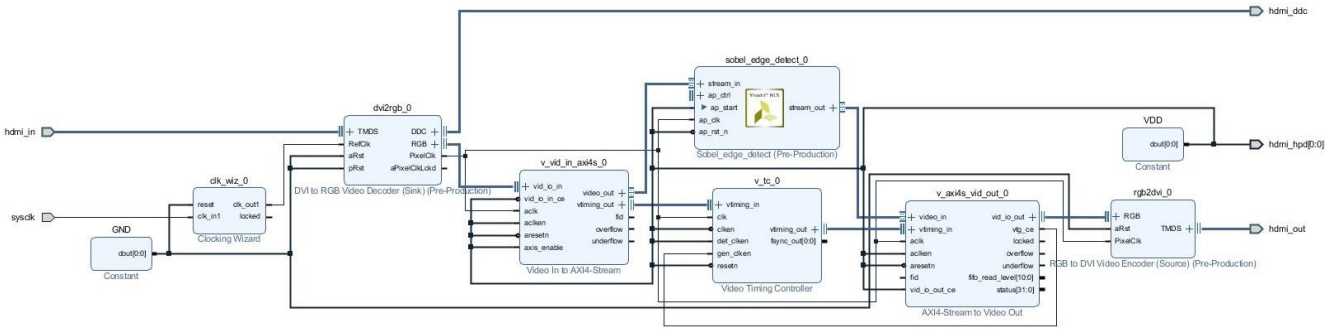
Σήματα εισόδου:

Συνδέουμε το ΄**RGB**΄ (΄DVI to RGB Video Decoder΄) με το ΄**vid_io_in**΄ (΄Video In to AXI-4 Stream ΄), το ΄**video_out**΄ (΄Video In to AXI-4 Stream΄) με το ΄**stream_in**΄ (΄Sobel_edge_detect΄), το ΄**vtiming_out**΄ (΄Video In to AXI-4 Stream΄) με το ΄**vtiming_in**΄ (΄Video Timing Controller΄).

Σήματα εξόδου:

Συνδέουμε το ΄**RGB**΄ (΄RGB to DVI Video Encoder΄) με το ΄**video_out**΄ (΄AXI-4 Stream to Video Out΄), το ΄**video_in**΄ (΄AXI-4 Stream to Video Out΄) με το ΄**stream_out**΄ (΄Sobel_edge_detect΄) και το ΄**vtiming_in**΄ (΄AXI-4 Stream to Video Out΄) με το ΄**vtiming_out**΄ (΄Video Timing Controller΄).

Το τελικό σχέδιο φαίνεται στην Εικόνα 5.40.



Εικόνα 5.40:Ολοκληρωμένη σχεδίαση εφαρμογής ανίχνευσης ακμών με φίτρο Sobel

Έπειτα, ενοποιούμε όλα τα υλικά (Create HDL Wrapper) και χρησιμοποιούμε τους φυσικούς περιορισμούς όπως στην εφαρμογή ροής HDMI (Εικόνα 5.41).

```
##HDMI Rx
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_cec }]; #IO_L13N_T2_MRCC_35 Sch=hdmi_rx_cec
set_property -dict { PACKAGE_PIN P19   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_clk_n }]; #IO_L13N_T2_MRCC_34 Sch=hdmi_rx_clk_n
set_property -dict { PACKAGE_PIN N18   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_clk_p }]; #IO_L13P_T2_MRCC_34 Sch=hdmi_rx_clk_p
set_property -dict { PACKAGE_PIN W20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_n[0] }]; #IO_L16N_T2_34 Sch=hdmi_rx_d_n[0]
set_property -dict { PACKAGE_PIN V20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_p[0] }]; #IO_L16P_T2_34 Sch=hdmi_rx_d_p[0]
set_property -dict { PACKAGE_PIN U20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_n[1] }]; #IO_L15N_T2_DQS_34 Sch=hdmi_rx_d_n[1]
set_property -dict { PACKAGE_PIN T20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_p[1] }]; #IO_L15P_T2_DQS_34 Sch=hdmi_rx_d_p[1]
set_property -dict { PACKAGE_PIN P20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_n[2] }]; #IO_L14N_T2_SRCC_34 Sch=hdmi_rx_d_n[2]
set_property -dict { PACKAGE_PIN N20   IOSTANDARD TMDS_33  } [get_ports { hdmi_in_data_p[2] }]; #IO_L14P_T2_SRCC_34 Sch=hdmi_rx_d_p[2]
set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports { hdmi_hpd }]; #IO_25_34 Sch=hdmi_rx_hpd
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { hdmi_ddc_scl_io }]; #IO_L11P_T1_SRCC_34 Sch=hdmi_rx_scl
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports { hdmi_ddc_sda_io }]; #IO_L11N_T1_SRCC_34 Sch=hdmi_rx_sda

##HDMI Tx
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_cec }]; #IO_L19N_T3_VREF_35 Sch=hdmi_tx_cec
set_property -dict { PACKAGE_PIN L17   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_clk_n }]; #IO_L11N_T1_SRCC_35 Sch=hdmi_tx_clk_n
set_property -dict { PACKAGE_PIN L16   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_clk_p }]; #IO_L11P_T1_SRCC_35 Sch=hdmi_tx_clk_p
set_property -dict { PACKAGE_PIN K18   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_n[0] }]; #IO_L12N_T1_MRCC_35 Sch=hdmi_tx_d_n[0]
set_property -dict { PACKAGE_PIN K17   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_p[0] }]; #IO_L12P_T1_MRCC_35 Sch=hdmi_tx_d_p[0]
set_property -dict { PACKAGE_PIN J19   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_n[1] }]; #IO_L10N_T1_AD11N_35 Sch=hdmi_tx_d_n[1]
set_property -dict { PACKAGE_PIN K19   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_p[1] }]; #IO_L10P_T1_AD11P_35 Sch=hdmi_tx_d_p[1]
set_property -dict { PACKAGE_PIN H18   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_n[2] }]; #IO_L14N_T2_AD4N_SRCC_35 Sch=hdmi_tx_d_n[2]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD TMDS_33  } [get_ports { hdmi_out_data_p[2] }]; #IO_L14P_T2_AD4P_SRCC_35 Sch=hdmi_tx_d_p[2]
#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports { hdmi_tx_hpdn }]; #IO_0_34 Sch=hdmi_tx_hpdn

create_clock -period 13.4 -name hdmi_in_clk_p -waveform {0.000 6.7} [get_ports hdmi_in_clk_p]


```

Εικόνα 5.41:Οι φυσικοί περιορισμοί

Ακολουθούμε την διαδικασία για την δημιουργίας του αρχείου bitstream και στη συνέχεια το μεταφέρουμε στο λογικό υλικό της πλακέτας.

Σε αυτήν την εφαρμογή χρησιμοποιώ σαν είσοδο εικόνας (Εικόνα 5.42) την έξοδο της κάρτας γραφικών του υπολογιστή και το αποτέλεσμα της ανίχνευσης ακμών προβάλλεται σε μία οθόνη (Εικόνα 5.43) σε πραγματικό χρόνο.

Εικόνα 5.42:Εικόνα εισόδου πριν την επεξεργασία

Εικόνα 5.43:Εικόνα εξόδου μετά την ανίχνευση ακμών

6 Συμπεράσματα

Ο σκοπός αυτής της εργασίας ήταν η επιτάχυνση εφαρμογών ρομποτικής όρασης με χρήση της ετερογενούς διάταξης του τσιπ Zynq. Επίσης, παρουσιάστηκε η πλακέτα Pynq - Z2 με τα διάφορα στοιχεία που την απαρτίζουν, ο τρόπος που βάζουμε σε λειτουργία την πλακέτα, η αλληλεπίδραση με τα μπουτόν και τους διακόπτες καθώς και ο τρόπος επικοινωνίας με τον υπολογιστή.

Ακόμη, εξηγήθηκαν τα βασικά εργαλεία της Xilinx με τα οποία μπορούν να υλοποιηθούν ποικίλες εφαρμογές. Επιπλέον, για την καλύτερη κατανόηση της χρήσης των FPGAs και συγκεκριμένα της πλακέτας Pynq - Z2, παρουσιάστηκαν κάποιες γενικές εφαρμογές δίνοντας εκτενώς τα βήματα που χρειάστηκαν να εκτελεστούν για την υλοποίηση τους.

Πιο συγκεκριμένα:

Στην εφαρμογή του μη προσημασμένου αθροιστή, φαίνεται πώς μπορούμε να συνδέσουμε την πλακέτα μόνο για την χρήση του λογικού υλικού του Zynq και πώς φτιάχνουμε ένα υλικό για την εκτέλεση μίας λειτουργίας, δίνοντας την περιγραφή του σε γλώσσα VHDL. Στη συνέχεια, γίνεται αντιληπτός ο τρόπος που προσθέτουμε τους φυσικούς περιορισμούς της πλακέτας και τέλος ο τρόπος παραγωγής του αρχείου bitstream (γίνεται η σύνθεση του σχεδίου και ύστερα η ανάλυση του) που απαιτείται για την τοποθέτηση του σχεδίου στο λογικό υλικό.

Στην εφαρμογή της αριθμομηχανής δεν χρησιμοποιούμε κάποιο εργαλείο αφού δεν χρησιμοποιούμε το λογικό υλικό για να σχεδιάσουμε κάτι. Σκοπός είναι να δούμε την λειτουργία του επεξεργαστή που δεν είναι διαφορετική από αυτήν ενός υπολογιστή. Έτσι λοιπόν, αποκτούμε πρόσβαση στο Linux Secure Shell ή χρησιμοποιούμε την διεπαγή του Jupyter Notebook για να δημιουργήσουμε ένα αρχείο σε γλώσσα Python που εκτελεί την λειτουργία μιας απλής αριθμομηχανής.

Για να δούμε πώς συνεργάζεται ο επεξεργαστής ARM με το λογικό υλικό δημιουργούμε μία εφαρμογή αριθμητικής και λογικής μονάδας όπου ελέγχουμε την λειτουργία της μέσω του επεξεργαστή. Για να γίνει αυτό, αρχικά δημιουργούμε μέσω του Vivado Design Tool ένα IP που εκτελεί αριθμητικές και λογικές πράξεις και χρησιμοποιεί το πρότυπο AXI4 - Lite για την επικοινωνία με τον ARM, αναφέροντας όλα τα βήματα που απαιτούνται. Το υλικό μας χρησιμοποιεί 4 καταχωρητές των 32bit μέσω των οποίων θα αλληλεπιδρά ο επεξεργαστής κάνοντας εγγραφή και ανάγνωση δεδομένων. Ύστερα, παρουσιάζεται ο τρόπος που δημιουργούμε ένα σχέδιο, προσθέτωντας το IP που φτιάξαμε και τον ARM, όλα τα υπόλοιπα που υπάρχουν στο σχέδιο δημιουργούνται αυτόματα από το Vivado. Επίσης, παρουσιάζεται το πλεονέκτημα του περιβάλλοντος Pynq, δηλαδή η χρήση της βιβλιοθήκης Overlay όπου χρειάζεται το αρχείο bitstream και block design, που

δημιουργήθηκαν από την σύνθεση και ανάλυση του σχεδίου, για να τοποθετηθεί το σχέδιο στο λογικό υλικό.

Όπως καταλαβαίνουμε, αυτή η εφαρμογή κάνει χρήση όλων των δυνατοτήτων του Zynq και του Pynq, καθιστώντας την ιδανική για την κατανόηση του συστήματος που παρουσιάζεται στην εργασία.

Σε επόμενο βήμα, δημιουργούμε μία πολύ χρήσιμη εφαρμογή αφού αποτελεί την βάση για σχεδίαση εφαρμογών επεξεργασίας ροής εικόνας. Έτσι, φτιάχνουμε ένα απλό σύστημα που δέχεται μία ροή δεδομένων εικόνας από θύρα HDMI και την προβάλλει σε έξοδο αντίστοιχης θύρας. Χρησιμοποιούμε blocks για την μετατροπή δεδομένων μορφής RGB σε DVI και το αντίστροφο και block για να δημιουργήσουμε σταθερό παλμό χρονισμού, μεγαλύτερο από αυτό που προσφέρει το σύστημα μας, σαν ρολόι αναφοράς του block ΄DVI to RGB Video Decoder΄. Επίσης, παρουσιάζεται ο τρόπος δημιουργίας παλμού ρολογιού για τον χρονισμό των pixels.

Αφού λοιπόν φτιάξαμε την βάση, οδηγούμαστε στην δημιουργία της εφαρμογής ανίχνευσης ακμών Sobel σε πραγματικό χρόνο για να κατανοήσουμε το μέγεθος επιτάχυνσης που μπορεί να προσφέρει αυτό το σύστημα FPGA, χωρίς όμως την χρήση του επεξεργαστή.

Χρησιμοποιούμε ένα έτοιμο IP που εκτελεί ανίχνευση ακμών, κατασκευασμένο από το Vivado HLS, και από την στιγμή που επεξεργάζεται ροή δεδομένων κάνει χρήση του προτύπου AXI4 stream. Αυτός είναι και ο λόγος που προσθέτουμε ακόμα 3 IP, για την μετατροπή των δεδομένων βίντεο σε ροή δεδομένων και αντίστροφα και τον συγχρονισμό αυτών.

Παρατηρούμε ότι η εφαρμογή του ανιχνευτή ακμών Sobel απαιτεί επεξεργασία πολλών δεδομένων σε πραγματικό χρόνο και όμως επιτυγχάνεται χωρίς καμία καθυστέρηση από την στιγμή που υπάρχει hardware για να επεξεργαστεί το βίντεο.

6.1 Μελλοντική επέκταση

Παρόλο που η εφαρμογή ανίχνευσης ακμών Sobel αποτελεί μία ολοκληρωμένη λύση, επιδέχεται βελτίωση.

Έτσι, θα μπορούσε να χρησιμοποιηθεί μία κάμερα για την λήψη του βίντεο μιας και χρησιμοποιούμε την έξοδο της κάρτας γραφικών του υπολογιστή.

Όπως αναφέραμε, με την ανίχνευση ακμών μπορούμε να εξάγουμε σημαντικές πληροφορίες για ένα χώρο. Έτσι λοιπόν, η εφαρμογή μπορεί να χρησιμοποιηθεί για τον έλεγχο κίνησης αυτόνομων, κινούμενων και μη,οχημάτων ή ρομπότ στον χώρο, σε συνδυασμό με την προσθήκη αλγορίθμων αποφυγής εμποδίων.

Επίσης, μπορεί να αντικατασταθεί το IP που ανιχνεύει τις ακμές, με άλλο block που εκτελεί διαφορετικές λειτουργίες όπως ανίχνευση οχημάτων, αντικειμένων, πινακίδων και φαναριών για χρήση σε έξυπνα αυτοκίνητα (ADAS: Advanced Driver Assistance Systems).

Μία σημαντική βελτίωση αποτελεί η προσθήκη του επεξεργαστή Zynq για να έχουμε έλεγχο της εφαρμογής μας όπως κάναμε στην εφαρμογή της αριθμητικής και λογικής μονάδας, αξιοποιώντας έτσι το πλεονέκτημα του περιβάλλοντος Pynq, δηλαδή της βιβλιοθήκης Overlay και της γλώσσας Python.

Βιβλιογραφικές Πηγές

[1] L. H. Crockett, R. A. Elliot, M. A. Enderwitz and R. W. Stewart, *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*, First Edition, Strathclyde Academic Media, 2014.
http://www.zynqbook.com

[2] Digital Image Processing / Rafael C. Gonzalez, University of Tennessee, RichardE. Woods, Interapptics.

[3] Volnei A. Pedroni - Circuit Design and Simulation with VHDL-The MIT Press (2010)

[4] PYNQ website
http://www.pynq.io/home.html

[5] PYNQ Documentation
https://pynq.readthedocs.io/en/v2.5.1/

[6] PYNQ Overlays
https://pynq.readthedocs.io/en/v2.5.1/pynq_overlays.html

[7] Digilent website.
http://www.digilentinc.com

[8] Xilinx website
https://www.xilinx.com

[9] Xilinx, Inc., "AXI Reference Guide", UG761, v14.3, November 2012.
http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf

[10] ARM website
https://www.arm.com/

[11] ARM Cortex -A9 Technical Reference Manual
https://developer.arm.com/documentation/100511/0401/?search=5eec6e5fe24a5e02d07b257c

[12] ARM Cortex-A9 MPCore Technical Reference Manual
https://developer.arm.com/documentation/100486/0401/?search=5eec6e5fe24a5e02d07b257c

[13] Cortex-A9 NEON Media Processing Engine Technical Reference Manual
https://developer.arm.com/documentation/ddi0409/i/?search=5eec6e5fe24a5e02d07b257c

[14] Cortex-A9 Floating-Point Unit Technical Reference Manual
https://developer.arm.com/documentation/ddi0408/i/?search=5eec6e5fe24a5e02d07b257c

[15] Arm Cortex-A53 MPCore Processor Technical Reference Manual
https://developer.arm.com/documentation/ddi0500/j/?search=5eec6e55e24a5e02d07b256e

[16] Cortex-R5 Technical Reference Manual
https://developer.arm.com/documentation/ddi0460/d/?search=5eec6e65e24a5e02d07b258d

[17] Cortex-M1 Technical Reference Manual
https://developer.arm.com/documentation/ddi0413/d/?search=5eec6e71e24a5e02d07b259e
[18] ARM Cortex -M3 Processor Technical Reference Manual
https://developer.arm.com/documentation/100165/0201/?search=5eec6e71e24a5e02d07b259c
[19] ARM Mali-400 GPU
https://developer.arm.com/ip-products/graphics-and-multimedia/mali-gpus/mali-400-gpu
[20] AMBA AXI and ACE Protocol Specification
https://developer.arm.com/documentation/ihi0022/latest
[21] RISC-V website
https://riscv.org/
[22] RISC-V Exchange: Cores & SoCs
https://riscv.org/exchange/cores-socs/
[23] FPGA History
https://hardwarebee.com/field-programmable-gate-array-fpga-history-applications/
[24] FPGAs Companies
https://hardwarebee.com/list-fpga-companies/
[25] MicroBlaze Soft Processor Core
https://www.xilinx.com/products/design-tools/microblaze.html
[26] MicroBlaze Processor Reference Guide
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug984-vivado-microblaze-ref.pdf
[27] PicoBlaze 8-bit Embedded Microcontroller User Guide
https://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf
[28] Spartan-7 Product Advantage
https://www.xilinx.com/products/silicon-devices/fpga/spartan-7.html#productAdvantages
[29] Artix-7 Product Advantage
https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html
[30] Virtex-7 FPGAs Product Advantage
https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html
[31] Kintex UltraScale Product Advantage
https://www.xilinx.com/products/silicon-devices/fpga/kintex-ultrascale.html
[32] Zynq-7000 SoC Product Advantages
https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html
[33] Zynq UltraScale+ MPSoC Product Advantages
https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[34] Vivado Design Suite
https://www.xilinx.com/products/design-tools/vivado.html
[35] Vivado High-Level Synthesis
https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html
[36] Vivado System Generator for DSP
https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html
[37] Vivado Model Composer
https://www.xilinx.com/products/design-tools/vivado/integration/model-composer.html
[38] Arty S7-50 board
https://www.xilinx.com/products/boards-and-kits/1-pnziih.html
[39] Xilinx Virtex-7 FPGA VC707 Evaluation Kit
https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html
[40] Zybo Z7-10: Zynq-7000 ARM/FPGA SoC Development Board
https://www.xilinx.com/products/boards-and-kits/1-pukimv.html
[41] PYNQ-Z2 board
http://www.tul.com.tw/ProductsPYNQ-Z2.html
[42] Python Overview
https://www.tutorialspoint.com/python/python_overview.htm
[43] Python website
https://www.python.org/
[44] Intel FPGAs & Programmable Devices website
https://www.intel.com/content/www/us/en/products/programmable.html
[45] Intel Max 10 FPGAs
https://www.intel.com/content/www/us/en/products/programmable/fpga/max-10.html
[46] Nios II Processor Reference Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu-nii5v1gen2.pdf
[47] Intel Cyclone 10 FPGAs
https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-10.html
[48] Intel Arria 10 FPGAs
https://www.intel.com/content/www/us/en/products/programmable/arria-series.html
[49] Intel Stratix FPGAs
https://www.intel.com/content/www/us/en/products/programmable/stratix-series.html
[50] Intel Agilex FPGAs & SoCs
https://www.intel.com/content/www/us/en/products/programmable/fpga/agilex.html

[51] Intel Cyclone V SoC FPGAs
https://www.intel.com/content/www/us/en/products/programmable/soc/cyclone-v.html
[52] Intel Arria V SoC FPGAs
https://www.intel.com/content/www/us/en/products/programmable/soc/arria-v.html
[53] Intel Arria 10 SoC FPGAs
https://www.intel.com/content/www/us/en/products/programmable/soc/arria-10.html
[54] Intel Stratix 10 SoC FPGAs
https://www.intel.com/content/www/us/en/products/programmable/soc/stratix-10.html
[55] Intel Quartus Prime Software Suite
https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html
[56] Intel High Level Synthesis Compiler
https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html
[57] Nios II Embedded Design Suite
https://www.intel.com/content/www/us/en/products/programmable/processor/nios-ii/design-tools.html
[58] DSP Builder for Intel FPGAs
https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/dsp-builder.html
[59] Intel Cyclone 10 LP FPGA Evaluation Kit
https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/cyclone-10-lp-evaluation-kit.html
[60] Intel Cyclone V DE1 SoC Board
https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/cyclone-v-se-device-family---de1-soc-board.html
[61] Intel Stratix 10 SX SoC Development Kit
https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/stratix-10-soc-development-kit.html
[62] Lattice Semiconductor website
http://www.latticesemi.com/
[63] iCE40 LP/HX/LM
https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40
[64] iCE40 Ultra / UltraLite
https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40Ultra
[65] iCE40 UltraPlus

https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus

[66] ECP5 / ECP5-5G
https://www.latticesemi.com/en/Products/FPGAandCPLD/ECP5

[67] MachXO3D
https://www.latticesemi.com/Products/FPGAandCPLD/MachXO3D

[68] Lattice Diamond Software
https://www.latticesemi.com/en/Products/DesignSoftwareAndIP/FPGAandLDS/LatticeDiamond

[69] iCEcube2 Design Software
https://www.latticesemi.com/Products/DesignSoftwareAndIP/FPGAandLDS/iCEcube2

[70] Lattice Radiant Software
https://www.latticesemi.com/Products/DesignSoftwareAndIP/FPGAandLDS/Radiant

[71] Neural Network Compiler
https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler

[72] LatticeMico System Development Tools
https://www.latticesemi.com/en/Products/DesignSoftwareAndIP/EmbeddedDesignSoftware/LatticeMicoSystem

[73] Lattice Mico8 Soft Microcontroller
http://www.latticesemi.com/Products/DesignSoftwareAndIP/IntellectualProperty/IPCore/IPCores02/Mico8.aspx

[74] LatticeMico32 Soft Processor
http://www.latticesemi.com/en/Products/DesignSoftwareAndIP/IntellectualProperty/IPCore/IPCores02/LatticeMico32.aspx

[75] ECP5 Versa Development Kit
https://www.latticesemi.com/products/developmentboardsandkits/ecp5versadevelopmentkit

[76] Serializer/Deserializer (SerDes)
https://semiengineering.com/knowledge_centers/communications-io/off-chip-communications/i-o-enabling-technology/serializer-deserializer-serdes/

[77] MachXO2 Control Development Kit
https://www.latticesemi.com/products/developmentboardsandkits/machxo2controldevkit

[78] iCE40 UltraLite Breakout Board
https://www.latticesemi.com/products/developmentboardsandkits/ice40ultralitebreakoutboard

[79] Microsemi website
https://www.microsemi.com/

[80] PolarFire FPGAs
https://www.microsemi.com/product-directory/fpgas/3854-polarfire-fpgas#overview

[81] IGLOO2 FPGAs
https://www.microsemi.com/product-directory/fpgas/1688-igloo2
[82] IGLOO FPGAs
https://www.microsemi.com/product-directory/fpgas/1689-igloo
[83] ProASIC3 FPGAs
https://www.microsemi.com/product-directory/fpgas/1690-proasic3
[84] PolarFire SoC
https://www.microsemi.com/product-directory/soc-fpgas/5498-polarfire-soc-fpga
[85] SmartFusion2 SoC
https://www.microsemi.com/product-directory/soc-fpgas/1692-smartfusion2
[86] Libero SoC v12.0 and later
https://www.microsemi.com/product-directory/design-resources/1750-libero-soc
[87] Libero SoC v11.9 and earlier
https://www.microsemi.com/product-directory/libero-soc/5507-libero-soc-v11-9-archive
[88] Libero IDE
https://www.microsemi.com/product-directory/design-resources/1751-libero-ide
[89] DSP Design Tools
https://www.microsemi.com/product-directory/dev-tools/5540-dsp-design-tools
[90] Soft CPUs
https://www.microsemi.com/product-directory/mi-v-ecosystem/5093-other-cpus
[91] ARM Cortex-M1 Processor
https://www.microsemi.com/product-directory/mi-v-ecosystem/5068-arm-cortex-m1-processor
[92] Mi-V RISC-V Ecosystem
https://www.microsemi.com/product-directory/fpga-soc/5210-mi-v-embedded-ecosystem
[93] PolarFire FPGA Video kit
https://www.microsemi.com/existing-parts/parts/150804
[94] SmartFusion2 Advanced Development Kit
https://www.microsemi.com/existing-parts/parts/143965
[95] IGLOO Nano AGLN250
https://www.microsemi.com/existing-parts/parts/144014
[96] Xilinx Vivado Design Suite 2019.1.
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-1.html
[97] Vivado Design Suite User Guide Getting Started

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug910-vivado-getting-started.pdf

[98] Vivado Design Suite User Guide Using the Vivado IDE 2019.1
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug893-vivado-ide.pdf

[99] Vivado Design Suite User Guide Design Flows Overview 2019.1
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug892-vivado-design-flows-overview.pdf

[100] Vivado Design Suite User Guide High-Level Synthesis
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug902-vivado-high-level-synthesis.pdf

[101] Vivado Design Suite User Guide Embedded Processor Hardware Design
https://china.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2019_1/ug898-vivado-embedded-design.pdf

[102] Vivado Design Suite User Guide Hierarchical Design
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug905-vivado-hierarchical-design.pdf

[103] Vivado Design Suite User Guide System-Level Design Entry
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug895-vivado-system-level-design-entry.pdf

[104] Vivado Design Suite User Guide Using Constraints
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug903-vivado-using-constraints.pdf

[105] Vivado Design Suite User Guide Designing with IP
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug896-vivado-ip.pdf

[106] Vivado Design Suite User Guide Designing IP Subsystems Using IP Integrator
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug994-vivado-ip-subsystems.pdf

[107] Vivado Design Suite User Guide Synthesis
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug901-vivado-synthesis.pdf

[108] Vivado Design Suite User Guide Implementation
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug904-vivado-implementation.pdf

[109] Vivado Design Suite User Guide Creating and Packaging Custom IP

https://china.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1119-vivado-creating-packaging-ip-tutorial.pdf

[110] Video Connectivity Using TMDS I/O in Spartan-3A FPGAs
https://www.xilinx.com/support/documentation/application_notes/xapp460.pdf

[111] 7 Series FPGAs Clocking Resources User Guide
https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

[112] Digital Visual Interface DVI
http://www.cs.unc.edu/Research/stc/FAQs/Video/dvi_spec-V1_0.pdf

[113] Digilent RGB to DVI User Guide
https://github.com/Digilent/vivado-library/blob/master/ip/rgb2dvi/docs/rgb2dvi.pdf

[114] DIgilent DVI to RGB User Guide
https://github.com/Digilent/vivado-library/blob/master/ip/dvi2rgb/docs/dvi2rgb.pdf

[115] Digilent Vivado IP Library
https://github.com/Digilent/vivado-library

[116] Adam Taylor's Sobel IP
https://github.com/ATaylorCEngFIET/Hackster

Λίστα όρων

A

ACP: Accelerator Coherency Port

ALU: Arithmetic and Logic Unit

AMBA: Advanced Microcontroller Bus Architecture

APU: Application Processing Unit

AXI: Advanced eXtensive Interface

C

CLB: Configurable Logic Block

D

DDR: Double Data Rate

DSP: Digital Signal Processing / Digital Signal Processor

DVI: Digital Visual Interface

F

FF: Flip Flop

FFT: Fast Fourier Transform

FIFO: First In First Out

FIR: Finite Impulse Response

FPGA: Field Programmable Gate Array

FPU: Floating Point Unit

G

GP: General Purpose

H

HDMI: High Definition Multimedia Interface

HLS: High Level Synthesis

HP: High Performance

I

IOB: Input / Output Block

IP: Intellectual Property

J

JTAG: Joint Test Action Group

L

LED: Light Emitting Diode

LUT: Lookup Table

M

MMU: Memory Management Unit

MPE: Media Processing Engine

O

OCM: On-Chip Memory

P

PL: Programmable Logic

Pmod: Peripheral module

PS: Proccessing System

R

RAM: Random Access Memory

ROM: Read Only Memory

RGB: Red Green Blue

RTL: Register Transfer Level

S

SCU: Snoop Control Unit

SD: Secure Digital

SIMD: Single Instruction Multiple Data

U

USB: Universal Serial Bus

V

VHDL: Very High Speed Integrated CircuitHardware Description Language

X

XDC: Xilinx Design Constraints