
**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΜΙΚΡΟΗΛΕΚΤΡΟΝΙΚΗ**


**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

# Συσχεδίαση υλικού-λογισμικού και υλοποίηση σε Xilinx Zedboard ενσωματωμένου συστήματος με βάση το πρωτόκολλο επικοινωνίας AXI4


ΑΛΕΞΑΝΔΡΟΣ Ν. ΠΑΝΤΕΛΟΥΚΑΣ

Επιβλέπων: ΑΝΤΩΝΗΣ ΠΑΣΧΑΛΗΣ, ΚΑΘΗΓΗΤΗΣ



ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2017

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

# Συσχεδίαση υλικού-λογισμικού και υλοποίηση σε Xilinx Zedboard ενσωματωμένου συστήματος με βάση το πρωτόκολλο επικοινωνίας AXI4

ΑΛΕΞΑΝΔΡΟΣ Ν. ΠΑΝΤΕΛΟΥΚΑΣ
Α.Μ.:ΜΜ260

ΕΠΙΒΛΕΠΩΝ: ΑΝΤΩΝΗΣ ΠΑΣΧΑΛΗΣ, Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

ΣΕΠΤΕΜΒΡΙΟΣ 2017

# ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια η σχεδίαση ενσωματωμένων συστημάτων έχει εξελιχθεί στον βαθμό που η επιμέρους σχεδίαση των δύο κυρίων συστατικών ενός συστήματος (υλικό και λογισμικό) γίνεται παράλληλα. Οι σχεδιαστικές επιλογές αναφέρονται όχι μόνο στην σχεδίαση ενός αποδοτικού και εύρωστου συστήματος αλλά και στο ποιες επιμέρους εργασίες θα εκτελέσει το υλικό και ποιες το λογισμικό.
Σκοπός αυτής της εργασίας είναι η σχεδίαση ενός ενσωματωμένου συστήματος που θα αποτελείται από προγραμματιζόμενη λογική (υλοποιημένη σε FPGA) και από την κύρια επεξεργαστική μονάδα (CPU). Η πλακέτα Zedboard της Xilinx προσφέρει την προγραμματιζόμενη λογική και την επεξεργαστική μονάδα μέσα στο ίδιο ολοκληρωμένο κύκλωμα με αποτέλεσμα την υψηλής απόδοσης επικοινωνία μεταξύ υλικού-λογισμικού.
Στην παρούσα εργασία η επικοινωνία μεταξύ υλικού-λογισμικού πραγματοποιήθηκε με βάση το πρωτόκολλο AXI4 [1]. Αποτελεί πρωτόκολλο τέταρτης γενιάς για επικοινωνία ARM επεξεργαστών με περιφερειακή λογική που περιλαμβάνεται σε ένα σύστημα.
Το εργαλείο της Xilinx Vivado (το οποίο είναι το κύριο εργαλείο που χρησιμοποιήθηκε στην εργασία) απλουστεύει και επιταχύνει την διαδικασία της σχεδίασης ενσωματωμένων συστημάτων λόγω της ευρείας προσφοράς σε προσχεδιασμένα IP-blocks (intellectual property) τα οποία επικοινωνούν με βάση το πρωτόκολλο AXI4.
Τα κύρια βήματα που ακολουθήθηκαν για την σχεδίαση και υλοποίηση του ενσωματωμένου συστήματος είναι τα εξής: α) Σχεδίαση απλής λογικής επεξεργασίας δεδομένων σε γλώσσα VHDL, β) Σχεδίαση ενσωματωμένου συστήματος με χρήση IP-blocks από το εργαλείο Vivado, γ) Επεξεργασία/σχεδίαση των AXI4 διεπαφών της προγραμματιζόμενης λογικής με την επεξεργαστική μονάδα, δ) Ανάπτυξη απλού προγράμματος σε γλώσσα προγραμματισμού C, το οποίο εκτελείται από την επεξεργαστική μονάδα, ε) ολοκλήρωση, υλοποίηση και επαλήθευση λειτουργίας του συστήματος σε πλακέτα Xilinx Zedboard.
Τα αποτελέσματα της εργασίας παρουσιάζουν εξαιρετικό ενδιαφέρον καθώς η συσχεδίαση υλικού-λογισμικού είναι ιδιαίτερα απλουστευμένη. Το σύστημα που σχεδιάστηκε δίνει τη δυνατότητα στον εκάστοτε σχεδιαστή υλικού ή λογισμικού να ενσωματώσει την λογική του (HDL κώδικα / C πρόγραμμα) και να παράγει γρήγορα και αποδοτικά ένα ολοκληρωμένο σύστημα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σχεδίαση Ολοκληρωμένων Συστημάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Σύστημα-σε-Ψηφίδα, πρωτόκολλο επικοινωνίας, συσχεδίαση υλικού-λογισμικού, υλοποίηση σε FPGA

# ABSTRACT

Embedded systems design has evolved to the extent on which hardware design and software development happen almost simultaneously and affect one another. Design decisions aim not only at the development of a high-performance and robust system, but also on the choice of processes that will be executed on hardware and processes that will be executed on software.

The purpose of the thesis is the design of an embedded system that comprises the programmable logic (FPGA implementation) and the central processing unit (CPU). The Zedboard of Xilinx offers the programmable logic and the processing unit on the same chip, thus enabling the design of a high-performance communication between hardware and software.

The hardware-software communication is implemented based on the AXI4 protocol [1]. AXI4 is a fourth generation interface on-chip protocol for ARM CPU cores that are desired to communicate with peripherals.

Xilinx's Vivado tool (which is the main tool used for the embedded system development) simplifies and accelerates the design process due to the wide variety of offered IP-blocks (intellectual property) which communicate with AXI4 interfaces.

The embedded system was designed as follows: a) Simple data processing logic design on VHDL, b) Embedded system design using IP-blocks offered by Vivado, c) Edit/design of AXI4 interfaces of the programmable logic with the CPU, d) Simple C-code development, executed by the CPU, e) integration, implementation and validation of the system on the Xilinx's Zedboard.

The results of the thesis look promising due to the fact that the hardware-software co-design is extremely simple. Designers can easily integrate their work (HDL code / C program) to the system and develop a fully functional embedded system.



SUBJECT AREA: Embedded System Design

KEYWORDS: System-on-Chip, interface protocol, hardware-software co-design, FPGA implementation

# ΠΕΡΙΕΧΟΜΕΝΑ

# ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

# ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

# ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

## ΠΡΟΛΟΓΟΣ

Η εργασία πραγματοποιήθηκε στα πλαίσια διπλωματικής εργασίας του διατμηματικού προγράμματος μεταπτυχιακών σπουδών στη Μικροηλεκτρονική, το οποίο λαμβάνει χώρα στο τμήμα πληροφορικής και τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών (ΕΚΠΑ).

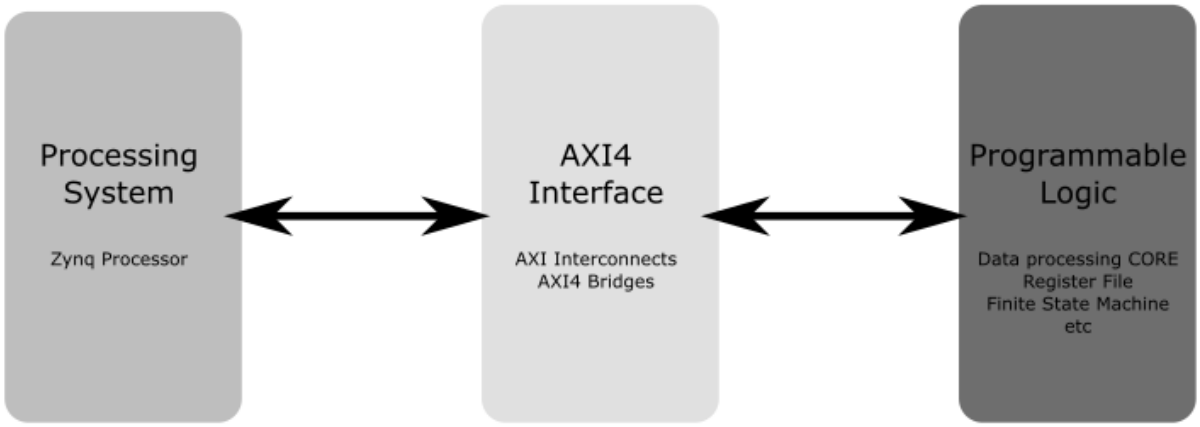
# 1. ΕΙΣΑΓΩΓΗ

Με την εξέλιξη της τεχνολογίας κατασκευής ολοκληρωμένων κυκλωμάτων οι σχεδιαστές έχουν πλέον στην διάθεση τους εξαιρετικές δυνατότητες όσον αφορά στην ολοκλήρωση (integration), την απόδοση (performance), την αξιοπιστία (reliability), την κατανάλωση ισχύος (power consumption) κτλ. Με την μείωση της κλίμακας μεγέθους των τρανζίστορ (και κατ' επέκταση την αύξηση του πλήθους τους) μέσα σε ένα ολοκληρωμένο κύκλωμα, οι σχεδιαστές είναι σε θέση να υλοποιούν ένα ολοκληρωμένο σύστημα-σε-ψηφίδα (system-on-chip) με τις δυνατότητες και τις προδιαγραφές που ορίζει η εκάστοτε έρευνα ή εργασία (project).

Ένα ενσωματωμένο σύστημα αποτελείται από υλικό (hardware) και λογισμικό (software). Η συνεργασία των δύο μερών είναι αναγκαία όσο οι στόχοι και οι απαιτήσεις του συστήματος ανεβαίνουν. Οι σχεδιαστές στοχεύουν, ανάλογα με την φύση και το σκοπό του συστήματος, στην βέλτιστη υλοποίηση, είτε αυτό σημαίνει υψηλότατη απόδοση πχ για ψηφιακή επεξεργασία σήματος σε πραγματικό χρόνο (real-time digital signal processing), είτε σημαίνει χαμηλότατη κατανάλωση ισχύος πχ υλοποίηση ασύρματου δικτύου αισθητήρων (wireless sensor network).

Για την υλοποίηση των δυνατοτήτων ενός συστήματος, οι σχεδιαστές πρέπει να επιλέξουν τις λειτουργίες που θα υλοποιηθούν στο υλικό και τις λειτουργίες που θα υλοποιηθούν σε λογισμικό. Το λογισμικό ή αλλιώς η κεντρική μονάδα επεξεργασίας (central processing unit - CPU), ανάλογα με το ρεπερτόριο εντολών της, έχει την δυνατότητα να εκτελέσει εξαιρετικό πλήθος εργασιών, μεγάλου φάσματος. Μάλιστα, με την ανάπτυξη πολυπύρηνων (multi-core) και πολυνηματικών (multi-thread) δυνατοτήτων στους επεξεργαστές, η υπολογιστική ισχύς είναι εξαιρετικά υψηλή. Παρόλα αυτά το υλικό, λόγω της εξειδικευμένης εργασίας που εκτελεί, πχ ένας διαιρέτης μιγαδικών αριθμών υλοποιημένος σε FPGA, μπορεί να εκτελέσει μια εργασία πολύ πιο γρήγορα και με χαμηλότερη κατανάλωση ισχύος από ότι θα την εκτελούσε ο επεξεργαστής. Επομένως, κρίνεται πλέον αναγκαία η συσχεδίαση του υλικού και του λογισμικού, στοχεύοντας στην βέλτιστη λύση.

Τέλος, ο συνδετικός κρίκος μεταξύ υλικού-λογισμικού είναι ο δίαυλος επικοινωνίας (bus ή interconnect). Τα ενσωματωμένα συστήματα χρησιμοποιούν τον δίαυλο για την μεταφορά των δεδομένων από το υλικό στο λογισμικό και το αντίστροφο. Όλες οι επικοινωνίες εντός του ολοκληρωμένου κυκλώματος αλλά και μεταξύ του ολοκληρωμένου με τον «έξω κόσμο» γίνονται βάσει πρωτοκόλλου. Στο Σχήμα 1 παρουσιάζεται ένα απλουστευμένο μπλοκ διάγραμμα ενός ενσωματωμένου συστήματος. Περιλαμβάνει τον επεξεργαστή (Processing system), τον δίαυλο επικοινωνίας (AXI Interface) βάσει του πρωτοκόλλου AXI4, το οποίο χρησιμοποιήθηκε στην παρούσα εργασία, και την προγραμματιζόμενη λογική (Programmable logic).




**Σχήμα 1: Μπλοκ διάγραμμα ενσωματωμένου συστήματος**

Το παραπάνω σύστημα υλοποιήθηκε σε Zedboard της Xilinx. Η μονάδα επεξεργασίας ή αλλιώς PS (processing system) βρίσκεται μέσα στο chip το οποίο περιέχει και το υλικό (fabric) ενός FPGA (programmable logic ή PL), κάνοντας την επικοινωνία υλικού-λογισμικού εξαιρετικά γρήγορη μέσω του δίαυλου επικοινωνίας.

# 2. ΤΟ ΠΡΩΤΟΚΟΛΛΟ ΕΠΙΚΟΙΝΩΝΙΑΣ AXI4

## 2.1 Γενικά Χαρακτηριστικά

Το AMBA AXI4 πρωτόκολλο επικοινωνίας [1] υποστηρίζει την σχεδίαση συστημάτων υψηλής απόδοσης και συχνότητας (high performance and frequency). Πιο συγκεκριμένα, το πρωτόκολλο:

- Αρμόζει σε συστήματα μεγάλου εύρους ζώνης (high-bandwidth) και χαμηλής υστέρησης (low-latency)
- Λειτουργεί σε υψηλές συχνότητες χωρίς να είναι αναγκαία μια πολύπλοκη λογική προσαρμογής ή αλλιώς «γεφύρωσης» (complex bridges)
- Καλύπτει τις απαιτήσεις ενός μεγάλου εύρους διεπαφών (interfaces)
- Αρμόζει για επικοινωνία με διαχειριστές μνήμης (memory controllers) μεγάλου χρόνου αρχικοποίησης (high initial access latency)
- Παρέχει ευελιξία (flexibility) στην σχεδίαση αρχιτεκτονικών με δίαυλο επικοινωνίας (interconnect architectures)
- Είναι συμβατό με παλαιότερες εκδόσεις του πρωτοκόλλου όπως το AHB ή το APB

Σημαντικά στοιχεία του AXI4 πρωτοκόλλου είναι τα εξής:

- Διαχωρισμός μεταφοράς διευθύνσεων/σημάτων ελέγχου (address/control) και δεδομένων (payload)
- Υποστήριξη μεταφοράς μη ευθυγραμμισμένων δεδομένων (unaligned data transfer) με την χρήση των σημάτων strobe (βλέπε Πίνακας 5)
- Συναλλαγές μεγάλου όγκου πληροφορίας (burst-based transactions) με χρήση μόνο της αρχικής διεύθυνσης μνήμης (start address)
- Διαχωρισμός καναλιών για εγγραφές και αναγνώσεις (read/write channel) με στόχο την Άμεση Πρόσβαση Μνήμης με χαμηλό κόστος (low-cost Direct Access Memory)
- Εκτέλεση συναλλαγών εκτός σειράς (out-of-order transaction completion)
- Εύκολη εισαγωγή καταχωρητών (εφαρμογή μεθόδου pipeline) για την επίτευξη των χρονικών απαιτήσεων του συστήματος (timing closure)

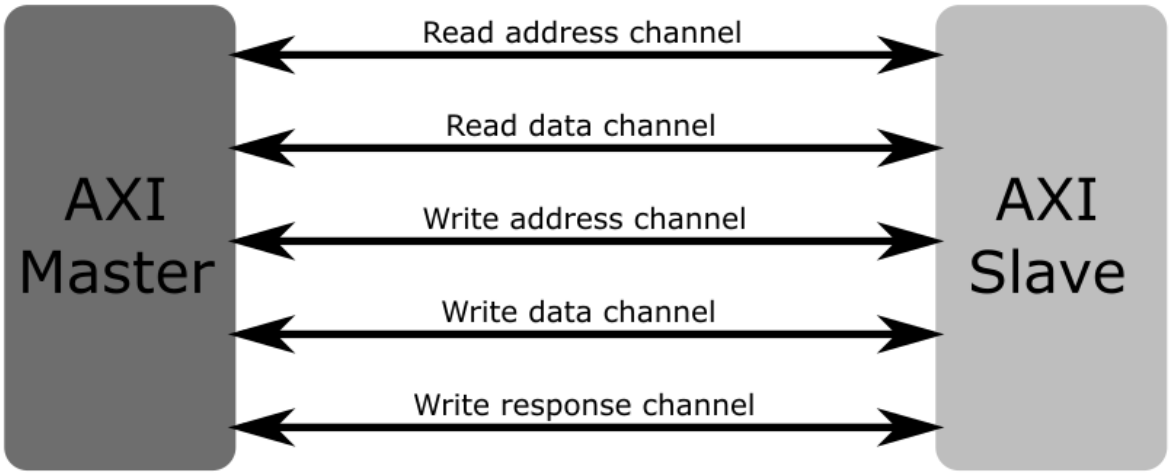
## 2.2 Βασική αρχιτεκτονική του πρωτοκόλλου AXI4

Για την επίτευξη οποιασδήποτε συναλλαγής σε ένα σύστημα που χρησιμοποιεί AXI4, ακολουθείται η αρχιτεκτονική Αφέντη/Σκλάβου (Master/Slave). Το μπλοκ που ξεκινά μια συναλλαγή ανάγνωσης ή εγγραφής ονομάζεται AXI Master. Το μπλοκ που δέχεται τα δεδομένα που αφορούν την συναλλαγή (διεύθυνση, σήματα ελέγχου κτλ) ονομάζεται AXI Slave. Ο AXI Master εκτελεί συναλλαγές ανάγνωσης και εγγραφής από και προς τον AXI Slave αντίστοιχα.

Στο πρωτόκολλο AXI4 ορίζονται τα εξής ανεξάρτητα κανάλια συναλλαγών:

- Κανάλι διεύθυνσης ανάγνωσης (Read address channel)
- Κανάλι δεδομένων ανάγνωσης (Read data channel)
- Κανάλι διεύθυνσης εγγραφής (Write address channel)
- Κανάλι δεδομένων εγγραφής (Write data channel)
- Κανάλι επιβεβαίωσης εγγραφής (Write response channel)

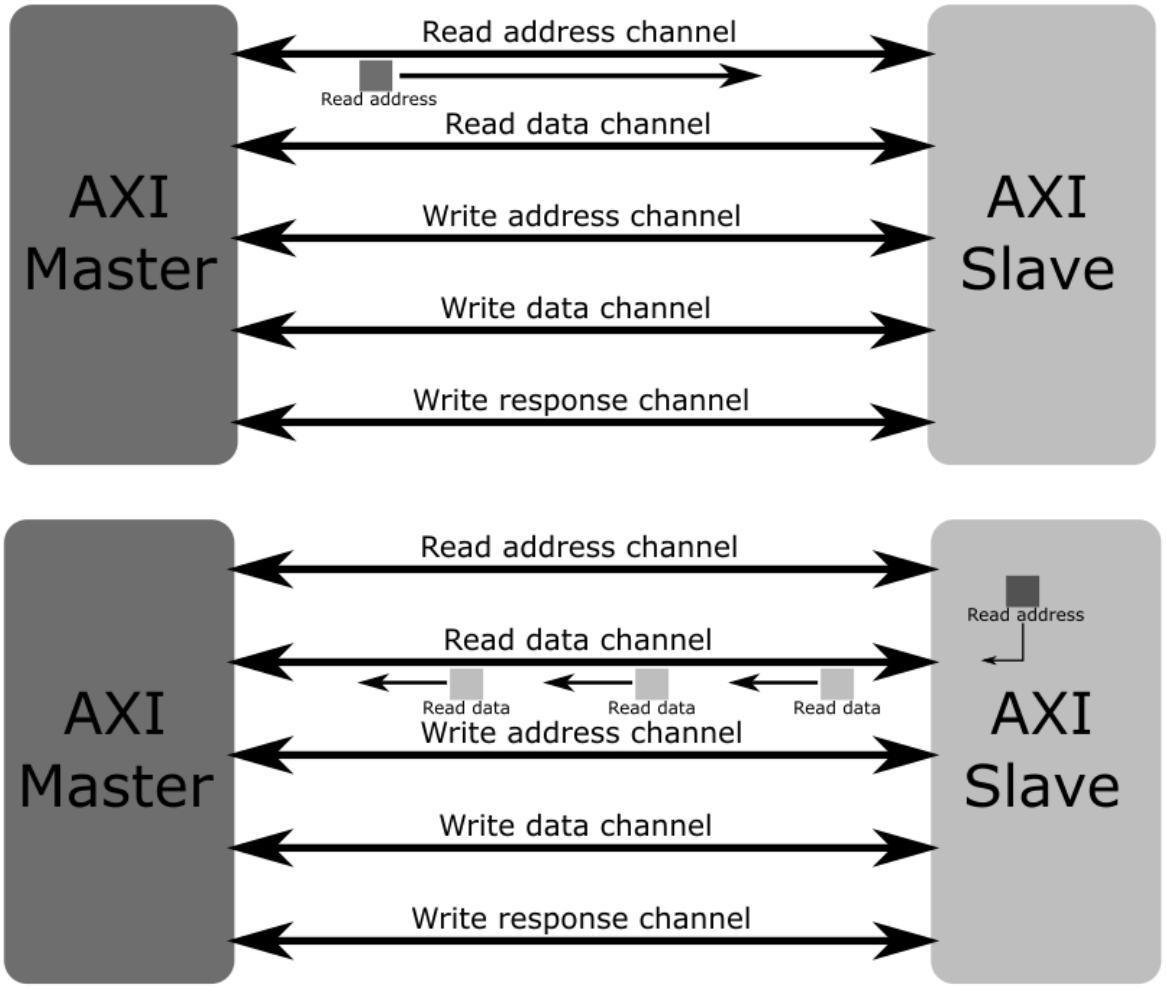
Στο Σχήμα 2 παρατίθεται η βασική αρχιτεκτονική μιας AXI4 διεπαφής μεταξύ ενός AXI Master και ενός AXI Slave.Όπως φαίνεται στο σχήμα, τα δύο μέρη επικοινωνούν μέσω των πέντε ανεξάρτητων καναλιών που αναφέρονται παραπάνω.




**Σχήμα 2: Βασική αρχιτεκτονική AXI4**

### 2.2.1 Συναλλαγή ανάγνωσης

Η συναλλαγή ανάγνωσης αναφέρεται ουσιαστικά στην μεταφορά δεδομένων από τον AXI Slave προς τον AXI Master. Ο AXI Master στέλνει αρχικά την διεύθυνση ανάγνωσης (και τα απαραίτητα σήματα ελέγχου) μέσω του read address channel, ο AXI Slave την δέχεται, και έπειτα από την απαραίτητη επεξεργασία, θα στείλει πίσω στον Master τα δεδομένα που είναι αποθηκευμένα στην διεύθυνση που δέχθηκε (Σχήμα 3).

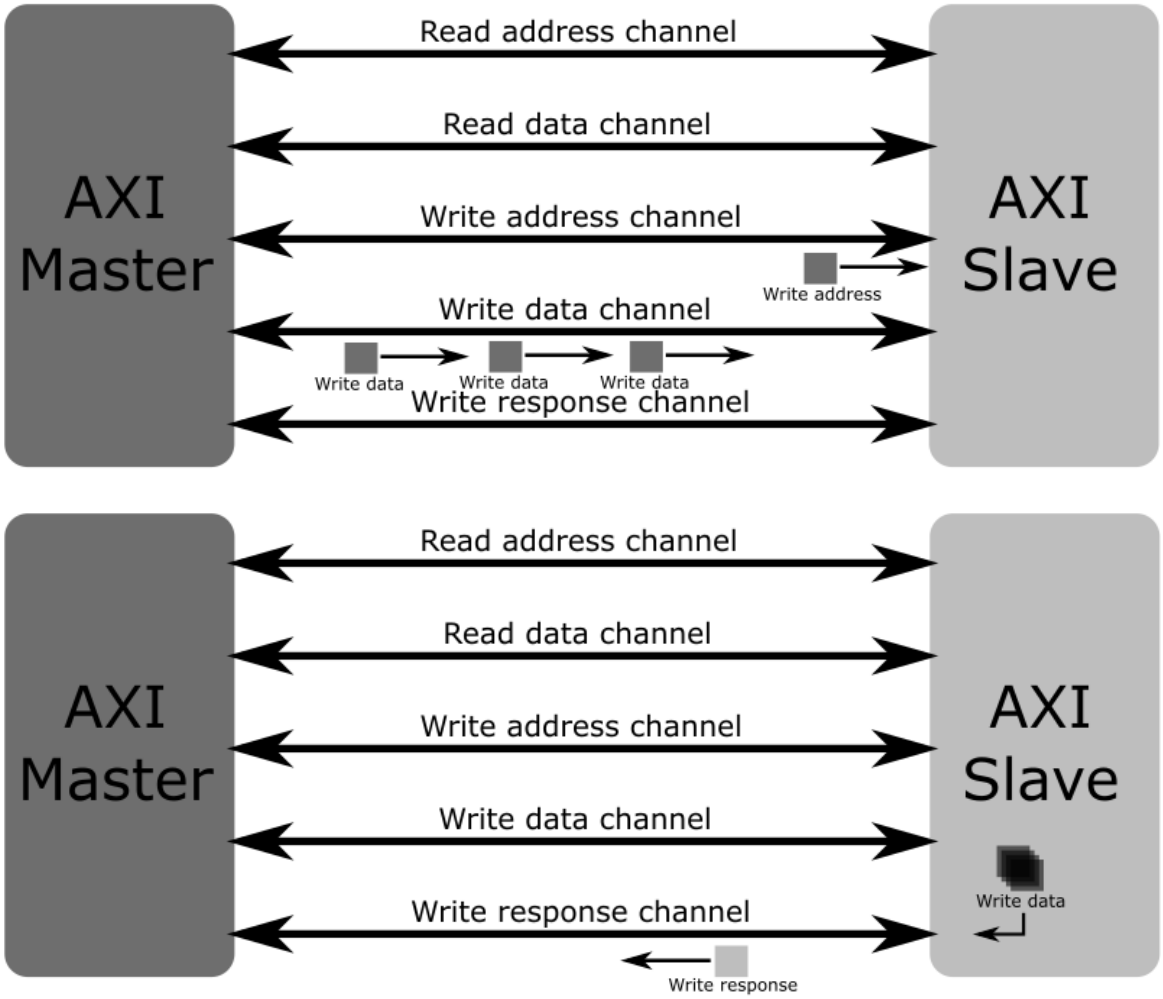



**Σχήμα 3: Συναλλαγή Ανάγνωσης**

Όπως αναφέρθηκε, ο AXI Master μπορεί να στείλει την αρχική διεύθυνση μνήμης και επίσης το μέγεθος της συναλλαγής (δηλαδή πόσα δεδομένα θέλει να διαβάσει) και ο AXI Slave θα στείλει με τη σειρά τα δεδομένα που ζητήθηκαν, ξεκινώντας από την διεύθυνση που δέχθηκε (βλέπε Πίνακας 2 και Πίνακας 3). Για παράδειγμα, ο AXI Master στέλνει την διεύθυνση 0x04 και το μέγεθος της συναλλαγής 0x03. Ο AXI Slave θα απαντήσει στέλνοντας πρώτα τα δεδομένα της διεύθυνσης 0x04, έπειτα τα δεδομένα της διεύθυνσης 0x08 και τέλος τα δεδομένα της διεύθυνσης 0x0C (έστω 4-byte οργανωμένη μνήμη).

### 2.2.2 Συναλλαγή εγγραφής

Η συναλλαγή εγγραφής αναφέρεται στην μεταφορά δεδομένων από τον AXI Master προς τον AXI Slave. Ο Master θα στείλει αρχικά την διεύθυνση στην οποία θέλει να γράψει τα δεδομένα και έπειτα θα στείλει τα δεδομένα (payload) για εγγραφή. Αφού έχει λάβει τα δεδομένα, ο AXI Slave θα απαντήσει, μέσω του Write response channel, ότι έλαβε ορθά ή όχι τα δεδομένα που στάλθηκαν (Σχήμα 4). Η αποστολή της διεύθυνσης εγγραφής δεν είναι απαραίτητο να γίνει πριν την αποστολή των δεδομένων. Εξαρτάται από την υλοποίηση της λογικής του AXI Slave. Όπως αναφέρθηκε και στην διαδικασία ανάγνωσης, ο AXI Master θα χρειαστεί να στείλει μόνο την αρχική διεύθυνση εγγραφής και τα δεδομένα θα εγγραφούν με τη σειρά στις επόμενες θέσεις μνήμης.




**Σχήμα 4: Συναλλαγή εγγραφής**

## 2.3 Περιγραφή σημάτων του πρωτοκόλλου AXI4

Σε αυτή την ενότητα περιγράφονται όλα τα σήματα που περιλαμβάνει το πρωτόκολλο καθώς και η χρήση τους. Ένας σχεδιαστής μπορεί να χρησιμοποιεί όλα ή ένα υποσύνολο των σημάτων για την ολοκλήρωση του συστήματός του.
Η περιγραφή των σημάτων διαχωρίζεται ανά κανάλι. Οι πέντε πίνακες αντιστοιχούν στα πέντε κανάλια επικοινωνίας μεταξύ ενός AXI Master και ενός AXI Slave.

### 2.3.1 Καθολικά σήματα του πρωτοκόλλου AXI4

Υπάρχουν δύο καθολικά (global) σήματα που χρησιμοποιούνται για την λειτουργία του συστήματος βασισμένο σε AXI4 πρωτόκολλο επικοινωνίας και είναι ανεξάρτητα καναλιών και αναφέρονται στον Πίνακας 1.

**Πίνακας 1: Περιγραφή καθολικών σημάτων του AXI4**

| ΣΗΜΑ | ΠΗΓΗ ΣΗΜΑΤΟΣ | ΠΕΡΙΓΡΑΦΗ ΣΗΜΑΤΟΣ |
|---|---|---|
| ACLK | Πηγή ρολογιού (DCM, PLL ή MMCM κτλ) | Αποτελεί το ρολόι του συστήματος για την λειτουργία των καταχωρητών/flip-flop |
| ARESETn | Λογική αρχικοποίησης (reset) | Αποτελεί το σήμα αρχικοποίησης. Το σήμα είναι αρνητικής λογικής, δηλαδή όταν βρίσκεται στο λογικό ‘0’ τότε ενεργοποιείται η αρχικοποίηση (ACTIVE LOW). |

ΣΗΜΕΙΩΣΗ: Όλοι ο καταχωρητές λειτουργούν με την ανοδική ακμή του ρολογιού (rising edge sampling).

### 2.3.2 Σήματα καναλιού διεύθυνσης ανάγνωσης

Ο Πίνακας 2 αναφέρεται στα σήματα του καναλιού διεύθυνσης ανάγνωσης (Read address channel).

**Πίνακας 2: Σήματα καναλιού διεύθυνσης ανάγνωσης**

| ΣΗΜΑ | ΠΗΓΗ ΣΗΜΑΤΟΣ | ΠΕΡΙΓΡΑΦΗ ΣΗΜΑΤΟΣ |
|---|---|---|
| ARID | AXI MASTER | Ταυτότητα διεύθυνσης ανάγνωσης (Read address ID). Το σήμα αυτό χρησιμοποιείται ως ετικέτα για την διάκριση των σημάτων του καναλιού διεύθυνσης ανάγνωσης (identification tag). |
| ARADDR | AXI MASTER | Διεύθυνση ανάγνωσης. Αποτελεί την αρχική διεύθυνση ανάγνωσης δεδομένων από τον AXI Slave. |
| ARLEN | AXI MASTER | Δηλώνει τον αριθμό των αναγνώσεων. Δηλαδή πόσες αναγνώσεις θα πραγματοποιηθούν μέχρι την ολοκλήρωση της συναλλαγής. |
| ARSIZE | AXI MASTER | Δηλώνει το μέγεθος των αναγνώσεων. Πόσα bytes πληροφορίας περιλαμβάνει η κάθε ανάγνωση. |

| | | |
|---|---|---|
| ARBURST | AXI MASTER | Τύπος συναλλαγής. Δηλώνει τον τρόπο με τον οποίο θα υπολογιστεί η επόμενη διεύθυνση ανάγνωσης. |
| ARLOCK | AXI MASTER | Όταν πραγματοποιηθεί μια συναλλαγή με ενεργοποιημένο αυτό το σήμα, τότε κανένας άλλος AXI Master δεν μπορεί να πραγματοποιήσει συναλλαγή ανάγνωσης με τον συγκεκριμένο AXI Slave. Μόλις πραγματοποιηθεί συναλλαγή ανάγνωσης με απενεργοποιημένο αυτό το σήμα τότε μπορούν και άλλοι AXI Masters να πραγματοποιήσουν αναγνώσεις με τον συγκεκριμένο AXI Slave. (το σήμα έχει νόημα ύπαρξης στην περίπτωση που υπάρχουν παραπάνω από ένας AXI Master στο σύστημα και έχουν όλοι πρόσβαση σε έναν ή περισσότερους AXI Slave) |
| ARCACHE | AXI MASTER | Τύπος μνήμης. Δηλώνει τον τύπο της μνήμης από την οποία διαβάζονται τα δεδομένα. |
| ARPROT | AXI MASTER | Τύπος προστασίας. Δηλώνει τον τύπο προστασίας της συναλλαγής, καθώς επίσης και το αν η συναλλαγή αποτελεί ανάγνωση δεδομένων ή εντολής (data ή instruction read για την περίπτωση επεξεργαστή που μιλά με μια cache μνήμη εντολών ή/και δεδομένων). |
| ARQOS | AXI MASTER | Ποιότητα υπηρεσίας (Quality of Service). Δηλώνει τον βαθμό ποιότητας της συναλλαγής. |
| ARREGION | AXI MASTER | Ταυτότητα περιοχής. Χρησιμοποιείται ώστε να αντιστοιχίζεται μια πραγματική διεπαφή (physical interface) σε πολλαπλές λογικές διεπαφές (logical interfaces) |
| ARUSER | AXI MASTER | Σήμα χρήστη. Μπορεί να χρησιμοποιηθεί για την μεταφορά επιπλέον πληροφορίας, ανάλογα με τις απαιτήσεις του συστήματος. |
| ARVALID | AXI MASTER | Δηλώνει ότι η διεύθυνση ανάγνωσης και τα σήματα ελέγχου είναι έγκυρα και έτοιμα προς μεταφορά (ενότητα 2.4). |
| ARREADY | AXI SLAVE | Δηλώνει ότι ο AXI Slave είναι έτοιμος να λάβει τη διεύθυνση ανάγνωσης και τα σήματα ελέγχου (ενότητα 2.4). |

### 2.3.3 Σήματα καναλιού δεδομένων ανάγνωσης

Ο Πίνακας 3 αναφέρεται στα σήματα του καναλιού δεδομένων ανάγνωσης (Read data channel).

**Πίνακας 3: Σήματα καναλιού δεδομένων ανάγνωσης**

| ΣΗΜΑ | ΠΗΓΗ ΣΗΜΑΤΟΣ | ΠΕΡΙΓΡΑΦΗ ΣΗΜΑΤΟΣ |
|---|---|---|
| RID | AXI SLAVE | Ετικέτα ταυτότητας (identification tag). Χρησιμοποιείται για την διάκριση των σημάτων του καναλιού δεδομένων ανάγνωσης. |
| RDATA | AXI SLAVE | Τα δεδομένα ανάγνωσης (payload). |
| RRESP | AXI SLAVE | Επιβεβαίωση ανάγνωσης. Ενημερώνει τον AXI Master για την κατάσταση (status) της συναλλαγής ανάγνωσης. |
| RLAST | AXI SLAVE | Το σήμα δηλώνει ότι η τρέχουσα ανάγνωση δεδομένων αποτελεί την τελευταία. Όταν η τελευταία μεταφορά δεδομένων σε μια συναλλαγή πραγματοποιείται, το σήμα αυτό ενεργοποιείται. |
| RUSER | AXI SLAVE | Σήμα χρήστη. Μπορεί να χρησιμοποιηθεί για την μεταφορά επιπλέον πληροφορίας, ανάλογα με τις απαιτήσεις του συστήματος. |
| RVALID | AXI SLAVE | Δηλώνει ότι τα δεδομένα ανάγνωσης είναι έγκυρα και έτοιμα προς μεταφορά (ενότητα 2.4). |
| RREADY | AXI MASTER | Δηλώνει ότι ο AXI Master είναι έτοιμος να λάβει τα δεδομένα ανάγνωσης (ενότητα 2.4). |

### 2.3.4 Σήματα καναλιού διεύθυνσης εγγραφής

Ο Πίνακας 4 αναφέρεται στα σήματα του καναλιού διεύθυνσης εγγραφής (Write address channel).

**Πίνακας 4: Σήματα καναλιού διεύθυνσης εγγραφής**

| ΣΗΜΑ | ΠΗΓΗ ΣΗΜΑΤΟΣ | ΠΕΡΙΓΡΑΦΗ ΣΗΜΑΤΟΣ |
|---|---|---|
| AWID | AXI MASTER | Ταυτότητα διεύθυνσης εγγραφής (Write address ID). Το σήμα αυτό χρησιμοποιείται ως ετικέτα για την διάκριση των σημάτων του καναλιού διεύθυνσης εγγραφής (identification tag). |
| AWADDR | AXI MASTER | Διεύθυνση εγγραφής. Αποτελεί την αρχική διεύθυνση εγγραφής δεδομένων στον AXI Slave. |
| AWLEN | AXI MASTER | Δηλώνει τον αριθμό των εγγραφών. Δηλαδή πόσες εγγραφές θα πραγματοποιηθούν μέχρι την ολοκλήρωση της συναλλαγής. |

| | | |
|---|---|---|
| AWSIZE | AXI MASTER | Δηλώνει το μέγεθος των εγγραφών. Πόσα bytes πληροφορίας περιλαμβάνει η κάθε εγγραφή. |
| AWBURST | AXI MASTER | Τύπος συναλλαγής. Δηλώνει τον τρόπο με τον οποίο θα υπολογιστεί η διεύθυνση εγγραφής. |
| AWLOCK | AXI MASTER | Όταν πραγματοποιηθεί μια συναλλαγή με ενεργοποιημένο αυτό το σήμα, τότε κανένας άλλος AXI Master δεν μπορεί να πραγματοποιήσει συναλλαγή εγγραφής με τον συγκεκριμένο AXI Slave. Μόλις πραγματοποιηθεί συναλλαγή εγγραφής με απενεργοποιημένο αυτό το σήμα τότε μπορούν και άλλοι AXI Masters να πραγματοποιήσουν εγγραφές με τον συγκεκριμένο AXI Slave. (το σήμα έχει νόημα ύπαρξης στην περίπτωση που υπάρχουν παραπάνω από ένας AXI Master στο σύστημα και έχουν όλοι πρόσβαση σε έναν ή περισσότερους AXI Slave) |
| AWCACHE | AXI MASTER | Τύπος μνήμης. Δηλώνει τον τύπο της μνήμης στην οποία γράφονται τα δεδομένα. |
| AWPROT | AXI MASTER | Τύπος προστασίας. Δηλώνει τον τύπο προστασίας της συναλλαγής, καθώς επίσης και το αν η συναλλαγή αποτελεί εγγραφή δεδομένων ή εντολής (data ή instruction write για την περίπτωση επεξεργαστή που μιλά με μια cache μνήμη εντολών ή/και δεδομένων). |
| AWQOS | AXI MASTER | Ποιότητα υπηρεσίας (Quality of Service). Δηλώνει τον βαθμό ποιότητας της συναλλαγής. |
| AWREGION | AXI MASTER | Ταυτότητα περιοχής. Χρησιμοποιείται ώστε να αντιστοιχίζεται μια πραγματική διεπαφή (physical interface) σε πολλαπλές λογικές διεπαφές (logical interfaces) |
| AWUSER | AXI MASTER | Σήμα χρήστη. Μπορεί να χρησιμοποιηθεί για την μεταφορά επιπλέον πληροφορίας, ανάλογα με τις απαιτήσεις του συστήματος. |
| AWVALID | AXI MASTER | Δηλώνει ότι η διεύθυνση εγγραφής και τα σήματα ελέγχου είναι έγκυρα και έτοιμα προς μεταφορά (ενότητα 2.4). |
| AWREADY | AXI SLAVE | Δηλώνει ότι ο AXI Slave είναι έτοιμος να λάβει τη διεύθυνση εγγραφής και τα σήματα ελέγχου (ενότητα 2.4). |

### 2.3.5 Σήματα καναλιού δεδομένων εγγραφής

Ο Πίνακας 5 αναφέρεται στα σήματα του καναλιού δεδομένων εγγραφής (Write data channel).

**Πίνακας 5: Σήματα καναλιού δεδομένων εγγραφής**

| ΣΗΜΑ | ΠΗΓΗ ΣΗΜΑΤΟΣ | ΠΕΡΙΓΡΑΦΗ ΣΗΜΑΤΟΣ |
|---|---|---|
| WID | AXI MASTER | Ετικέτα ταυτότητας (identification tag). Χρησιμοποιείται για την διάκριση των σημάτων του καναλιού δεδομένων εγγραφής. |
| WDATA | AXI MASTER | Τα δεδομένα εγγραφής (payload). |
| WSTRB | AXI MASTER | Τα σήματα write strobes δηλώνουν ποια bytes είναι έγκυρα δεδομένα κατά την εγγραφή μέσω ενός δίαυλου. Ο AXI Master συσχετίζει ένα σήμα strobe για κάθε byte δεδομένων. Πχ για 32-bit λέξεις δεδομένων αντιστοιχίζονται 4 σήματα strobe, για 64-bit λέξεις αντιστοιχίζονται 8 σήματα strobe κοκ. |
| WLAST | AXI MASTER | Το σήμα δηλώνει ότι η τρέχουσα εγγραφή δεδομένων αποτελεί την τελευταία. Όταν η τελευταία μεταφορά δεδομένων σε μια συναλλαγή πραγματοποιείται, το σήμα αυτό ενεργοποιείται. |
| WUSER | AXI MASTER | Σήμα χρήστη. Μπορεί να χρησιμοποιηθεί για την μεταφορά επιπλέον πληροφορίας, ανάλογα με τις απαιτήσεις του συστήματος. |
| WVALID | AXI MASTER | Δηλώνει ότι τα δεδομένα εγγραφής είναι έγκυρα και έτοιμα προς μεταφορά (ενότητα 2.4). |
| WREADY | AXI SLAVE | Δηλώνει ότι ο AXI Slave είναι έτοιμος να λάβει τα δεδομένα εγγραφής (ενότητα 2.4). |

### 2.3.6 Σήματα καναλιού επιβεβαίωσης εγγραφής

Ο Πίνακας 6 αναφέρεται στα σήματα του καναλιού επιβεβαίωσης εγγραφής (Write response channel).

**Πίνακας 6: Σήματα καναλιού επιβεβαίωσης εγγραφής**

| ΣΗΜΑ | ΠΗΓΗ ΣΗΜΑΤΟΣ | ΠΕΡΙΓΡΑΦΗ ΣΗΜΑΤΟΣ |
|---|---|---|
| BID | AXI SLAVE | Ετικέτα ταυτότητας (identification tag). Χρησιμοποιείται για την διάκριση των σημάτων του καναλιού επιβεβαίωσης εγγραφής. |
| BRESP | AXI SLAVE | Τα δεδομένα επιβεβαίωσης εγγραφής. |
| BUSER | AXI SLAVE | Σήμα χρήστη. Μπορεί να χρησιμοποιηθεί για την μεταφορά επιπλέον πληροφορίας, ανάλογα με τις απαιτήσεις του συστήματος. |

| | | |
|---|---|---|
| BVALID | AXI SLAVE | Δηλώνει ότι τα δεδομένα επιβεβαίωσης εγγραφής είναι έγκυρα και έτοιμα προς μεταφορά (ενότητα 2.4). |
| BREADY | AXI MASTER | Δηλώνει ότι ο AXI Master είναι έτοιμος να λάβει τα δεδομένα εγγραφής (ενότητα 2.4). |

## 2.4 Περιγραφή απαιτήσεων του πρωτοκόλλου AXI4

Σε αυτή την ενότητα περιγράφονται οι απαιτήσεις (requirements) του πρωτοκόλλου για την ορθή υλοποίηση συστημάτων βασισμένα σε AXI4. Επίσης, δίνεται μια πιο λεπτομερής επεξήγηση των σημάτων της AXI4 διεπαφής που περιγράφεται στην ενότητα 2.3.

- Ρολόι συστήματος (ACLK)

Κάθε υπομονάδα (component) που επικοινωνεί με AXI4 πρωτόκολλο χρησιμοποιεί ένα μοναδικό ρολόι. Όλοι οι καταχωρητές/flip-flop λειτουργούν με την ανερχόμενη ακμή του ρολογιού και οι αλλαγές στην τιμή οποιουδήποτε σήματος πρέπει να συμβαίνουν μετά την ανερχόμενη ακμή του ρολογιού.

- Αρχικοποίηση συστήματος (ARESETn)

Το πρωτόκολλο AXI4 χρησιμοποιεί ένα μοναδικό σήμα αρχικοποίησης (reset)αρνητικής λογικής (ACTIVELOW). Το σήμα μπορεί να ενεργοποιηθεί (λογική τιμή ‘0’) ασύγχρονα αλλά η απενεργοποίησή του (λογική τιμή ‘1’) θα πρέπει να γίνεται σύγχρονα με την ανερχόμενη ακμή του ρολογιού.

Κατά την ενεργοποίηση του ARESETn θα πρέπει να συμβαίνουν τα εξής:

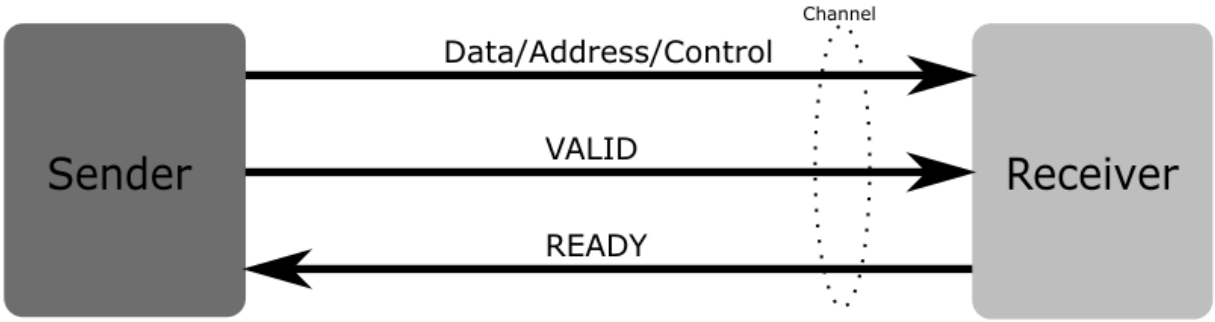
1) Ο AXI Master πρέπει να οδηγεί τα σήματα AWVALID, ARVALID και WVALID στην λογική τιμή ‘0’.
2) Ο AXI Slave πρέπει να οδηγεί τα σήματα RVALID και BVALID στην λογική τιμή ‘0’.
3) Όλα τα υπόλοιπα σήματα μπορούν να οδηγούνται σε οποιαδήποτε τιμή.

Με λίγα λόγια, όλα τα σήματα valid θα πρέπει να οδηγούνται στο ‘0’ κατά την αρχικοποίηση.

ΣΗΜΕΙΩΣΗ: Ο AXI Master μπορεί να οδηγήσει τα valid σήματα του σε λογικό ‘1’ (δηλαδή να ξεκινήσει μια συναλλαγή) μόλις απενεργοποιηθεί το ARESETn και έρθει η πρώτη ανερχόμενη ακμή του ACLK (μετά από την απενεργοποίηση του ARESETn).

- READY/VALID χειραψία

Για την επιτυχημένη μεταφορά δεδομένων μέσω ενός καναλιού, χρησιμοποιείται η τεχνική «χειραψίας» ready/valid (Σχήμα 5). Ο αποστολέας (sender) ενεργοποιεί το σήμα valid όταν έχει έτοιμα δεδομένα (Data/Address/Control) προς αποστολή και ο παραλήπτης (receiver) ενεργοποιεί το σήμα ready όταν είναι έτοιμος να λάβει νέα δεδομένα. Η αποστολή επιτυγχάνεται όταν και τα δύο σήματα είναι ενεργοποιημένα.




**Σχήμα 5: Ready/Valid "χειραψία"**

Στο Σχήμα 6 παρουσιάζονται τα χρονικά διαγράμματα για τρεις διαφορετικές περιπτώσεις «χειραψίας» όπου στην πρώτη περίπτωση το valid σήμα είναι ενεργοποιημένο πριν το ready σήμα, στη δεύτερη περίπτωση το valid σήμα ενεργοποιείται μετά το ready σήμα και στην τρίτη περίπτωση τα σήματα ενεργοποιούνται ταυτόχρονα. Και στις τρεις περιπτώσεις η μεταφορά της πληροφορίας γίνεται όταν βρίσκονται ενεργοποιημένα και τα δύο σήματα (διακεκομμένη γραμμή - handshake) και έρθει η ανερχόμενη ακμή του ρολογιού (clock).

ΣΗΜΕΙΩΣΗ: Είναι πολύ σημαντικό, κατά την σχεδίαση της λογικής του αποστολέα, το valid σήμα να μην αναμένει την ενεργοποίηση του ready σήματος για να ενεργοποιηθεί. Ο αποστολέας κρατά ενεργοποιημένο το valid σήμα μέχρις ότου ενεργοποιηθεί το ready σήμα και εκτελεστεί η μεταφορά. Για το ready σήμα δεν ισχύει το ίδιο, καθώς το σήμα μπορεί να ενεργοποιηθεί πριν ή μετά την ενεργοποίηση του valid σήματος. Εξαρτάται από την υλοποίηση.

Κάθε ένα από τα πέντε κανάλια που αναφέρονται στην ενότητα 2.2 διαθέτει το δικό του ζευγάρι σημάτων ready/valid ώστε να πραγματοποιείται η μεταφορά της πληροφορίας πάνω στο εκάστοτε κανάλι. Όπως ίσως έγινε αντιληπτό, στα κανάλια Read address, Write address και Write data ο αποστολέας (sender) θεωρείται ο AXI Master, ενώ στα κανάλια Read data και Write response ο αποστολέας θεωρείται ο AXI Slave. Στην ενότητα 2.3 αναφέρεται λεπτομερώς η κατεύθυνση των σημάτων.

- Η δομή της συναλλαγής

Όπως ήδη έχει αναφερθεί, για να πραγματοποιηθεί μια συναλλαγή από και προς την οποιαδήποτε μνήμη, ο AXI Master στέλνει την αρχική διεύθυνση (του πρώτου byte που θέλει να γράψει ή να διαβάσει) και ο υπολογισμός των επόμενων θέσεων μνήμης ανάγνωσης ή εγγραφής αποτελεί λειτουργία του AXI Slave. Για αυτό το λόγο χρησιμοποιούνται τα σήματα ελέγχου AWLEN, AWSIZE, ARLEN και ARSIZE.

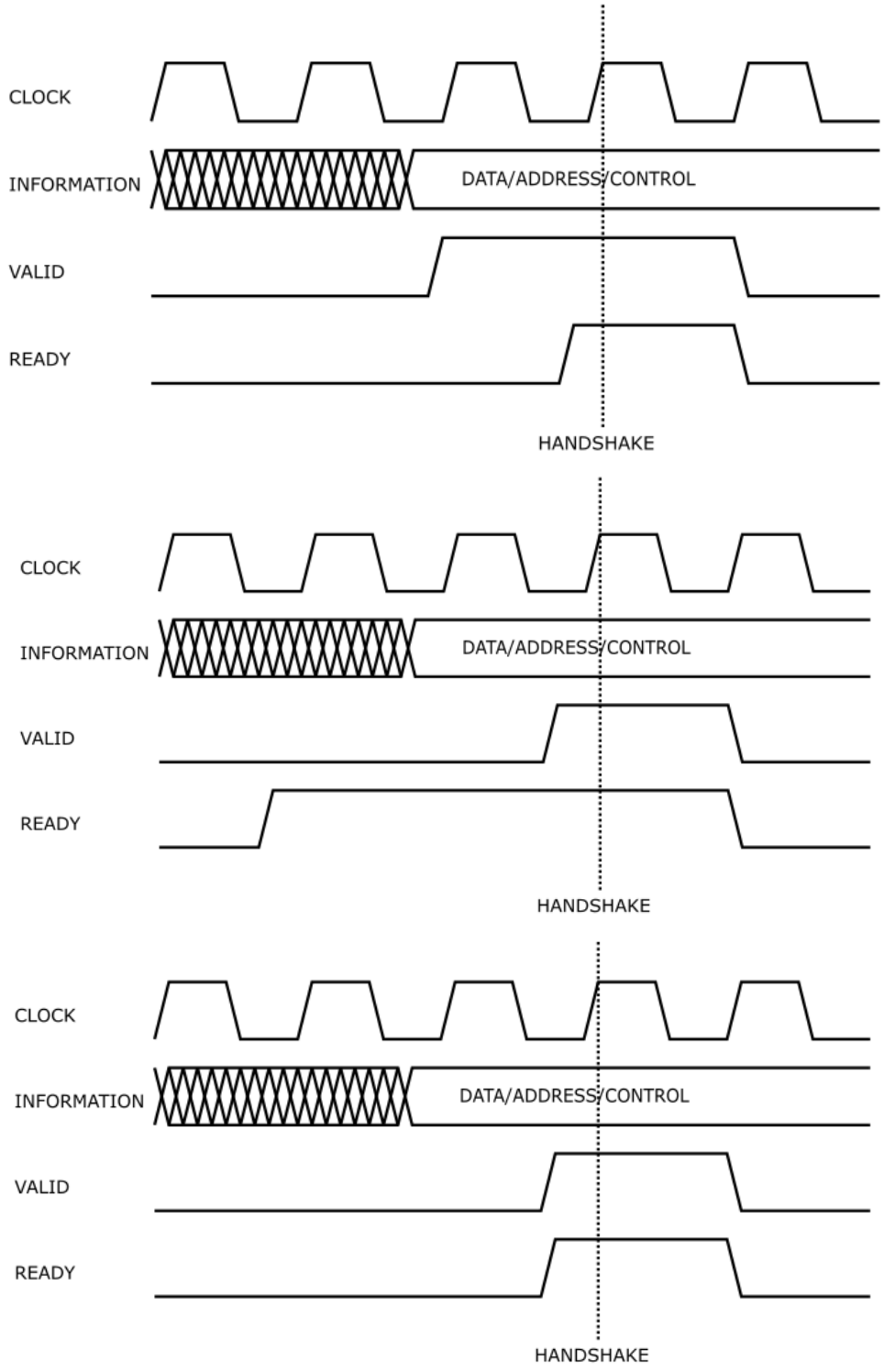
1) Το μήκος της συναλλαγής (AWLEN/ARLEN) ορίζει τον αριθμό των εγγραφών ή αναγνώσεων που θα πραγματοποιηθούν σε μία συναλλαγή. Το σήμα AWLEN/ARLEN είναι 8-bit και ισχύει: μήκος_συναλλαγής_εγγραφής = AWLEN[7:0] + 1. Η επιλογή του μήκους σε συνδυασμό με την αρχική διεύθυνση δεν θα πρέπει να «διασχίζουν» όρια της μνήμης πολλαπλάσια των 4KB.

ΣΗΜΕΙΩΣΗ: Σε περίπτωση που μια συναλλαγή, για κάποιο λόγο πρέπει να τερματιστεί πρόωρα, η συναλλαγή θα πρέπει να εκτελεστεί μέχρι τέλους (να μεταφερθεί όλος ο αριθμός των δεδομένων που ορίστηκε στο μήκος) και ο AXI Master χρησιμοποιεί τα σήματα strobe ώστε ο AXI Slave να μην λαμβάνει υπόψη του τα δεδομένα. Αντίστοιχα σε μία ανάγνωση, ο AXI Master μπορεί να αγνοεί τα εισερχόμενα δεδομένα.

2) Το μέγεθος της συναλλαγής (AWSIZE/ARSIZE) ορίζει τον μέγιστο αριθμό των έγκυρων bytes που περιλαμβάνει κάθε εγγραφή ή ανάγνωση. Το σήμα AWSIZE/ARSIZE είναι 3-bit και ορίζει το μέγεθος σύμφωνα με τον Πίνακας 7.

3) Ο τύπος της συναλλαγής (AWBURST/ARBURST) ορίζει τον τρόπο με τον οποίο υπολογίζονται οι επόμενες θέσεις μνήμης (ανάγνωσης ή εγγραφής) στον AXI Slave. Στον Πίνακας 8 περιγράφονται οι τρεις διαφορετικές περιπτώσεις υπολογισμού.
4) Τα σήματα strobe (WSTRB) ενημερώνουν τον AXI Slave ποια bytes είναι έγκυρα δεδομένα κατά την εγγραφή. Έτσι η λογική του AXI Slave γνωρίζει ποια δεδομένα να κρατήσει και ποια αποτελούν άχρηστη πληροφορία.

Ακολουθεί ένα σχηματικό παράδειγμα συναλλαγής εγγραφής για να γίνει αντιληπτή η χρήση όλων των παραπάνω σημάτων ελέγχου.




**Σχήμα 6: Ready/Valid "χειραψία"**

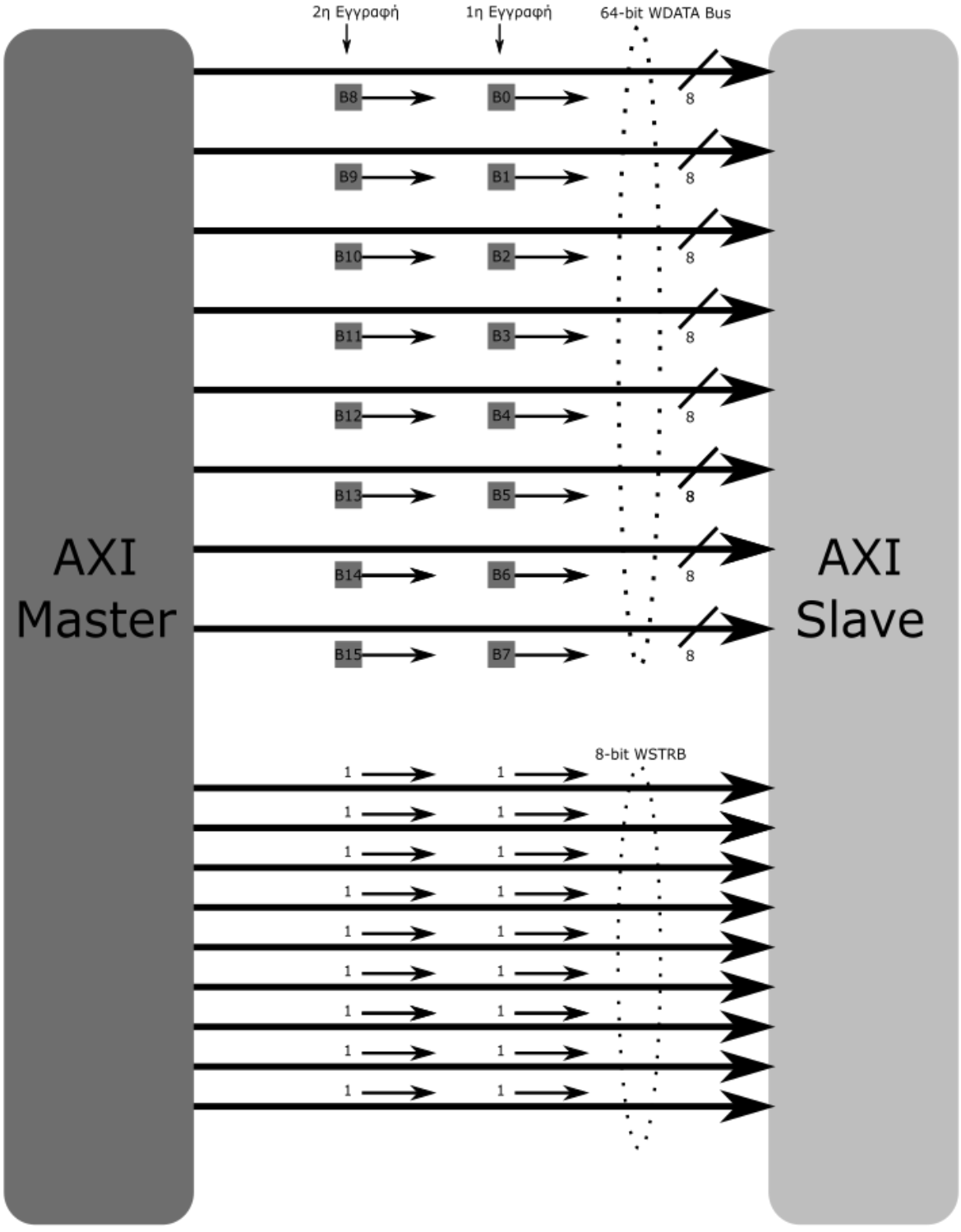
**Πίνακας 7: Κωδικοποίηση AWSIZE/ARSIZE**

| AWSIZE/ARSIZE (Δυαδική) | Bytesσε μία εγγραφή/ανάγνωση |
|---|---|
| 000 | 1 |
| 001 | 2 |
| 010 | 4 |
| 011 | 8 |
| 100 | 16 |
| 101 | 32 |
| 110 | 64 |
| 111 | 128 |

**Πίνακας 8: Κωδικοποίηση AWBURST/ARBURST**

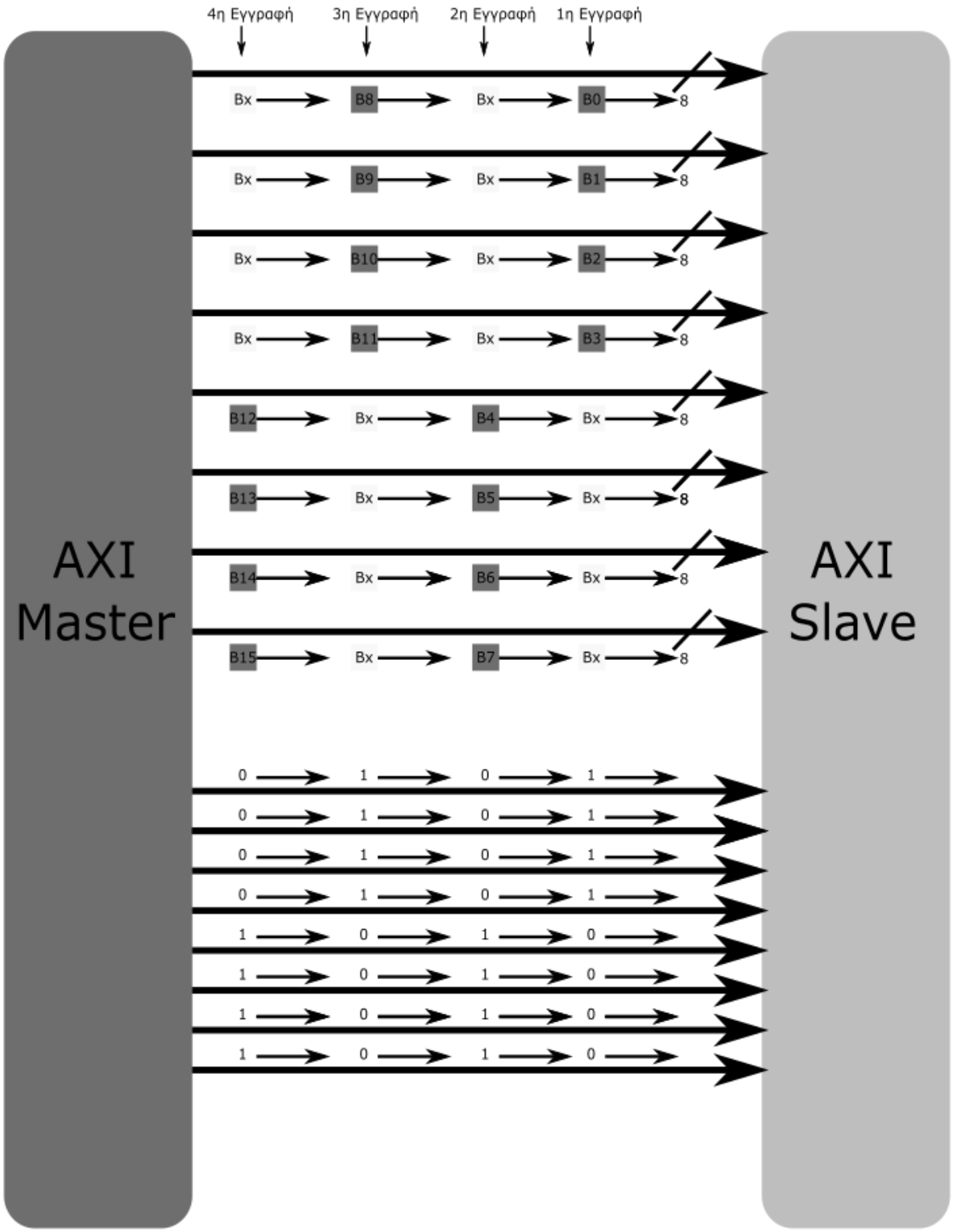
| AWBURST/ARBURST (Δυαδική) | Τύπος συναλλαγής | Περιγραφή τύπου συναλλαγής |
|---|---|---|
| 00 | FIXED | Χρησιμοποιείται όταν η διεύθυνση η οποία χρησιμοποιείται για ανάγνωση ή εγγραφή είναι μία και αφορά μια δομή δεδομένων όπως η ΣΤΟΙΒΑ (LIFO) ή η ΟΥΡΑ (FIFO). |
| 01 | INCREMENTING | Η διεύθυνση της επόμενης εγγραφής/ανάγνωσης είναι το αποτέλεσμα της προηγούμενης διεύθυνσης συν κάποια τιμή. Η τιμή αυτή ορίζεται από το μέγεθος (AWSIZE/ARSIZE) της εγγραφής/ανάγνωσης. Για παράδειγμα η νέα διεύθυνση σε μια συναλλαγή με μέγεθος 4-byte θα είναι η τρέχουσα τιμή συν 4. |
| 10 | WRAP | Όμοια με την περίπτωση INCREMENTING με τη διαφορά ότι όταν ο μετρητής της διεύθυνσης φτάσει στο τέλος της μνήμης, επιστρέφει στην αρχή της μνήμης για να αποθηκεύσει ή να διαβάσει τα επόμενα δεδομένα. |
| 11 | Reserved | - |

Για το παράδειγμα ορίζουμε ένα δίαυλο δεδομένων (Databus) εύρους 64-bit. Έστω ότι ο AXI Master θέλει να γράψει 16 bytes στον AXI Slave με INCREMENT τύπο συναλλαγής. Τα παρακάτω σχήματα παρουσιάζουν διαφορετικές περιπτώσεις μήκους και μεγέθους της συναλλαγής με το ίδιο αποτέλεσμα.




**Σχήμα 7: Συναλλαγή εγγραφής (a)**

Στην περίπτωση που ο AXI Master χειρίζεται 64-bit λέξεις τότε το μήκος της συναλλαγής είναι 2 (δύο εγγραφές) και το μέγεθος της κάθε εγγραφής είναι 8 εφόσον σε κάθε εγγραφή μεταφέρονται 8 bytes έγκυρης πληροφορίας. Επίσης, το κάθε byte συνοδεύεται από το strobe σήμα του και στην περίπτωση αυτή είναι όλα τα σήματα στο λογικό ΄1΄ γιατί κάθε byte αντιστοιχεί σε έγκυρη πληροφορία.




**Σχήμα 8: Συναλλαγή εγγραφής (b)**

Στο δεύτερο σχήμα παρουσιάζεται η περίπτωση όπου ο AXI Master χειρίζεται 32-bit λέξεις. Τώρα το μήκος της συναλλαγής είναι 4 εγγραφές και το μέγεθος της κάθε εγγραφής είναι 4 bytes. Επίσης, παρατηρούμε ότι τα strobe σήματα που αντιστοιχούν στο κάθε byte τώρα παίρνουν τιμή ΄0΄ για τα bytes που δεν περιέχουν έγκυρη πληροφορία. Τέλος, η μεταφορά δεδομένων γίνεται κατά αυτόν τον τρόπο (πρώτα στα 32 LSB και έπειτα στα 32 MSB κοκ) λόγω της INCREMENTING επιλογής στον τύπο της συναλλαγής. Παρόμοια συμπεριφορά θα είχαμε για διαχείριση 16-bit λέξεων με μήκος συναλλαγής 8 και μέγεθος εγγραφής 2 bytes κοκ.

5) Ο τύπος της μνήμης με την οποία γίνονται οι συναλλαγές ορίζεται από τα σήματα AWCACHE/ARCACHE και η κωδικοποίηση του σήματος ορίζεται σύμφωνα με τον Πίνακας 9.

**Πίνακας 9: Κωδικοποίηση AWCACHE/ARCACHE**

| ARCACHE (Δυαδική) | AWCACHE (Δυαδική) | Τύπος μνήμης |
|---|---|---|
| 0000 | 0000 | Device Non-bufferable |
| 0001 | 0001 | Device Bufferable |
| 0010 | 0010 | Normal Non-cachable Non-bufferable |
| 0011 | 0011 | Normal Non-cachableBufferable |
| 1010 | 0110 | Write-through No-allocate |
| 1110 | 0110 | Write-through Read-allocate |
| 1010 | 1110 | Write-through Write-allocate |
| 1110 | 1110 | Write-through Read/Write-allocate |
| 1011 | 0111 | Write back No-allocate |
| 1111 | 0111 | Write back Read-allocate |
| 1011 | 1111 | Write back Write-allocate |
| 1111 | 1111 | Write back Read/Write-allocate |

6) Ο τύπος προστασίας της συναλλαγής ορίζεται από τα σήματα AWPROT και ARPROT για εγγραφή και ανάγνωση αντίστοιχα και κωδικοποιείται σύμφωνα με τον Πίνακας 10.

**Πίνακας 10: Κωδικοποίηση AWPROT/ARPROT**

| AWPROT/ARPROT | Περιγραφή |
|---|---|
| Bit 0 | 0: Μη προνομιούχα πρόσβαση (Non-privileged access)<br>1: Προνομιούχα πρόσβαση (Privileged access) |
| Bit 1 | 0: Ασφαλής πρόσβαση (Secure access)<br>1: Μη ασφαλής πρόσβαση (Non-secure access) |
| Bit 2 | 0: Πρόσβαση για δεδομένα (Data access)<br>1: Πρόσβαση για εντολή (Instruction access) |

7) Τέλος περιγράφεται η κωδικοποίηση των σημάτων επιβεβαίωσης RRESP και BRESP στον Πίνακας 11.

**Πίνακας 11: Κωδικοποίηση RRESP/BRESP**

| RRESP/BRESP (Δυαδική) | Τύπος επιβεβαίωσης | Περιγραφή τύπου επιβεβαίωσης |
|---|---|---|
| 00 | OKAY | Η συναλλαγή ήταν επιτυχής |
| 01 | EXOKAY | Η αποκλειστική συναλλαγή ήταν επιτυχής. (Η αποκλειστική συναλλαγή Exclusive access περιγράφεται στην ενότητα A7.2 του [1]) |
| 10 | SLVERR | Η πρόσβαση στον AXI Slave ήταν επιτυχής αλλά ο AXI Slave θέλει να ενημερώσει τον AXI Master για λάθος. Πχ η FIFO που γράφεται έχει γεμίσει ή γίνεται απόπειρα εγγραφής σε διεύθυνση που μπορεί μόνο να διαβαστεί (read-only) |
| 11 | DECERR | Λάθος αποκωδικοποίησης. Το λάθος αυτό συμβαίνει όταν σε ένα δίαυλο επικοινωνίας (interconnect) δεν είναι δυνατό να γίνει πρόσβαση στον AXI Slave που ζητήθηκε. |

## 2.5 Περιγραφή του AXI4-Lite

Στις ενότητες 2.2, 2.3 και 2.4 περιγράφεται το πρωτόκολλο AXI4 με το μεγαλύτερο μέρος των δυνατοτήτων του ούτως ώστε να επιτευχθεί επικοινωνία με στοιχεία μνήμης. Σε αυτή την ενότητα περιγράφεται ένα υποσύνολο του πρωτοκόλλου, το AXI4-Lite. Το AXI4-Lite είναι ιδανικό για την πραγματοποίηση πιο απλών συναλλαγών, συνήθως από και προς ένα αρχείο καταχωρητών (register file), το οποίο δεν χρειάζεται το σύνολο όλων των δυνατοτήτων του AXI4 που περιγράφεται στις προηγούμενες ενότητες.

Τα βασικά χαρακτηριστικά που διαχωρίζουν το Lite είναι τα εξής:

- Το μήκος της συναλλαγής είναι πάντα 1.
- Το μέγεθος της εγγραφής/ανάγνωσης είναι πάντα ίσο με το πλάτος του δίαυλου δεδομένων. Πχ για 32-bit δίαυλο, οι λέξεις δεδομένων είναι 32-bit. Το AXI4-Lite υποστηρίζει πλάτος δίαυλου ίσο με 32-bit ή 64-bit.
- Αποκλειστικές (exclusive) συναλλαγές δεν υποστηρίζονται

Έτσι η AXI4-Lite έκδοση χρησιμοποιεί ένα υποσύνολο σημάτων της διεπαφής AXI4, το οποίο περιγράφεται στον Πίνακας 12.

**Πίνακας 12: Σήματα διεπαφήςAXI4-Lite**

| Καθολικά | Κανάλι διεύθυνσης εγγραφής | Κανάλι δεδομένων εγγραφής | Κανάλι επιβεβαίωσης εγγραφής | Κανάλι διεύθυνσης ανάγνωσης | Κανάλι δεδομένων ανάγνωσης |
|---|---|---|---|---|---|
| ACLK | AWVALID | WVALID | BVALID | ARVALID | RVALID |
| ARESETn | AWREADY | WREADY | BREADY | ARREADY | RREADY |
| - | AWADDR | WDATA | BRESP | ARADDR | RDATA |
| - | AWPROT | WSTRB | - | ARPROT | RRESP |

Από τον Πίνακας 12 προκύπτουν οι εξής συνεπαγωγές:

1) Για τα σήματα BRESP και RRESP δεν ορίζεται η τιμή EXOKAY, εφόσον η αποκλειστική συναλλαγή δεν υποστηρίζεται από το Lite.
2) Τα σήματα AWLEN, ARLEN δεν έχουν νόημα χρήσης εφόσον οι συναλλαγές είναι πάντα μήκους 1.
3) Τα σήματα AWSIZE, ARSIZE δεν έχουν νόημα χρήσης εφόσον οι συναλλαγές χρησιμοποιούν πάντα όλο το εύρος του δίαυλου επικοινωνίας (32-bit ή 64-bit).
4) Τα σήματα AWBURST, ARBURST δεν έχουν νόημα χρήσης εφόσον οι συναλλαγές είναι πάντα μήκους 1.
5) Τα σήματα AWLOCK, ARLOCK δεν έχουν νόημα χρήσης εφόσον δεν υποστηρίζονται αποκλειστικές συναλλαγές.
6) Τα σήματα AWCACHE, ARCACHE δεν έχουν νόημα χρήσης εφόσον οι συναλλαγές με χρήση του AXI4-Lite γίνονται συνήθως από και προς αρχείο καταχωρητών.
7) Τα σήματα WLAST, RLAST δεν έχουν νόημα χρήσης εφόσον οι συναλλαγές είναι πάντα μήκους 1.
8) Τα σήματα WSTRB δίνουν τη δυνατότητα το αρχείο καταχωρητών να υποστηρίζει 8-bit ή/και 16-bit καταχωρητές.

Ένα καίριο θέμα είναι η επικοινωνία AXI Master και AXI Slave υπομονάδων που χρησιμοποιούν AXI4 και AXI4-Lite στο ίδιο σύστημα. Στον παρακάτω πίνακα φαίνεται η συμβατότητα μεταξύ των διαφόρων περιπτώσεων.

| AXI Master | AXI Slave | Συμβατότητα |
|---|---|---|
| AXI4 | AXI4 | Πλήρως συμβατά |
| AXI4 | AXI4-Lite | Χρειάζεται προσαρμογή |
| AXI4-Lite | AXI4 | Πλήρως συμβατά |
| AXI4-Lite | AXI4-Lite | Πλήρως συμβατά |

Η μόνη περίπτωση που θέλει προσαρμογή είναι η δεύτερη όπου ένας AXI4 Master επικοινωνεί με ένα AXI4-Lite Slave. Ο AXI Master κατά την επιβεβαίωση εγγραφής χρειάζεται το σήμα BID ώστε να γνωρίζει σε ποια συναλλαγή αναφέρεται η επιβεβαίωση. Ο AXI4-Lite Slave θα πρέπει λοιπόν να στείλει στον Master την πληροφορία που χρειάζεται.

## 2.6 Περιγραφή του AXI4-Stream

Μια άλλη παραλλαγή του πρωτοκόλλου AXI4 είναι το AXI4-Stream [2]. Πρόκειται για την πραγματοποίηση συναλλαγών μεγάλου όγκου δεδομένων χωρίς την επιπλέον πληροφορία ελέγχου. Χρησιμοποιείται απλώς για μεταφορά πληροφορίας (payload), δηλαδή δεδομένων από έναν AXI Master σε έναν AXI Slave.

Η σειρά δεδομένων που μεταφέρεται ονομάζεται stream. Το stream αποτελείται από μια σειρά bytes τα οποία μπορούν να κατηγοριοποιηθούν ως εξής:

- Data byte. Byte έγκυρης πληροφορίας που μεταφέρεται από έναν AXI Master σε έναν AXI Slave.
- Position byte. Byte μη έγκυρης πληροφορίας που χρησιμοποιείται για την εξακρίβωση της ακριβής θέσης του κάθε byte μέσα στο stream.
- Null byte. Byte μη έγκυρης πληροφορίας που δεν έχει κάποια χρηστική αξία.

Η διάκριση των bytes γίνεται σύμφωνα με τα σήματα ελέγχου που περιλαμβάνει η AXI4-Stream διεπαφή. Στον Πίνακας 13 περιγράφονται τα σήματα της διεπαφής.

**Πίνακας 13: Σήματα διεπαφής AXI4-Stream**

| Σήμα | Πηγή Σήματος | Περιγραφή Σήματος |
|---|---|---|
| ACLK | Πηγή ρολογιού (DCM ή PLL ή MMCM) | Ρολόι συστήματος. |
| ARESETn | Λογική αρχικοποίησης | Σήμα αρχικοποίησης συστήματος (ACTIVE LOW). |
| TVALID | AXI Master | Δηλώνει ότι τα δεδομένα είναι έγκυρα και έτοιμα προς μεταφορά (ενότητα 2.4). |
| TREADY | AXI Slave | Δηλώνει ότι ο AXI Slave είναι έτοιμος να λάβει τα δεδομένα (ενότητα 2.4). |
| TDATA | AXI Master | Δεδομένα προς αποστολή (payload). |
| TSTRB | AXI Master | Τα σήματα write strobes δηλώνουν ποια bytes είναι έγκυρα δεδομένα κατά την εγγραφή μέσω ενός δίαυλου. Ο AXI Master συσχετίζει ένα σήμα strobe για κάθε byte δεδομένων. Πχ για 32-bit λέξεις δεδομένων αντιστοιχίζονται 4 σήματα strobe, για 64-bit λέξεις αντιστοιχίζονται 8 σήματα strobe κοκ. |
| TKEEP | AXI Master | Το σήμα αυτό χρησιμοποιείται σε συνδυασμό με το TSTRB σήμα για να κατηγοριοποιηθεί το αντίστοιχο byte σε data byte, position byte ή null byte. (Βλέπε Πίνακας 14). |
| TLAST | AXI Master | Το σήμα αυτό δηλώνει το τέλος ενός πακέτου. Ένα stream μπορεί να διαιρεθεί σε πακέτα πληροφορίας. Κάθε TLAST σήμα σηματοδοτεί την λήξη ενός πακέτου (και την αρχή του επόμενου). |

| | | |
|---|---|---|
| TID | AXI Master | Ετικέτα ταυτότητας του stream. Μπορεί να χρησιμοποιηθεί για τον διαχωρισμό των πακέτων (πχ να δηλώνει την αφετηρία του πακέτου) |
| TDEST | AXI Master | Δηλώνει τον προορισμό του πακέτου. Το σήμα έχει νόημα όταν το σύστημα αποτελείται από δρομολογητές ή δίαυλο διασύνδεσης (interconnect) όπου η πληροφορία δρομολογείται για να φτάσει στον προορισμό της. |
| TUSER | AXI Master | Σήμα χρήστη. Μπορεί να χρησιμοποιηθεί για την μεταφορά επιπλέον πληροφορίας, ανάλογα με τις απαιτήσεις του συστήματος. |

**Πίνακας 14: Κατηγοριοποίηση bytes**

| TKEEP | TSTRB | Κατηγοριοποίηση του byte |
|---|---|---|
| 0 | 0 | NULL BYTE |
| 0 | 1 | Reserved |
| 1 | 0 | POSITION BYTE |
| 1 | 1 | DATA BYTE |

Κλείνοντας την περιγραφή του πρωτοκόλλου βλέπουμε τις εξαιρετικές δυνατότητες που προσφέρει το AXI4 πρωτόκολλο για την επικοινωνία υπομονάδων μέσα στο ολοκληρωμένο κύκλωμα. Ο κύριος επεξεργαστής μπορεί να επικοινωνεί με περιφερειακή λογική γρήγορα και αποδοτικά. Στην περίπτωση που θέλει να επικοινωνήσει με μια περιφερειακή μνήμη, η χρήση του AXI4 είναι ιδανική και παρέχει όλη την πληροφορία και τον έλεγχο που ορίζουν οι ανάγκες της αγοράς. Για επικοινωνία με μια απλή υπομονάδα όπως ένα αρχείο καταχωρητών που διαμορφώνει (configures) ή διαχειρίζεται (controls) μια υπομονάδα (πχ μια ALU ή έναν DSP) τότε το AXI4-Lite είναι ιδανικό. Και τέλος, αν η ανάγκη ενός συστήματος είναι η αποδοτική μεταφορά μεγάλου όγκου πληροφορίας τότε το AXI4-Stream, μπορεί να χρησιμοποιηθεί ως ιδανική επιλογή.

# 3. ΠΕΡΙΓΡΑΦΗ ΥΛΙΚΟΥ (HARDWARE)

Σε αυτό το κεφάλαιο περιγράφεται του κομμάτι του συστήματος που υλοποιείται στην προγραμματιζόμενη λογική (FPGA). Όπως αναφέρθηκε και στην εισαγωγή, ο επεξεργαστής επικοινωνεί, μέσω AXI4, με τη προγραμματιζόμενη λογική ώστε να επεξεργαστεί δεδομένα.
Στο Σχήμα 9 παρουσιάζεται το μπλοκ διάγραμμα του συστήματος. Το σύστημα αποτελείται από τις εξής υπο-μονάδες:

- ZYNQ επεξεργαστής (ZYNQ Processing System). Κεντρική μονάδα επεξεργασίας (CPU) η οποία “διευθύνει” το σύστημα. Το λογισμικό το οποίο αναπτύχθηκε και εκτελείται από τον επεξεργαστή, για την πραγματοποίηση συναλλαγών με AXI4 πρωτόκολλο καθώς και την επιθυμητή επεξεργασία δεδομένων, περιγράφεται στο Κεφάλαιο 5.
- Μονάδα επεξεργασίας δεδομένων (CORE). Αποτελεί τον πυρήνα της επεξεργασίας δεδομένων.
- Αρχείο Καταχωρητών (Register File). Μονάδα καταχωρητών για τον έλεγχο (control) και την διαμόρφωση (configuration) της μονάδας επεξεργασίας δεδομένων.
- AXI Άμεση Προσπέλαση Μνήμης (AXI Direct Memory Access). Έτοιμη υπο-μονάδα της Xilinx που πραγματοποιεί την μεταφορά δεδομένων από τη μνήμη προς την μονάδα επεξεργασίας δεδομένων και αντίστροφα.
- AXI Stream FIFOs. Έτοιμα IP cores της Xilinx για την προσωρινή αποθήκευση (buffering) των επεξεργασμένων, και μη, δεδομένων.
- AXI δίαυλος διασύνδεσης (AXI Interconnect). Έτοιμη υπο-μονάδα της Xilinx για την διασύνδεση του επεξεργαστή με το αρχείο καταχωρητών και με την AXI DMA υπο-μονάδα.
- Μνήμη δεδομένων (Memory). Κεντρική μνήμη με την οποία επικοινωνεί ο επεξεργαστής για την ανάγνωση και εγγραφή των επεξεργασμένων, και μη, δεδομένων.

Στην συνέχεια του κεφαλαίου ακολουθεί η περιγραφή και η λειτουργικότητα των παραπάνω υπο-μονάδων.

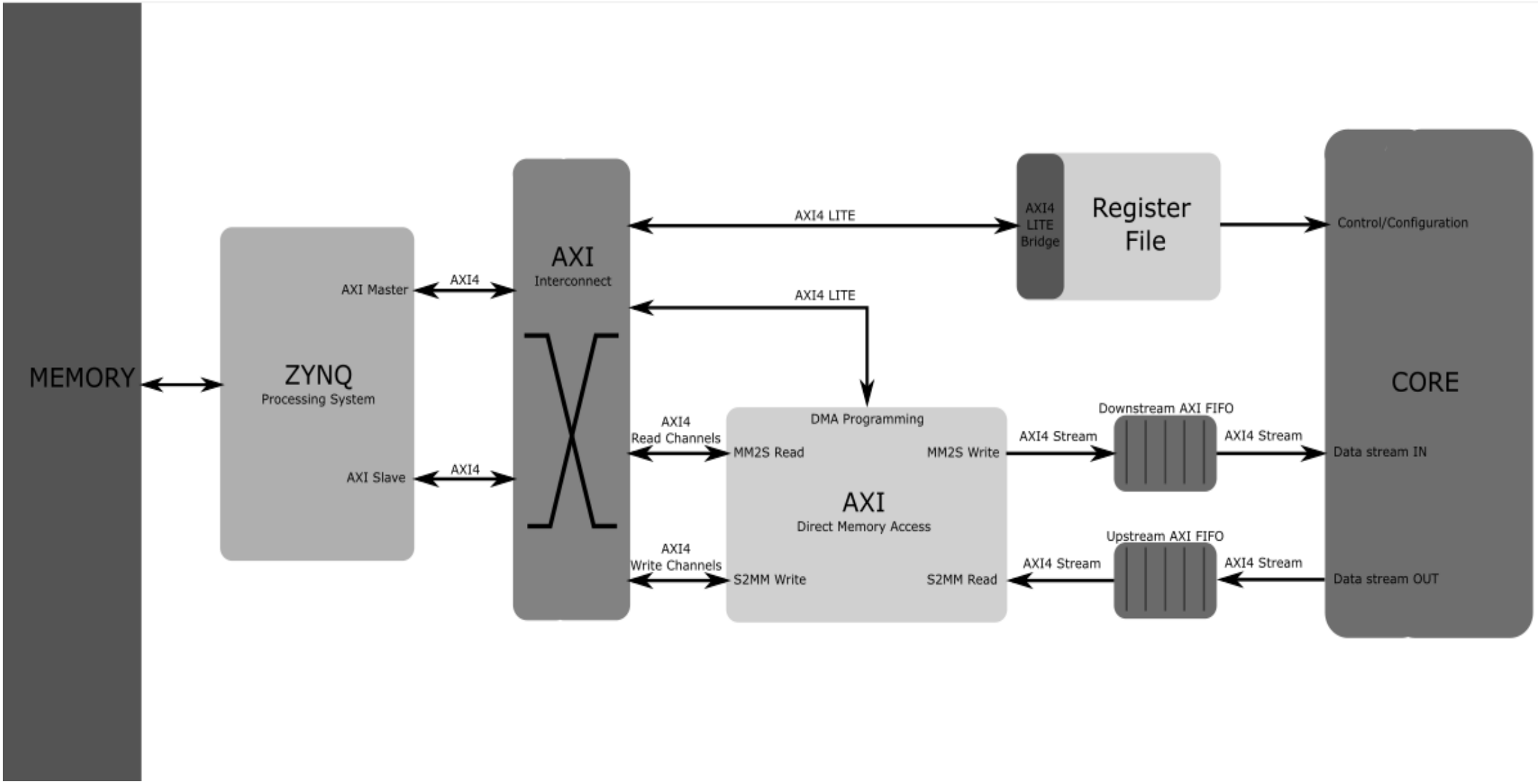
## 3.1 Περιγραφή λειτουργίας συστήματος (Overview)

Για την επεξεργασία των δεδομένων, ο επεξεργαστής εκτελεί μια αλληλουχία λειτουργιών:

1) Εγγραφή αρχείου καταχωρητών για τον έλεγχο και τη διαμόρφωση της επεξεργασίας. Για τον έλεγχο και την διαμόρφωση οποιασδήποτε επεξεργαστικής μονάδας (core), χρησιμοποιείται ένα αρχείο καταχωρητών στο οποίο ο κάθε καταχωρητής χρησιμοποιείται για ένα σκοπό. Ο ZYNQ εκτελεί εγγραφές προς το αρχείο καταχωρητών μέσω AXI4 LITE συναλλαγών, ανανεώνοντας τις τιμές των καταχωρητών. Οι τιμές των καταχωρητών λαμβάνονται από το CORE με αποτέλεσμα τον έλεγχο των λειτουργιών του.
2) Ανάγνωση μνήμης και μεταφορά δεδομένων για επεξεργασία. Για την μεταφορά των δεδομένων από τη μνήμη προς το CORE χρησιμοποιείται η μονάδα AXI DMA. Ο ZYNQ αρχικά προγραμματίζει το AXI DMA, δηλαδή ορίζει την διεύθυνση από την οποία θα αρχίσει την ανάγνωση και το μέγεθος (σε bytes) της συναλλαγής, δηλαδή πόσα δεδομένα θα μεταφερθούν. Μόλις εκτελεστεί ο προγραμματισμός, μέσω AXI4 LITE συναλλαγών, η μονάδα DMA εκτελεί μια AXI4 ανάγνωση (MM2S Read channels) από τη μνήμη, μέσω της AXI Slave διεπαφής του ZYNQ. Τα δεδομένα μεταφέρονται στην DMA μονάδα και στέλνονται προς το CORE μέσω AXI4 Stream συναλλαγών (MM2S Write).
3) Επεξεργασία δεδομένων. Εφόσον το αρχείο καταχωρητών έχει ενημερωθεί με την επιθυμητή λειτουργία (η οποία αντιστοιχεί στις διάφορες τιμές των καταχωρητών), και

τα δεδομένα μεταφερθούν προς το CORE (Data stream IN), η επεξεργασία εκτελείται.

4) Μεταφορά επεξεργασμένων δεδομένων και εγγραφή στη μνήμη. Για την μεταφορά των δεδομένων προς τη μνήμη ακολουθείται η ίδια διαδικασία. Ο ZYNQ, μέσω της AXI Master διεπαφής, προγραμματίζει με AXI4 LITE συναλλαγές, την μονάδα AXI DMA, δηλαδή ορίζει την διεύθυνση στην οποία θα εγγραφούν τα επεξεργασμένα δεδομένα, καθώς και το μέγεθος σε bytes της συναλλαγής. Μόλις εκτελεστεί ο προγραμματισμός, τα επεξεργασμένα δεδομένα μεταφέρονται μέσω του AXI DMA (S2MM Read), το οποίο εκτελεί μια AXI4 εγγραφή (S2MM Write) προς την AXI Slave διεπαφή του ZYNQ.




**Σχήμα 9: Τελικό Σύστημα**

## 3.2 Αρχείο καταχωρητών (Register File)

Το αρχείο καταχωρητών αποτελεί την μονάδα ελέγχου και διαμόρφωσης του CORE. Αποτελείται από μια σειρά καταχωρητών (registers) οι οποίοι χρησιμοποιούνται από το CORE για την εκτέλεση της επιθυμητής επεξεργασίας. Ο αριθμός των καταχωρητών εξαρτάται από τις ανάγκες του CORE και μπορεί να επιλεγεί μέσω generic σταθεράς.
Κάθε καταχωρητής είναι 32-bit και κάθε bit του καταχωρητή δυνητικά χρησιμοποιείται για την ενεργοποίηση ή απενεργοποίηση μιας συγκεκριμένης λειτουργίας του CORE. Κάθε καταχωρητής του αρχείου μπορεί επίσης να διαβαστεί από τον επεξεργαστή, και ανάλογα την τιμή αυτή να υποδηλώνει μια κατάσταση λειτουργίας του CORE (status), π.χ. μια τιμή να δηλώνει ότι η επεξεργασία βρίσκεται σε εξέλιξη, μια άλλη τιμή να δηλώνει ότι η επεξεργασία ολοκληρώθηκε ή ότι η επεξεργασία απέτυχε κοκ. Αυτό σημαίνει ότι οι συγκεκριμένοι καταχωρητές κατάστασης εγγράφονται από το CORE το οποίο εξάγει την κατάσταση στην οποία βρίσκεται (στην παρούσα εργασία οι καταχωρητές χρησιμοποιούνται μόνο για τον έλεγχο και τη διαμόρφωση του CORE και όχι για την ανάγνωση της κατάστασής του).
Η παρούσα εργασία έχει ως κύριο άξονα μελέτης την διασύνδεση του επεξεργαστή με μια μονάδα επεξεργασίας στο υλικό (hardware) μέσω του πρωτοκόλλου επικοινωνίας AXI4. Για τον σκοπό αυτό αναπτύχθηκε μια απλή μονάδα επεξεργασίας δεδομένων, με εξαιρετικά περιορισμένες δυνατότητες. Η μονάδα αυτή χρησιμοποιεί τρεις καταχωρητές οι οποίοι περιγράφονται στους παρακάτω πίνακες.

**Πίνακας 15: Καταχωρητής λειτουργίας (Operation register)**

| Αριθμός bit | Όνομα πεδίου | Περιγραφή λειτουργίας | Τιμές λειτουργίας |
|---|---|---|---|
| 31-12 | RESERVED | - | - |
| 11 | Multiply | Ενεργοποίηση πολλαπλασιασμού δεδομένων | 0: Απενεργοποίηση πολλαπλασιασμού<br>1: Ενεργοποίηση πολλαπλασιασμού |
| 10 | Shift variable | Ολίσθηση δεδομένων κατά μεταβλητή τιμή | 0: Απενεργοποίηση ολίσθησης με μεταβλητή τιμή<br>1: Ενεργοποίηση ολίσθησης με μεταβλητή τιμή |
| 9 | Load upper immediate enable | Ολίσθηση δεδομένων κατά 16 bits | 0: Απενεργοποίηση ολίσθησης<br>1: Ενεργοποίηση ολίσθησης |
| 8-4 | Shift amount | Μέγεθος ολίσθησης | 0: Μηδενική ολίσθηση<br>1: Ολίσθηση κατά ένα bit<br>2: Ολίσθηση κατά δυο bits<br>…<br>31: Ολίσθηση κατά 31 |

| | | | |
|---|---|---|---|
| | | | bits |
| 3-0 | ALU operation | Λειτουργία εκτέλεσης | 0000: Λογική ολίσθηση αριστερά<br>0010: Λογική ολίσθηση δεξιά<br>0011: Αριθμητική ολίσθηση δεξιά<br>0110: Set less than<br>0111: Set less than (μη προσημασμένη)<br>1000: Πρόσθεση<br>1001: Πρόσθεση (μη προσημασμένη)<br>1010: Αφαίρεση<br>1011: Αφαίρεση (μη προσημασμένη)<br>1100: Λογική πράξη AND<br>1101: Λογική πράξη OR<br>1110: Λογική πράξη XOR<br>1111: Λογική πράξη NOR |

**Πίνακας 16: Καταχωρητής ελέγχου (Control register)**

| Αριθμός bit | Όνομα πεδίου | Περιγραφή λειτουργίας | Τιμές λειτουργίας |
|---|---|---|---|
| 31-4 | RESERVED | - | - |
| 3 | Bypass | Όταν αυτό το bit τεθεί σε ΄1΄ τότε τα δεδομένα τα οποία στέλνονται προς το CORE δεν επιδέχονται καμία επεξεργασία και μεταφέρονται ανέπαφα στη FIFO για μεταφορά προς τη μνήμη. Διαφορετικά τα δεδομένα επεξεργάζονται κανονικά. | 0: Κανονική λειτουργία επεξεργασίας<br>1: Ενεργοποίηση bypass |
| 2-1 | RESERVED | - | - |
| 0 | Calculate | Ενεργοποίηση επεξεργασίας δεδομένων | 0: Απενεργοποίηση επεξεργασίας<br>1: Ενεργοποίηση |

| | | | |
|---|---|---|---|
| | | | επεξεργασίας |

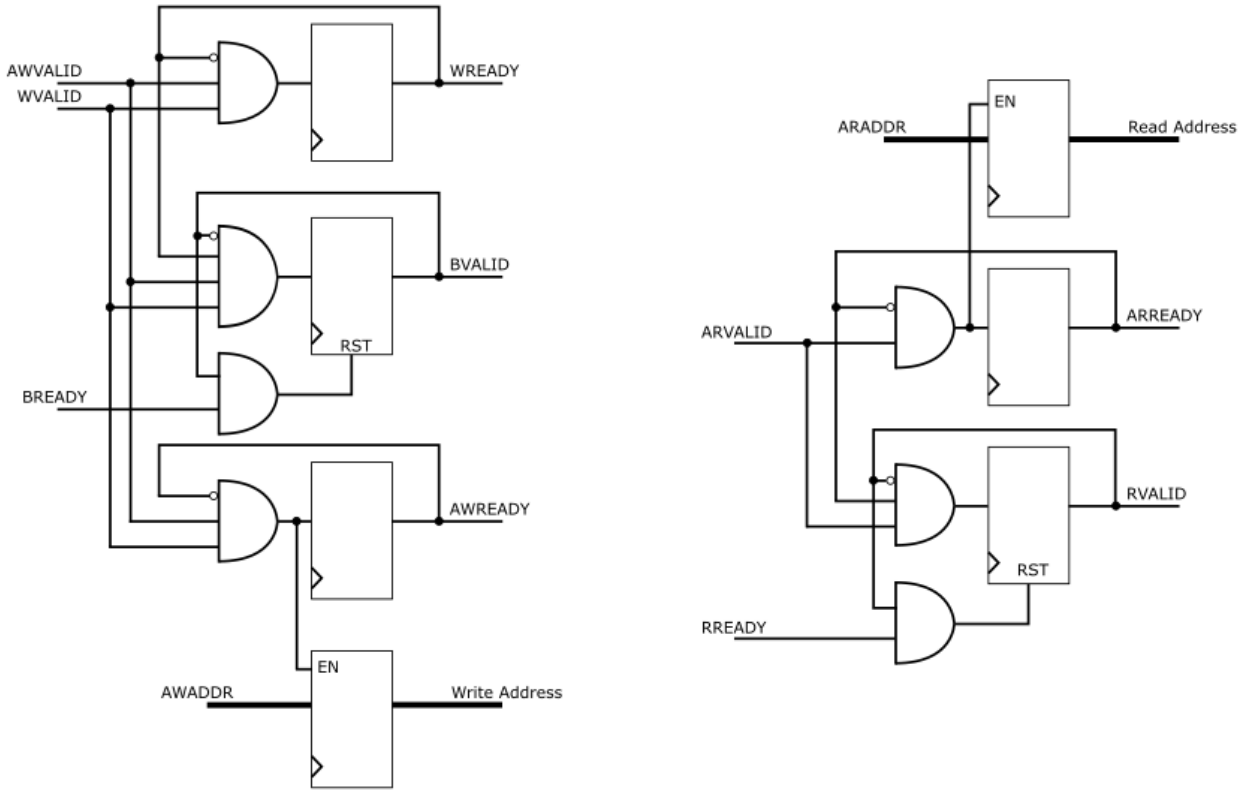
**Πίνακας 17: Καταχωρητής σταθεράς (Constant register)**

| Αριθμός bit | Όνομα πεδίου | Περιγραφή λειτουργίας | Τιμές λειτουργίας |
|---|---|---|---|
| 31-0 | Constant | Η σταθερά αυτή χρησιμοποιείται για την εκτέλεση επεξεργασίας των δεδομένων. Π.χ. αν η σταθερά τεθεί ίση με 5 και η λειτουργία τεθεί ως πρόσθεση, τότε τα δεδομένα που θα ληφθούν στο CORE θα αυξήσουν την τιμή τους κατά 5. | - |

Το αρχείο καταχωρητών αποτελεί μια μονάδα AXI Slave η οποία επικοινωνεί με AXI4 LITE. Για την σωστή εγγραφή και ανάγνωση των καταχωρητών είναι απαραίτητη η χρήση των σημάτων ελέγχου (Πίνακας 12) και η δημιουργία μιας λογικής “γεφύρωσης” ή αλλιώς προσαρμογής αυτών των σημάτων ώστε να επιτευχθεί η ανάγνωση/εγγραφή του σωστού καταχωρητή. Στο Σχήμα 10 παρουσιάζεται η λογική των σημάτων ελέγχου του AXI4 LITE για την σωστή λήψη των δεδομένων και διευθύνσεων.

**ΣΗΜΕΙΩΣΗ:** Η εγγραφή δεδομένων στην επιθυμητή διεύθυνση (Write address) θα πραγματοποιηθεί μόνο όταν τα σήματα AWVALID, WVALID, AWREADY και WREADY είναι στο λογικό ΄1΄.

**ΣΗΜΕΙΩΣΗ:** Η ανάγνωση δεδομένων από την επιθυμητή διεύθυνση (Read address) θα πραγματοποιηθεί μόνο όταν τα σήματα ARVALID, RVALID και ARREADY είναι στο λογικό ΄1΄.




**Σχήμα 10: Λογική σημάτων ελέγχου AXI4 LITE εγγραφής και ανάγνωσης**

## 3.3 Μονάδα επεξεργασίας δεδομένων

Όπως ήδη αναφέρθηκε, σκοπός της εργασίας είναι η επικοινωνία του επεξεργαστή με την προγραμματιζόμενη λογική μέσω AXI4. Η μονάδα επεξεργασίας δεδομένων που υλοποιήθηκε χρησιμοποιείται απλώς για την επαλήθευση της μεταφοράς δεδομένων και του ελέγχου/διαμόρφωσης του CORE. Το CORE αποτελείται από μια Αριθμητική/ Λογική μονάδα (ALU) η οποία εκτελεί αριθμητικές και λογικές πράξεις στα δεδομένα, καθώς επίσης και ολισθήσεις όπως ακριβώς περιγράφονται στον Πίνακας 15. Η ALU δέχεται στην μια είσοδο την τιμή του καταχωρητή σταθεράς (Πίνακας 17) και στην άλλη δέχεται τα δεδομένα που μεταφέρονται από την μνήμη. Τα δεδομένα αποτελούν πίνακες στους οποίους εκτελούνται οι επιλεγμένες πράξεις.

Η είσοδος και η έξοδος των δεδομένων εκτελείται σύμφωνα με το AXI4 Stream το οποίο, όπως εξηγείται στο κεφάλαιο 2, διαθέτει τον δίαυλο δεδομένων (data bus) και τα σήματα ελέγχου ready/valid για την εκτέλεση “χειραψίας” και τον έλεγχο της ροής δεδομένων (Σχήμα 5).

## 3.4 AXI Stream FIFOs

Οι δύο FIFOs που φαίνονται στο Σχήμα 9 επικοινωνούν με το CORE μέσω AXI4 Stream διεπαφών. Η κάθε FIFO εξάγει ένα σήμα ready προς την λογική από την οποία

δέχεται δεδομένα, δηλώνοντας την διαθεσιμότητα σε ελεύθερες θέσεις για προσωρινή αποθήκευση. Επίσης, η FIFO εξάγει προς την λογική στην οποία θα στείλει τα δεδομένα ένα σήμα valid το οποίο δηλώνει ότι υπάρχουν έγκυρα δεδομένα μέσα στη FIFO. Κανείς θα μπορούσε να θεωρήσει τα εξής:

Ready = not Full και Valid = not Empty

Δηλαδή, όταν η FIFO δεν είναι γεμάτη, είναι έτοιμη να δεχθεί δεδομένα και όταν η FIFO δεν είναι άδεια, τότε διαθέτει έγκυρα δεδομένα. Ο έλεγχος ροής δεδομένων από και προς το CORE εκτελείται με βάση τα σήματα ready/valid των FIFOs και το CORE δεν εκτελεί κανένα είδος ελέγχου ροής δεδομένων (Data flow control).

## 3.5 AXI Άμεση Προσπέλαση Μνήμης (Direct Memory Access)

Όπως αναφέρθηκε στην ενότητα 3.1, η μονάδα AXI DMA είναι υπεύθυνη για την μεταφορά των δεδομένων από και προς τη μνήμη, μέσω του ZYNQ. Το IP Core της Xilinx όπως περιγράφεται στο [3] διαθέτει ένα εσωτερικό αρχείο καταχωρητών για τον προγραμματισμό του, καθώς επίσης και δυο κανάλια δεδομένων. Το κανάλι εγγραφής (Stream to Memory Mapped – S2MM) και το κανάλι ανάγνωσης (Memory Mapped to Stream – MM2S) όπως φαίνεται στο Σχήμα 9.

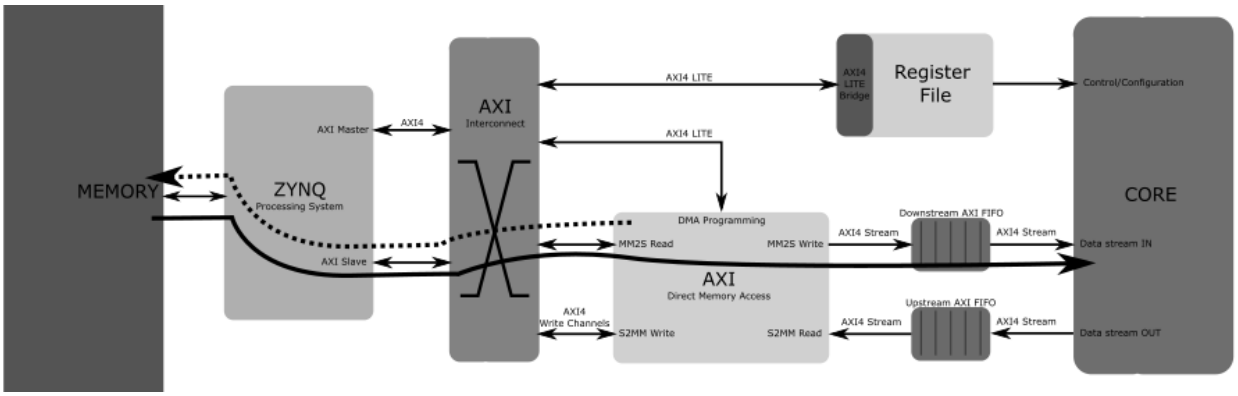
### Ανάγνωση μνήμης και μεταφορά δεδομένων προς το CORE

Για την μεταφορά δεδομένων από τη μνήμη προς το οποιοδήποτε CORE απαιτείται ο προγραμματισμός του καναλιού ανάγνωσης. Πιο συγκεκριμένα, χρειάζεται η ενεργοποίηση του καναλιού, ο ορισμός της διεύθυνσης ανάγνωσης και τέλος το μέγεθος της συναλλαγής, δηλαδή πόσα bytes θα μεταφερθούν από τη μνήμη. Στο [3] αναφέρονται οι διευθύνσεις και τα πεδία των καταχωρητών που διαθέτει η μονάδα. Για την έναρξη της συναλλαγής θα πρέπει να εκτελεστεί:

1) Μια συναλλαγή εγγραφής AXI4 LITE στην διεύθυνση καταχωρητή 0x00 που αντιστοιχεί στο MM2S Control register και να γραφτεί η τιμή 0x00000001 η οποία ενεργοποιεί το κανάλι ανάγνωσης.
2) Μια συναλλαγή εγγραφής AXI4 LITE στην διεύθυνση καταχωρητή 0x18 που αντιστοιχεί στο MM2S Source address register και να γραφτεί η τιμή της επιθυμητής διεύθυνσης ανάγνωσης από τη μνήμη.
3) Μια συναλλαγή εγγραφής AXI4 LITE στην διεύθυνση καταχωρητή 0x28 που αντιστοιχεί στο MM2S Length register και να γραφτεί η τιμή του επιθυμητού μήκους ανάγνωσης (σε πλήθος bytes).

ΣΗΜΕΙΩΣΗ: Οι διευθύνσεις καταχωρητών που αναφέρονται παραπάνω αποτελούν την διαφορά (offset) από την διεύθυνση βάση (base address) του DMA (βλέπε ενότητα 3.6).

Με την εκτέλεση και της τρίτης εγγραφής, το DMA έχει προγραμματιστεί και ξεκινά η συναλλαγή ανάγνωσης. Από την διεπαφή MM2S Read (η οποία αποτελεί μια AXI Master διεπαφή η οποία διαθέτει μόνο τα κανάλια δεδομένων ανάγνωσης και διεύθυνσης ανάγνωσης) στέλνεται η διεύθυνση ανάγνωσης η οποία προγραμματίστηκε (διακεκομμένη γραμμή στο Σχήμα 11). Ο ZYNQ, ως AXI Slave, λαμβάνει την διεύθυνση και διαβάζει την μνήμη. Τα δεδομένα μεταφέρονται από την AXI Slave διεπαφή του ZYNQ και εισέρχονται στην MM2S Read διεπαφή του DMA. Το DMA δρομολογεί τα δεδομένα στην διεπαφή MM2S Write η οποία αποτελεί μια AXI4 Stream διεπαφή πάνω στην οποία μεταφέρονται τα δεδομένα προς επεξεργασία (πλήρης γραμμή στο Σχήμα 11).




**Σχήμα 11: DMA Ανάγνωσης και μεταφορά δεδομένων προς το CORE**

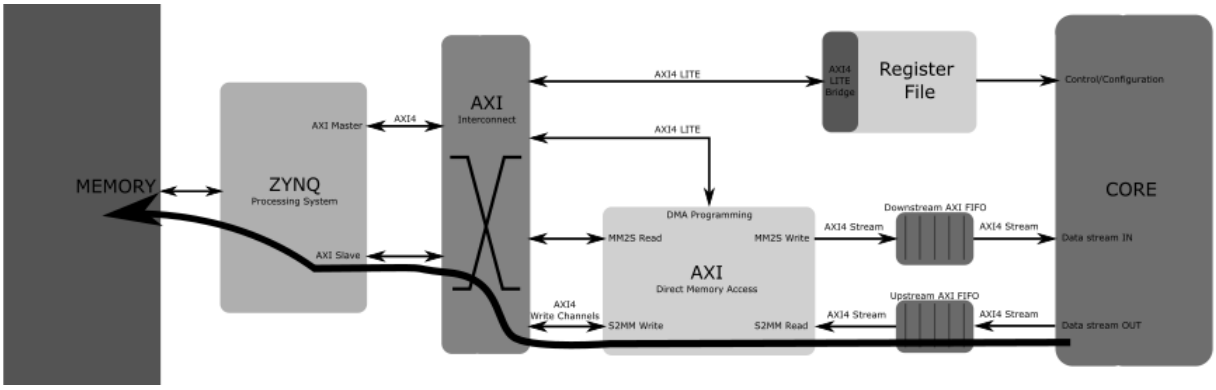
## Μεταφορά δεδομένων από το CORE και εγγραφή στη μνήμη

Για την μεταφορά δεδομένων από οποιοδήποτε CORE προς τη μνήμη απαιτείται ο προγραμματισμός του καναλιού εγγραφής. Πιο συγκεκριμένα, χρειάζεται η ενεργοποίηση του καναλιού, ο ορισμός της διεύθυνσης εγγραφής και τέλος το μέγεθος της συναλλαγής, δηλαδή πόσα bytes θα μεταφερθούν προς τη μνήμη. Στο [3] αναφέρονται οι διευθύνσεις και τα πεδία των καταχωρητών που διαθέτει η μονάδα. Για την έναρξη της συναλλαγής θα πρέπει να εκτελεστεί:

1) Μια συναλλαγή εγγραφής AXI4 LITE στην διεύθυνση καταχωρητή 0x30 που αντιστοιχεί στο S2MM Control register και να γραφτεί η τιμή 0x00000001 η οποία ενεργοποιεί το κανάλι εγγραφής.
2) Μια συναλλαγή εγγραφής AXI4 LITE στην διεύθυνση καταχωρητή 0x48 που αντιστοιχεί στο S2MM Destination address register και να γραφτεί η τιμή της επιθυμητής διεύθυνσης εγγραφής στη μνήμη.
3) Μια συναλλαγή εγγραφής AXI4 LITE στην διεύθυνση καταχωρητή 0x58 που αντιστοιχεί στο S2MM Length register και να γραφτεί η τιμή του επιθυμητού μήκους εγγραφής (σε πλήθος bytes).

ΣΗΜΕΙΩΣΗ: Οι διευθύνσεις καταχωρητών που αναφέρονται παραπάνω αποτελούν την διαφορά (offset) από την διεύθυνση βάση (base address) του DMA (βλέπε ενότητα 3.6).

Με την εκτέλεση και της τρίτης εγγραφής, το DMA έχει προγραμματιστεί και ξεκινά η συναλλαγή εγγραφής. Από την διεπαφή S2MM Read (η οποία αποτελεί μια AXI4 Stream διεπαφή) ξεκινά η ανάγνωση των δεδομένων από το CORE. Τα δεδομένα δρομολογούνται μέσω του DMA στην διεπαφή S2MM Write (η οποία αποτελεί μια AXI Master διεπαφή που διαθέτει μόνο τα κανάλια διεύθυνσης εγγραφής, δεδομένων εγγραφής και επιβεβαίωσης εγγραφής). Το DMA εκτελεί μια συναλλαγή εγγραφής προς την διεπαφή AXI Slave του ZYNQ και στέλνει τη διεύθυνση εγγραφής η οποία προγραμματίστηκε, καθώς και τα δεδομένα που διαβάστηκαν από το CORE. Ο ZYNQ ως AXI Slave λαμβάνει τη διεύθυνση εγγραφής και τα δεδομένα εγγραφής και στέλνει τα δεδομένα στη μνήμη. Τέλος, ο ZYNQ επιστρέφει την επιβεβαίωση εγγραφής πίσω στο DMA για την ολοκλήρωση της συναλλαγής.




**Σχήμα 12: Μεταφορά δεδομένων από το CORE και DMA Εγγραφής**

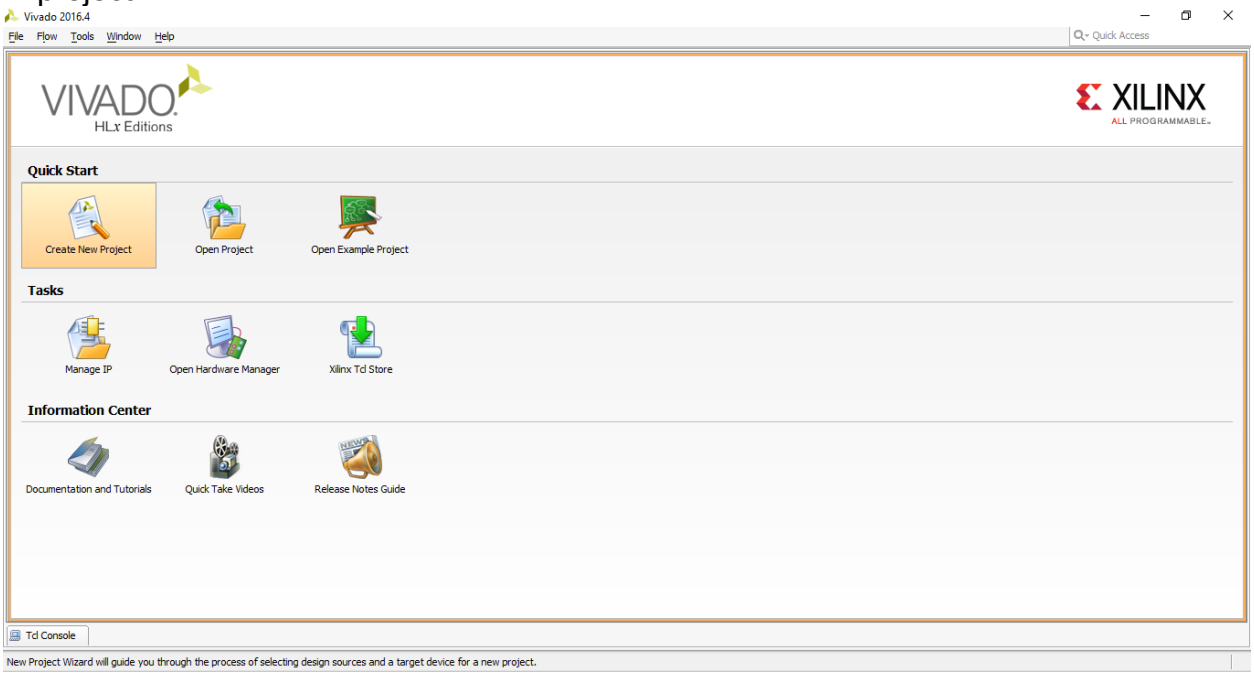
## 3.6 AXI δίαυλος διασύνδεσης

Ολοκληρώνοντας την περιγραφή του συστήματος, θα πρέπει να χρησιμοποιηθεί η υπομονάδα της Xilinx AXI Interconnect η οποία επιτυγχάνει την δρομολόγηση της πληροφορίας προς την σωστή υπομονάδα.
Πιο συγκεκριμένα, η AXI Master διεπαφή του ZYNQ θα πρέπει να είναι σε θέση να εκτελεί συναλλαγές προς το αρχείο καταχωρητών και προς την διεπαφή προγραμματισμού του AXI DMA. Για την σωστή δρομολόγηση των συναλλαγών προς την σωστή κατεύθυνση ανατίθεται μια περιοχή διευθύνσεων (address space) η οποία αφορά στην εκάστοτε υπομονάδα. Για παράδειγμα, για την μονάδα του αρχείου καταχωρητών ορίστηκε η περιοχή διευθύνσεων 0x43C00000 μέχρι 0x43C0FFF. Αυτό σημαίνει ότι αν το AXI Interconnect δεχθεί μια διεύθυνση μέσα σε αυτή την περιοχή, σημαίνει ότι η συναλλαγή αφορά στο αρχείο καταχωρητών. Τα LS bytes αυτής της διεύθυνσης κάνουν σαφή την επιλογή του συγκεκριμένου καταχωρητή που θα διαβαστεί ή θα γραφτεί πχ η διεύθυνση 0x43C00000 αντιστοιχεί στον πρώτο καταχωρητή του αρχείου καταχωρητών. Η διεύθυνση 0x43C00004 αντιστοιχεί στον δεύτερο καταχωρητή του αρχείου καταχωρητών κοκ. Η αρχική διεύθυνση 0x43C00000 ονομάζεται διεύθυνση βάση (base address) του αρχείου καταχωρητών και η διαφορά από αυτή την τιμή ονομάζεται offset.
Αντίστοιχα, για τον προγραμματισμό του DMA ανατέθηκε άλλη περιοχή διευθύνσεων και η επιλογή των καταχωρητών προγραμματισμού γίνεται προσθέτοντας την διαφορά (offset) στην διεύθυνση βάσης.
Τέλος, το AXI Interconnect είναι υπεύθυνο για την δρομολόγηση των συναλλαγών από τις διεπαφές MM2S Read και S2MM Write προς την διεπαφή AXI Slave του ZYNQ. Η υπομονάδα είναι υπεύθυνση για την επιβολή προτεραιότητας (arbitration) και την δρομολόγηση της αντίστοιχης συναλλαγής προς τον ZYNQ.

# 4. ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΣΤΟ VIVADO

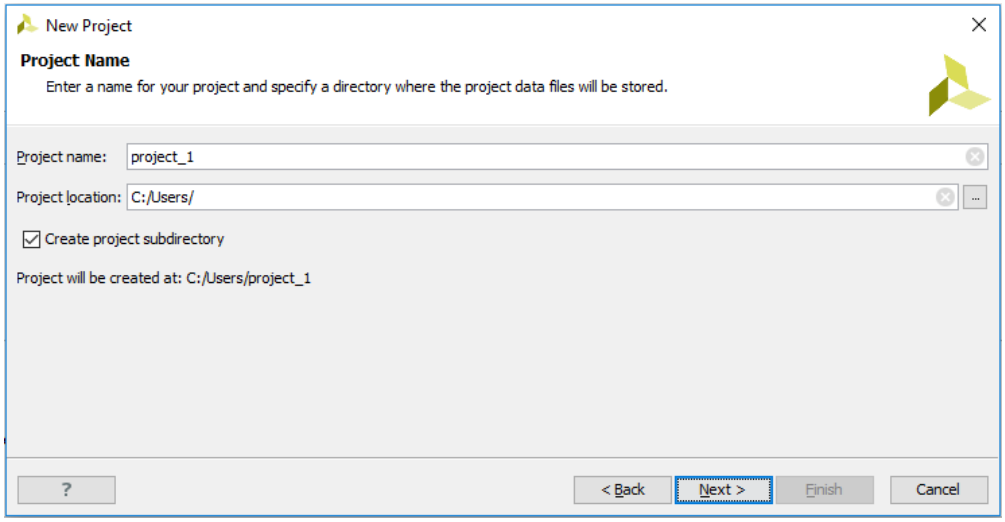
Σε αυτό το κεφάλαιο περιγράφεται η χρήση του εργαλείου Vivado της Xilinx και όλα τα βήματα για τον σχεδιασμό και την υλοποίηση του συστήματος [4] [5]. Το Vivado με την χρήση των έτοιμων IP υποσυστημάτων (IP Cores) δίνει τη δυνατότητα για εύκολη και γρήγορη σχεδίαση συστημάτων.
Παρακάτω ακολουθούν οδηγίες χρήσης του Vivado για τη σχεδίαση και υλοποίησης ενός συστήματος. Η περιγραφή σχεδίασης αναφέρεται στην έκδοση 2016.4 του Vivado.

1) Πρώτο βήμα είναι η δημιουργία ενός νέου project στο Vivado. Στο αρχικό παράθυρο όπως φαίνεται στην Εικόνα 1, πατώντας το “Create New Project” δημιουργείται το project.



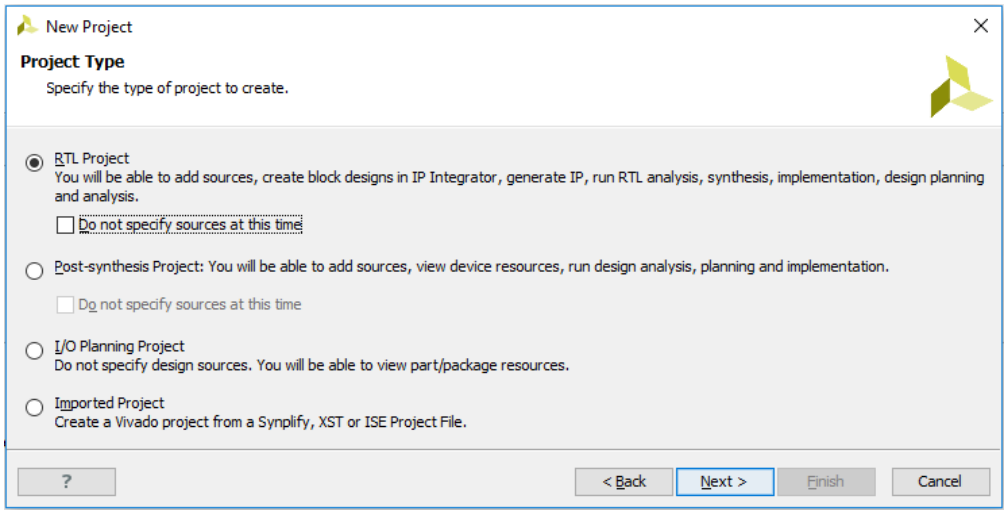

**Εικόνα 1: Δημιουργία νέου project**

2) Στο παράθυρο που εμφανίζεται πατήστε “Next”. Στο επόμενο παράθυρο ζητούνται πληροφορίες για το όνομα του project και την θέση αποθήκευσης του project. Επιλέξτε το όνομα που επιθυμείτε και την τοποθεσία και σιγουρευτείτε ότι το “Create project subdirectory” είναι επιλεγμένο (ώστε να δημιουργηθεί ένας καθολικός φάκελος του project). Πατήστε “Next”.



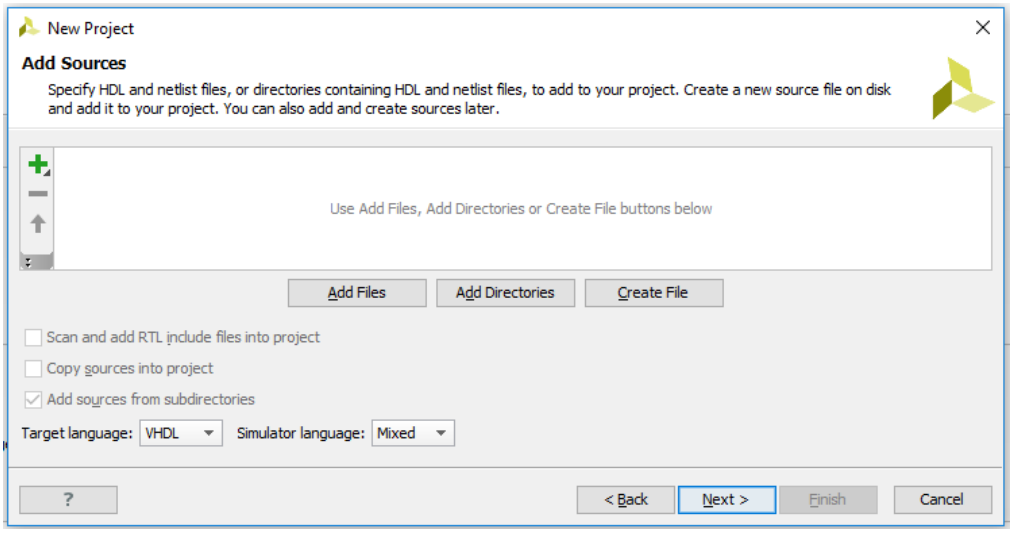

**Εικόνα 2: Ονοματοθεσία και φάκελος αποθήκευσης του project**

3) Στο επόμενο βήμα ζητείται ο τύπος του project. Επιλέξτε “RTL Project” και μην επιλέξτε το “Do not specify sources at this time”, ώστε να μπορέσετε να επιλέξετε τώρα τα αρχεία περιγραφής υλικού. Πατήστε “Next”.



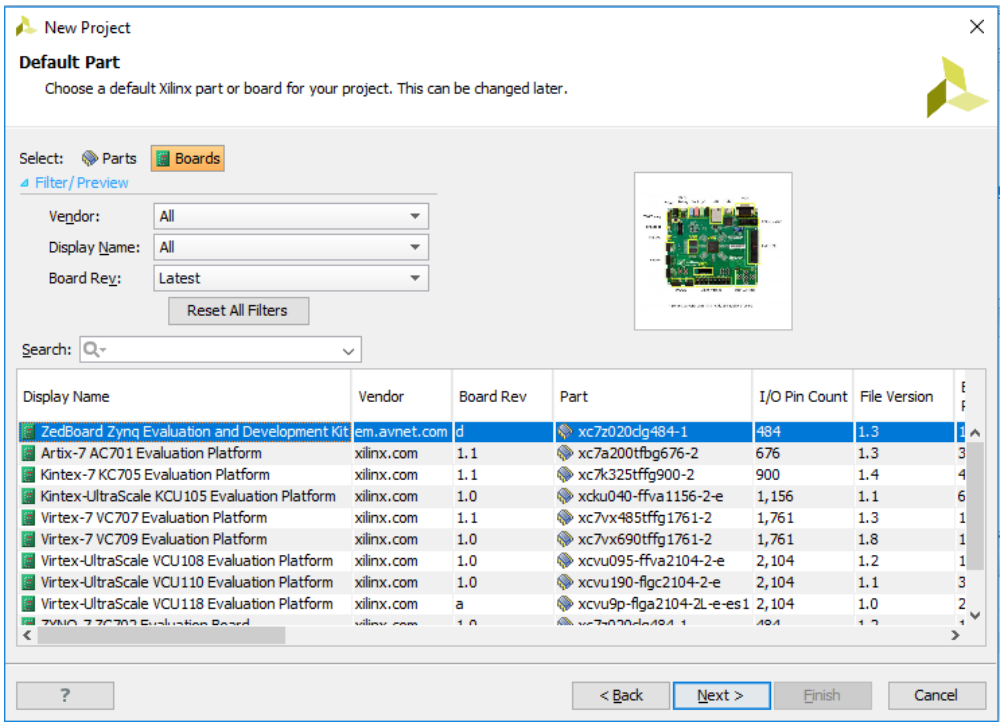

**Εικόνα 3: Επιλογή τύπου project**

4) Στο επόμενο παράθυρο μπορούμε να επιλέξουμε τα αρχεία κώδικα περιγραφής υλικού που θέλουμε να συμπεριλάβουμε στο project. Πατώντας το πλήκτρο “Add files” επιλέξτε τα αρχεία. Στο πεδίο “Target language” επιλέξτε την γλώσσα περιγραφής υλικού (VHDL ή Verilog) και στο πεδίο “Simulator language” επιλέξτε “Mixed”. Πατήστε “Next”. Τα επόμενα παράθυρα είναι ίδιας λειτουργίας αλλά αναφέρονται στην επιλογή έτοιμων IP (Existing IP) και σε αρχεία περιορισμών (Constraint files). Πατήστε “Next”.




**Εικόνα 4: Επιλογή αρχείων κώδικα και γλώσσας περιγραφής υλικού**

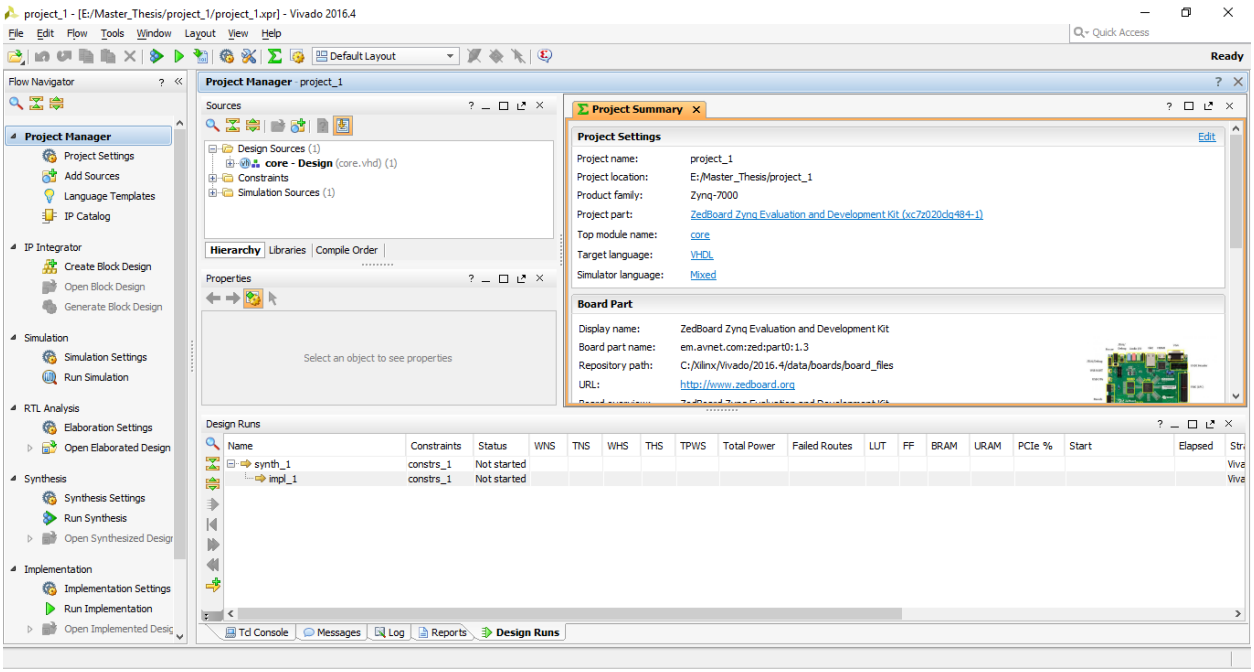
5) Το επόμενο βήμα είναι η επιλογή του FPGA ή/και της πλακέτας στην οποία θα υλοποιηθεί το σύστημα. Επιλέγοντας “Parts” ή “Boards” η λίστα επιλογών ανανεώνεται ανάλογα με την επιλογή. Επιλέξτε το FPGA ή το Board που επιθυμείτε και πατήστε “Next”. Στην παρούσα εργασία επιλέχθηκε το “Zedboard Zynq Evaluation and Development Kit” για την χρήση του ενσωματωμένου επεξεργαστή ARM (Zynq Processing System) όπως αναφέρεται στα επόμενα βήματα. Στο επόμενο παράθυρο εμφανίζεται η περίληψη των επιλογών που έγιναν μέχρι τώρα. Μπορείτε να ελέγξετε ότι είναι οι επιθυμητές επιλογές και πατήστε “Finish” για την τελική δημιουργία του project.




**Εικόνα 5: Επιλογή FPGA/Board υλοποίησης**

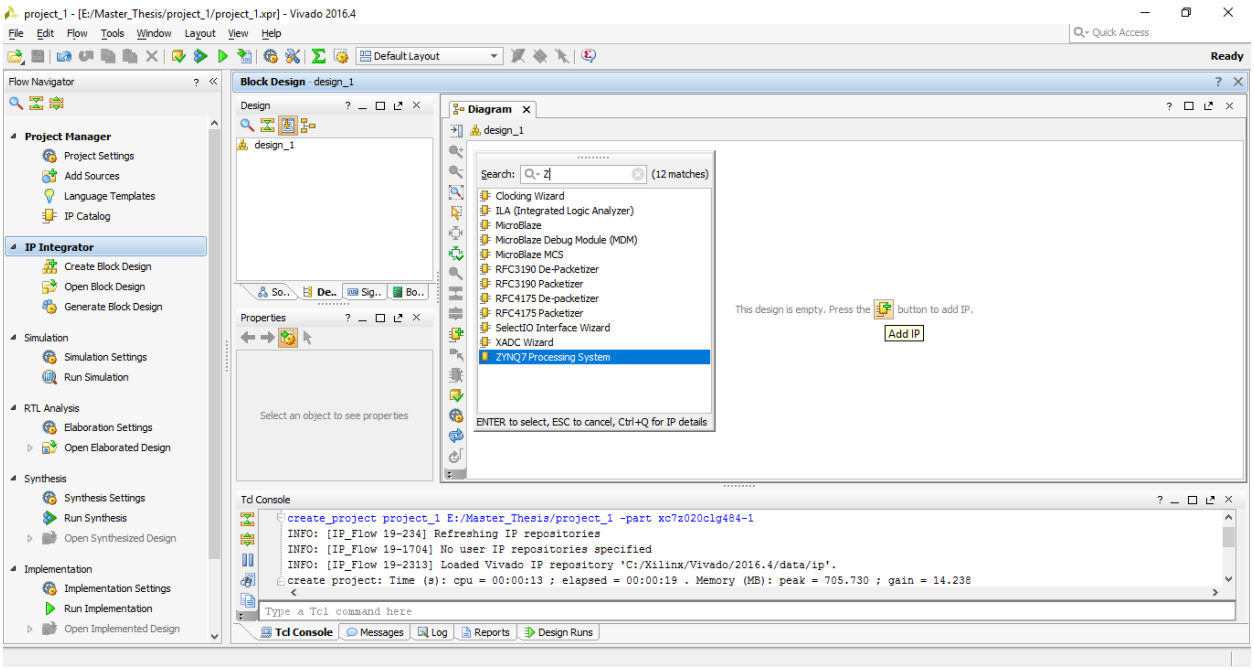
6) Με την ολοκλήρωση της δημιουργίας του project ανοίγει το κεντρικό παράθυρο εργασιών του Vivado όπως φαίνεται στην Εικόνα 6. Στο υπο-παράθυρο “Sources” θα πρέπει να εμφανίζονται τα αρχεία κώδικα που επιλέξατε (εφόσον επιλέξατε). Για την δημιουργία του συστήματος πατήστε το πλήκτρο “Create Block Design” που

βρίσκεται στο πεδίο “IP Integrator” του υπο-παραθύρου “Flow Navigator” στα αριστερά. Επιλέξτε το όνομα του block design και πατήστε OK. Σε αυτό το σημείο δημιουργείται ένα κενό block design στο οποίο μπορούμε να εισάγουμε IP και υποσυστήματα που τα περιγράφουν αρχεία κώδικα (custom components), να τα συνδέσουμε και να υλοποιηθεί το σύστημα.



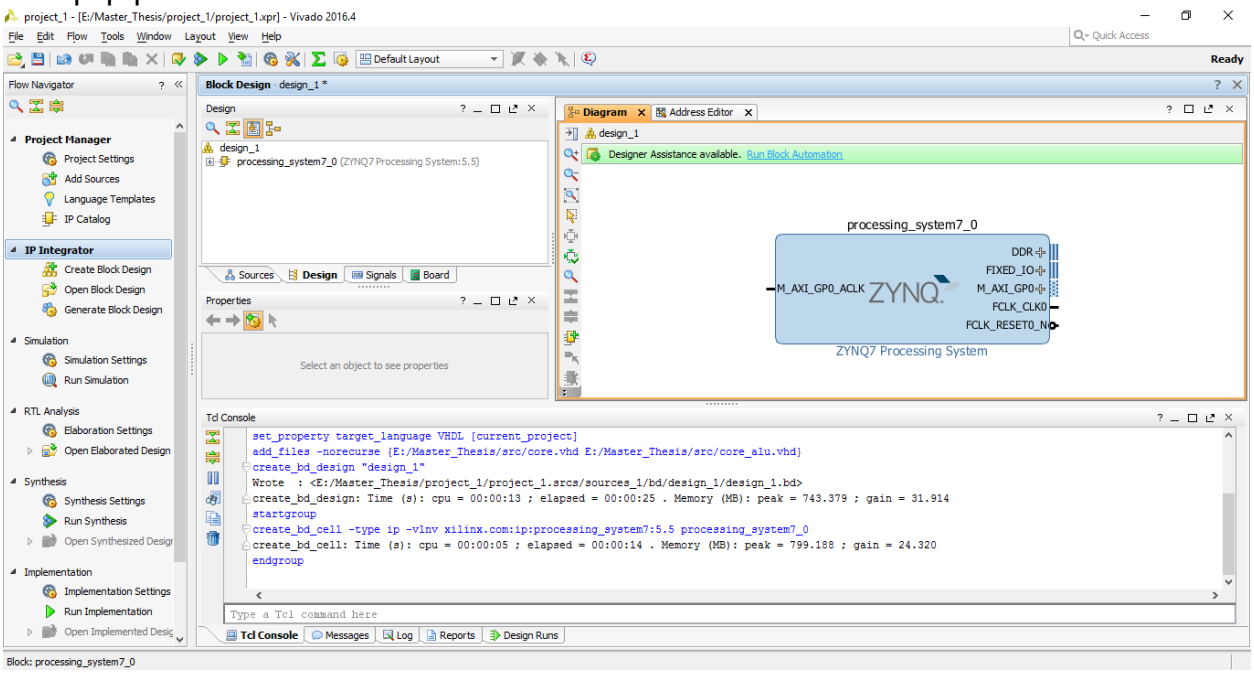

**Εικόνα 6: Κεντρικό παράθυρο του Vivado**

7) Από το βήμα αυτό και έπειτα περιγράφεται η σχεδίαση του συστήματος, όπως παρουσιάστηκε και επεξηγήθηκε στο Σχήμα 9 του Κεφαλαίου 3.
Αρχικά εισάγουμε τον Zynq επεξεργαστή πατώντας στο πλήκτρο Add IP (ή πατώντας Ctrl + I) και πληκτρολογώντας “Zynq7 Processing System” όπως φαίνεται στην Εικόνα 7. Κάντε διπλό κλικ όταν εμφανιστεί στη λίστα επιλογής.



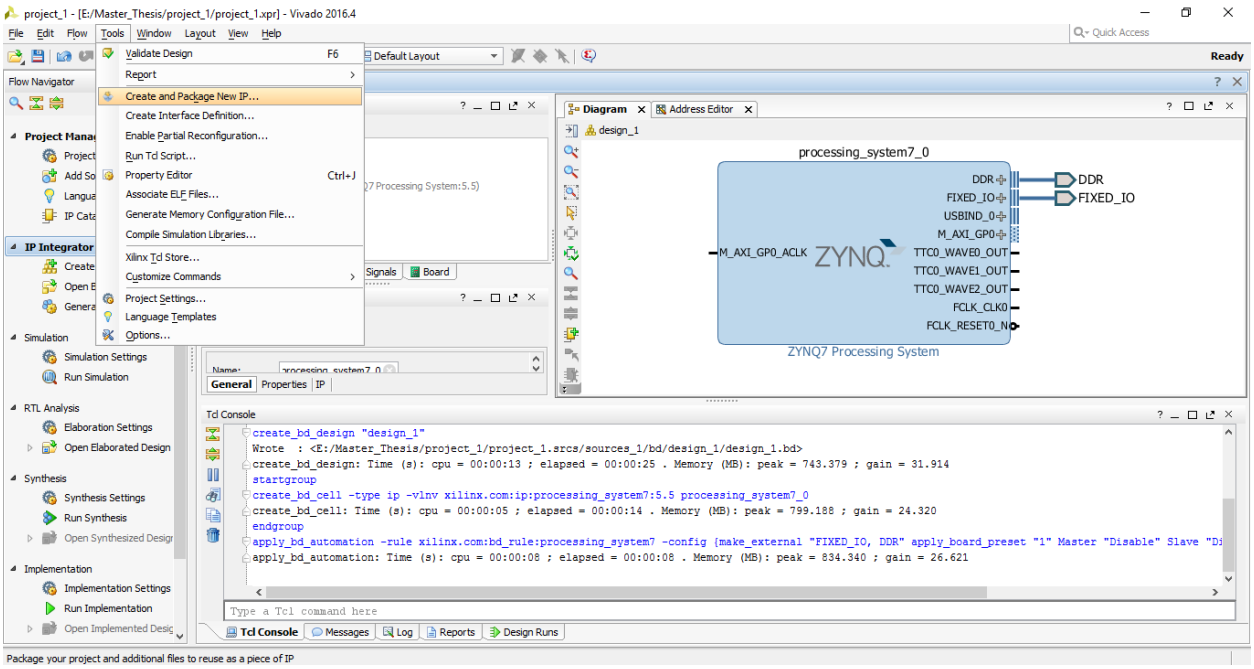

**Εικόνα 7: Εισαγωγή Zynq επεξεργαστή στο σύστημα**

8) Όπως φαίνεται στην Εικόνα 8 ο επεξεργαστής έχει εμφανιστεί στο block design και επίσης εμφανίστηκε η επιλογή “Run Block Automation” από πάνω. Πατήστε πάνω στην επιλογή “Run block Automation” (και έπειτα OK στο παράθυρο που εμφανίζεται, αφήνοντας τις επιλογές στις αρχικές, προκαθορισμένες τιμές τους). Πατώντας αυτή την επιλογή του Vivado εκτελεί αυτόματα εσωτερικές και εξωτερικές συνδέσεις μεταξύ υπο-συστημάτων του Zynq επεξεργαστή, ώστε να μπορεί να επικοινωνεί με περιφερειακά.



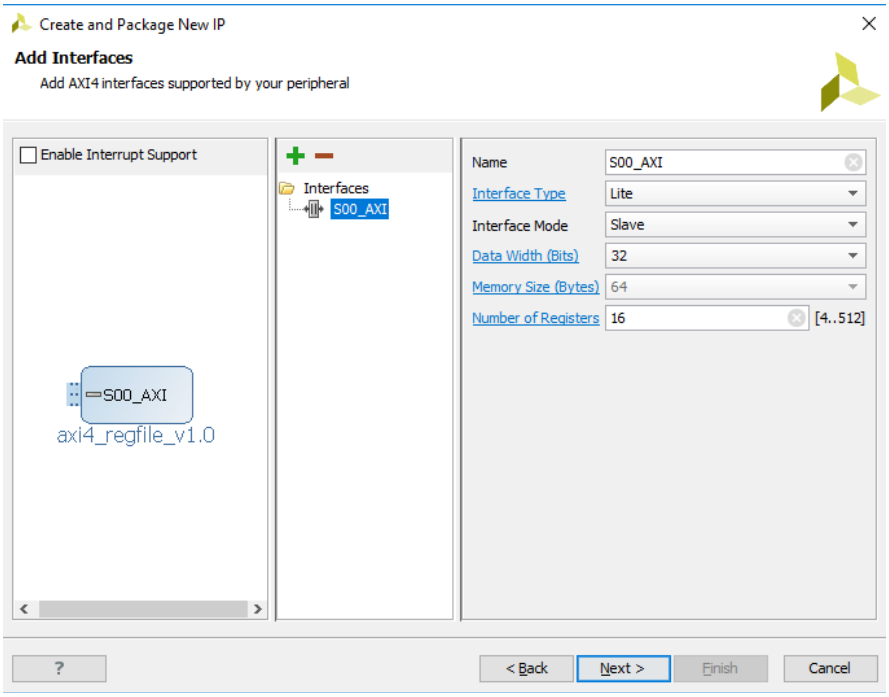

**Εικόνα 8: Διαμόρφωση του Zynq επεξεργαστή**

9) Εφόσον ο επεξεργαστής είναι έτοιμος, είμαστε σε θέση να δημιουργήσουμε τα περιφερειακά υπο-συστήματα. Για το αρχείο καταχωρητών εργαζόμαστε ως εξής. Από τη λίστα Tools επιλέγουμε το “Create and Package a New IP” (Εικόνα 9). Στο παράθυρο που εμφανίζεται επιλέξτε “Create a new AXI4 peripheral” και πατήστε Next. Επιλέξτε το όνομα και την θέση αποθήκευσης του νέου IP core και πατήστε Next.




**Εικόνα 9: Δημιουργία περιφερειακού**

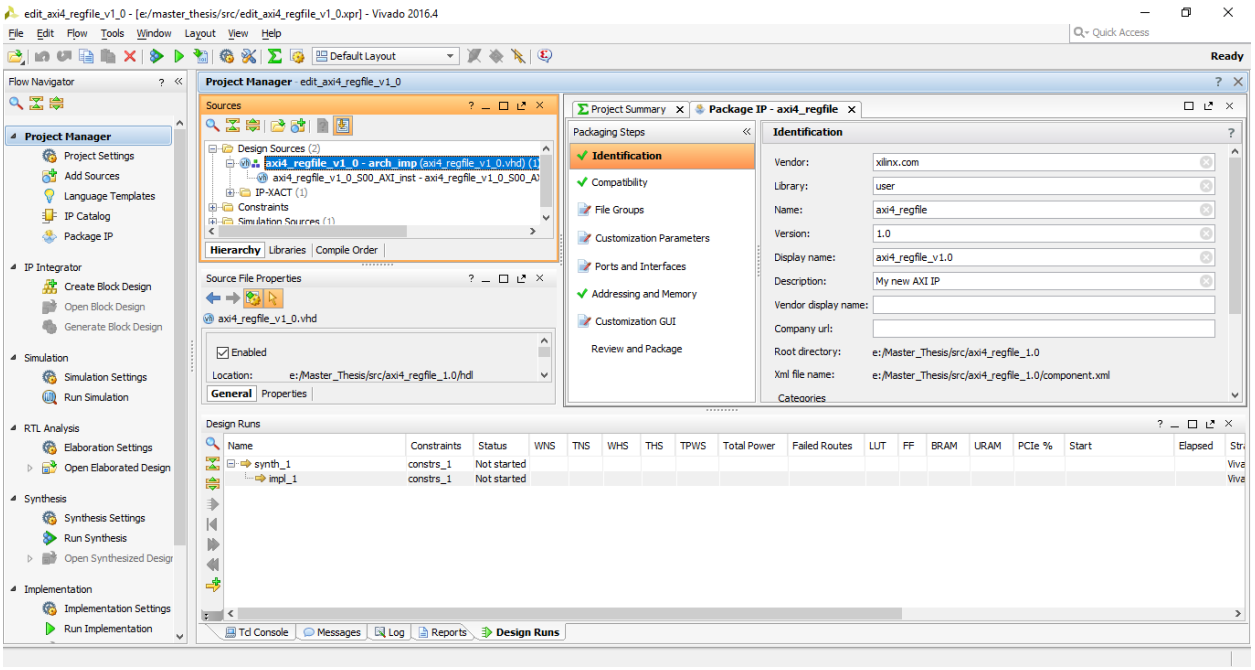
10) Στο παράθυρο της Εικόνα 10 μπορούμε να επιλέξουμε τα χαρακτηριστικά του περιφερειακού. Στην παρούσα εργασία επιλέγουμε ένα AXI4 Lite Slave 32-bit Interface με π.χ. 16 καταχωρητές. Έτσι, ο Zynq επεξεργαστής, ως AXI Master, θα μπορεί να εκτελεί συναλλαγές ανάγνωσης και εγγραφής προς το περιφερειακό.




**Εικόνα 10: Διαμόρφωση περιφερειακού**

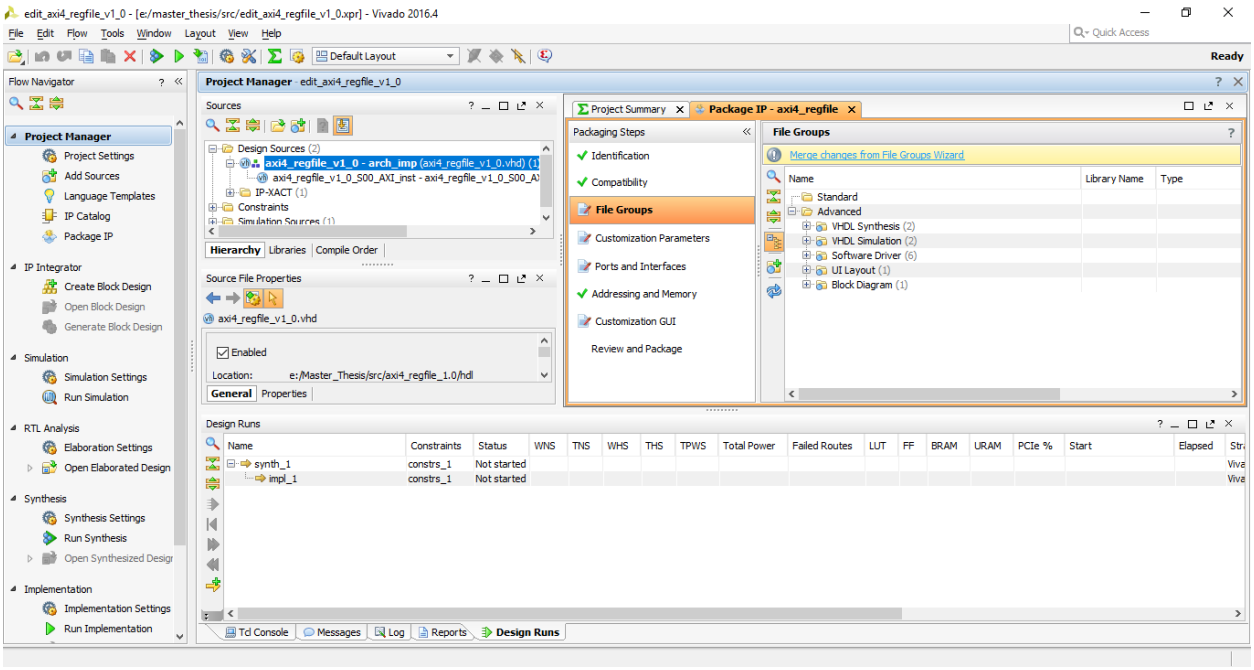
11) Αφού ολοκληρώσετε τις επιλογές διαμόρφωσης και πατήστε Next, θα εμφανιστεί παράθυρο περίληψης των επιλογών καθώς και η επιλογή να τροποποιήσετε τον κώδικα του περιφερειακού. Επιλέξτε “Edit IP” και Finish. Θα ανοίξει ένα νέο παράθυρο επεξεργασίας του περιφερειακού όπως φαίνεται στην Εικόνα 11. Στο παράθυρο Sources φαίνονται τα αρχεία κώδικα του περιφερειακού, και μπορεί ο χρήστης να τα επεξεργαστεί κάνοντας διπλό κλικ πάνω τους. Στην παρούσα

εργασία τα αρχεία επεξεργάστηκαν έτσι ώστε να εισάγουμε την λογική “γεφύρωσης” του AXI4 Lite που περιγράφτηκε στο προηγούμενο κεφάλαιο (αντί της λογικής που δημιουργήθηκε αυτόματα από το εργαλείο) και επίσης, τροποποιήθηκε το interface τους ώστε οι έξοδοι των καταχωρητών να είναι διαθέσιμες προς χρήση από την μονάδα επεξεργασίας δεδομένων (core).

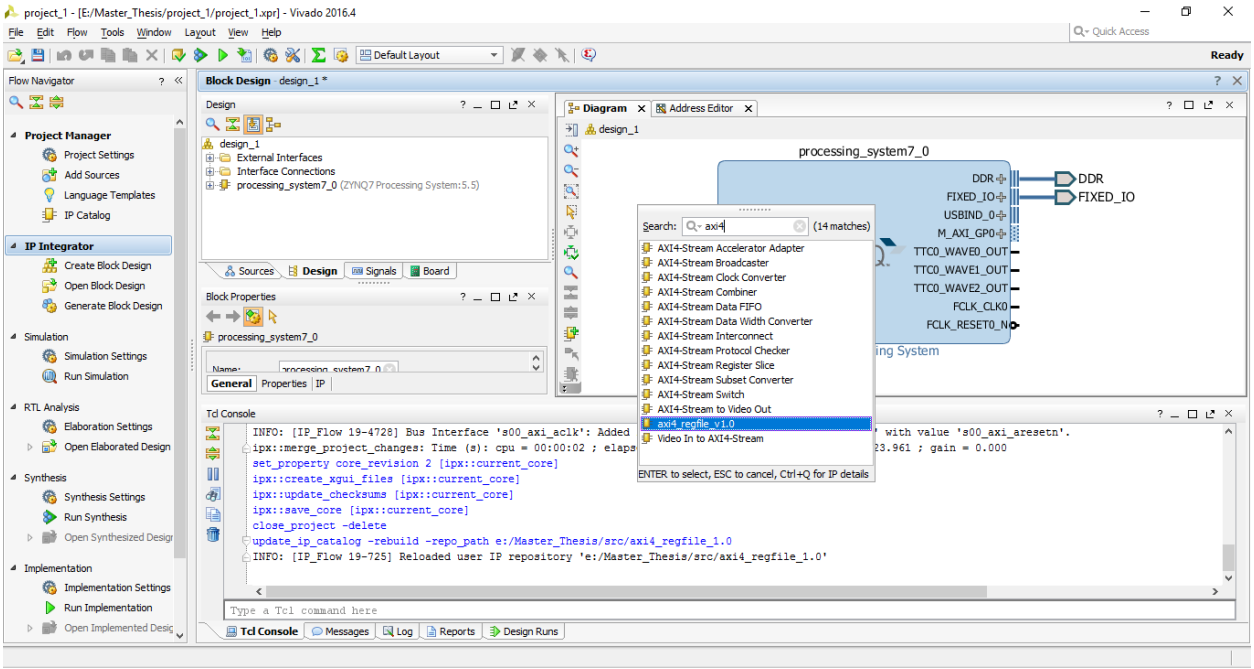



**Εικόνα 11: Παράθυρο επεξεργασίας περιφερειακού**

12) Αφού ολοκληρώσετε τις αλλαγές στον κώδικα, στην καρτέλα (tab) “Package IP” επιλέξτε “File Groups” και πατήστε “Merge changes from File Groups Wizard” (Εικόνα 12). Με αυτό τον τρόπο οι αλλαγές που έγιναν στον κώδικα ενσωματώνονται στο περιφερειακό. Η ίδια διαδικασία πρέπει να γίνει για το “Customization Parameters” και “Ports and Interfaces” έτσι ώστε, τελικά όλες οι επιλογές να έχουν το πράσινο “v”. Η διαδικασία επεξεργασίας του περιφερειακού ολοκληρώνεται με την επιλογή “Review and Package” και πατώντας το πλήκτρο “Re-Package IP”. Θα εμφανιστεί μήνυμα να κλείσει του project επεξεργασίας, πατήστε Close.
Τώρα, το περιφερειακό έχει εισαχθεί στον κατάλογο των IP και μπορούμε να το εισάγουμε στο block design όπως φαίνεται στην Εικόνα 13.

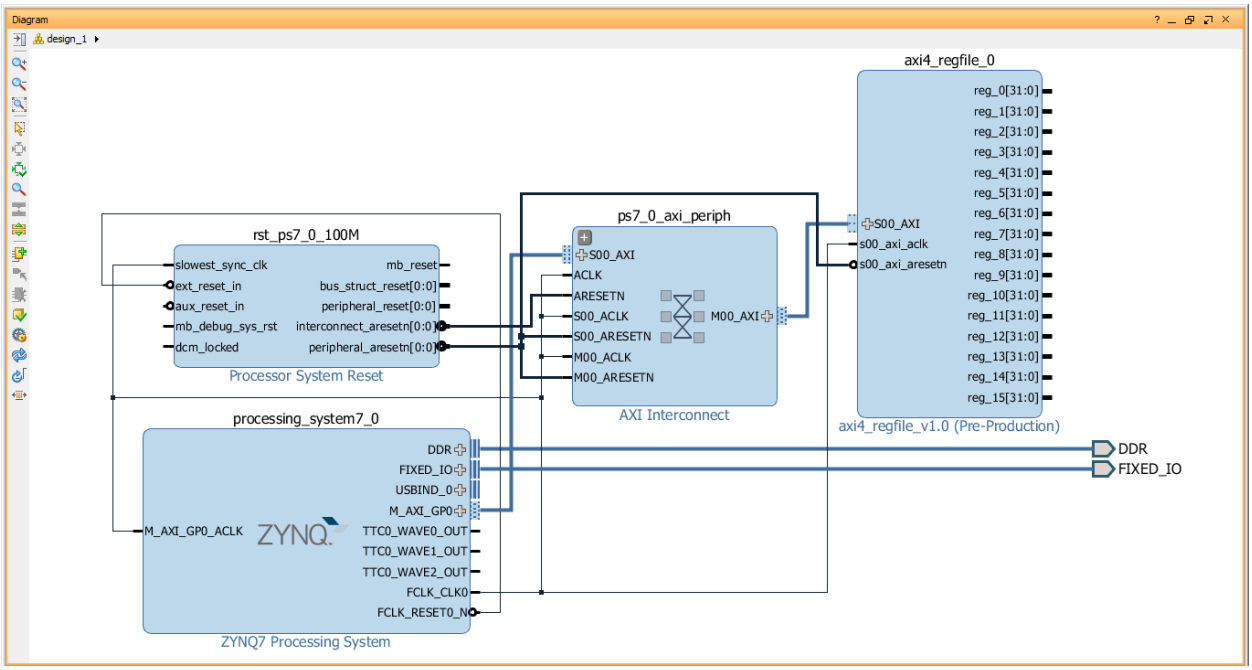



**Εικόνα 12: Ενσωμάτωση αλλαγών στο περιφερειακό**



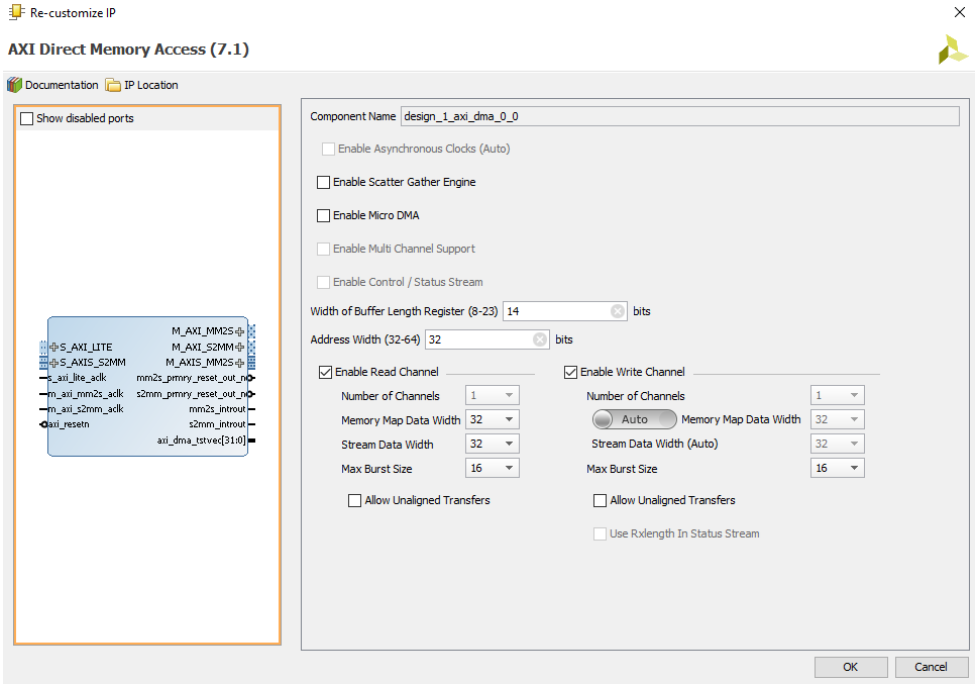

**Εικόνα 13: Εισαγωγή περιφερειακού αρχείου καταχωρητών**

13) Μόλις εισαχθεί το αρχείο καταχωρητών στο block design εμφανίζεται η επιλογή “Run Connection Automation”. Η επιλογή αυτή μας δίνει τη δυνατότητα να συνδεθούν οι διεπαφές AXI4 αυτόματα από το εργαλείο. Πατώντας την επιλογή και OK στο παράθυρο που εμφανίζεται (το παράθυρο αναφέρει τις διεπαφές προς σύνδεση, δηλαδή την διεπαφή Slave του αρχείου καταχωρητών), παρατηρούμε το αποτέλεσμα της Εικόνα 14. Το εργαλείο εισήγαγε ένα AXI Interconnect (ps7_0_axi_periph) έτσι ώστε να μπορέσει η AXI Master διεπαφή του Zynq να επικοινωνήσει με την AXI Slave Lite διεπαφή του αρχείου καταχωρητών (το AXI Interconnect λειτουργεί ως δρομολογητής πληροφορίας σε περίπτωση περισσότερων περιφερειακών, όπως θα δούμε στη συνέχεια).




**Εικόνα 14: Διασύνδεση αρχείου καταχωρητών με τον Zynq επεξεργαστή**

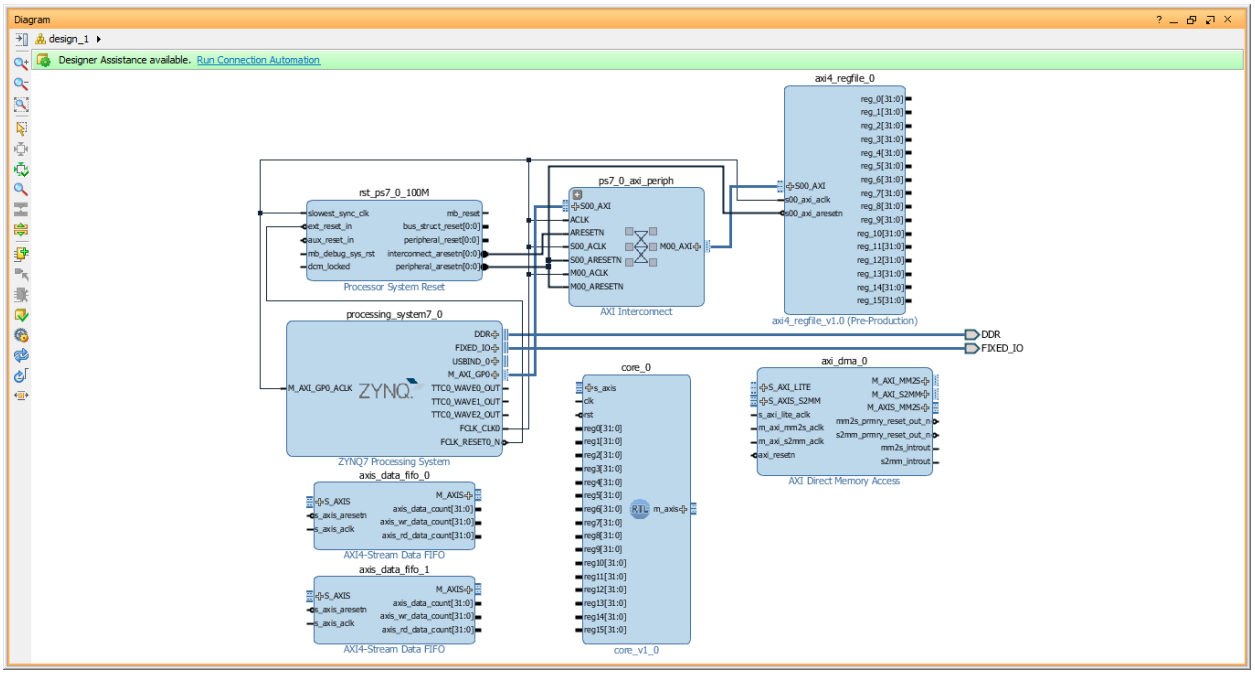
14) Σε αυτό το σημείο έχουμε ολοκληρώσει την λογική ελέγχου (control) της μονάδας επεξεργασίας δεδομένων (core) και θα δημιουργήσουμε την λογική για την μεταφορά των δεδομένων (datapath) από τον επεξεργαστή (δηλαδή από τη μνήμη) προς το core και αντίστροφα.
Για την δημιουργία του datapath επιλέγουμε από τη λίστα των IP, το “AXI Direct Memory Access”. Κάνοντας διπλό κλικ πάνω στο μπλοκ μπορούμε να το διαμορφώσουμε κατά βούληση. Όπως φαίνεται στην Εικόνα 15, μην επιλέξετε το “Enable Scatter Gather Engine” και αφήστε τις υπόλοιπες επιλογές ως έχουν. Οι διεπαφές του μπλοκ μειώθηκαν όπως φαίνεται στο παράθυρο προεπισκόπησης. Πατήστε OK.




**Εικόνα 15: Διαμόρφωση του AXI DMA μπλοκ**

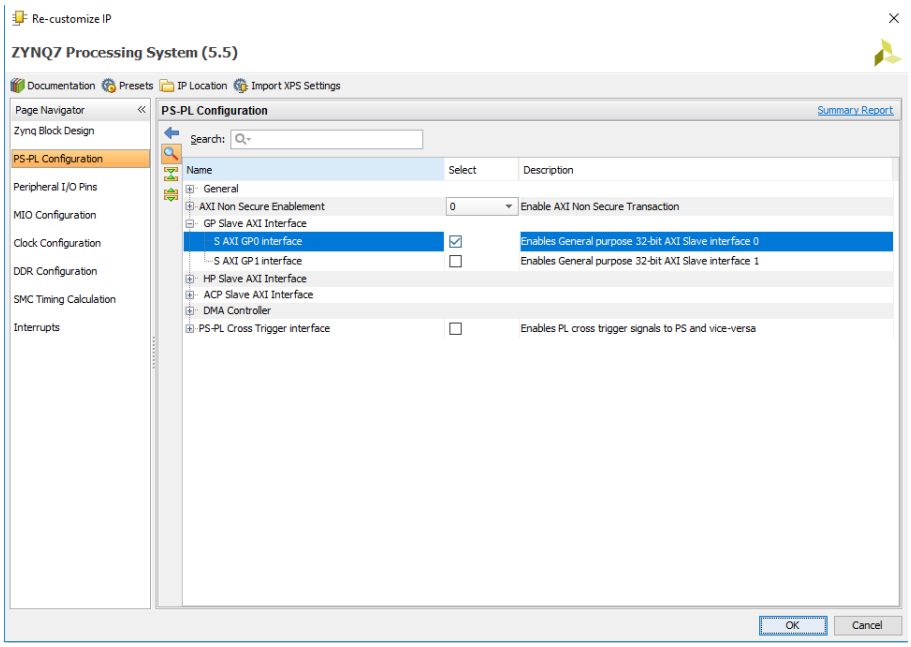
Επίσης από τη λίστα των IP επιλέξτε δύο “AXI4 Stream Data FIFO” για την χρήση τους από και προς το core όπως αναφέρθηκε στο Κεφάλαιο 3.

15) Τέλος, εισάγουμε την λογική επεξεργασίας δεδομένων (core) πατώντας δεξί κλικ στο block design και επιλέγοντας “Add Module”. Θα εμφανιστεί ένα παράθυρο για να επιλέξουμε ποιο από τα αρχεία HDL που έχουμε ήδη εισάγει στο project (βλέπε βήμα 4) θα εισαχθεί στο block design. Εφόσον υπάρχει ιεραρχία στο core, επιλέγουμε το top level αρχείο.
Έτσι έχουμε τέσσερα υποσυστήματα που πρέπει να συνδέσουμε για την δημιουργία του datapath (core, AXI DMA και 2 AXI-Stream FIFOs).



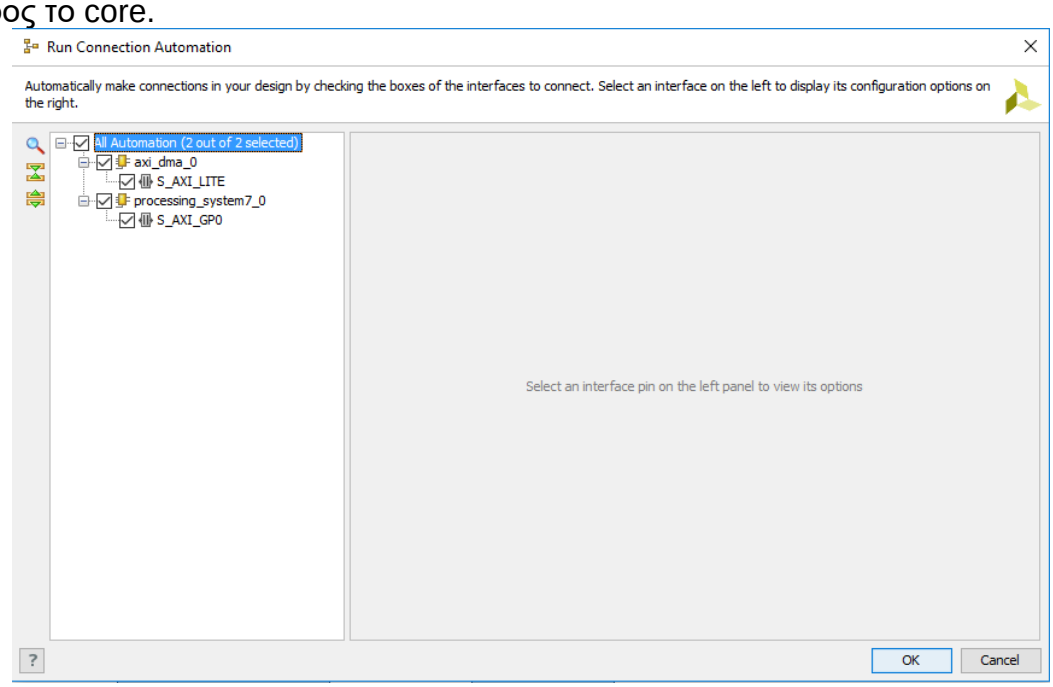

**Εικόνα 16: Δημιουργία του datapath**

16) Πριν επιλέξουμε το Run Connection Automation θα πρέπει να ενεργοποιήσουμε μια AXI Slave διεπαφή στον Zynq επεξεργαστή, έτσι ώστε τα δεδομένα από το core να μπορούν να καταλήξουν στον επεξεργαστή (μεσω του AXI DMA). Κάνοντας διπλό κλικ στον επεξεργαστή και επιλέγοντας το PL-PS Configuration, μπορούμε να δούμε όλες τις διεπαφές του επεξεργαστή. Επιλέγουμε μια “GP Slave AXI Interface” όπως φαίνεται στην Εικόνα 17.




**Εικόνα 17: Ενεργοποίηση AXI Slave διεπαφής στον επεξεργαστή**

17) Τώρα επιλέγουμε το Run Connection Automation για την δημιουργία διασύνδεσης των AXI διεπαφών. Όπως φαίνεται στην Εικόνα 18, επιλέξτε όλες τις διεπαφές για να γίνει η διασύνδεση τους. Η διεπαφή AXI_LITE του DMA θα πρέπει να διασυνδεθεί με τον επεξεργαστή (μέσω AXI Interconnect) και η AXI_GP0 του επεξεργαστή θα πρέπει να συνδεθεί με το κανάλι AXI Master MM2S (μέσω AXI Interconnect , βλέπε Κεφ 3) για την μεταφορά δεδομένων από τον επεξεργαστή προς το core.




**Εικόνα 18: Διασύνδεση AXI διεπαφών**

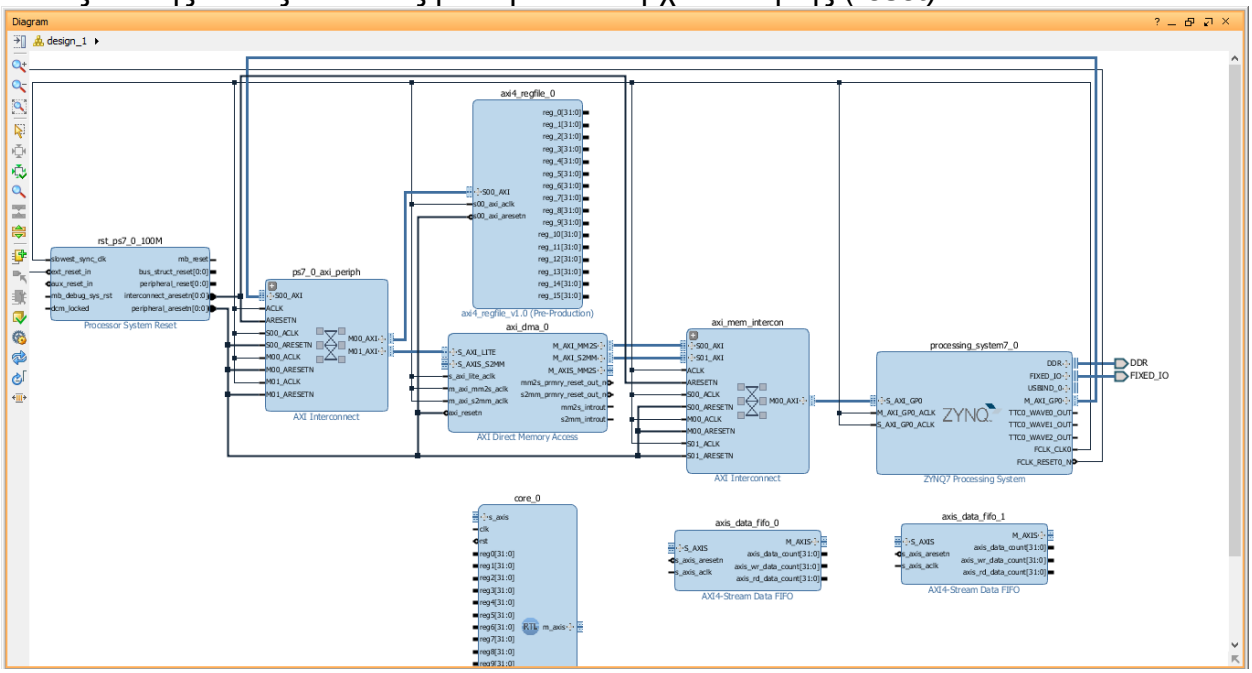
Επίσης, θα πρέπει να γίνει Connection Automation για το κανάλι AXI Master S2MM το οποίο μέσω AXI Interconnect συνδέεται με το AXI_GP0 του επεξεργαστή, για την μεταφορά δεδομένων από το core προς τον επεξεργαστή.
Στην Εικόνα 19 βλέπουμε το αποτέλεσμα των παραπάνω:

- Το AXI Interconnect ps7_0_axi_periph πλέον έχει δύο AXI Master διεπαφές έτσι ώστε ο επεξεργαστής να μπορεί να επικοινωνήσει και με το αρχείο καταχωρητών, αλλά και με το AXI DMA. Μέσω του AXI Interconnect γίνεται η δρομολόγηση της πληροφορίας για το επιθυμητό περιφερειακό.
- Οι AXI Master MM2S και S2MM διεπαφές του AXI DMA έχουν συνδεθεί μέσω ενός νέου AXI Interconnect (axi_mem_interconnect) με την AXI Slave διεπαφή του επεξεργαστή για την μεταφορά δεδομένων από και προς το core.

Σε αυτό το σημείο μένει να γίνει η σύνδεση των AXI Slave S2MM και AXI Master MM2S διεπαφών του AXI DMA για την ολοκλήρωση του datapath από και προς το core αντίστοιχα. Αυτές οι δύο διεπαφές αποτελούν AXI-Stream διεπαφές, επομένως μπορούμε να συνδέσουμε:

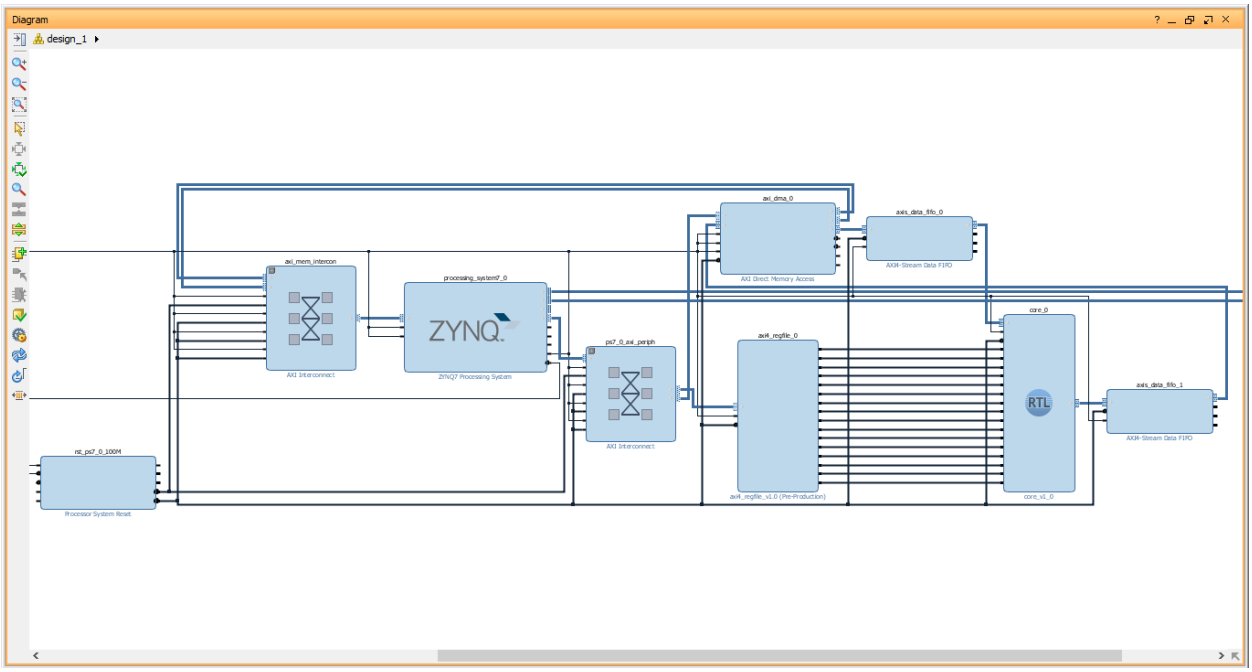
- ➢ την είσοδο της πρώτης AXI Stream FIFO με την AXI Master MM2S διεπαφή (για την μεταφορά δεδομένων προς το core)
- ➢ την έξοδο της πρώτης AXI Stream FIFO με την AXI-Stream Slave διεπαφή του core
- ➢ την είσοδο της δεύτερης AXI-Stream FIFO με την AXI-Stream Master διεπαφή του core
- ➢ την έξοδο της δεύτερης AXI-Stream FIFO με την AXI Slave S2MM διεπαφή (για την μεταφορά δεδομένων από το core)

Τέλος, συνδέουμε της εξόδους του αρχείου καταχωρητών με τις εισόδους του core, καθώς επίσης και τις εισόδους ρολογιού και αρχικοποίησης (reset).




**Εικόνα 19: Διασύνδεση του AXI DMA με τον επεξεργαστή**

Στην Εικόνα 20 βλέπουμε το τελικό σύστημα έπειτα από τις τελευταίες συνδέσεις για το datapath. Πατώντας F6 μπορούμε να κάνουμε έναν έλεγχο ότι το block design είναι σωστά σχεδιασμένο.

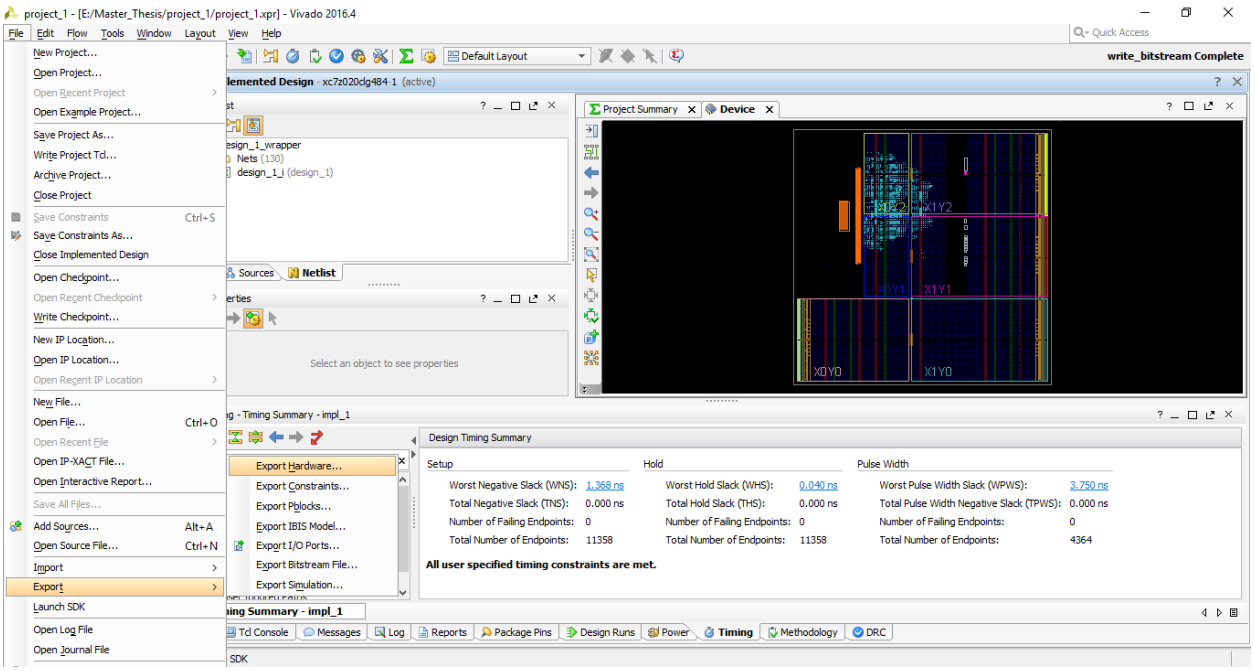



**Εικόνα 20: Τελικό σύστημα**

18) Εφόσον ολοκληρώθηκε η σχεδίαση του block design επιτυχώς, στο παράθυρο Sources επιλέγουμε το design και κάνουμε δεξί κλικ, επιλέγοντας το “Create HDL Wrapper”. Η επιλογή αυτή δημιουργεί ένα HDL αρχείο το οποίο αποτελεί το top level αρχείο του block design. Έπειτα στο αρχείο wrapper που δημιουργήθηκε με δεξί κλικ και “Set as Top” ορίζουμε ως top level το αρχείο wrapper ώστε η σύνθεση και υλοποίηση του συστήματος να γίνει για όλο το σύστημα.

19) Για την σύνθεση και υλοποίηση του συστήματος πατάμε “Generate bitstream” στο παράθυρο Flow Navigator, το οποίο εκτελεί πρώτα σύνθεση, έπειτα υλοποίηση και δημιουργεί το .bit αρχείο το οποίο χρησιμοποιούμε για τον προγραμματισμό του FPGA.

20) Αφού ολοκληρωθεί η υλοποίηση, επιλέξτε File → Export → Export Hardware… Το βήμα αυτό είναι απαραίτητο για την μετέπειτα σχεδίαση του λογισμικού με βάση το σύστημα που υλοποιήθηκε. Στο παράθυρο που εμφανίζεται βεβαιωθείτε ότι το “Include Bitstream” είναι επιλεγμένο.
Τέλος, επιλέξτε File → Launch SDK το οποίο ανοίγει το εργαλείο Software Development Kit της Xilinx για την δημιουργία του λογισμικού με βάση το σύστημα που υλοποιήθηκε.

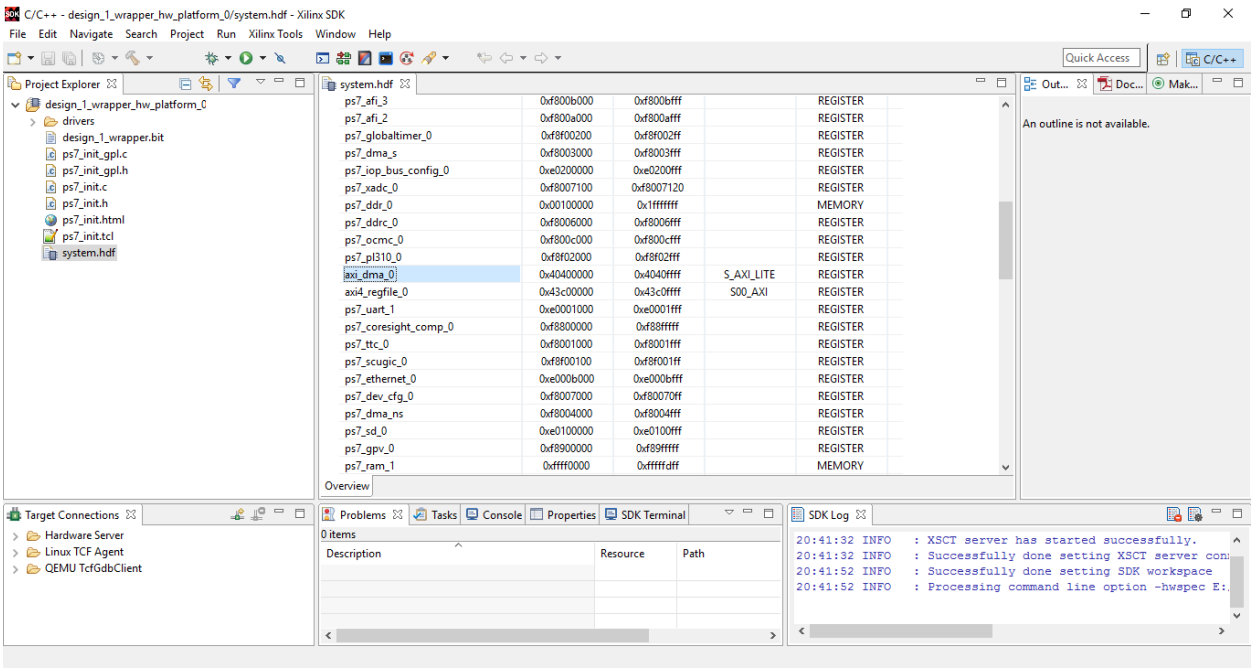



**Εικόνα 21: Εξαγωγή του συστήματος για την ανάπτυξη λογισμικού**

Η ανάπτυξη του λογισμικού καθώς επίσης ο προγραμματισμός του FPGA και η επαλήθευση (validation) του συστήματος, περιγράφεται στο επόμενο κεφάλαιο.

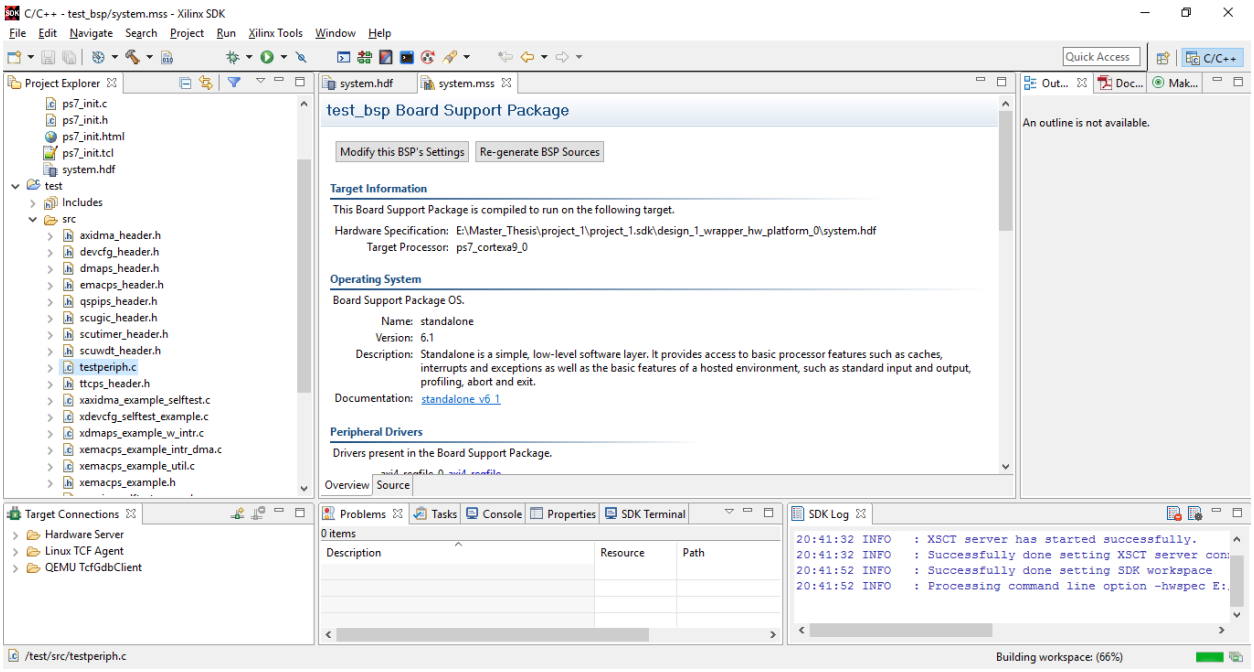
# 5. ΕΠΑΛΗΘΕΥΣΗ ΣΥΣΤΗΜΑΤΟΣ (SYSTEM VALIDATION)

Για την επαλήθευση της ορθής λειτουργίας του συστήματος αναπτύχθηκε απλό λογισμικό που εκτελεί AXI4 συναλλαγές εγγραφής και ανάγνωσης. Για την ανάπτυξη λογισμικού χρησιμοποιήθηκε του εργαλείο Software Development Kit (SDK) της Xilinx. Ακολουθώντας τα βήματα του προηγούμενου κεφαλαίου, το SDK θα πρέπει να έχει ανοίξει. Όπως φαίνεται στην Εικόνα 22, πατώντας στον Project Explorer στον φάκελο design_1_wrapper_hw_platform_0 → Drivers → system.hdf, παρουσιάζεται (εκτός άλλων) ο χάρτης διευθύνσεων (address map) του ZYNQ. Ο ZYNQ εκτελεί συναλλαγές σύμφωνα με αυτό το χάρτη και όπως φαίνεται στην Εικόνα 22, η μονάδα AXI DMA (axi_dma_0) έχει προστεθεί στο χάρτη, με διευθύνσεις αυτές οι οποίες ορίστηκαν στο Vivado (default). Επίσης βλέπουμε το αρχείο καταχωρητών (axi4_regfile_0) με τις αντίστοιχες διευθύνσεις που ορίστηκαν.
Στο αρχείο system.hdf αναφέρονται επίσης οι drivers οι οποίοι δημιουργούνται κατά την εκτέλεση του τελευταίου βήματος του Κεφαλαίου 4. Οι drivers δημιουργούνται ώστε ο ZYNQ να μπορεί να εκτελέσει τις επιθυμητές συναλλαγές προς την AXI DMA μονάδα και προς το αρχείο καταχωρητών.



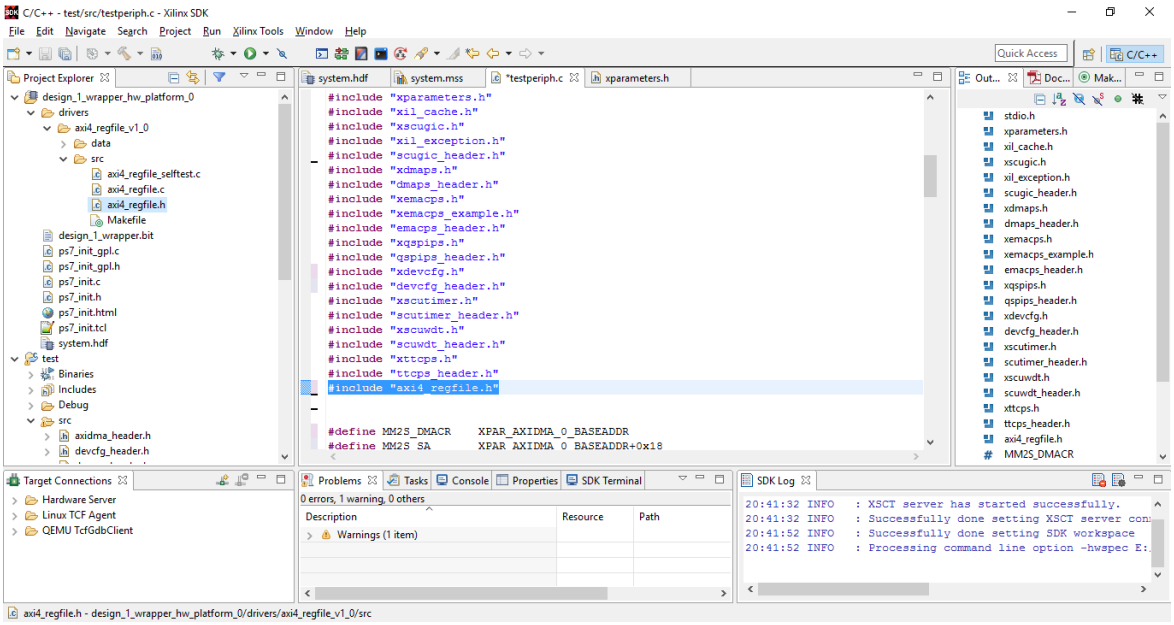

**Εικόνα 22: Χάρτης Διευθύνσεων του ZYNQ (Address map)**

Σε αυτό το σημείο δημιουργούμε ένα νέο project στο SDK ώστε να μπορέσουμε να υλοποιήσουμε το λογισμικό προς εκτέλεση. Για τη δημιουργία του project πατήστε File → New → Application Project. Στο παράθυρο που εμφανίζεται μπορούμε να επιλέξουμε το όνομα και τον τύπο του project. Επιλέξτε Peripheral Tests για τον έλεγχο περιφερειακών. Τα περιφερειακά του συστήματος που υλοποιήθηκε, είναι η μονάδα AXI DMA και το αρχείο καταχωρητών.
Με την δημιουργία του project παράγεται ένας φάκελος ο οποίος περιλαμβάνει το λογισμικό που θα εκτελεστεί από τον ZYNQ. Το αρχείο το οποίο περιλαμβάνει το λογισμικό ονομάζεται testperiph.c όπως φαίνεται στην Εικόνα 23. Πρόκειται για ένα αρχείο γλώσσας C μέσα στο οποίο αναπτύχθηκε, αυτόματα, πρόγραμμα για τον αυτοέλεγχο (self-test) των περιφερειακών. Μέσα σε αυτό το αρχείο μπορούμε να κάνουμε τροποποιήσεις ώστε να επαληθεύσουμε την ορθή λειτουργία του συστήματος.




**Εικόνα 23: Δημιουργία νέου project**

Αρχικά, για να μπορέσει ο ZYNQ να επικοινωνήσει με το αρχείο καταχωρητών θα πρέπει να εκτελεστούν συναλλαγές εγγραφής στις σωστές διευθύνσεις, όπως ορίζονται στον χάρτη διευθύνσεων που αναφέρθηκε παραπάνω. Οι διευθύνσεις οι οποίες αντιστοιχούν στον εκάστοτε καταχωρητή του αρχείου αποτελούν μια διαφορά (offset) από την διεύθυνση βάσης (base address) του περιφερειακού. Όπως φαίνεται στην Εικόνα 24, με την εκτέλεση του βήματος “Export Hardware” (τελευταίο βήμα στο προηγούμενο κεφάλαιο), έχει δημιουργηθεί ένα αρχείο axi4_regfile.h το οποίο περιλαμβάνει ορίσματα των διευθύνσεων για κάθε καταχωρητή ξεχωριστά. Οι ορισμένες αυτές σταθερές αποτελούν στην ουσία την διαφορά (offset) του εκάστοτε καταχωρητή από την διεύθυνση βάσης. Έτσι, περιλαμβάνοντας το αρχείο axi4_regfile.h στο testperiph.c μπορούμε να αναφερόμαστε στη διεύθυνση του κάθε καταχωρητή με το αντίστοιχο όρισμά του προσθέτοντάς το στη διεύθυνση βάσης.




**Εικόνα 24: Συμπερίληψη αρχείου ορισμάτων διευθύνσεων του αρχείου καταχωρητών**

Για την επαλήθευση του συστήματος τα βήματα που υλοποιήθηκαν στο C πρόγραμμα είναι τα εξής:

1. Ορισμός διευθύνσεων μνήμης επεξεργασμένων, και μη, δεδομένων όπου αποθηκεύονται. Από τη διεύθυνση source θα διαβαστούν τα μη επεξεργασμένα δεδομένα, και στη διεύθυνση destination θα αποθηκευτούν τα επεξεργασμένα δεδομένα. Ο ορισμός διευθύνσεων σε C αποτελεί ουσιαστικά την δημιουργία δεικτών (pointers) και ορίζουμε την αρχική τους τιμή ίση με την διεύθυνση μνήμης που επιθυμούμε (start address).
2. Ορισμός είδους επεξεργασίας δεδομένων. Το βήμα αυτό αποτελεί την διαμόρφωση (configuration) του CORE μέσω του αρχείου καταχωρητών. Επομένως, περιλαμβάνουμε την εντολή:

   Xil_Out32 ( base_address + register_offset, register_value )

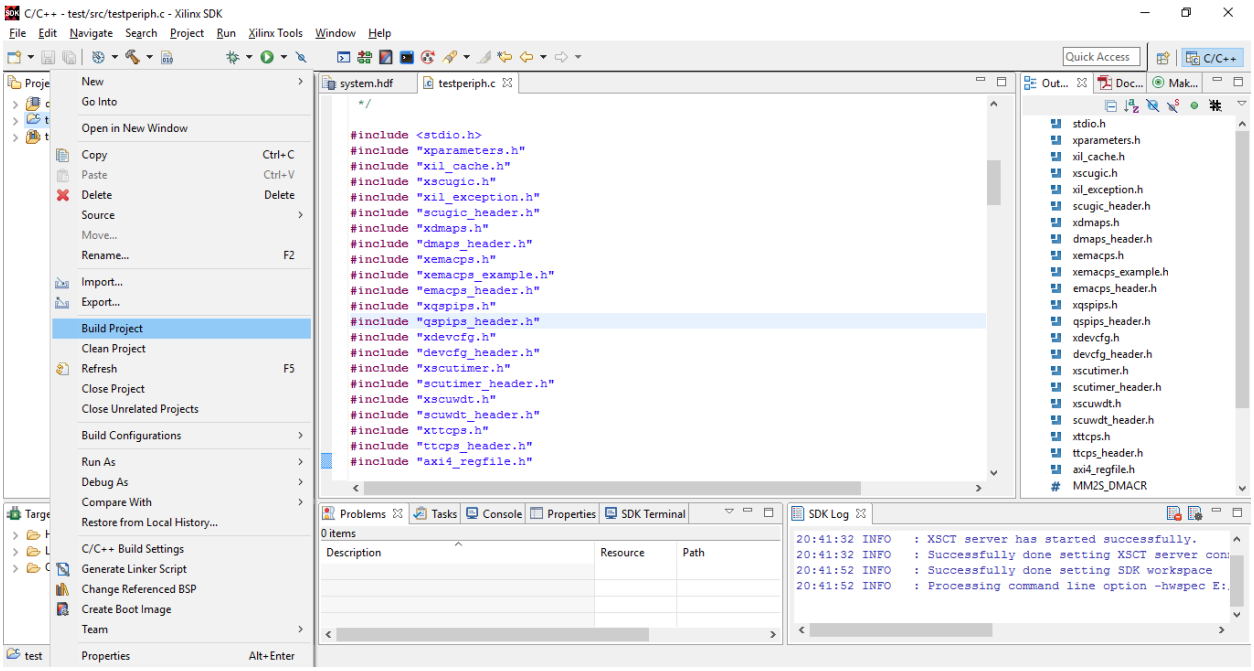
   Η εντολή αυτή εκτελεί μια AXI4 συναλλαγή εγγραφής της τιμής register_value στην διεύθυνση base_address + register_offset. Έτσι μπορούμε να διαμορφώσουμε το CORE κατά βούληση γράφοντας την τιμή που θέλουμε στον αντίστοιχο καταχωρητή διαμόρφωσης. Για παράδειγμα, για την εκτέλεση πρόσθεσης ως επεξεργασία δεδομένων θα γράψουμε την τιμή 0x00000008 (σύμφωνα με τον Πίνακας 15) στην διεύθυνση με offset 0x00. Για την ενεργοποίηση της επεξεργασίας (και την απενεργοποίηση του bypass) θα γράψουμε την τιμή 0x00000001 (σύμφωνα με τον Πίνακας 16) στην διεύθυνση με offset 0x04 (ευθυγραμμισμένες διευθύνσεις κατά 32-bit). Τέλος, ορίζεται μια σταθερά επεξεργασίας (σύμφωνα με τον Πίνακας 17) στην διεύθυνση με offset 0x0C. Έτσι, μόλις το CORE λάβει τα δεδομένα, θα εκτελέσει πρόσθεση του κάθε δεδομένου (όρισμα πίνακα στην C) με την τιμή σταθεράς.
   Σε περίπτωση που το αρχείο καταχωρητών περιελάμβανε καταχωρητές κατάστασης (status registers) θα θέλαμε το λογισμικό να διαβάζει αυτές τις τιμές. Θα έπρεπε να συμπεριληφθεί η εντολή:

   status_register = Xil_In32 ( base_address + register_offset )

   Η εντολή αυτή εκτελεί μια AXI4 συναλλαγή ανάγνωσης από την διεύθυνση base_address + register_offset και αποθηκεύει την τιμή του καταχωρητή στην μεταβλητή status_register.
3. Δημιουργία δεδομένων για επεξεργασία. Ένας απλός τρόπος δημιουργίας δεδομένων είναι η ανάθεση τιμών (με for loop) στον pointer που ορίστηκε ως δείκτης της διεύθυνσης μη επεξεργασμένων δεδομένων. Έτσι ο ZYNQ αποθηκεύει τα δεδομένα στις αντίστοιχες θέσεις μνήμης, ξεκινώντας από την διεύθυνση που ορίσαμε στο πρώτο βήμα.
4. Προγραμματισμός του AXI DMA για εκτέλεση ανάγνωσης. Σε αυτό το σημείο μπορούμε να προγραμματίσουμε την μονάδα AXI DMA ώστε να μεταφερθούν τα δεδομένα από τη μνήμη στο CORE. Για τον προγραμματισμό εκτελούμε τις τρεις εγγραφές στους καταχωρητές όπως περιγράφεται στην ενότητα 3.5 (Ανάγνωση μνήμης και μεταφορά δεδομένων προς το CORE). Με την εκτέλεση και της τρίτης εγγραφής η μεταφορά δεδομένων ξεκινά.
5. Προγραμματισμός του AXI DMA για εκτέλεση εγγραφής. Εφόσον ολοκληρωθεί η επεξεργασία, προγραμματίζουμε την AXI DMA μονάδα για να εκτελέσει την μεταφορά δεδομένων προς τη μνήμη. Εκτελούμε τις τρεις εγγραφές στους καταχωρητές όπως περιγράφεται στην ενότητα 3.5 (Μεταφορά δεδομένων από το CORE και εγγραφή στη μνήμη). Με την εκτέλεση και της τρίτης εγγραφής η μεταφορά δεδομένων ξεκινά. Με την ολοκλήρωση της μεταφοράς τα δεδομένα έχουν αποθηκευτεί στην διεύθυνση που ορίσαμε στο βήμα 1 ως διεύθυνση destination.

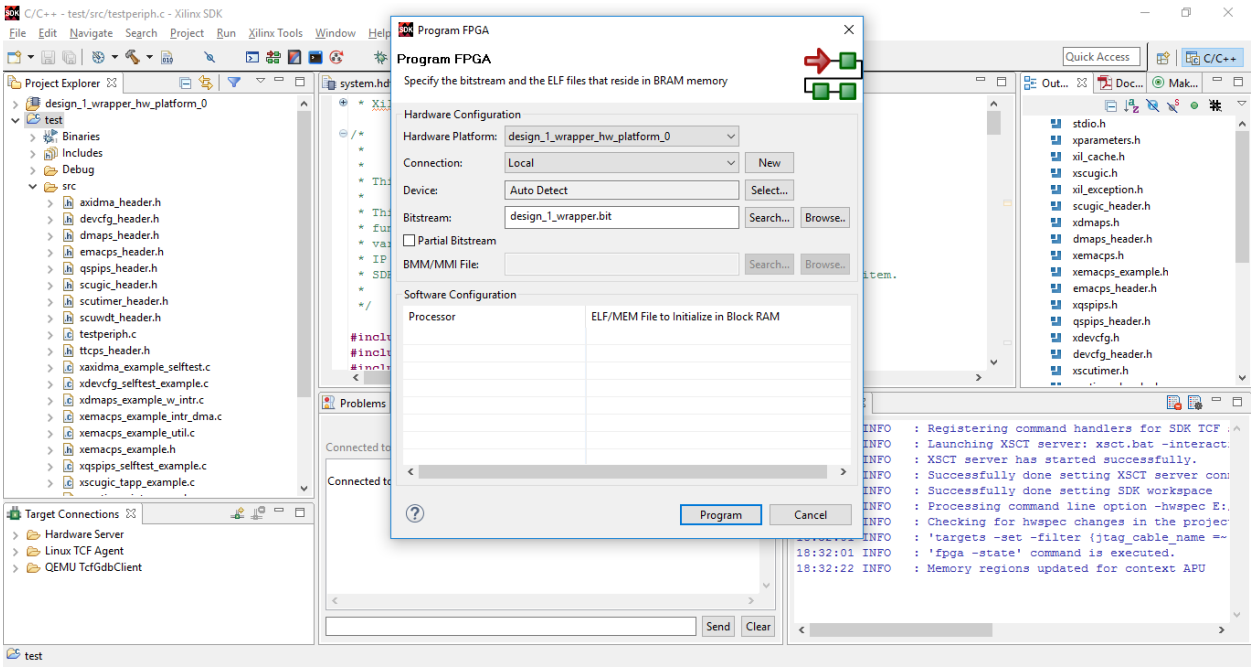
6. Για την επαλήθευση της λειτουργίας αρκεί να εκτυπώσουμε (printf) τις τιμές στις οποίες δείχνει ο pointer της διεύθυνσης destination και να διαπιστώσουμε ότι τα αρχικά δεδομένα υπέστησαν επεξεργασία (πρόσθεση με σταθερά).

Με την ολοκλήρωση της ανάπτυξης του προγράμματος, είμαστε σε θέση να προγραμματίσουμε το FPGA και να εκτελέσουμε του πρόγραμμα. Αρχικά θα πρέπει να κάνουμε δεξί κλικ στον φάκελο του project (στον Project Explorer) και να επιλέξουμε Build Project όπως φαίνεται στην παρακάτω εικόνα.



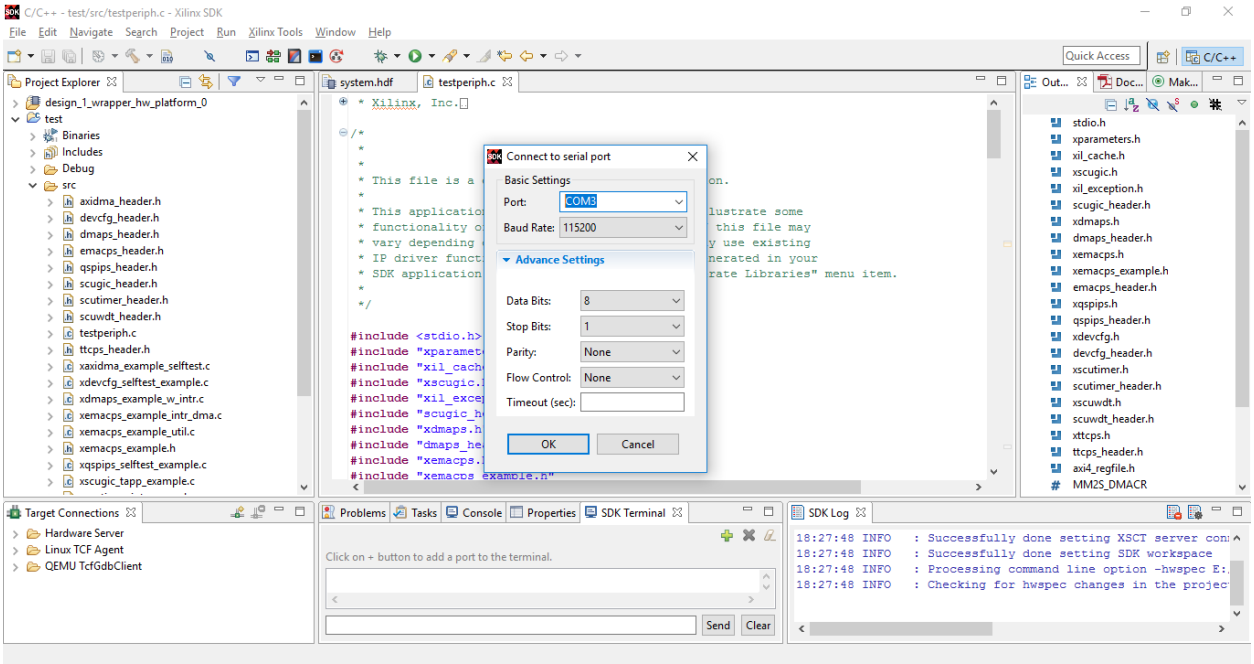

**Εικόνα 25: Build Project**

Έπειτα θα πρέπει να συνδέσουμε το Zedboard στον υπολογιστή στον οποίο εργαζόμαστε, ώστε να πραγματοποιήσουμε τον προγραμματισμό και την εκτέλεση του προγράμματος. Συνδέουμε με καλώδιο, σε θύρες USB, τις διεπαφές JTAG και UART του Zedboard [6] για τον προγραμματισμό και την εκτέλεση του προγράμματος, αντίστοιχα. Μόλις ολοκληρώσουμε τις συνδέσεις, ανάβουμε του Zedboard.
Για τον προγραμματισμό του FPGA πατάμε το πλήκτρο Program FPGA στο περιβάλλον του SDK. Βεβαιωθείτε ότι το bitstream που είναι επιλεγμένο είναι το σωστό και πατήστε Program. Το FPGA προγραμματίζεται μέσω του JTAG καλωδίου.




**Εικόνα 26: Προγραμματισμός FPGA**

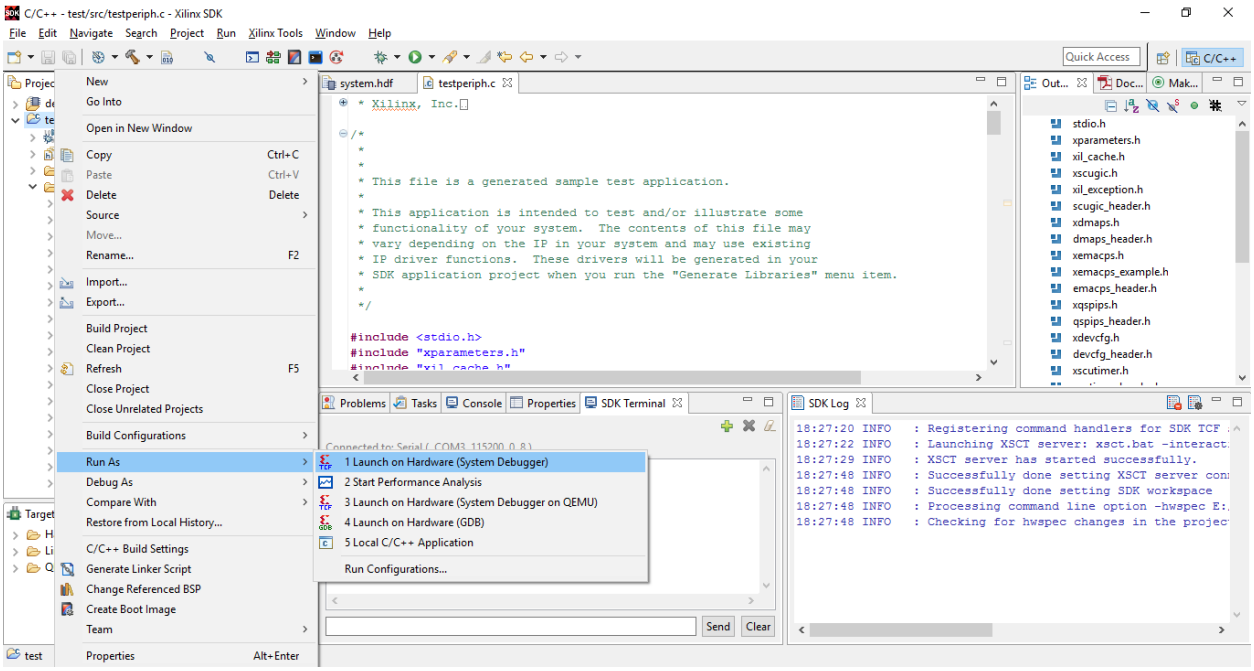
Πλέον το σύστημα υλοποιείται στο FPGA. Για να εκτελεστεί το λογισμικό που αναπτύχθηκε, θα πρέπει να συνδεθεί ο υπολογιστής με την θύρα UART του Zedboard. Στο περιβάλλον του SDK, στο κάτω μέρος στην καρτέλα SDK terminal πατήστε το σύμβολο συν (+) το οποίο ονομάζεται Connect to serial port. Με αυτό τον τρόπο συνδέεται ο υπολογιστής με την UART θύρα του Zedboard. Στο παράθυρο που εμφανίζεται επιλέξτε την θύρα USB στην οποία συνδέσατε το καλώδιο που καταλήγει στην UART (πχ COM3) και πατήστε OK. Η σύνδεση του πραγματοποιήθηκε με την ένδειξη “Connected to COM3 at 115200”.




**Εικόνα 27: Σύνδεση με τη θύρα UART του Zedboard**

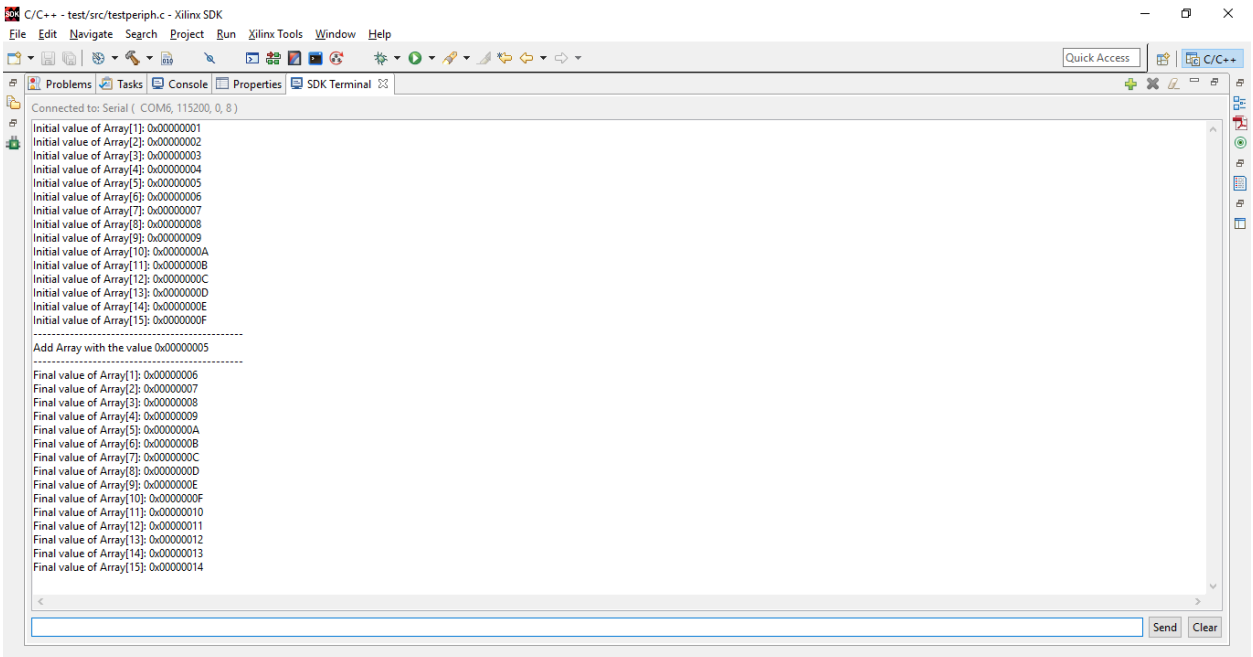
Για να εκτελεστεί το πρόγραμμα κάνουμε δεξί κλικ στο φάκελο του project (στον Project explorer) και στην επιλογή Run as επιλέξτε Launch on Hardware (System Debugger),

όπως φαίνεται στην παρακάτω εικόνα. Ο υπολογιστής στέλνει τα δεδομένα (πρόγραμμα) μέσω της UART στο FPGA, και το πρόγραμμα εκτελείται από τον ZYNQ.

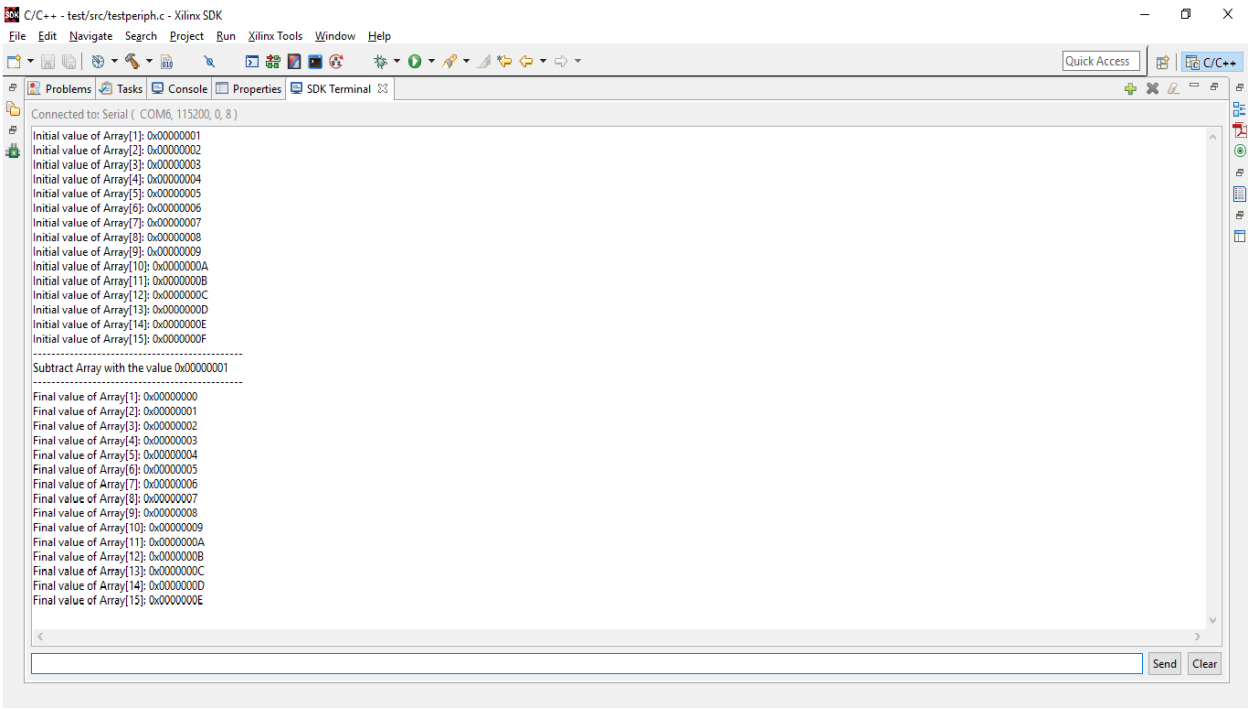



**Εικόνα 28: Εκτέλεση προγράμματος**

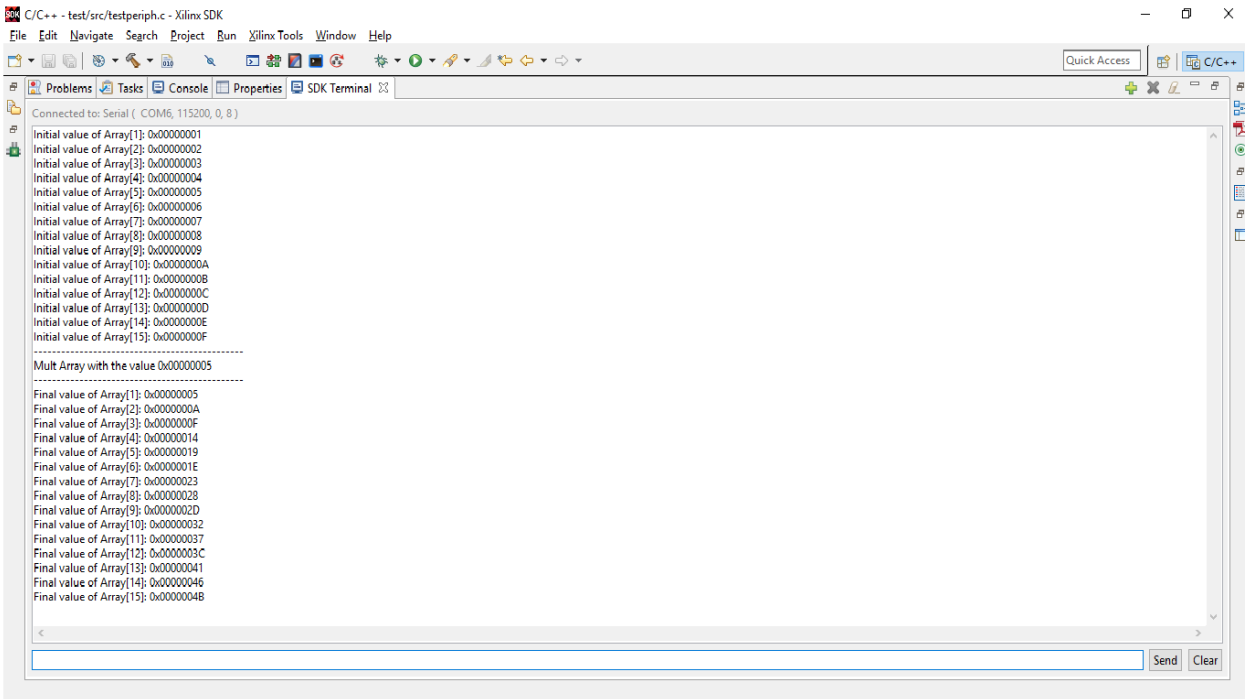
Μόλις ολοκληρωθεί η εκτέλεση του προγράμματος, τα αποτελέσματα θα έχουν εκτυπωθεί στην καρτέλα SDK terminal (σύμφωνα με το πρόγραμμα C). Στις παρακάτω εικόνες παρουσιάζονται μερικά από τα αποτελέσματα της λειτουργίας του συστήματος. Τα σενάρια αναφέρονται στην επεξεργασία ενός πίνακα 32-bit αριθμών με μήκος 15 (15 ορίσματα με τιμές 0x00000001 έως 0x0000000F). Οι εικόνες δείχνουν το αποτέλεσμα μετά από πρόσθεση, αφαίρεση και πολλαπλασιασμό αντίστοιχα.




**Εικόνα 29: Πρόσθεση πίνακα με σταθερά**




**Εικόνα 30: Αφαίρεση πίνακα με σταθερά**




**Εικόνα 31: Πολλαπλασιασμός πίνακα με σταθερά**

# 6. ΕΠΙΛΟΓΟΣ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Οι απαιτήσεις σε πόρους του FPGA για αυτό το απλό σύστημα παρατίθενται στον παρακάτω πίνακα. Όπως είναι εμφανές η λογική διασύνδεσης του ZYNQ επεξεργαστή με την προγραμματιζόμενη λογική έχει ελάχιστες απαιτήσεις όσων αφορά σε πόρους του FPGA. Σε αυτούς συμπεριλαμβάνεται και το απλό CORE και το αρχείο καταχωρητών που υλοποιήθηκαν για την επαλήθευση της επικοινωνίας (test design). Ως εκ τούτου, θα μπορούσε κανείς να χρησιμοποιήσει την μικρή λογική διασύνδεσης και να υλοποιήσει ένα εξαιρετικά μεγάλο (σε απαιτήσεις πόρων) CORE το οποίο διαθέτει διεπαφές AXI4 για την επικοινωνία με τον ZYNQ επεξεργαστή.

**Πίνακας 18: Χρήση πόρων του ZYNQ FPGA**

| Κατηγορία πόρων FPGA | Σε Χρήση | Διατίθενται | Χρήση (επί %) |
|---|---|---|---|
| LUT (Look-Up Tables) | 3113 | 53200 | 5.85 |
| LUTRAM (Look-Up Tables RAM) | 167 | 17400 | 0.96 |
| FF(Flip Flops) | 3994 | 106400 | 3.75 |
| BRAM (Block RAM) | 4.5 | 140 | 3.21 |
| DSP (Digital Signal Processor) | 3 | 220 | 1.36 |
| BUFG (Global clock buffer) | 1 | 32 | 3.13 |

Το σύστημα που υλοποιήθηκε λειτουργεί με ένα ρολόι το οποίο παράγεται μέσω του ZYNQ επεξεργαστή και λειτουργεί σε συχνότητα 100 MHz.

Είναι εμφανές ότι η ταχύτητα της μεταφοράς δεδομένων, εκτός από την διασύνδεση μνήμης με τον επεξεργαστή εξαρτάται από την μονάδα AXI DMA η οποία στην ουσία εκτελεί την μεταφορά των δεδομένων προς το CORE. Ανάλογα με την εκάστοτε υλοποίηση, στην παρακάτω εικόνα, παρατίθενται μερικές ενδεικτικές τιμές μέγιστων συχνοτήτων στις οποίες μπορεί να λειτουργήσει η μονάδα [3]. Η εικόνα αναφέρεται σε τρεις διαφορετικές οικογένειες FPGA καθώς επίσης και τρεις βαθμούς ταχύτητας (speed grade).

**Πίνακας 19: Μέγιστες συχνότητες λειτουργίας AXI DMA**

| Family | Speed Grade | Fmax (MHz) | | |
|---|---|---|---|---|
| | | AXI4 | AXI4-Stream | AXI4-Lite |
| Virtex®-7 | -1 | 200 | 200 | 180 |
| Kintex®-7 | | 200 | 200 | 180 |
| Artix®-7 | | 150 | 150 | 120 |
| | | | | |
| Virtex-7 | -2 | 240 | 240 | 200 |
| Kintex-7 | | 240 | 240 | 200 |
| Artix-7 | | 180 | 180 | 140 |
| | | | | |
| Virtex-7 | -3 | 280 | 280 | 220 |
| Kintex-7 | | 280 | 280 | 220 |
| Artix-7 | | 200 | 200 | 160 |

Τέλος, στην αναφορά [3] αναφέρονται οι καθυστερήσεις σε κύκλους ρολογιού (latency) και η απόδοση μεταφοράς δεδομένων (throughtput). Παρακάτω παρατίθενται οι πίνακες που αναφέρονται.

**Πίνακας 20: Καθυστέρηση (latency) της μονάδας AXI DMA**

| Description | Clocks |
|---|---|
| **MM2S Channel** | |
| Tail Descriptor write to m_axi_sg_arvalid | 10 |
| m_axi_sg_arvalid to m_axi_mm2s_arvalid | 28 |
| m_axi_mm2s_arvalid to m_axis_mm2s_tvalid | 6 |
| **S2MM Channel** | |
| Tail Descriptor write to m_axi_sg_arvalid | 10 |
| s_axis_s2mm_tvalid to m_axi_s2mm_awvalid | 39 |

**Πίνακας 21: Απόδοση μεταφοράς δεδομένων (throughput) της μονάδας AXI DMA**

| Channel | Clock Frequency (MHz) | Bytes Transferred | Total Throughput (MB/s) | Percent of Theoretical |
|---|---|---|---|---|
| MM2S[(2)] | 100 | 10,000 | 399.04 | 99.76 |
| S2MM[(3)] | 100 | 10,000 | 298.59 | 74.64 |

Εν κατακλείδι, στόχος αυτής της εργασίας είναι η διασύνδεση του ZYNQ επεξεργαστή με την προγραμματιζόμενη λογική με έναν αποδοτικό και βέλτιστο τρόπο με ελάχιστες απαιτήσεις από πλευράς πόρων του FPGA. Η ουσιαστική διασύνδεση του υλικού με το λογισμικό δίνει την δυνατότητα στον σχεδιαστή, όχι μόνο να παράγει ένα πλήρες και εξαιρετικά αποδοτικό σύστημα-σε-ψηφίδα (System-on-Chip), αλλά επίσης και την εξαιρετικά μεγάλη ευελιξία επιλογής και διαχωρισμού διεργασιών υλικού/λογισμικού, οι οποίες παλαιότερα ήταν εξαιρετικά περιορισμένες.

# ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

| **Ξενόγλωσσος όρος** | **Ελληνικός Όρος** |
|---|---|
| Robust | Εύρωστος |
| System-on-Chip | Σύστημα-σε-Ψηφίδα |
| Embedded system | Ενσωματωμένο σύστημα |
| FPGA | Διάταξη επαναπρογραμματιζόμενης λογική |
| Performance | Απόδοση |
| Latency | Καθυστέρηση |
| Throughput | Απόδοση μεταφοράς δεδομένων στη μονάδα του χρόνου |
| Digital Signal Processing | Ψηφιακή επεξεργασία σήματος |
| Sensor Network | Δίκτυο αισθητήρων |
| Interface | Διεπαφή |
| Interconnect | Δίαυλος επικοινωνίας (δρομολογητής) |
| Bus | Δίαυλος μεταφοράς δεδομένων |
| Software | Λογισμικό |
| Hardware | Υλικό |
| Validation | Επαλήθευση |
| DMA | Άμεση Προσπέλαση Μνήμης |

# ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

| | |
|---|---|
| FPGA | Field Programmable Gate Array |
| SoC | System on Chip |
| DSP | Digital Signal Processor |
| BUFG | Global buffer |
| BRAM | Block Random Access Memory |
| LUT | Look-up Talbe |
| FF | Flip Flop |
| DMA | Direct Memory Access |

# ΑΝΑΦΟΡΕΣ

[1] AMBA® AXI™ and ACE™ Protocol Specification. Copyright © 2003, 2004, 2010, 2011 ARM. All rights reserved. ARM IHI 0022D (ID102711)
[2] AMBA® 4 AXI4-Stream Protocol Version: 1.0 Specification Copyright © 2010 ARM. All rights reserved. ARM IHI 0051A (ID030510)
[3] AXI DMA v7.1 LogiCORE IP product guide. PG201 October 5, 2016
[4] The ZYNQ Book Tutorial. Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, Robert W. Stewart. Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK. V1.2 September 2014
[5] Vivado Design Suite Tutorial. Embedded Processor Hardware Design UG940 (v2016.1) April 6, 2016
[6] Zynq Evaluation and Development Hardware User's Guide. v2.2 January 27, 2014

Χρήσιμες Διαδικτυακές πηγές:

http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html
https://www.youtube.com/watch?v=0Dt8rWJdiJo
https://www.youtube.com/watch?v=meQcwzC4Vtk&t=15s
https://www.youtube.com/watch?v=Vs0h0kue7p4&t=1985s
https://www.youtube.com/watch?v=cDc9B2zAPz4
https://www.youtube.com/watch?v=y7Smc49YqNU
https://www.youtube.com/watch?v=EXULPtmJWbs
https://www.youtube.com/watch?v=K7CX9sSYVao
https://www.youtube.com/watch?v=sl8xCMu39Mk
https://www.youtube.com/watch?v=HyluUU2DGi8
https://www.youtube.com/watch?v=vvsOdj2ewkI
https://www.youtube.com/watch?v=ZhnX2u59Om4&t=179s
https://www.youtube.com/watch?v=9FrgIfE2xfU&t=1424s
https://www.youtube.com/watch?v=p96eKzQabWI
https://www.youtube.com/watch?v=8hzzVhPw6uw
https://www.youtube.com/watch?v=CNnrFVWveek&t=4s