Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Επιτάχυνση προσομοίωσης με χρήση της τεχνικής FPGA-in-the-Loop**
Ονοματεπώνυμο Φοιτητή	**Εμμανουήλ Λουκιανός**
Πατρώνυμο	**Λουκάς**
Αριθμός Μητρώου	**ΜΠΣΠ 11016**
Επιβλέπων	**Μιχάλης Ψαράκης**

Ημερομηνία Παράδοσης **Ιούλιος 2013**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Μιχάλης Ψαράκης	Δημήτρης Γκιζόπουλος	Άγγελος Πικράκης
Επίκουρος Καθηγητής	Αναπληρωτής Καθηγητής	Λέκτορας

Περιεχόμενα

Περίληψη

Η προσομοίωση είναι αναπόσπαστο μέρος της ανάπτυξης συστημάτων. Χρησιμοποιείται για την δοκιμή των σχεδιάσεων κάτω από διάφορα σενάρια και για την βελτίωση των χαρακτηριστικών τους. Επίσης χρησιμοποιείται σε σχεδιάσεις που είναι εξαιρετικά πολύπλοκες για να υπολογισθεί η απόδοσή τους με αναλυτικές μεθόδους. Σε απλές σχεδιάσεις η προσομοίωση ολοκληρώνεται σε λίγους κύκλους και ο απαιτούμενος χρόνος είναι σχετικά σύντομος. Δεν απαιτούν όμως όλες οι προσομοιώσεις ένα μικρό αριθμό κύκλων για να ολοκληρωθούν. Σε πολλές περιπτώσεις το κριτήριο τερματισμού βασίζεται στην επεξεργασία ενός μεγάλου πλήθους δεδομένων. Μία τέτοια περίπτωση είναι ο υπολογισμός του BER (Bit Error Rate – ρυθμός σφαλμάτων στα bit) σε ένα σύστημα επικοινωνίας. Για να σχεδιασθεί ένα μόνο σημείο σε ένα διάγραμμα BER χρειάζεται η επεξεργασία ενός πολύ μεγάλου αριθμού δειγμάτων. Για να σχεδιασθεί διάγραμμα με τιμές BER μέχρι 10^{-9} χρειάζεται η επεξεργασία 10^{9} δειγμάτων από τα οποία μόνο ένα θα είναι εσφαλμένο. Προκειμένου να ολοκληρωθεί ένα πλήρες διάγραμμα με καμπύλες που αποτελούνται από πολλά σημεία, η διαδικασία θα χρειαστεί πολλές ώρες.

Η προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop μπορεί να επιταχύνει σημαντικά αυτήν την διαδικασία. Καθώς η παραλληλία είναι στην φύση των συσκευών FPGA, υλοποιώντας ορισμένα τμήματα του μοντέλου σε hardware, ο χρόνος προσομοίωσης μπορεί να μειωθεί σημαντικά. Με την προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop η προσομοίωση σε πολύπλοκες σχεδιάσεις μπορεί να ολοκληρωθεί σε πολύ λιγότερο χρόνο.

Πολλοί παράγοντες καθορίζουν τον βαθμό επιτάχυνσης κατά την προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop. Σ' αυτή την εργασία δοκιμάζονται απλές αλλά και πολύπλοκες σχεδιάσεις προκειμένου να γίνει μια εκτίμηση για το όφελος που μπορεί να αποδώσει η τεχνική αυτή. Επίσης γίνεται παρουσίαση των εργαλείων που χρησιμοποιήθηκαν και των μοντέλων που δοκιμάστηκαν.

Abstract

Simulation is an integral part of system development. It is used to test designs under various scenarios and for the improvement of their characteristics. It is also used in designs that are too complicated for analytical solutions. Simulation of simple designs is completed in a few cycles and the required time is relatively short. But not all simulations require a small number of cycles to be completed. In many cases termination criteria of a simulation is based on the processing of a large amount of samples. Such a case is the calculation of BER (Bit Error Rate) of a communication system. To calculate a single point on a BER diagram, a large number of data must be processed. To plot a diagram with BER points up to 10^{-9}, the system must process 10^{9} bits of which only one will be erroneous. To create a plot that contains sufficient points to draw a curve, many hours would be required.

Hardware Co-Simulation can accelerate significally this procedure. Considering the parallel nature of FPGA devices, by implementing some parts of the model in hardware, simulation time can be reduced substantially. Hardware Co-Simulation can make it possible to complete simulation of complex designs in a much shorter period of time.

Many factors define the acceleration of simulation time by using the FPGA-in-the-Loop technique. In this thesis, simple and complex designs are being put to test in order to estimate the benefit that hardware Co-Simulation can attribute. The tools and the models used are also presented.

1. Εισαγωγή – Σύντομη περιγραφή Προβλήματος

Τα περιβάλλοντα προσομοίωσης μας επιτρέπουν να μελετήσουμε την συμπεριφορά ενός συστήματος χωρίς το σύστημα αυτό να έχει πραγματικά υλοποιηθεί. Χρησιμοποιούνται κατά την σχεδίαση συστημάτων καθώς βοηθούν στην μείωση του κόστους, στην επίτευξη καλύτερων χαρακτηριστικών του συστήματος και στην επιτάχυνση της διαδικασίας σχεδίασης. Η προσομοίωση χρησιμοποιείται σε πολλούς διαφορετικούς χώρους πέρα από την ανάπτυξη συστημάτων όπως στην μελέτη φυσικών φαινομένων, χημικών αντιδράσεων και σε μοντέλα που είναι πολύ πολύπλοκα για να επιλυθούν με αναλυτικές μεθόδους. Ένα εργαλείο που χρησιμοποιείται για την προσομοίωση συστημάτων είναι το Simulink.

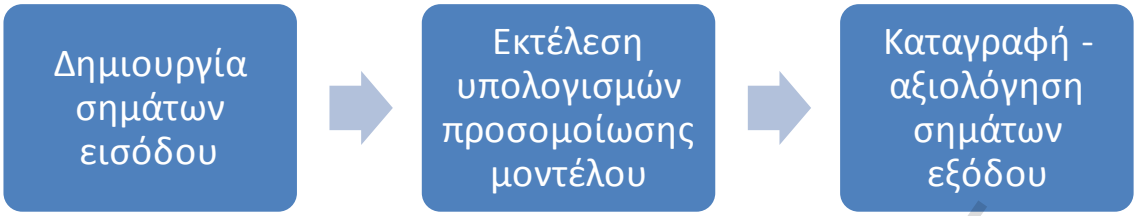
Το εργαλείο Simulink αποτελεί μια επέκταση του εργαλείου MATLAB. Προσφέρει την δυνατότητα μοντελοποίησης και προσομοίωσης συστημάτων. Το περιβάλλον του είναι γραφικό. Τα μοντέλα σχεδιάζονται στην οθόνη χρησιμοποιώντας μπλοκ βασικών λειτουργιών. Πολλά στοιχεία επεξεργασίας σημάτων είναι διαθέσιμα καθώς και εικονικές συσκευές εισόδου και εξόδου. Οι συσκευές εισόδου παράγουν τα σήματα εισόδου και σε κάθε κύκλο του Simulink υπολογίζεται η αντίστοιχη τιμή εξόδου.

Η προσομοίωση εκτελείται σε κύκλους. Σε κάθε κύκλο δημιουργούνται τα σήματα εισόδου και εκτελούνται οι υπολογισμοί που υπαγορεύει το μοντέλο προσομοίωσης. Κατόπιν τα σήματα εξόδου καταγράφονται και ξεκινά ένας νέος κύκλος με νέα σήματα εισόδου. Όταν η προσομοίωση απαιτεί λίγους κύκλους ή όταν το μοντέλο είναι σχετικά απλό, η προσομοίωση ολοκληρώνεται γρήγορα. Υπάρχουν όμως περιπτώσεις που η προσομοίωση που παρέχει το Simulink είναι ανεπαρκής. Αυτό συμβαίνει όταν απαιτείται η επεξεργασία ενός πολύ μεγάλου πλήθους δεδομένων προκειμένου να βγουν χρήσιμα συμπεράσματα για την λειτουργία του μοντέλου ή όταν ο διαθέσιμος χρόνος για την επεξεργασία των σημάτων είναι περιορισμένος. Π.χ. στην επεξεργασία σήματος video σε πραγματικό χρόνο, ο χρόνος που διατίθεται για την επεξεργασία κάθε πλαισίου του οπτικού σήματος είναι αυστηρά περιορισμένος. Ακόμα και για πλαίσιο εικόνας χαμηλής ευκρίνειας, της τάξης του 128X128 εικονοστοιχείων, ο διαθέσιμος χρόνος του 1/25 sec δεν αρκεί για την επεξεργασία των 16384 εικονοστοιχείων που συνθέτουν την εικόνα.

Ένα παράδειγμα όπου απαιτείται επεξεργασία μεγάλου όγκου δεδομένων είναι η δημιουργία γραφημάτων για την απόδοση ενός ζεύγους κωδικοποιητή – αποκωδικοποιητή κατά την μετάδοση ενός σήματος μέσα από ένα κανάλι που υπόκειται σε θόρυβο. Το BER (Bit Error Rate) ενός συστήματος μπορεί να παίρνει τιμές της τάξης του 10^{-9}. Αυτό σημαίνει ότι θα υπάρχει ένα λανθασμένο bit κατά την μετάδοση ενός δισεκατομμυρίου bit. Για να χαραχθεί με ακρίβεια ένα διάγραμμα με σημεία BER μπορεί να χρειαστεί να προσομοιωθεί η μετάδοση πολλών δισεκατομμυρίων bit. Μία τέτοια προσομοίωση θα πάρει πολλές ώρες στο Simulink.

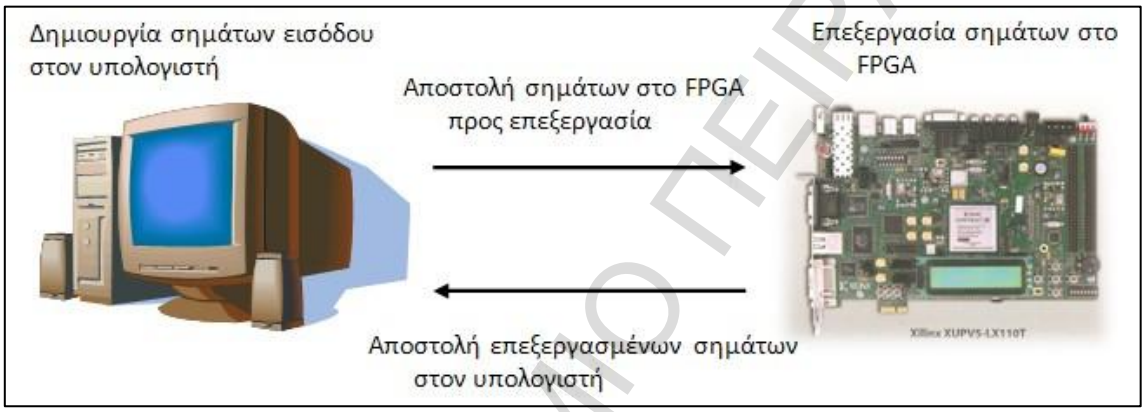
Η λύση στο πρόβλημα είναι η τεχνική FPGA-In-the-Loop. Στην τεχνική αυτή χρησιμοποιείται μία πλακέτα με FPGA στο οποίο συντίθεται κύκλωμα που εκτελεί τους πολύπλοκους υπολογισμούς που χρειάζονται πάρα πολύ χρόνο για να εκτελεσθούν από το πρόγραμμα προσομοίωσης. FPGA είναι τα αρχικά των λέξεων Field Programmable Gate Array που αποδίδεται στα ελληνικά ως «επιτόπου προγραμματιζόμενοι πίνακες πυλών». Πρόκειται για ένα ολοκληρωμένο κύκλωμα το οποίο διαμορφώνεται από τον σχεδιαστή της εφαρμογής αφότου αυτό έχει κατασκευαστεί, σε αντίθεση με τα ASIC (Application Specific Integrated Circuit) τα οποία διαμορφώνονται στο εργοστάσιο κατασκευής και καμία περεταίρω αλλαγή δεν είναι δυνατή.

Στην προσομοίωση συστημάτων με software ο υπολογιστής παράγει μια σειρά από σήματα που αποτελούν τα σήματα εισόδου του συστήματος (εικόνα 1). Κατόπιν εκτελούνται οι υπολογισμοί που καθορίζονται από το μοντέλο που προσομοιώνεται. Αυτό είναι και το πιο χρονοβόρο τμήμα της προσομοίωσης. Κατόπιν τα αποτελέσματα των υπολογισμών της προσομοίωσης καταγράφονται και αξιολογούνται.



Εικόνα 1 Στάδια στην Προσομοίωση με software

Στην προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop τα σήματα εισόδου δημιουργούνται στον υπολογιστή. Κατόπιν αυτά τα σήματα στέλνονται στην πλακέτα που ενσωματώνει το FPGA. Στο FPGA έχει υλοποιηθεί κατάλληλο κύκλωμα που εκτελεί τους υπολογισμούς της προσομοίωσης. Τα αποτελέσματα των υπολογισμών στέλνονται κατόπιν πίσω στον υπολογιστή (εικόνα 2).



Εικόνα 2 Προσομοίωση με την τεχνική FPGA-In-the-Loop

Οι υπολογισμοί που εκτελούνται στο FPGA χρειάζονται πολύ λιγότερο χρόνο από τον αντίστοιχο χρόνο εκτέλεσης στον υπολογιστή. Αν και το FPGA εργάζεται με συχνότητα ρολογιού χαμηλότερη από αυτήν ενός υπολογιστή, της τάξης των 100-500 MHZ, η απόδοση είναι πολύ μεγαλύτερη λόγω της αξιοποίησης της παραλληλίας. Ενώ στον υπολογιστή οι πράξεις εκτελούνται η μια μετά την άλλη στην αριθμητική-λογική μονάδα (ALU), στο FPGA δημιουργείται ξεχωριστό κύκλωμα για τον κάθε υπολογισμό. Έτσι μπορούν να εκτελούνται πολλοί υπολογισμοί παράλληλα. Το FPGA της πλακέτας που χρησιμοποιήθηκε στην εργασία διαθέτει 64 τεμάχια (slices) DSP48, το κάθε ένα από τα οποία περιέχει πολλαπλασιαστή, αθροιστή και συσσωρευτή των 48 bit. Στο παράδειγμα με το φίλτρο FIR που παρουσιάζεται αξιοποιούνται τα 62 απ' αυτά.

Στην προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop υπάρχει μια συνεχής ροή δεδομένων ανάμεσα στον υπολογιστή και στην πλακέτα. Σε κάθε κύκλο προσομοίωσης αποστέλλονται στην πλακέτα νέα σήματα εισόδου και επιστρέφουν στον υπολογιστή τα αντίστοιχα αποτελέσματα των υπολογισμών. Ο όγκος των δεδομένων που μεταφέρονται μπορεί να είναι μεγάλος και η μεταφορά τους να απαιτεί υπολογίσιμο χρόνο. Είναι δυνατό το όφελος από την επιτάχυνση των υπολογισμών να χαθεί από τον χρόνο που απαιτεί η μεταφορά των δεδομένων. Παίζει λοιπόν σημαντικό ρόλο το μέσο μετάδοσης των σημάτων. Η μεταφορά με καλώδιο Ethernet είναι ο πιο γρήγορος τρόπος αλλά δεν υποστηρίζεται από όλες τις πλακέτες FPGA. Ο τρόπος μεταφοράς μέσω του αγωγού JTAG που χρησιμοποιείται για τον προγραμματισμό του FPGA της πλακέτας μπορεί να εφαρμοσθεί σε κάθε πλακέτα αλλά είναι πιο αργός από την μεταφορά μέσω του καλωδίου Ethernet.

Ένας τρόπος να μειωθούν οι καθυστερήσεις από την μεταφορά δεδομένων είναι η χρήση κοινόχρηστων μνημών. Μία μνήμη RAM εισόδου και μία μνήμη RAM εξόδου στις οποίες έχει πρόσβαση και το κύκλωμα του FPGA και το software του υπολογιστή υλοποιούνται στο FPGA. Στην μνήμη εισόδου αποστέλλονται μαζικά τα δεδομένα που θα τροφοδοτήσουν τους υπολογισμούς πολλών κύκλων προσομοίωσης. Στο τέλος των υπολογισμών τα δεδομένα της

μνήμης RAM εξόδου μεταφέρονται μαζικά στον υπολογιστή. Έτσι δεν έχουμε καθυστέρηση για την μεταφορά δεδομένων σε κάθε κύκλο προσομοίωσης.

Το εργαλείο που χρησιμοποιήθηκε σε αυτήν την εργασία είναι το System Generator. Για την εκτέλεση του εργαλείου απαιτείται η εγκατάσταση δύο πακέτων λογισμικού: του Matlab/Simulink της Mathworks και του ISE της Xilinx. Με την εγκατάσταση του δεύτερου μία επιπλέον συλλογή προστίθεται στην βιβλιοθήκη του Simulink, με τα μπλοκ της οποίας μπορούμε να συνθέσουμε το μοντέλο μας. Η μεγάλη διαφορά είναι ότι το μοντέλο που συνθέτουμε με μπλοκ της Xilinx μπορεί να υλοποιηθεί σε πραγματικό hardware, σε FPGA της Xilinx. Το μοντέλο αυτό μπορεί να προσομοιωθεί από το Simulink με δύο τρόπους. Ο πρώτος είναι ο γνωστός, καθαρά με χρήση software. Ο δεύτερος είναι αυτός που έχει το μεγαλύτερο ενδιαφέρον και αποτελεί και το αντικείμενο αυτής της εργασίας: Το System Generator δημιουργεί το κατάλληλο αρχείο με το οποίο προγραμματίζει το FPGA και έτσι το μοντέλο υλοποιείται σε πραγματικό hardware. Το Simulink παράγει τα σήματα εισόδου για την προσομοίωση τα οποία στέλνονται στο FPGA. Εκεί γίνονται οι υπολογισμοί και από την έξοδο του FPGA τα αποτελέσματα στέλνονται πίσω στο Simulink όπου μπορούν να καταγραφούν και να αξιολογηθούν.

Στην εργασία αυτή παρουσιάζεται πρώτα το εργαλείο προσομοίωσης Simulink. Ένα απλό μοντέλο δημιουργείται με τα μπλοκ λειτουργιών που διατίθενται. Κατόπιν γίνεται μια σύντομη περιγραφή των βασικών μπλοκ της Xilinx και το ίδιο μοντέλο δημιουργείται με τα μπλοκ αυτά. Ακολουθεί μια σύντομη αναφορά στην τεχνολογία και τον τρόπο σχεδίασης των κυκλωμάτων FPGA.

Η κάρτα που χρησιμοποιήθηκε στις δοκιμές ήταν η Virtex5 με το FPGA XC5VLX110T. Το FPGA αυτό διαθέτει 64 τεμάχια (slices) DSP48, το κάθε ένα από τα οποία περιέχει ένα 25X18 bit πολλαπλασιαστή, έναν αθροιστή και έναν συσσωρευτή (accumulator). Επίσης διαθέτει 17.280 τεμάχια virtex5, το κάθε ένα από τα οποία περιέχει τέσσερα flip-flop και τέσσερεις πίνακες LUT. Διαθέτει τέλος 5.328Kb RAM σε μπλοκ των 18K και των 36K.

Προκειμένου να γίνει προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop απαιτείται μια διαδικασία για την εγκατάσταση της πλακέτας που διατίθεται. Η διαδικασία αυτή παρουσιάζεται αναλυτικά.

Τέσσερα μοντέλα διαφορετικής πολυπλοκότητας παρουσιάζονται και περιγράφεται η διαδικασία προσομοίωσης με χρήση της τεχνικής FPGA-in-the-Loop. Γίνονται μετρήσεις της ταχύτητας προσομοίωσης του κάθε μοντέλου.

- Το πρώτο μοντέλο αφορά ένα απλό φίλτρο. Σ' αυτό κάθε τιμή του σήματος αντικαθίσταται από το μέσο όρο τριών διαδοχικών τιμών του αρχικού σήματος. Στο μοντέλο αυτό η software προσομοίωση ήταν κατά πολύ πιο γρήγορη από την προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop. Οι υπολογισμοί που απαιτούνται είναι πολύ απλοί και το όφελος από την ταχεία εκτέλεση των υπολογισμών στο FPGA είναι μηδαμινό σε σχέση με τον χρόνο που ξοδεύεται για την μεταφορά των δεδομένων προς και από την πλακέτα.
- Το δεύτερο μοντέλο αφορά την σύνθεση ενός φίλτρου FIR (Finite Impulse Response) με χρήση του εργαλείου σχεδίασης φίλτρων FDA tool. Και πάλι η software προσομοίωση ήταν κατά πολύ πιο γρήγορη από την προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop. Ένας μεγάλος αριθμός κύκλων προσομοίωσης απαιτείται για τον υπολογισμό της απόδοσης του φίλτρου στις διάφορες συχνότητες. Ο μεγάλος όγκος δεδομένων που χρειάζεται να μετακινηθεί μεταξύ υπολογιστή και FPGA δημιουργεί μεγάλη καθυστέρηση και κάνει την προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop μη αποδοτική.
- Στο τρίτο μοντέλο γίνεται επεξεργασία εικόνας σε σήμα βίντεο. Ένας αριθμός από πλαίσια εικόνας τροφοδοτούν έναν ανιχνευτή ακμών Sobel. Εδώ χρησιμοποιείται η τεχνική μεταφοράς σημάτων με κοινόχρηστες μνήμες RAM. Ολόκληρο το πλαίσιο μιας εικόνας μεταφέρεται στην μνήμη RAM εισόδου του FPGA και κατόπιν γίνεται επεξεργασία κάθε ξεχωριστού εικονοστοιχείου. Τα επεξεργασμένα εικονοστοιχεία αποθηκεύονται στην μνήμη RAM εξόδου και κατόπιν ολόκληρη η εικόνα μεταφέρεται στον υπολογιστή. Αυτός ο τρόπος μεταφοράς δεδομένων αύξησε την απόδοση της προσομοίωσης με χρήση της τεχνικής FPGA-in-the-Loop η οποία χρειάστηκε παραπλήσιο χρόνο στην εκτέλεσή της με την προσομοίωση με software.

- Το τέταρτο και πολυπλοκότερο μοντέλο αφορά μετρήσεις Bit Error Rate (BER) σε συστήματα επικοινωνιών. Για να υπολογισθεί ένα σημείο σε ένα γράφημα BER απαιτείται η επεξεργασία πολλών εκατοντάδων χιλιάδων bit. Μιας και τα δεδομένα εισόδου είναι τυχαία, δεν χρειάζεται να δημιουργηθούν στον υπολογιστή και να μεταφερθούν στην πλακέτα FPGA. Μπορούν να δημιουργηθούν από το ίδιο το FPGA, να γίνει προσομοίωση της μετάδοσής τους μέσα από ένα περιβάλλον στο οποίο εκτίθενται σε θόρυβο, να αποκωδικοποιηθούν και να μετρηθούν τα λάθη που προκλήθηκαν. Όλη αυτή η διαδικασία μπορεί να γίνει χωρίς να χρειαστεί αλληλεπίδραση με τον υπολογιστή. Η επικοινωνία με τον υπολογιστή αφορά μόνο την αποστολή των παραμέτρων προσομοίωσης στην πλακέτα και την αποστολή των αποτελεσμάτων πίσω στον υπολογιστή. Σε αυτό το παράδειγμα υπάρχει πολύ μεγάλο πλήθος υπολογισμών και ελάχιστη μεταφορά δεδομένων μεταξύ software και FPGA. Το αποτέλεσμα είναι η προσομοίωση με χρήση της τεχνικής FPGA-in-the-Loop να είναι εκατοντάδες φορές γρηγορότερη από την προσομοίωση με software.

Για την καλύτερη κατανόηση της λειτουργίας του μοντέλου μετρήσεων BER προηγείται μια παρουσίαση των αρχών κωδικοποίησης-αποκωδικοποίησης και των όρων που χρησιμοποιούνται. Το μοντέλο μετρήσεων BER πρέπει να προσαρμοστεί στην εκάστοτε πλακέτα που διατίθεται. Αυτή η προσαρμογή απαιτεί μια μακροσκελή διαδικασία η οποία παρουσιάζεται λεπτομερώς.

Τέλος από τις μετρήσεις για τον βαθμό της επιτάχυνσης της προσομοίωσης του κάθε μοντέλου βγαίνουν συμπεράσματα για τους παράγοντες που την καθορίζουν:

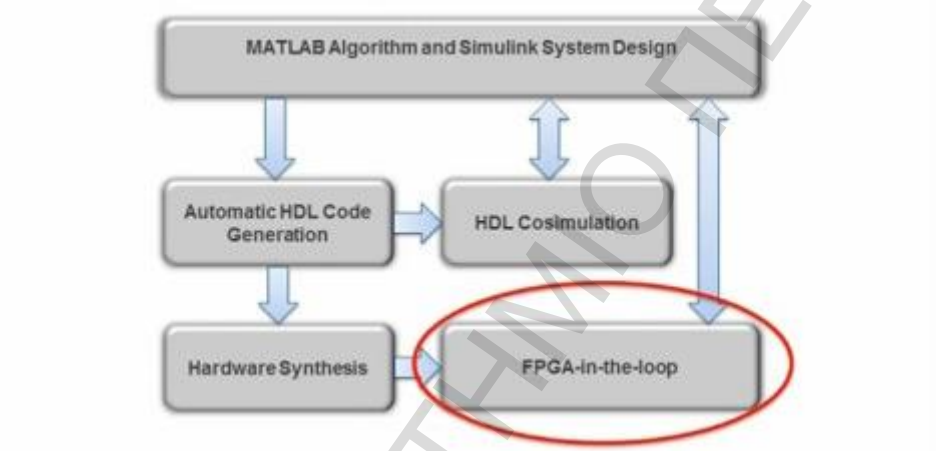
- Βλέπουμε ότι το FPGA μπορεί να επεξεργαστεί δεδομένα με ταχύτητες πολλαπλάσιες αυτών που μπορεί να πετύχει ένας υπολογιστής, χάρις την παραλληλία των FPGA.
- Το αδύνατο σημείο της προσομοίωσης με χρήση της τεχνικής FPGA-In-the-Loop βρίσκεται στον χρόνο που απαιτείται για την επικοινωνία μεταξύ FPGA και υπολογιστή.
- Σημαντικό ρόλο παίζει η πολυπλοκότητα των υπολογισμών. Θα πρέπει στο FPGA να εκτελούνται πολλοί υπολογισμοί σε κάθε κύκλο προσομοίωσης ώστε να ελπίζουμε σε κάποιο όφελος από την χρήση της τεχνικής FPGA-In-the-Loop.
- Η μαζική μεταφορά δεδομένων μεταξύ FPGA και υπολογιστή με χρήση κοινόχρηστων μνημών βοηθά στην επίτευξη καλύτερης απόδοσης της προσομοίωσης.

2. FPGA-In-the-Loop (FIL)

Στην τεχνική FPGA-In-the-Loop μία πλακέτα με FPGA συνδέεται στον υπολογιστή και συμμετέχει στην επεξεργασία των δεδομένων. Ο σκοπός μπορεί να είναι η επιτάχυνση των υπολογισμών ή η δοκιμή του κυκλώματος που έχει υλοποιηθεί στην πλακέτα.

2.1.1. Hardware Co-Simulation

Καθώς τα FPGA γίνονται όλο και πιο πολύπλοκα η λειτουργική επαλήθευση (verification) των σχεδιάσεων μόνο με προσομοιωτές HDL (Hardware Description Language – Γλώσσα Περιγραφής Υλικού) δεν είναι αρκετή για να ελέγξει κάθε λεπτομέρεια των κυκλωμάτων. Στα πολύπλοκα κυκλώματα η ταχύτητα προσομοίωσης είναι χαμηλή και εξαντλητικές δοκιμές κάτω από όλες τις συνθήκες απαιτούν ιδιαίτερα μεγάλο χρονικό διάστημα. Ένας τρόπος να επιταχυνθεί η διαδικασία είναι αντί να προσομοιώσουμε τον κώδικα HDL, να συνθέσουμε τον κώδικα αυτό ώστε να υλοποιηθεί το αντίστοιχο κύκλωμα σε ένα FPGA (εικόνα 3). Κατόπιν μπορούμε να τροφοδοτήσουμε το FPGA με σετ δεδομένων ελέγχου. Συνδέοντας κάρτες με FPGA στον υπολογιστή και σε συνεργασία με το εργαλείο προσομοίωσης μπορούν να γίνουν δοκιμές πάνω σε μεγάλα σετ δεδομένων σε σύντομο χρόνο για διαπιστωθεί η σωστή λειτουργία της σχεδίασης πάνω σε πραγματικά σενάρια.

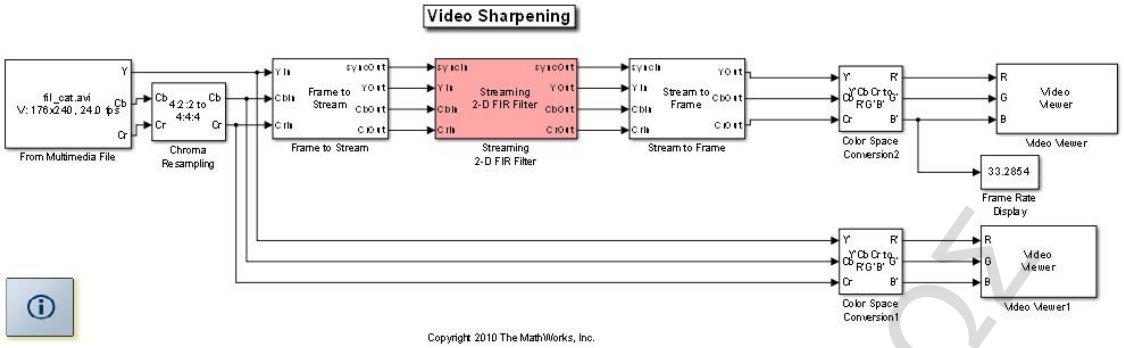


Εικόνα 3 Τεχνική FPGA-In-the-Loop για δοκιμή της σχεδίασης

Όπως φαίνεται στο διάγραμμα της εικόνας 3, η τεχνική FPGA-In-the-Loop μπορεί να είναι μέρος της ανάπτυξης ενός συστήματος. Αφού πρώτα δημιουργηθεί το μοντέλο του συστήματος με ένα εργαλείο όπως το MATLAB/Simulink μπορεί μετά από έναν πρώτο έλεγχο με τον προσομοιωτή HDL να δοκιμασθεί διεξοδικά με την τεχνική FPGA-In-the-Loop. Η τεχνική αυτή προσφέρει ταχύτητα και ακρίβεια.

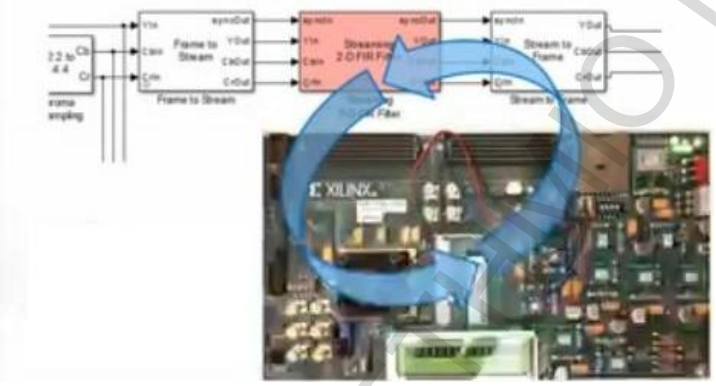
2.1.2. Simulation Acceleration

Υπάρχουν πολλά εργαλεία που προσφέρουν την δυνατότητα να προσομοιώσουμε την συμπεριφορά διαφόρων συστημάτων. Τα συστήματα αυτά μπορεί να προέρχονται από εντελώς διαφορετικά πεδία της επιστήμης όπως αεροδυναμική, βιολογία, ηλεκτρονικά, μετεωρολογία, χημεία και πολλά άλλα. Η προσομοίωση που προσφέρεται δεν έχει πάντα την απαιτούμενη ταχύτητα. Π.χ. στην εικόνα 4 φαίνεται ένα μοντέλο επεξεργασίας σήματος βίντεο (αύξησης της ευκρίνειας) υλοποιημένο στο εργαλείο Matlab της Mathworks (αναφορά 14 στην βιβλιογραφία). Το μπλοκ στα αριστερά είναι ένα αρχείο πολυμέσων που τροφοδοτεί το μοντέλο με πλαίσια εικόνας. Αυτά μετατρέπονται σε ακολουθία bit με το μπλοκ «Frame to Stream» και οδηγούνται στο μπλοκ «2D FIR Filter». Εκεί γίνεται επεξεργασία με σκοπό την αύξηση της ευκρίνειας της εικόνας και κατόπιν η ακολουθία bit ανασυντάσσεται σε πλαίσια εικόνας με το μπλοκ «Stream to Frame».



Εικόνα 4 Μοντέλο κυκλώματος επεξεργασίας εικόνας

Η προσομοίωση με software έχει πολύ χαμηλή ταχύτητα, της τάξης του 0,03 πλαίσια/δευτερόλεπτο. Η καθυστέρηση προκαλείται από τον μεγάλο χρόνο που απαιτείται για την επεξεργασία του κάθε bit του πλαισίου εικόνας στο μπλοκ «2D FIR Filter». Μπορούμε να επιταχύνουμε την διαδικασία με την τεχνική FPGA-In-the-Loop. Αντικαθιστούμε το μπλοκ αυτό (μπλοκ με κόκκινο χρώμα στην εικόνα 4) με κύκλωμα που θα υλοποιηθεί στο FPGA. Τα υπόλοιπα μπλοκ εξακολουθούν να προσομοιώνονται στον υπολογιστή. Τα δεδομένα που τροφοδοτούν το μπλοκ επεξεργασίας στέλνονται μέσω του κατάλληλου αγωγού στην πλακέτα που ενσωματώνει το FPGA (εικόνα 5). Εκεί οι υπολογισμοί εκτελούνται σε χρόνο πολύ μικρότερο από τον χρόνο εκτέλεσής τους με software. Κατόπιν τα επεξεργασμένα δεδομένα επιστρέφουν στον υπολογιστή μέσω του ίδιου αγωγού.



Εικόνα 5 Τεχνική FPGA-In-the-Loop για επιτάχυνση προσομοίωσης

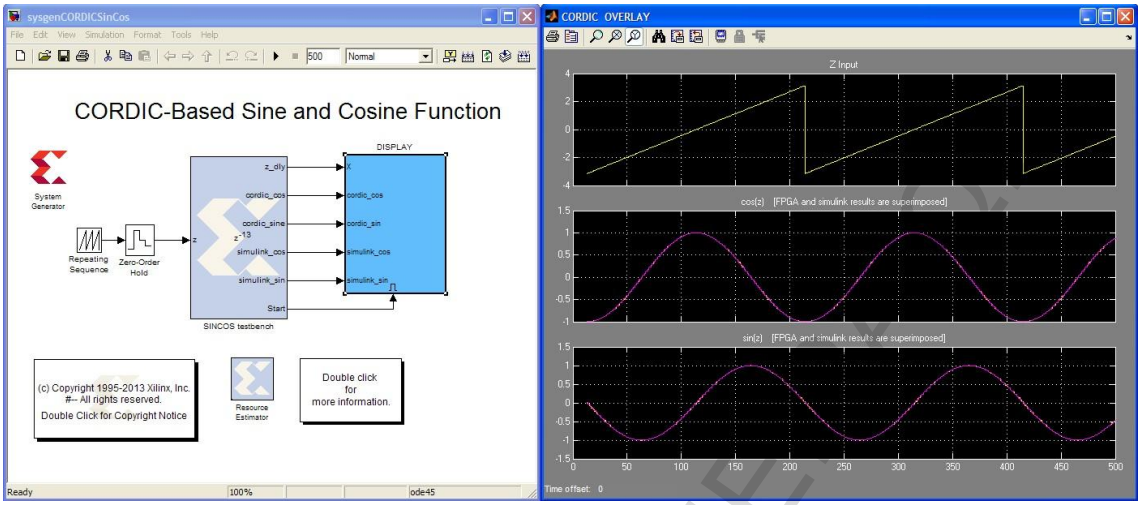
Με την τεχνική FPGA-In-the-Loop η προσομοίωση επιταχύνεται και ο ρυθμός επεξεργασίας ανεβαίνει στα 33 πλαίσια/δευτερόλεπτο. Αν και το FPGA εργάζεται με συχνότητα ρολογιού χαμηλότερη από αυτήν ενός υπολογιστή, η ταχύτητα με την οποία ολοκληρώνονται οι υπολογισμοί είναι πολύ μεγάλη λόγω της παραλληλίας που υπάρχει στα FPGA. Ενώ στον υπολογιστή οι πράξεις εκτελούνται η μια μετά την άλλη στην αριθμητική-λογική μονάδα (ALU), στο FPGA δημιουργείται ξεχωριστό κύκλωμα για τον κάθε υπολογισμό. Έτσι μπορούν να εκτελούνται πολλοί υπολογισμοί παράλληλα. Αποτέλεσμα είναι το σύνολο των υπολογισμών να απαιτεί λίγους μόνο κύκλους ρολογιού.

2.2. Περιβάλλοντα FIL (FPGA-In-the-Loop)

2.2.1 Matlab/Simulink + Xilinx ISE

Το εργαλείο Matlab παρέχει ένα περιβάλλον για αριθμητικούς υπολογισμούς. Χειρίζεται πίνακες και δημιουργεί γραφήματα συναρτήσεων και δεδομένων. Αλγόριθμοι μπορούν να περιγραφούν σε γλώσσα υψηλού επιπέδου. Με την προσθήκη του εργαλείου μοντελοποίησης-προσομοίωσης Simulink προσφέρει την δυνατότητα δημιουργίας μοντέλων με γραφικό τρόπο και προσομοίωσής τους. Η βιβλιοθήκη του Simulink περιλαμβάνει μπλοκ για την δημιουργία μοντέλων που αφορούν πολλούς διαφορετικούς τομείς της επιστήμης. Με την προσθήκη του

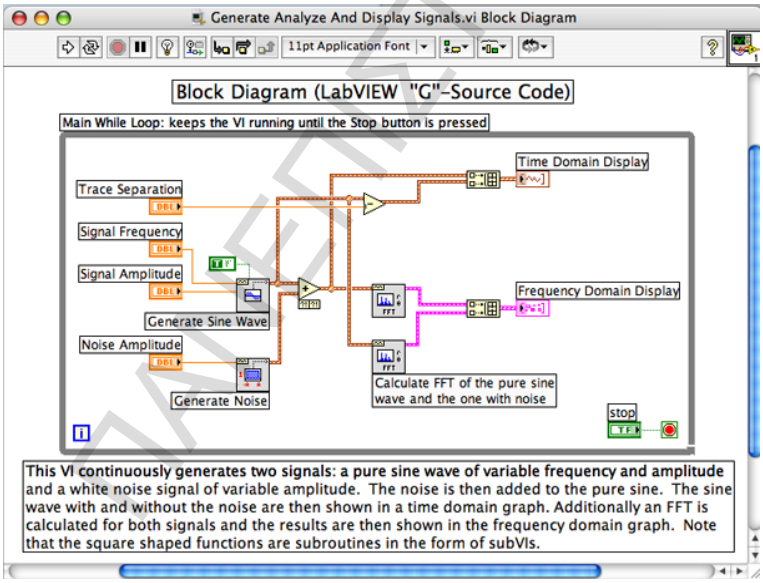
περιβάλλοντος ανάπτυξης ISE της Xilinx προστίθεται στην βιβλιοθήκη του Simulink μια ομάδα από μπλοκ που αντιστοιχίζονται σε υλικό σε FPGA. Πέρα από την προσομοίωση της σχεδίασης με software (εικόνα 6), μπορεί να γίνει υλοποίηση του κυκλώματος σε FPGA και σε συνεργασία με το περιβάλλον του Simulink να γίνει hardware co-simulation στο μοντέλο που έχει σχεδιαστεί.



Εικόνα 6 Matlab/Simulink + Xilinx ISE

2.2.2. NI Labview

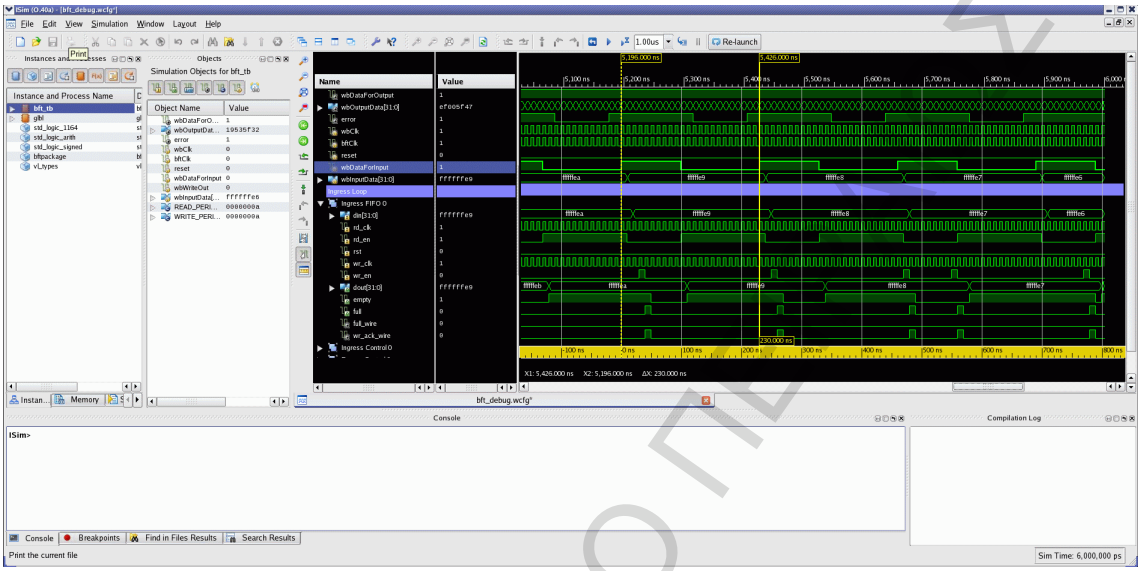
Το εργαλείο LabVIEW έχει κατασκευαστεί από την National Instruments. Είναι μια πλατφόρμα σχεδίασης συστημάτων σε γραφικό περιβάλλον (εικόνα 7). Για την περιγραφή του μοντέλου χρησιμοποιεί την γλώσσα προγραμματισμού «G» την οποία επίσης έχει αναπτύξει η National Instruments. Μπορεί να χρησιμοποιηθεί για την διεξαγωγή μετρήσεων, για δοκιμές και για αυτοματισμό. Με την προσθήκη του εργαλείου LabVIEW-FPGA μπορεί να χρησιμοποιηθεί για την ανάπτυξη συστημάτων σε FPGA χρησιμοποιώντας γραφικό περιβάλλον, χωρίς να χρειάζεται η χρήση γλωσσών περιγραφής υλικού. Μπορεί να χρησιμοποιηθεί με την τεχνική FPGA-In-the-Loop για επιτάχυνση προσομοίωσης ή επαλήθευση των χαρακτηριστικών της σχεδίασης με την εγκατάσταση του πρόσθετου εργαλείου NI HIL Simulator.



Εικόνα 7 NI LabVIEW

2.2.3. Xilinx Isim

Το Isim είναι ένας πλήρης προσομοιωτής HDL ο οποίος είναι ενσωματωμένος στο περιβάλλον ανάπτυξης ISE της Xilinx. Υποστηρίζει τις γλώσσες περιγραφής υλικού VHDL-93 και VERILOG-2001. Έχει δυνατότητα σχεδίασης των κυματομορφών του κυκλώματος την λειτουργία του οποίου προσομοιώνει (εικόνα 8). Διαθέτει δυνατότητα προσομοίωσης με την τεχνική hardware co-simulation με την οποία μπορεί να επιταχύνει την προσομοίωση σε σχεδιάσεις που απαιτούν πολλούς υπολογισμούς αλλά και να επαληθεύσει την λειτουργία των κυκλωμάτων.



Εικόνα 8 Xilinx ISim

3. Περιβάλλον Μοντελοποίησης και Προσομοίωσης Συστημάτων

3.1. Εισαγωγή

Πολλές φορές η συλλογή και ο έλεγχος κάποιων στοιχείων είναι πολύ δύσκολο να πραγματοποιηθούν και ο πειραματισμός ανέφικτος για πρακτικούς, οικονομικούς ή ηθικούς λόγους. Είναι όμως δυνατόν να δημιουργήσουμε μαθηματικά μοντέλα πραγματικών καταστάσεων τα οποία επιτρέπουν την παρακολούθηση της εξέλιξης τέτοιων καταστάσεων, αλλάζοντας μόνο τις τιμές κάποιων παραμέτρων. Με την εξέλιξη του υλικού και του λογισμικού των υπολογιστών όλο και περισσότερα περιβάλλοντα προσομοίωσης συστημάτων χτίζονται καθημερινά. Τα περιβάλλοντα αυτά δεν αντικαθιστούν τις πραγματικές συνθήκες ενός συστήματος αλλά έχει αποδειχτεί ότι τα συμπεράσματα που εξάγονται σε καλά οργανωμένα περιβάλλοντα προσομοίωσης βρίσκονται πολύ κοντά στην πραγματικότητα.

3.2. Σχεδίαση και προσομοίωση συστημάτων στο Simulink

Το εργαλείο Simulink είναι ένα περιβάλλον σχεδίασης και προσομοίωσης συστημάτων. Χρησιμοποιεί μπλοκ λειτουργιών με τα οποία μπορούν να μοντελοποιηθούν συστήματα που καλύπτουν πολλά διαφορετικά πεδία της επιστήμης και της τεχνολογίας. Μπορεί να χρησιμοποιηθεί για σχεδίαση συστημάτων, αυτόματη παραγωγή κώδικα και για δοκιμές και επαλήθευση των χαρακτηριστικών των ενσωματωμένων συστημάτων. Το Simulink παρέχει γραφικό περιβάλλον και βιβλιοθήκες με προκαθορισμένα μπλοκ λειτουργιών με τα οποία συντίθεται το σύστημα. Καθώς το Simulink είναι ενσωματωμένο στο εργαλείο MATLAB, είναι δυνατή η εισαγωγή αλγορίθμων από το MATLAB στα μοντέλα του Simulink και η εξαγωγή των αποτελεσμάτων της προσομοίωσης στο MATLAB για περαιτέρω ανάλυση.

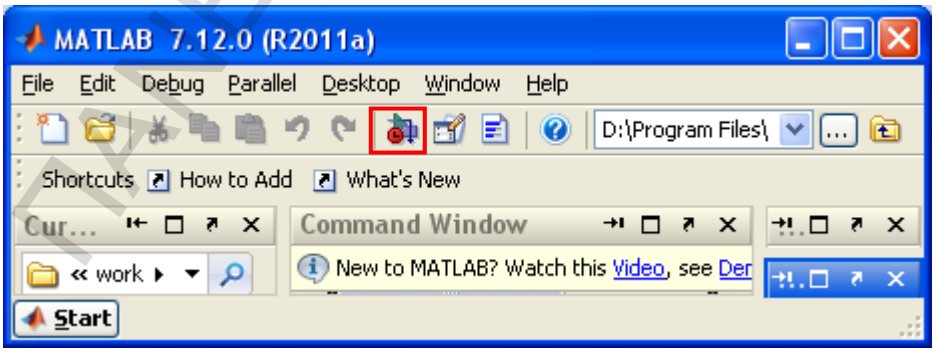
Ανάμεσα στα χαρακτηριστικά του εργαλείου Simulink περιλαμβάνονται:

- Γραφικό περιβάλλον για την δημιουργία και την διαχείριση σχεδιάσεων συστημάτων.
- Βιβλιοθήκες από προκαθορισμένα μπλοκ για την μοντελοποίηση συστημάτων συνεχούς και διακριτού χρόνου.
- Προσομοίωση συστημάτων με επίλυση σταθερού ή μεταβλητού βήματος.
- Παλμογράφοι και οθόνες ενδείξεων για εμφάνιση των αποτελεσμάτων της προσομοίωσης.
- Εργαλεία διαχείρισης έργων και δεδομένων για τον χειρισμό αρχείων μοντέλων και δεδομένων.
- Μπλοκ λειτουργιών για την εισαγωγή αλγορίθμων του MATLAB στα μοντέλα.
- Δυνατότητα για εισαγωγή κώδικα C και C++ στα μοντέλα.

Το Simulink ξεκινά από το Matlab είτε γράφοντας:

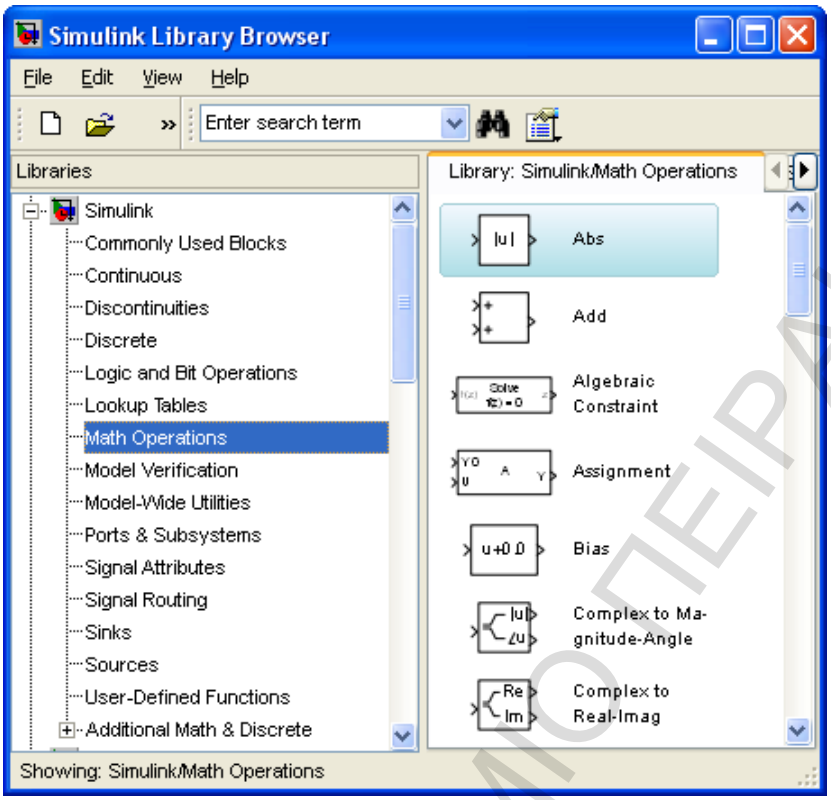
Simulink

στο παράθυρο εντολών του Matlab είτε κάνοντας κλικ στο εικονίδιο «Simulink Library Browser» στο πάνω μέρος του Matlab command window (εικόνα 9).



Εικόνα 9 Simulink button

Ο browser της βιβλιοθήκης του Simulink εμφανίζεται. Τα μπλοκ που θα χρειαστούν για την μοντελοποίηση συστημάτων βρίσκονται σε υποφακέλους μέσα στο κύριο φάκελο του Simulink (εικόνα 10).



Εικόνα 10 Simulink Library Browser

3.2.1.Βασικά στοιχεία

Υπάρχουν δύο κύριες κλάσεις στοιχείων στο Simulink: Μπλοκ και γραμμές. Τα μπλοκ χρησιμοποιούνται για να δημιουργήσουν, να τροποποιήσουν, να συνδυάσουν, να εξάγουν και να παρουσιάσουν σήματα. Οι γραμμές χρησιμοποιούνται για να μεταφέρουν τα σήματα από το ένα μπλοκ στο άλλο.

3.2.2. Μπλοκ

Οι υποφάκελοι κάτω από τον φάκελο Simulink περιέχουν τις γενικές κλάσεις μπλοκ που διατίθενται για χρήση:

Continuous: γραμμικά και συστήματα συνεχούς χρόνου.

Discontinuities: μη γραμμικά συστήματα.

Discrete: γραμμικά συστήματα διακριτού χρόνου.

Logic and Bit operations: πράξεις λογικής και χειρισμού bit.

Model Verification: έλεγχος διαφόρων συνθηκών επί των σημάτων.

Lookup Tables: Έτοιμοι πίνακες π.χ. με τιμές ημιτόνων αλλά και δυνατότητα δημιουργίας πινάκων μίας ή δύο διαστάσεων.

Math operations: Αριθμητικές πράξεις.

Model wide utilities: Μπλοκ που επιτρέπουν την εισαγωγή τεκμηρίωσης και οδηγιών στο μοντέλο.

Ports and subsystems: Μπλοκ δημιουργίας υποσυστημάτων.

Signal attributes: χειρισμός των παραμέτρων του συστήματος.

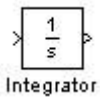
Signal routing: διακόπτες, πολυπλέκτες κλπ.

Sinks: Χρησιμοποιείται για την έξοδο και την παρουσίαση σημάτων.

Sources: Χρησιμοποιείται για δημιουργία διαφόρων σημάτων.

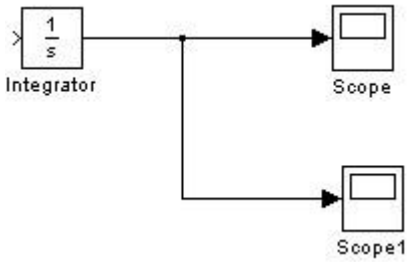
User defined functions: Δημιουργία μπλοκ με λειτουργία καθορισμένη από τον χρήστη σε κώδικα matlab.

Τα μπλοκ έχουν από μηδέν έως αρκετές εισόδους και από μηδέν έως αρκετές εξόδους. Οι ασύνδετες είσοδοι παρουσιάζονται σαν ένα μικρό βέλος και οι ασύνδετες έξοδοι παρουσιάζονται σαν ένα μικρό τρίγωνο. Το παρακάτω μπλοκ έχει μία ασύνδετη είσοδο στα αριστερά και μία ασύνδετη έξοδο στα αριστερά.



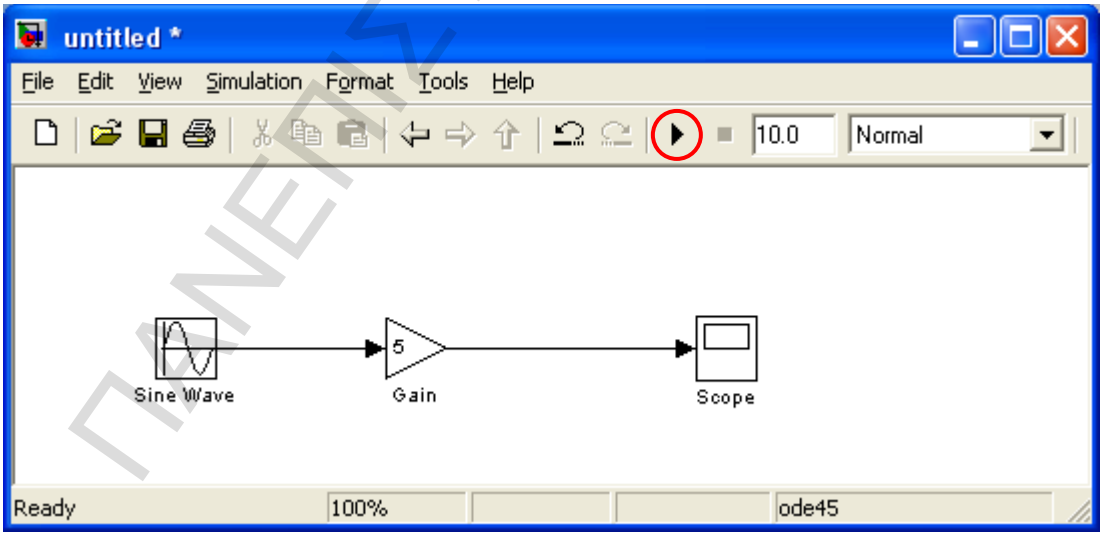
3.2.3.Γραμμές

Οι γραμμές μεταφέρουν τα σήματα από μία έξοδο σε μία ή περισσότερες εισόδους. Το βέλος δείχνει την κατεύθυνση. Δύο έξοδοι δεν μπορούν να συνδεθούν μεταξύ τους με μία γραμμή προκειμένου να οδηγηθούν μαζί σε μια είσοδο.



3.3. Σχεδίαση ενός συστήματος στο Simulink

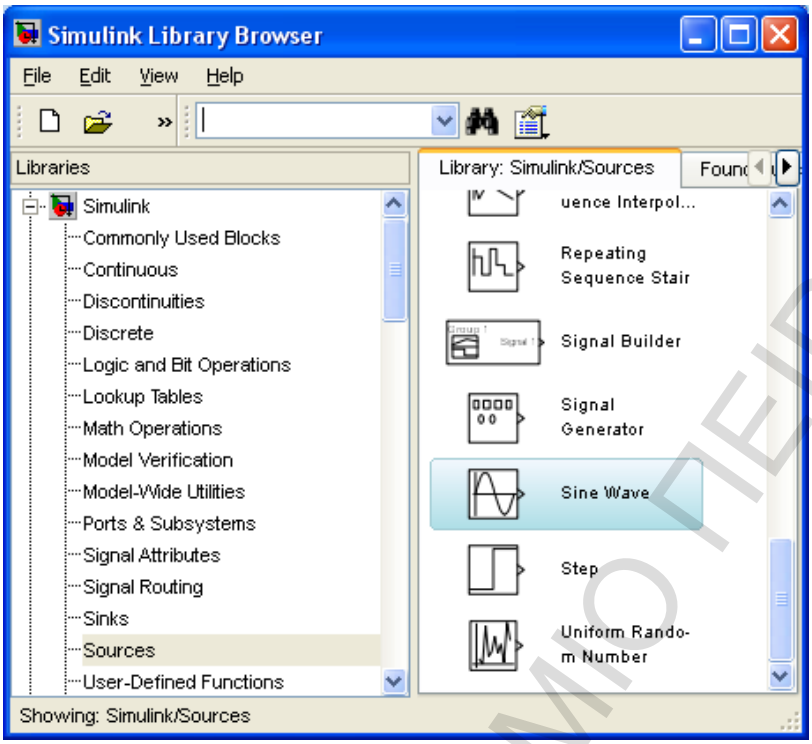
Η διαδικασία σχεδίασης ενός συστήματος θα φανεί καλύτερα μέσα από ένα παράδειγμα. Έστω πως θέλουμε να σχεδιάσουμε το σύστημα που φαίνεται στην εικόνα 11.



Εικόνα 11 Παράδειγμα μοντέλου Simulink

Το μοντέλο αυτό αποτελείται από τρία μπλοκ: Γεννήτρια ημιτονοειδούς σήματος (sine wave), ρύθμιση κέρδους (gain) και παλμογράφος (scope).

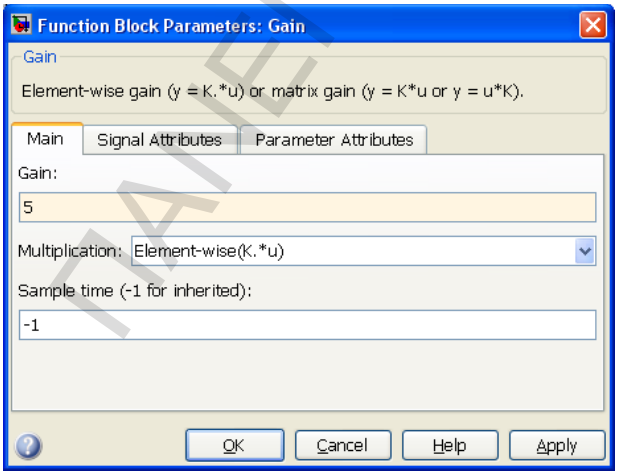
Κατ' αρχάς δημιουργούμε ένα νέο μοντέλο: File→New→Model από το Simulink Library Browser. Κατόπιν συλλέγουμε τα απαραίτητα μπλοκ. Η γεννήτρια ημιτονοειδούς σήματος βρίσκεται μέσα στον υποφάκελο sources της κλάσης Simulink όπως φαίνεται στη εικόνα 12. Για να την χρησιμοποιήσουμε κάνουμε κλικ επάνω στο μπλοκ και το σέρνουμε στο κενό παράθυρο του μοντέλου που δημιουργήσαμε. Συνεχίζουμε προσθέτοντας το μπλοκ gain από τον υποφάκελο math και το μπλοκ scope από το υποφάκελο sinks.



Εικόνα 12 Επιλογή μπλοκ

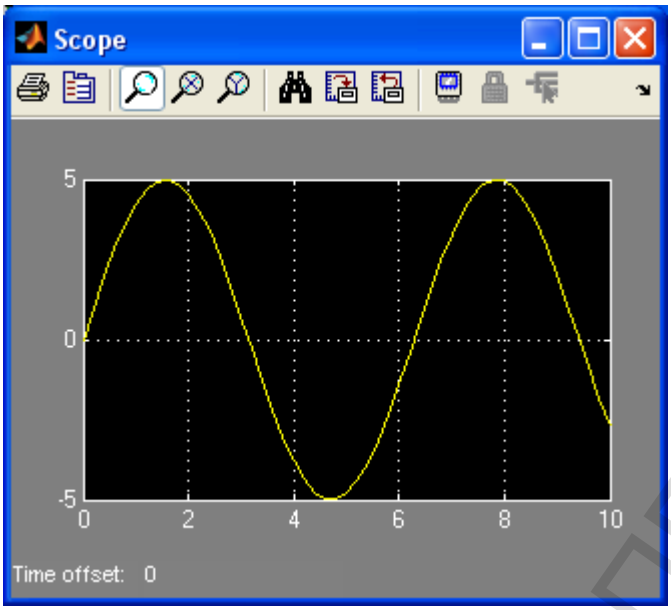
Για να συνδέσουμε τα μπλοκ μεταξύ τους κάνουμε κλικ στην έξοδο της γεννήτριας ημιτονοειδούς σήματος και σέρνουμε τον δείκτη στην είσοδο του μπλοκ gain. Ένας πιο εύκολος τρόπος για να το πετύχουμε αυτό είναι να κάνουμε ένα κλικ στην γεννήτρια και κατόπιν να κάνουμε κλικ στο μπλοκ gain κρατώντας πατημένο το πλήκτρο ctrl. Παρομοίως συνδέουμε την έξοδο του μπλοκ gain στον παλμογράφο (scope).

Θα πρέπει στη συνέχεια να ρυθμίσουμε τις παραμέτρους των μπλοκ. Θέλουμε να ορίσουμε το κέρδος του μπλοκ gain στην τιμή 5. Κάνουμε διπλό κλικ επάνω στο μπλοκ και στο πλαίσιο Gain εισάγουμε την τιμή 5 (εικόνα 13). Πατάμε το κουμπί OK.



Εικόνα 13 Ρύθμιση παραμέτρων του μπλοκ Gain

Τέλος κάνουμε διπλό κλικ πάνω στο μπλοκ του παλμογράφου για να εμφανισθεί η έξοδος του συστήματος (εικόνα 14). Για να ξεκινήσει η προσομοίωση κάνουμε κλικ στο «βέλος δεξιά» στην γραμμή εργαλείων του μοντέλου (εικόνα 11).

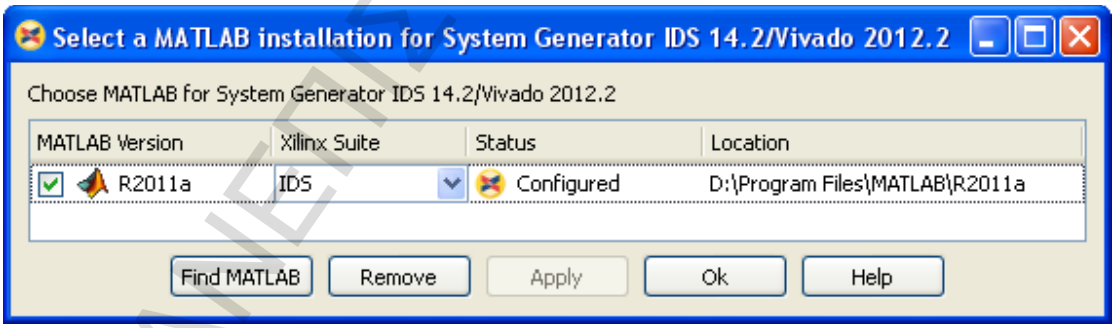


Εικόνα 14 Έξοδος του συστήματος

3.4. Xilinx Toolkit

Το εργαλείο System Generator μας επιτρέπει να χρησιμοποιήσουμε το εργαλείο Simulink για να συνθέσουμε και να προσομοιώσουμε ένα μοντέλο που προορίζεται να υλοποιηθεί με hardware σε ένα fpga.

Αφού εγκαταστήσουμε το περιβάλλον ανάπτυξης Xilinx Ise Design Suite και το εργαλείο Matlab της Mathworks σε ένα σύστημα μπορούμε να χρησιμοποιήσουμε την επιλογή «System Generator MATLAB Configurator» (εικόνα 15) που υπάρχει στο μενού έναρξης του Xilinx Design Tools για να συνδέσουμε τις δύο εφαρμογές. Με αυτόν τον τρόπο προστίθεται στις βιβλιοθήκες εργαλείων του Simulink μία νέα βιβλιοθήκη με μπλοκ της Xilinx.

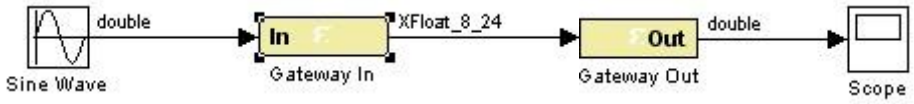


Εικόνα 15 Σύνδεση του Matlab με το System Generator

Τα μπλοκ αυτά τα χρησιμοποιούμε όπως και τα μπλοκ του Simulink για να δημιουργήσουμε το μοντέλο μας. Η διαφορά είναι ότι τα μπλοκ αυτά μπορούν να αντιστοιχηθούν σε hardware και να υλοποιηθούν σε ένα fpga. Αφότου συνθέσουμε το μοντέλο μας μπορούμε να κάνουμε προσομοίωση με χρήση software, αλλά το σημαντικό είναι πως υπάρχει η δυνατότητα να γίνει η προσομοίωση με το μοντέλο υλοποιημένο σε πραγματικό hardware σε fpga.

Ένα μοντέλο μπορεί να περιέχει και μπλοκ του Simulink και μπλοκ της Xilinx. Τα όρια του τμήματος του μοντέλου που θα υλοποιηθεί στο fpga είναι τα μπλοκ gateway in και gateway out.

Ανάμεσα σε αυτά τα δύο μπλοκ μπορούν να υπάρχουν μόνο μπλοκ της Xilinx. Το τμήμα αυτό θα υλοποιηθεί στο fpga.



Το μπλοκ gateway in μετατρέπει τα σήματα τύπου double (κινητής υποδιαστολής) του Simulink σε έναν από τέσσερεις τύπους δεδομένων που υποστηρίζει το System Generator :

1. Fixed Unsigned - Μη προσημασμένοι για λειτουργίες σε θετικούς αριθμούς μόνο.
2. Fixed Signed - Προσημασμένοι με συμπλήρωμα ως προς δύο, για λειτουργίες που αφορούν θετικούς και αρνητικούς αριθμούς.
3. Floating point.
4. Boolean για ενέργειες σε επίπεδο bit.

Στους τύπους fixed το πλήθος των bit που χρησιμοποιούνται για το ακέραιο και για το δεκαδικό μέρος του αριθμού μπορούν να καθορισθούν ανά ένα bit και όχι υποχρεωτικά ανά βήματα των οκτώ bit. Αντίστοιχα στους τύπους floating μπορεί να καθορισθεί το πλήθος των bit σε έναν από τους δύο προκαθορισμένους τύπους single και double ή να καθορισθεί ελεύθερα. Προκειμένου να πετύχουμε την καλύτερη δυνατή απόδοση θα πρέπει να χρησιμοποιούμε τον μικρότερο αριθμό bit που απαιτείται για να αναπαρασταθούν τα μεγέθη των σημάτων.

Στις ρυθμίσεις του gateway in υπάρχει δυνατότητα να καθορίσουμε τι θα συμβεί σε περίπτωση υπερχείλισης (overflow). Μπορούμε να επιλέξουμε ανάμεσα στην αναδίπλωση (wrap) που είναι η προκαθορισμένη επιλογή και στον κορεσμό (saturate) που κοστίζει σε επιπλέον υλικό. Υπάρχει επίσης η δυνατότητα flag as error όπου ένα flag δείχνει ότι συνέβη υπερχείλιση στην είσοδο και αναλαμβάνουμε οι ίδιοι τον χειρισμό της.

Τέλος υπάρχει δυνατότητα να επιλέξουμε τι θα συμβεί στο τελευταίο bit του σήματος εισόδου κατά την κβαντοποίηση. Μπορούμε να επιλέξουμε στρογγυλοποίηση (round) στην κοντινότερη τιμή του σήματος, επιλογή που κοστίζει σε υλικό ή μπορούμε να επιλέξουμε αποκοπή (truncate) των υπόλοιπων ψηφίων για απλότητα αλλά με μικρότερη ακρίβεια.

3.4.1. Ενέργειες σε επίπεδο bit

Στα συστήματα επεξεργασίας σημάτων υπάρχουν λειτουργίες που δεν μπορούν να εκφρασθούν με μαθηματικούς τύπους αλλά απαιτείται να γίνει άμεσος χειρισμός στα bit που αναπαριστούν την τιμή του σήματος. Τέτοιες ενέργειες γίνονται με τα μπλοκ που παρουσιάζονται παρακάτω.

- Reinterpret. Το μπλοκ αυτό αλλάζει τον τύπο των δεδομένων χωρίς να επηρεάζει τα ίδια τα bit. Π.χ. ο αριθμός 4 αναπαριστάται σε τύπο μη προσημασμένο [4 1] ως 1000. Εάν αυτά τα bit περάσουν από ένα μπλοκ Reinterpret και μεταφραστούν σε τύπο μη προσημασμένο [4 0] τότε η τιμή που θα αναπαριστούν τα bit 1000 θα είναι 8.
- Convert. Αυτό το μπλοκ αλλάζει τα bit που αναπαριστούν τον αριθμό διατηρώντας την τιμή του αριθμού αναλλοίωτη. Μπορεί να αλλάξει το πλήθος των bit που χρησιμοποιούνται για να αναπαραστήσουν τον αριθμό ή να αλλάξει τον τύπο του αριθμού από προσημασμένο σε μη προσημασμένο και αντίστροφα. Συνήθως χρησιμοποιείται για να αποκόψει τα δεκαδικά bit μετά από την πράξη του πολλαπλασιασμού.
- Concat. Το μπλοκ αυτό ενώνει δύο εισόδους σε μία έξοδο. Έχει δύο εισόδους, την HI και την LOW. Τα bit της εισόδου HI παίρνουν την θέση των σημαντικότερων bit (msb) στο σήμα εξόδου ενώ τα bit της εισόδου LOW παίρνουν την θέση των λιγότερο σημαντικών bit (lsb) στο σήμα εξόδου. Χρησιμοποιείται κυρίως για να προσθέσει μηδενικά bit στα msb ή στα lsb ενός σήματος.
- Slice. Μπορεί να χρησιμοποιηθεί για να δώσει πρόσβαση σε συγκεκριμένα bit του σήματος. Για παράδειγμα μπορεί να επιλέξει μόνο το msb ενός σήματος εισόδου ή μόνο τα bit του ακέραιου τμήματος ή μόνο τα τρία πρώτα από τα bit του δεκαδικού μέρους.
- BitBasher. Το μπλοκ αυτό χρησιμοποιεί την σύνταξη της verilog για να χειριστεί τα bit του σήματος εισόδου.

- Expression. Το μπλοκ αυτό εκτελεί λειτουργίες and, or και xor στα bit των δύο σημάτων. Οι είσοδοι μπορούν να έχουν περισσότερα από ένα bit.
  Τα σύμβολα που χρησιμοποιούνται για τις πράξεις είναι:

  Προτεραιότητα ()

  NOT ~

  AND &

  OR |

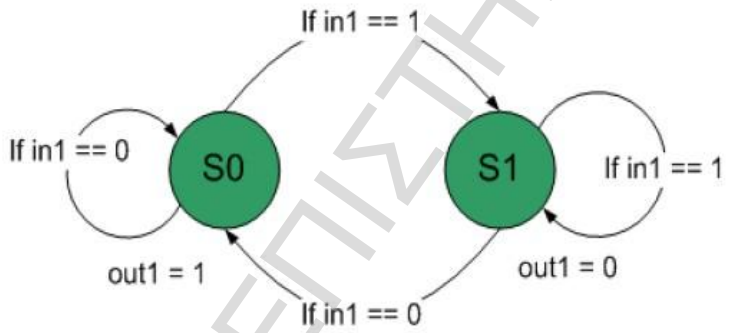
  XOR ^

  Το πλήθος των εισόδων καθορίζεται από την συνάρτηση. Π.χ. η συνάρτηση « a & b | c » υποδεικνύει πως στο μπλοκ αυτό υπάρχουν τρεις είσοδοι.

- MCode. Αυτό το μπλοκ επιτρέπει την εισαγωγή κώδικα matlab ο οποίος θα μεταφραστεί σε vhdl. Δεν είναι κατάλληλο για να περιγράψει αλγοριθμικούς χειρισμούς όπως ένα φίλτρο FIR ή για να υπολογίσει τον αντίστροφο ενός πίνακα. Προορίζεται για να περιγράψει με εύκολο τρόπο μηχανές καταστάσεων (state machines). Η Xilinx παρέχει την εντολή xl_state στο matlab. Με αυτήν πρέπει να δοθεί αρχική τιμή στην μεταβλητή τύπου persistent η οποία θα δείχνει την κατάσταση στην οποία βρίσκεται η μηχανή καταστάσεων. Η xl_state δέχεται δύο παραμέτρους: την αρχική κατάσταση της μεταβλητής και το πλήθος των bit που απαιτούνται για την αναπαράσταση της μεταβλητής αυτής.

3.4.2. Παράδειγμα μηχανής καταστάσεων

Το σχήμα που ακολουθεί (εικόνα 16) δείχνει το διάγραμμα καταστάσεων μίας απλής μηχανής καταστάσεων με δύο καταστάσεις.



Εικόνα 16 Διάγραμμα καταστάσεων

Με την εντολή xl_state μία μεταβλητή που ονομάζεται state ορίζεται ως τύπος persistent με μέγεθος δύο bit με αναπαράσταση ως μη προσημασμένου αριθμού. Οι εντολές switch - case χρησιμοποιούνται για να αποκωδικοποιήσουν τις εισόδους, να καθορίσουν την επόμενη κατάσταση και να ορίσουν τις εξόδους. Ακολουθεί ο κώδικας Matlab που περιγράφει την λειτουργία της μηχανής καταστάσεων.

```
function [out1] = fsm(in1)

persistent state,
state=xl_state(0, {xlUnsigned, 2, 0});

switch double(state)
```

```
case 0
        if in1==1;
                out1=0;
                state=1
        else
                out1=1;
                state=0;
        end
case 1
        if in1==0
                out1=1;
                state=0;
        else
                out1=0;
                state=1;
        end
otherwise
        state=0;
        out1=0;
end
```

Αυτό είναι ένα απλό παράδειγμα που εύκολα μπορεί να επεκταθεί σε πιο πολύπλοκα συστήματα.

Τα περισσότερα από τα μπλοκ του system generator τα οποία έχουν κάποιο είδος μνήμης διαθέτουν επιλογές για εισόδους reset και clock enable. Αυτές οι είσοδοι πρέπει να συνδεθούν σε κάποιο Boolean σήμα. Αν οι είσοδοι αυτές επιλεγούν να μην φαίνονται τότε συνδέονται αυτόματα στο κεντρικό reset και clock enable αντίστοιχα.

3.4.3. Παράδειγμα σχεδίασης κυκλώματος με το xilinx toolkit

Θα παρουσιάσουμε ένα παράδειγμα σχεδίασης ενός απλού μοντέλου με το xilinx toolkit. Συγκεκριμένα θα κατασκευάσουμε ένα κύκλωμα που θα υπολογίζει την έκφραση:

Z= (5*x) + (3*y)

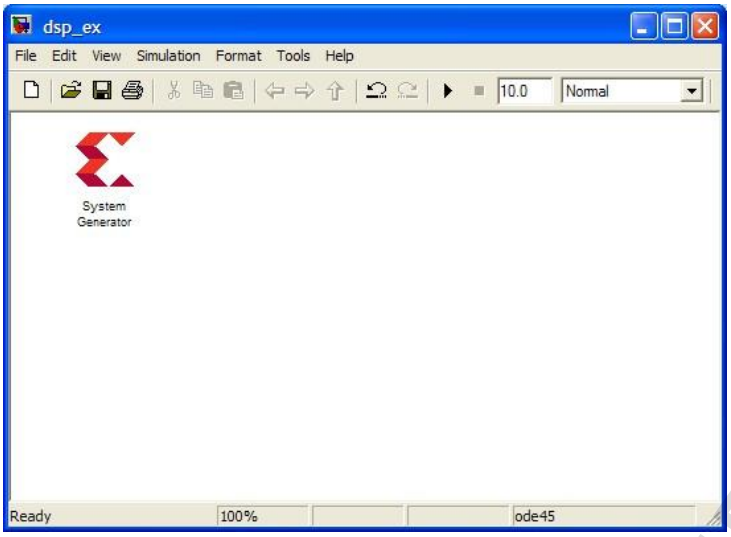
Οι κύριες λειτουργίες που θα υλοποιηθούν είναι:

- Δύο λειτουργίες πολλαπλασιασμού.
- Μία λειτουργία πρόσθεσης.
- Δύο στοιχεία αποθήκευσης όπου θα αποθηκευθούν οι συντελεστές 5 και 3. Σ' αυτό το παράδειγμα θα χρησιμοποιήσουμε το μπλοκ σταθεράς για αυτό το σκοπό.

Υπάρχουν μπλοκ για κάθε μία από αυτές τις λειτουργίες. Επίσης θα χρειαστεί και ένα μπλοκ «system generator block» το οποίο είναι απαραίτητο σε κάθε σχεδίαση που περιέχει μπλοκ της Xilinx.

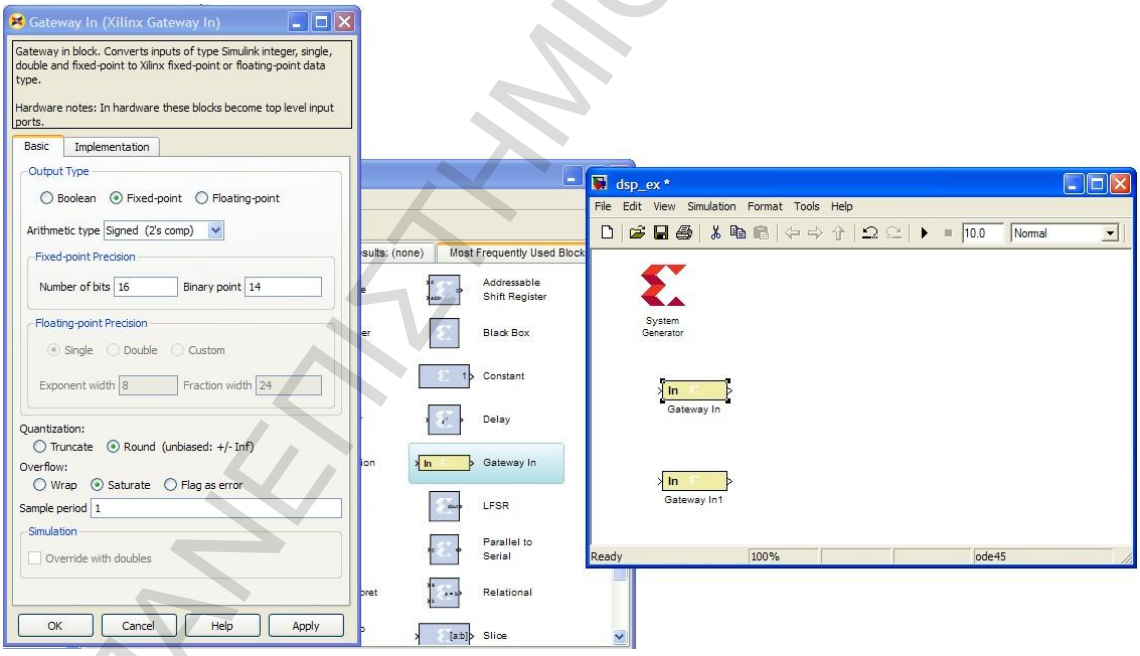
Ξεκινάμε το system generator και αρχίζουμε την σχεδίαση δημιουργώντας ένα νέο μοντέλο. Από το Simulink Library Browser επιλέγουμε file→new→model.

Το πρώτο μπλοκ που θα τοποθετήσουμε είναι το system generator block. Αυτό βρίσκεται στην κατηγορία μπλοκ του Simulink: Xilinx blockset→Basic Elements→System Generator. Τοποθετούμε το μπλοκ αυτό στο νέο παράθυρο όπως στην εικόνα 17.



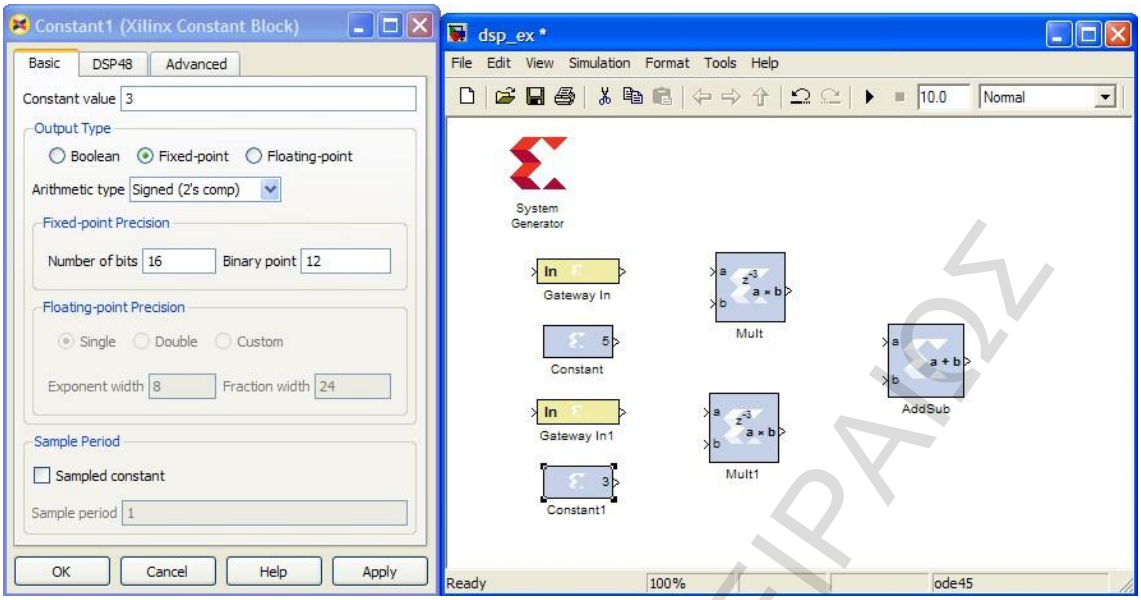
Εικόνα 17 Δημιουργία νέας σχεδίασης

Στην συνέχεια προσθέτουμε στην σχεδίασή μας δύο μπλοκ gateway in. Αυτά μετατρέπουν τους αριθμούς κινητής υποδιαστολής (floating point) του Simulink σε αριθμούς σταθερής υποδιαστολής (fixed point) που χειρίζεται το FPGA. Τα μπλοκ αυτά υπάρχουν στην κατηγορία μπλοκ: Xilinx blockset→Basic Elements→Gateway in. Κάνοντας διπλό κλικ πάνω στα μπλοκ gateway in μπορούμε να αλλάξουμε τον τρόπο αναπαράστασης των αριθμών (εικόνα 18). Θέτουμε τον τύπο εξόδου σε «συμπλήρωμα ως προς δύο» (signed 2's comp.), πλήθος bit 16, και σημείο υποδιαστολής στο 12^{o} bit. Επαναλαμβάνουμε τις ρυθμίσεις και για το δεύτερο μπλοκ gateway in.



Εικόνα 18 Προσθήκη Gateway in στο μοντέλο.

Στην συνέχεια θα προσθέσουμε δύο πολλαπλασιαστές. Τα μπλοκ πολλαπλασιασμού βρίσκονται στο Xilinx Blockset→Math→Mult. Η έκφραση που θέλουμε να υπολογίσουμε απαιτεί τον πολλαπλασιασμό δύο μεταβλητών με δύο σταθερές. Οι μεταβλητές θα δίνονται από τα μπλοκ gateway in και οι σταθερές θα δίνονται από μπλοκ σταθερών. Αυτά βρίσκονται στο Xilinx Blockset→Basic Elements→Constant. Προσθέτουμε δύο μπλοκ σταθερών όπως φαίνεται στην εικόνα 19.

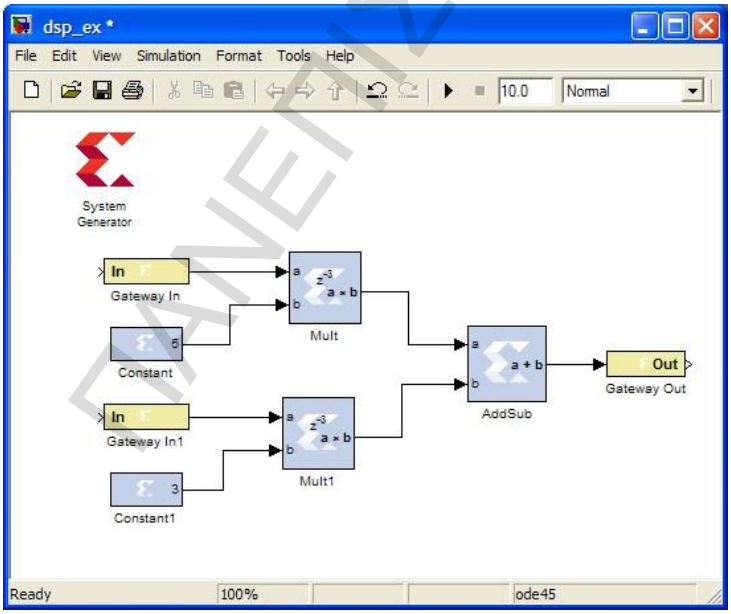


Εικόνα 19 Προσθήκη σταθερών

Κάνουμε διπλό κλικ πάνω στο μπλοκ της σταθεράς. Πρέπει να επιλέξουμε το πλήθος των bit του ακέραιου μέρους να είναι αρκετά για την αναπαράσταση της σταθεράς. Επιλέγουμε συνολικό πλήθος bit: 16 και υποδιαστολή στο 12^{o} bit. Δίνουμε τιμή στην σταθερά ίση με 5. Επαναλαμβάνουμε την ρύθμιση για την δεύτερη σταθερά με τιμή ίση με 3 (εικόνα 19).

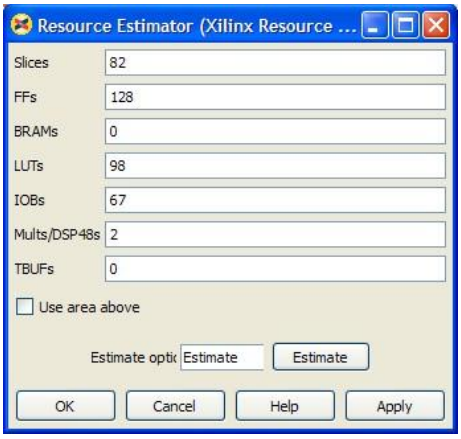
Τώρα πρέπει να αθροίσουμε τα γινόμενα των δύο πολλαπλασιαστών. Χρησιμοποιούμε γι' αυτό το σκοπό ένα μπλοκ πρόσθεσης - αφαίρεσης που βρίσκεται στο Xilinx Blockset→Math→AddSub. Κάνοντας διπλό κλικ επάνω στο μπλοκ μπορούμε να επιλέξουμε εάν θέλουμε η λειτουργία του μπλοκ να είναι πρόσθεση ή αφαίρεση. Επιλέγουμε την πρόσθεση. Το πλήθος των bit εισόδου δεν χρειάζεται να καθορισθεί καθώς προσαρμόζεται αυτόματα στην μορφή των δεδομένων εισόδου. Το πλήθος των bit εξόδου μπορεί να προσαρμοστεί αυτόματα αλλά υπάρχει η δυνατότητα και για χειροκίνητο ορισμό. Το αφήνουμε στο αυτόματο.

Τέλος προσθέτουμε ένα μπλοκ Gateway Out. Αυτό μετατρέπει τα δεδομένα fixed point που χειρίζεται το FPGA σε αριθμούς floating point που δέχεται το Simulink. Το μπλοκ αυτό υπάρχει στην κατηγορία μπλοκ: Xilinx blockset→Basic Elements→Gateway out. Μπορούμε τώρα να προσθέσουμε τις γραμμές που θα συνδέσουν τα μπλοκ (εικόνα 20).



Εικόνα 20 Ολοκληρωμένη σχεδίαση

Μπορούμε να ζητήσουμε εκτίμηση για τους πόρους που θα καταλάβει η σχεδίασή μας. Κάνουμε διπλό κλικ στο μπλοκ του System Generator. Στο πλαίσιο Compilation επιλέγουμε HDL Netlist και στο πλαίσιο part επιλέγουμε το FPGA της πλακέτας μας. Για την Virtex5 της εργασίας επιλέγουμε: Virtex5→xc5vlx110t→-1→ff1136. Προσθέτουμε στην σχεδίασή μας το μπλοκ resource estimator από την βιβλιοθήκη Xilinx blocksets→tools→resource estimator. Κάνουμε διπλό κλικ στο μπλοκ resource estimator και κατόπιν κάνουμε κλικ στο κουμπί estimate.



Εικόνα 21 Εκτίμηση απαιτούμενων πόρων

Μετά από μερικά δευτερόλεπτα υπολογίζονται οι τιμές που φαίνονται στην εικόνα 21. Ο εκτιμητής πόρων μας πληροφορεί πως για την υλοποίηση του μοντέλου μας θα χρειαστούν δύο DSP48, 128 flip flop, 98 Look Up Tables, 82 slices (τεμάχια) της virtex5 και 67 IOB's

4. Field Programmable Gate Array (FPGA)

FPGA είναι τα αρχικά των λέξεων Field Programmable Gate Array που αποδίδεται στα ελληνικά ως «επιτόπου προγραμματιζόμενοι πίνακες πυλών». Πρόκειται για ένα ολοκληρωμένο κύκλωμα το οποίο διαμορφώνεται από τον σχεδιαστή της εφαρμογής αφότου αυτό έχει κατασκευαστεί, σε αντίθεση με τα ολοκληρωμένα κυκλώματα ASIC (Application Specific Integrated Circuit) τα οποία διαμορφώνονται στο εργοστάσιο κατασκευής και καμία περεταίρω αλλαγή δεν είναι δυνατή.

Η σχεδίαση γίνεται συνήθως χρησιμοποιώντας κάποια γλώσσα περιγραφής υλικού όπως η VHDL ή η Verilog. Υπάρχει δυνατότητα η σχεδίαση να γίνει με σχηματικό διάγραμμα αλλά αυτό γίνεται σπάνια. Τα σύγχρονα FPGA ενσωματώνουν ένα μεγάλο αριθμό από πύλες λογικής και μνήμες RAM για να υλοποιήσουν πολύπλοκα κυκλώματα.

Μερικά από τα πλεονεκτήματα που προσφέρουν τα FPGA έναντι των ASIC είναι:

- δυνατότητα να αναβαθμισθεί η διαμόρφωσή τους αφότου φύγουν από το εργοστάσιο κατασκευής,
- μερική επαναδιαμόρφωση (partial re-configuration) ώστε να προσαρμοσθεί η λειτουργία τους στις τρέχουσες ανάγκες,
- μικρότερο κόστος σχεδίασης.

Στα πρώτα χρόνια της εμφάνισής τους τα FPGA κατανάλωναν περισσότερη ισχύ και απαιτούσαν σαράντα φορές μεγαλύτερη επιφάνεια από ότι τα ASIC. Με τον καιρό, καθώς η τεχνολογία των FPGA εξελίσσεται τα μειονεκτήματα αυτά τείνουν να εξαλειφθούν.

Τα FPGA περιέχουν προγραμματιζόμενες μονάδες που ονομάζονται logic block (μπλοκ λογικής) και μια ιεραρχία από αναπρογραμματιζόμενες διασυνδέσεις που επιτρέπουν στα μπλοκ να συνδεθούν μεταξύ τους με διάφορους τρόπους. Τα μπλοκ μπορούν να προγραμματισθούν ώστε εκτελούν πολύπλοκες συνδυαστικές λειτουργίες ή πιο απλές λειτουργίες όπως πύλες AND OR και XOR. Τα FPGA περιέχουν κάποιες μονάδες μνήμης οι οποίες μπορεί να είναι απλά FLIP FLOP ή μνήμες RAM.

Το πρώτο FPGA εμπορικού σκοπού με τον κωδικό XC2064 κατασκευάστηκε από την XILINX το 1985. Το chip αυτό περιελάμβανε 64 μπλοκ λογικής με πίνακες LUT (Look Up Table) των τριών εισόδων. Η XILINX συνέχισε να αναπτύσσεται στον χώρο των FPGA χωρίς ανταγωνιστή μέχρι τα μέσα της δεκαετίας του 90 οπότε και άλλες εταιρίες όπως η ACTEL και η ALTERA εμφανίσθηκαν και πήραν μερίδιο στην αγορά. Κατά την διάρκεια της δεκαετίας αυτής αυξήθηκαν τόσο η πολυπλοκότητα των FPGA όσο και το πλήθος των εφαρμογών όπου χρησιμοποιούνταν. Αρχικά χρησιμοποιήθηκαν στις τηλεπικοινωνίες και στα δίκτυα αλλά γρήγορα εξαπλώθηκαν και στα αυτοκίνητα, στην βιομηχανία και σε άλλες καταναλωτικές συσκευές.

Τα σύγχρονα FPGA μπορούν να περιέχουν επεξεργαστές καθώς και τα περιφερειακά τους. Αυτά μπορούν να υλοποιηθούν χρησιμοποιώντας τα υπάρχοντα μπλοκ (soft core processor – εύπλαστοι επεξεργαστές) ή να ενσωματωθούν κατά την κατασκευή του chip. Οι επεξεργαστές microblaze, Nios-II και ARM είναι δημοφιλείς εύπλαστοι επεξεργαστές. Στο chip Zynq 7000 της XILINX μπορεί να ενσωματωθεί ένας 32 bit, 1 GHZ επεξεργαστής ARM Cortex A9. Οι σειρές Virtex-II και Virtex4 ενσωματώνουν έναν ή περισσότερους επεξεργαστές PowerPC.

4.1. Λειτουργική σχεδίαση

Η διαδικασία της σχεδίασης ξεκινά διαιρώντας το σύστημα σε μικρότερα υποσυστήματα. Αυτά μπορεί να διαιρεθούν ξανά σε μικρότερα υποσυστήματα μέχρις ότου η πολυπλοκότητά τους να είναι σε διαχειρίσιμο επίπεδο. Προτού ξεκινήσουμε την διαίρεση σε υποσυστήματα μπορούμε να δημιουργήσουμε ένα μοντέλο συμπεριφοράς (behavioral model) του συστήματος το οποίο θα βρίσκεται σε ένα ενδιάμεσο επίπεδο αφαίρεσης. Το μοντέλο συμπεριφοράς μπορεί να περιλαμβάνει την περιγραφή του αλγόριθμου που θα υλοποιηθεί χωρίς τις ακριβείς λεπτομέρειες του χρονισμού. Ο σκοπός του μοντέλου συμπεριφοράς είναι να επαληθεύσει την σωστή λειτουργία των υποσυστημάτων προτού προχωρήσουμε στην λεπτομερή υλοποίηση της σχεδίασης.

Υπάρχουν διάφορες προσεγγίσεις προκειμένου να υλοποιηθεί ένα υποσύστημα. Ένας τρόπος είναι να σχεδιάσουμε κάθε εξάρτημα από την αρχή. Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε εξαρτήματα από προηγούμενες σχεδιάσεις ή από βιβλιοθήκες. Υπάρχουν επίσης έτοιμα εξαρτήματα που παρέχονται από εταιρίες, τα οποία είναι γνωστά σαν IP – Intellectual Properties - πνευματικές ιδιοκτησίες. Αυτά έχουν σχεδιασθεί απ' αρχής για να χρησιμοποιηθούν σε διάφορες εφαρμογές και έχουν ελεγχθεί λεπτομερώς για την καλή λειτουργία τους. Έτσι μπορεί να εξοικονομηθεί πολύς χρόνος από την σχεδίαση και τις δοκιμές. Ακόμα και αν το IP δεν έχει ακριβώς τις συνδέσεις που χρειαζόμαστε μπορούμε να το ενσωματώσουμε σε ένα wrapper (περιτύλιγμα) που θα ρυθμίζει τις διαφορές. Αν τελικά αποφασίσουμε να σχεδιάσουμε απ' αρχής ένα εξάρτημα, θα πρέπει να το κάνουμε με τέτοιο τρόπο ώστε να είναι επαναχρησιμοποιήσιμο. Ο επιπλέον χρόνος για δοκιμές κάτω από όλες τις συνθήκες και για τη λεπτομερή τεκμηρίωσή του θα μας επιστραφεί όταν θα χρειαστεί να επαναχρησιμοποιήσουμε το εξάρτημα που σχεδιάσαμε.

Άλλος τρόπος σχεδίασης εξαρτημάτων είναι η γεννήτρια πυρήνων (core generator). Η γεννήτρια πυρήνων παράγει εξαρτήματα για όλων των ειδών τις λειτουργίες όπως μνήμες, αριθμητικές μονάδες, ψηφιακή επεξεργασία σήματος και μηχανές καταστάσεων.

4.2. Σύνθεση

Μετά την σχεδίαση του μοντέλου ακολουθεί η σύνθεση σε επίπεδο RTL (Register Transfer Level). Όπως υποδεικνύει το όνομα, πρόκειται για αντιστοίχηση των διαφόρων τμημάτων του μοντέλου σε τμήματα που μπορούν να υλοποιηθούν στο hardware. Δεν είναι όλα τα χαρακτηριστικά των γλωσσών περιγραφής υλικού συνθέσιμα. Πολλά χαρακτηριστικά είναι κατάλληλα μόνο για λειτουργική περιγραφή του μοντέλου σε υψηλό επίπεδο και για το γράψιμο μοντέλων δοκιμών και δεν μπορούν να συντεθούν. Επίσης τα πιο πολλά εργαλεία σύνθεσης μπορούν να συνθέσουν δομές μόνο στην βασική τους μορφή. Ο κώδικας πρέπει να ακολουθεί συγκεκριμένη τυποποιημένη μορφή προκειμένου να περιγράψει συγκεκριμένο υλικό. Π.χ. υπάρχει συγκεκριμένος κώδικας για να περιγραφεί ένας καταχωρητής που ενεργοποιείται στην ανοδική πλευρά του παλμού του ρολογιού. Οι λόγοι για τους περιορισμούς αυτούς είναι ιστορικοί και εμπορικοί. Τα πρώτα εργαλεία σύνθεσης αναγνώριζαν σχετικά απλές δομές του υλικού στον κώδικα HDL. Οι επόμενες εκδόσεις εστίαζαν στην βελτίωση του υλικού που παρήγαγαν τα εργαλεία σύνθεσης, παρά στο να επεκτείνουν τις μορφές του αποδεκτού κώδικα. Άλλωστε δεν υπήρχε μεγάλη απαίτηση να μειωθούν οι περιορισμοί αφού η δουλειά μπορούσε να γίνει με τους υπάρχοντες περιορισμούς.

Καθώς διαφορετικά εργαλεία σύνθεσης δέχονταν διαφορετικά σύνολα εντολών δημιουργήθηκε η ανάγκη για την καθιέρωση ενός συνόλου εντολών που θα γίνεται καθολικά αποδεκτό. Έτσι το IEEE καθόρισε δύο στάνταρ σύνολα εντολών, ένα για την VHDL και ένα για την VERILOG τα οποία γίνονται δεκτά από όλα τα εργαλεία σύνθεσης.

Ένα εργαλείο σύνθεσης αρχικά αναλύει το μοντέλο για να ελέγξει ότι ο κώδικας ακολουθεί τις επιταγές του. Επίσης κάνει ελέγχους σχεδίασης όπως έλεγχο για ασύνδετες εισόδους ή εξόδους, ή πολλαπλές εξόδους συνδεμένες μαζί. Κατόπιν το εργαλείο σύνθεσης αντιστοιχεί κυκλώματα (hardware constructs). Γίνονται ενέργειες όπως:

- Ανάλυση των καθορισμών των σημάτων για να αποφασισθεί η κωδικοποίηση και ο αριθμός των bit που απαιτούνται για την αναπαράσταση των δεδομένων.
- Ανάλυση των εκφράσεων και των ορισμών για να αναγνωρισθούν συνδυαστικά κυκλώματα όπως αθροιστές και πολλαπλασιαστές και να αναγνωρισθούν οι είσοδοι, οι έξοδοι και τα ενδιάμεσα σήματα.
- Ανάλυση των εντολών διαδικασιών (process) για να αναγνωρισθούν το ρολόι και τα άλλα σήματα ρολογιού και να επιλεχθούν τα κατάλληλα είδη καταχωρητών και flip flop που θα χρησιμοποιηθούν.

Για κάθε στοιχείο του υλικού που έχει εντοπισθεί, αντιστοιχίζεται ένα κύκλωμα από μια συλλογή από έτοιμα κυκλώματα που ονομάζεται βιβλιοθήκη τεχνολογίας. Η βιβλιοθήκη περιέχει κυκλώματα όπως πύλες, πολυπλέκτες, κυκλώματα υπολογισμού του bit κρατούμενου και flip flop.

Η διαδικασία της δημιουργίας του κυκλώματος από την γλώσσα περιγραφής υλικού ακολουθεί τις προσταγές των περιορισμών που έχουμε θέσει. Οι περιορισμοί αφορούν κυρίως τον αριθμό των κύκλων ρολογιού για να ολοκληρωθεί μία λειτουργία ή την καθυστέρηση διάδοσης ενός σήματος. Αν σε κάποιο κύκλωμα δεν τηρούνται οι περιορισμοί τότε δοκιμάζονται εναλλακτικές υλοποιήσεις της ίδιας λειτουργίας.

Αν και στηριζόμαστε στο εργαλείο σύνθεσης για την δημιουργία του κυκλώματος, μερικές φορές χρειάζεται να ενσωματώσουμε προσχεδιασμένα κυκλώματα. Τέτοια περίπτωση είναι τα εξαρτήματα που δημιουργεί η γεννήτρια πυρήνων (core generator). Ο τρόπος που εισάγονται αυτά τα κυκλώματα διαφέρει από εργαλείο σε εργαλείο.

Αφού ολοκληρωθεί η διαδικασία της σύνθεσης ακολουθεί η προσομοίωση της σχεδίασης. Αυτή περιλαμβάνει δύο στάδια: την χρονική ανάλυση (timing analysis) και την λειτουργική προσομοίωση. Κατά την χρονική ανάλυση ελέγχεται ότι τηρούνται οι χρονικοί περιορισμοί της σχεδίασής. Στην λειτουργική προσομοίωση ελέγχεται η σωστή λειτουργία του συστήματος. Και τα δύο αυτά στάδια μπορούν να γίνουν σε δύο επίπεδα λεπτομέρειας: Αρχικά σε επίπεδο RTL (Register Transfer Level) και κατόπιν σε επίπεδο πύλης που είναι πιο λεπτομερής και ακριβής. Η χρονική ανάλυση σε επίπεδο πύλης λαμβάνει υπ' όψιν τις εκτιμήσεις χρόνου των εξαρτημάτων που διαθέτουν οι βιβλιοθήκες τεχνολογίας (technology library). Έτσι μπορεί να αναδείξει προβλήματα που δεν είναι εμφανή με την προσομοίωση σε επίπεδο RTL.

Επίσης προβλήματα συμπεριφοράς μπορεί να ανιχνευθούν με την προσομοίωση σε επίπεδο πύλης. Είναι δυνατό να γραφεί κώδικας μοντέλου RTL που παράγει διαφορετικά αποτελέσματα στην προσομοίωση RTL και διαφορετικά στο hardware μετά την σύνθεση.

4.3. Φυσική σχεδίαση

Τα βασικά βήματα στην σχεδίαση του FPGA δεν διαφέρουν από αυτά της σχεδίασης των ASIC. Παρόλα αυτά τα εργαλεία που χρησιμοποιούνται εφαρμόζουν διαφορετικές τεχνικές. Τα βήματα αυτά είναι: χωροθέτηση (floorplanning), τοποθέτηση (placement) και δρομολόγηση (routing). Το πρώτο βήμα αφορά την θέση που θα πάρει κάθε τμήμα της σχεδίασης πάνω στο chip. Υπάρχουν παράγοντες που επηρεάζουν την χωροθέτηση:

- Τμήματα που έχουν πολλές συνδέσεις μεταξύ τους θα πρέπει να τοποθετηθούν κοντά το ένα στο άλλο ώστε να μειωθεί το μήκος της καλωδίωσης και αποφευχθεί συμφόρηση.
- Τμήματα που συνδέονται με εξωτερικούς ακροδέκτες θα πρέπει να τοποθετηθούν προς την περιφέρεια του chip.
- Σύνδεση και διανομή των σημάτων ρολογιού στα διάφορα τμήματα.
- Κανάλια διασύνδεσης των διαφόρων τμημάτων.

Μια καλή χωροθέτηση των τμημάτων σε ένα FPGA μειώνει τον αριθμό των διασυνδέσεων μεγάλου μήκους, και επιτυγχάνει γρηγορότερη υλοποίηση της δρομολόγησης.

Στις μικρές εφαρμογές για FPGA η χωροθέτηση που δημιουργείται αυτόματα από τα εργαλεία σχεδίασης είναι ικανοποιητική. Σε μεγαλύτερες σχεδιάσεις ή στις περιπτώσεις που το πρόγραμμα σχεδίασης αδυνατεί να ταιριάξει την εφαρμογή στον συγκεκριμένο FPGA, τότε μπορούμε να επιχειρήσουμε να βελτιώσουμε την χωροθέτηση χειροκίνητα ή να επιλέξουμε ένα μεγαλύτερο FPGA.

Πριν προχωρήσουμε στην τοποθέτηση και στην δρομολόγηση πρέπει να γίνει η χαρτογράφηση (mapping). Αυτή αφορά την αντιστοίχιση των πόρων του FPGA στα εξαρτήματα που θα χρησιμοποιηθούν στην σχεδίαση. Πύλες και πολυπλέκτες θα αντιστοιχισθούν σε πίνακες (Look Up Tables), Flip Flop θα αντιστοιχισθούν στα Flip Flop που διαθέτει το FPGA. Επίσης εξαρτήματα που αφορούν συγκεκριμένους πόρους του FPGA όπως μπλοκ μνήμης RAM ή μπλοκ DSP48 θα αντιστοιχηθούν σε αυτό το στάδιο.

Η τοποθέτηση και δρομολόγηση έχει στόχο να εντοπίσει συγκεκριμένα μπλοκ και να τα συνδέσει με τέτοιο τρόπο ώστε να ελαχιστοποιηθεί ο χώρος που αυτά καταλαμβάνουν και να μειωθεί η καθυστέρηση μετάδοσης των σημάτων. Και αυτό το στάδιο γίνεται αυτόματα από τα εργαλεία σχεδίασης. Αν χρειαστεί να βελτιώσουμε κάποιο σημείο, αυτό γίνεται θέτοντας περιορισμούς και όχι επεμβαίνοντας άμεσα.

Το τελικό αποτέλεσμα είναι ένα αρχείο με μια ακολουθία bit (bitfile) που καθορίζει την παραμετροποίηση του FPGA. Τώρα που είναι γνωστές οι λεπτομερείς πληροφορίες χρονισμού των μπλοκ και των διασυνδέσεων μπορούμε να κάνουμε τις τελικές δοκιμές προσομοίωσης.

Σε μια απλοποιημένη ροή σχεδίασης η φυσική σχεδίαση είναι το τελευταίο στάδιο της σχεδίασης. Στην βιομηχανία είναι πιο ρεαλιστικό η σύνθεση και η φυσική σχεδίαση να δοκιμάζονται σε εναλλακτικές υλοποιήσεις που θα αποδώσουν την βέλτιστη χρονική απόκριση και μέγιστη αξιοποίηση του υλικού.

Προκειμένου το FPGA να μπορεί να αναπρογραμματισθεί, η ακολουθία bit που καθορίζει την λειτουργία του γράφεται σε μνήμη RAM μέσα στο FPGA. Αυτό κάνει αναγκαία την ύπαρξη μίας μνήμης τύπου FLASH στην οποία θα αποθηκεύεται μόνιμα. Η μνήμη FLASH δεν είναι ενσωματωμένη στο FPGA και αυτό έχει συνέπεια η ακολουθία bit να είναι εκτεθειμένη. Αν και η αποκωδικοποίηση του (reverse engineering) θεωρείται αδύνατη, υπάρχει κίνδυνος αντιγραφής. Σήμερα πολλοί κατασκευαστές προσφέρουν δυνατότητες κρυπτογράφησης της ακολουθίας bit ώστε να αποτραπεί αυτός ο κίνδυνος.

5. Προετοιμασία για την Προσομοίωση στο Simulink με χρήση της τεχνικής FPGA-In-the-Loop

Για να γίνει προσομοίωση με την τεχνική FPGA-In-the-Loop χρειάζεται να εγκατασταθεί το κατάλληλο λογισμικό στον υπολογιστή. Η διαδικασία αυτή δεν έχει ιδιαίτερη δυσκολία και περιγράφεται στην αρχή του κεφαλαίου αυτού. Ακολουθεί μια σύντομη περιγραφή της πλακέτας FPGA που χρησιμοποιήθηκε στην εργασία. Το εργαλείο system generator έχει προ-εγκατεστημένες μερικές πλακέτες FPGA της Xilinx. Εάν η πλακέτα που διαθέτουμε δεν είναι μια από αυτές, μπορούμε να την προσθέσουμε. Η διαδικασία εγκατάστασης της πλακέτας Virtex5 που χρησιμοποιήθηκε περιγράφεται στην συνέχεια.

5.1. Εγκατάσταση λογισμικού

Το λογισμικό που απαιτείται για το system generator είναι το ISE Design Suite της Xiinx και το MatLab/Simulink της Mathworks. Η έκδοση του ISE Design Suite που χρησιμοποιήθηκε είναι η 14.2. Οι συμβατές εκδόσεις MatLab είναι οι 2011a και 2011b. Στις δοκιμές χρησιμοποιήθηκαν και οι δύο χωρίς να παρουσιαστούν διαφορές.

Οι δύο αυτές εφαρμογές εγκαταστάθηκαν σε υπολογιστή Pentium 4 στα 2.66 GHZ με 2 GB RAM και με λειτουργικό σύστημα «Windows XP Professional English με SP3». Δοκιμές στα νεώτερα «Windows 7» είχαν προβλήματα συμβατότητας. Συγκεκριμένα η διαδικασία generation χρειαζόταν πολλαπλάσιο χρόνο περαίωσης από ότι στα Windows XP. Αυτό είναι σοβαρό πρόβλημα αν λάβουμε υπ' όψη ότι ο χρόνος αυτός είναι περίπου πέντε λεπτά για ένα πολύ απλό σύστημα και μπορεί να φθάσει τα σαράντα λεπτά ή και περισσότερο για μεγάλα συστήματα. Με ελληνικά Windows XP το σύστημα λειτουργεί κανονικά στην πλειοψηφία των μοντέλων που δοκιμάστηκαν. Στο μοντέλο με τους κωδικοποιητές – αποκωδικοποιητές 3GPP εμφάνιζε σφάλμα λόγω της κωδικοποίησης των χαρακτήρων του μοντέλου. Το πρόβλημα αυτό ξεπερνιέται γράφοντας στο παράθυρο εντολών του Matlab:

bdclose all

slcharacterencoding windows-1252

Αυτό θα πρέπει να γίνεται κάθε φορά που ξεκινά το system generator .

Τέλος η ύπαρξη των 2 GB RAM είναι απαραίτητη για λόγους ταχύτητας. Με λιγότερη μνήμη το σύστημα λειτουργεί αλλά ο δίσκος δουλεύει συνεχώς κατά την διάρκεια του generation και ο απαιτούμενος χρόνος περαίωσης διπλασιάζεται.

5.1.1. Περιγραφή της πλακέτας XUPV5-LX110T

Η πλακέτα που χρησιμοποιήθηκε στην εργασία είναι η XUPV5-LX110T της σειράς Virtex5 της Xilinx. Περιέχει μπλοκ που περιλαμβάνουν 36-Kbit block RAM/FIFOs, τεμάχια DSP δεύτερης γενιάς 25 x 18 bit, δυνατότητα παρακολούθησης του συστήματος και διαχείριση του ρολογιού με PLL (phase locked loop). Διαθέτει μπλοκ για σειριακή επικοινωνία υψηλής ταχύτητας και χαμηλής κατανάλωσης. Οι δυνατότητες σύνδεσης περιλαμβάνουν υποδοχή PCI Express και Ethernet MACs (Media Access Controllers).

5.1.2. Εγκατάσταση πλακέτας

Κατ' αρχάς συνδέουμε την πλακέτα με το καλώδιο USB και την τροφοδοτούμε. Οι απαραίτητοι drivers θα εγκατασταθούν αυτόματα εφόσον έχουμε εγκαταστήσει ήδη το ISE Design Suite.

Για να μπορέσουμε να εγκαταστήσουμε την πλακέτα που θα χρησιμοποιηθεί για την προσομοίωση θα πρέπει πρώτα να ανοίξουμε ένα μοντέλο του system generator. Μπορούμε να χρησιμοποιήσουμε ένα από τα έτοιμα παραδείγματα που υπάρχουν στο :

« C:\Xilinx\14.2\ISE_DS\ISE\sysgen\examples\getting_started_training\lab7\solution».

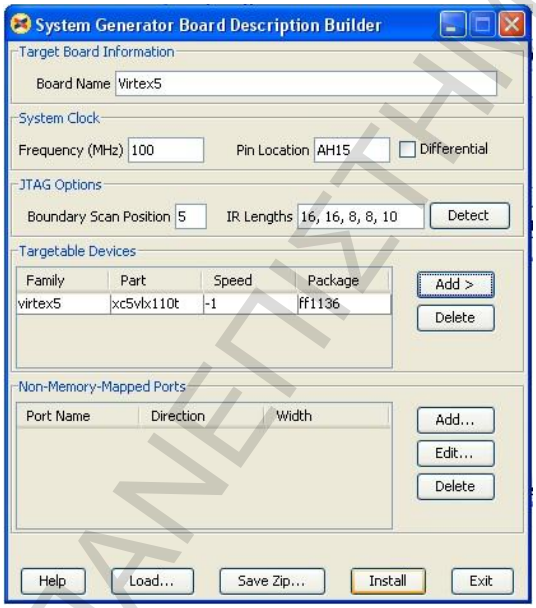
Ανοίγουμε το σχετικό μοντέλο κάνοντας διπλό κλικ πάνω στο αρχείο «lab7.mdl» που υπάρχει στο παράθυρο current folder στα αριστερά της οθόνης.

Στην συνέχεια κάνουμε διπλό κλικ στο εικονίδιο του system generator. Εμφανίζεται το παράθυρο με τις επιλογές της πλακέτας (εικόνα 22).

Ελέγχουμε ότι έχει επιλεγεί η καρτέλα «compilation», κάνουμε κλικ στο κουμπί με το βέλος στα αριστερά πλαισίου «compilation» και κατεβάζουμε το ποντίκι στην επιλογή «Hardware Co_Simulation». Ανοίγει ένα παράθυρο με τις πλακέτες που υποστηρίζονται. Επιλέγουμε την τελευταία επιλογή «New Compilation Target…»

Εικόνα 22 Προσθήκη νέας πλακέτας

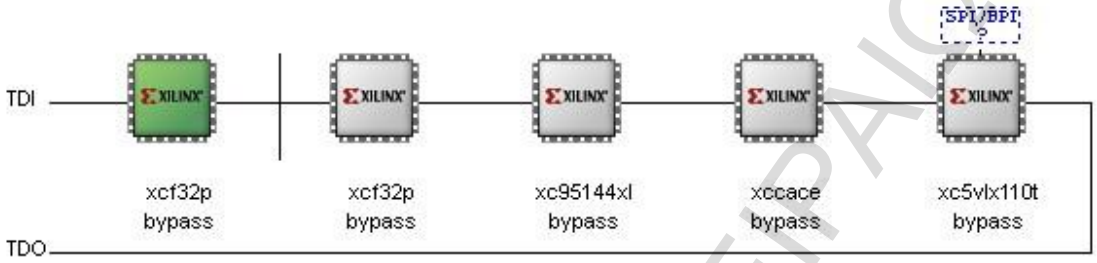
Στο παράθυρο που ανοίγει συμπληρώνουμε για την πλακέτα Virtex-5 τις ρυθμίσεις που αναφέρονται παρακάτω.



Εικόνα 23 Ρυθμίσεις για την Virtex5

Board Name	:Virtex5 (ή όπως αλλιώς θέλουμε)
Frequency (MHZ)	:100
Pin Location	:AH15
Differential	:κενό
Boundary Scan Position	:5
IR Lengths	:16, 16, 8, 8, 10

Για άλλες πλακέτες, για την συχνότητα του ρολογιού και την θέση του αντίστοιχου pin συμβουλευόμαστε το manual της πλακέτας. Την τιμή «Boundary Scan Position» μπορούμε να την βρούμε χρησιμοποιώντας το εργαλείο Impact που βρίσκεται στα design tools στο μενού έναρξης. Μόλις το ανοίξουμε πατάμε cancel στο παράθυρο που μας ζητά να ξεκινήσουμε νέο project και κάνουμε διπλό κλικ στην επιλογή «Boundary Scan». Μετά κάνουμε δεξί κλικ στο παράθυρο στα δεξιά και επιλέγουμε «initialize chain». Εμφανίζεται η εικόνα της δομής της πλακέτας όπως η εικόνα 24 όπου βλέπουμε ότι το fpga είναι πέμπτο στην αλυσίδα. Άρα βάζουμε «5» στην επιλογή «Boundary Scan Position». Το Impact δεν χρειάζεται άλλο και μπορούμε να το κλείσουμε. Κατόπιν μπορούμε να πατήσουμε το κουμπί Detect για να συμπληρωθεί αυτόματα το πλαίσιο «IR Lengths»



Εικόνα 24 Δομή πλακέτας Virtex5

Στην συνέχεια πατάμε το κουμπάκι «Add» και επιλέγουμε τον τύπο FPGA που περιέχει η πλακέτα μας. Για την Virtex5 της εργασίας η επιλογή είναι virtex5→xc5vlx110t→-1→ff1136. Κατόπιν πατάμε το κουμπάκι «Save Zip…». Η εγκατάσταση έχει ολοκληρωθεί. Κλείνουμε το παράθυρο του System Generator και κατόπιν το ξανανοίγουμε. Η πλακέτα μας θα υπάρχει πλέον ανάμεσα στις διαθέσιμες για Hardware Co-Simulation. Αυτή η διαδικασία χρειάζεται να γίνει άπαξ στον κάθε υπολογιστή. Είμαστε πλέον έτοιμοι να χρησιμοποιήσουμε την πλακέτα για hardware co-simulation.

Όλες οι πλακέτες FPGA της Xilinx μπορούν να χρησιμοποιηθούν για προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop μέσω της σύνδεσης JTAG. Η επικοινωνία μεταξύ υπολογιστή και FPGA με τον τρόπο αυτό είναι αργή. Ορισμένες πλακέτες έχουν σχεδιασθεί με σκοπό να χρησιμοποιηθούν για προσομοίωση με συνεργασία υλικού και υποστηρίζουν επικοινωνία μέσω Ethernet. Αυτή η δυνατότητα είναι σημαντική όταν μεγάλος όγκος δεδομένων πρέπει να ανταλλαγεί μεταξύ υπολογιστή και FPGA κατά την διάρκεια προσομοίωσης.

6. Hardware Co-Simulation

Στο κεφάλαιο αυτό παρουσιάζεται βήμα-βήμα η δημιουργία ενός μοντέλου στο οποίο θα γίνει προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop. Αρχικά δημιουργείται ένα μοντέλο με μπλοκ του Simulink. Στην συνέχεια δημιουργείται το αντίστοιχο μοντέλο με μπλοκ της Xilinx. Παρόλο που υπάρχουν μπλοκ της Xilinx με την ίδια λειτουργία με τα αντίστοιχα μπλοκ του Simulink, η κατασκευή του δεύτερου μοντέλου έχει μεγαλύτερο βαθμό δυσκολίας από την κατασκευή του πρώτου μοντέλου. Ο λόγος είναι ότι στο μοντέλο Simulink όλα τα σήματα αναπαριστώνται και επεξεργάζονται με την μορφή floating point. Στο μοντέλο Xilinx τα σήματα αναπαριστώνται ως fixed point. Είναι αρμοδιότητα του σχεδιαστή να μεριμνήσει για την σωστή επιλογή του πλήθους των bit που θα χρησιμοποιηθούν για το ακέραιο και για το δεκαδικό μέρος των αριθμών.

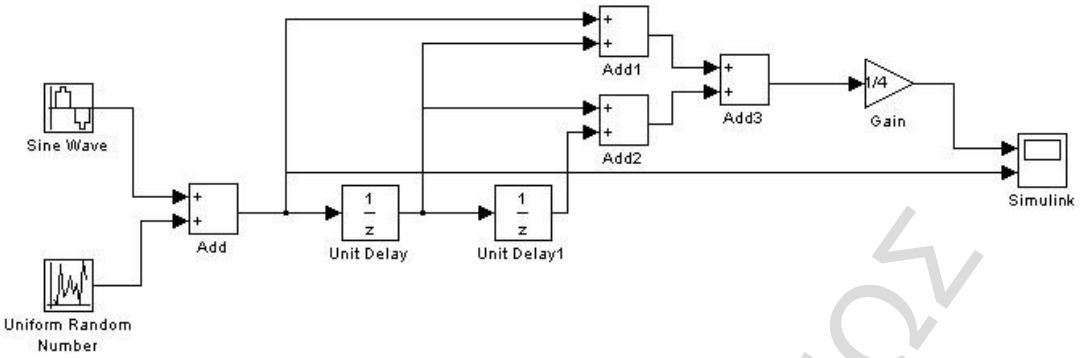
Στο δεύτερο μέρος του κεφαλαίου γίνεται προσομοίωση ενός φίλτρου FIR. Γίνεται παρουσίαση του εργαλείου σχεδίασης φίλτρων «FDA Tool» και της διαδικασίας που χρειάζεται ώστε εργαλείο αυτό να χρησιμοποιηθεί στο περιβάλλον της Xilinx.

6.1. Μεθοδολογία προσομοίωσης

Η μεθοδολογία της προσομοίωσης με χρήση της τεχνικής FPGA-In-the-Loop θα παρουσιασθεί με ένα απλό παράδειγμα. Θα φτιάξουμε αρχικά ένα μοντέλο με μπλοκ του Simulink και κατόπιν θα φτιάξουμε ένα μοντέλο που θα έχει την ίδια λειτουργία αλλά θα είναι κατασκευασμένο με μπλοκ της Xilinx. Αφού ελέγξουμε με προσομοίωση ότι η λειτουργία τους είναι ταυτόσημη θα υλοποιήσουμε το μοντέλο στο fpga. Κατόπιν θα γίνει η προσομοίωση χρησιμοποιώντας την τεχνική FPGA-In-the-Loop. Το Simulink θα δημιουργεί τα σήματα εισόδου και θα τα στέλνει στο FPGA το οποίο θα εκτελεί τους υπολογισμούς. Κατόπιν τα δεδομένα θα στέλνονται πίσω στον υπολογιστή όπου το Simulink θα καταγράφει τα σήματα εξόδου. Στο τέλος θα μετρήσουμε πόσο χρόνο χρειάζεται η προσομοίωση του μοντέλου με μπλοκ του Simulink και πόσο χρόνο χρειάζεται η προσομοίωση του μοντέλου με μπλοκ της Xilinx και με χρήση της τεχνικής FPGA-In-the-Loop.

Στο παράδειγμα θα προσομοιώσουμε ένα απλό φίλτρο. Η λειτουργία του βασίζεται στην αντικατάσταση της τιμής του σήματος d_k με τον μέσο όρο των διαδοχικών τιμών d_k, 2Xd_{k-1}, d_{k-2}. Στην είσοδο του φίλτρου το σήμα περνάει από δύο μονάδες καθυστέρησης (unit delay). Τα μπλοκ αυτά εμφανίζουν στην έξοδό τους το σήμα εισόδου με καθυστέρηση κατά έναν κύκλο ρολογιού. Με αυτόν τον τρόπο έχουμε συνεχώς διαθέσιμες τρεις διαδοχικές τιμές του σήματος, την τρέχουσα τιμή d_k και τις δύο προηγούμενες d_{k-1}και d_{k-2} των οποίων υπολογίζουμε το άθροισμα. Η τιμή dk_{-1} συμμετέχει διπλά ώστε η τιμή της να έχει μεγαλύτερη βαρύτητα στην έξοδο. Το άθροισμα κατόπιν διαιρείται με το 4 για να προκύψει ο μέσος όρος των τιμών ο οποίος και θα αποτελεί την νέα τιμή του σήματος.

Ξεκινάμε το System generator και ορίζουμε ως τρέχοντα φάκελο (current folder) τον «C:\Xilinx\14.2\ISE_DS\ISE\sysgen\». Δημιουργούμε ένα φάκελο με όνομα «simple filter». Κάνουμε διπλό κλικ πάνω στον φάκελο αυτό και προστίθεται στη διαδρομή. Από το Simulink Library Browser δημιουργούμε ένα νέο κενό μοντέλο. Χρησιμοποιώντας τα μπλοκ του Simulink δημιουργούμε το μοντέλο που φαίνεται στην εικόνα 25 .



Εικόνα 25 Μοντέλο Simulink

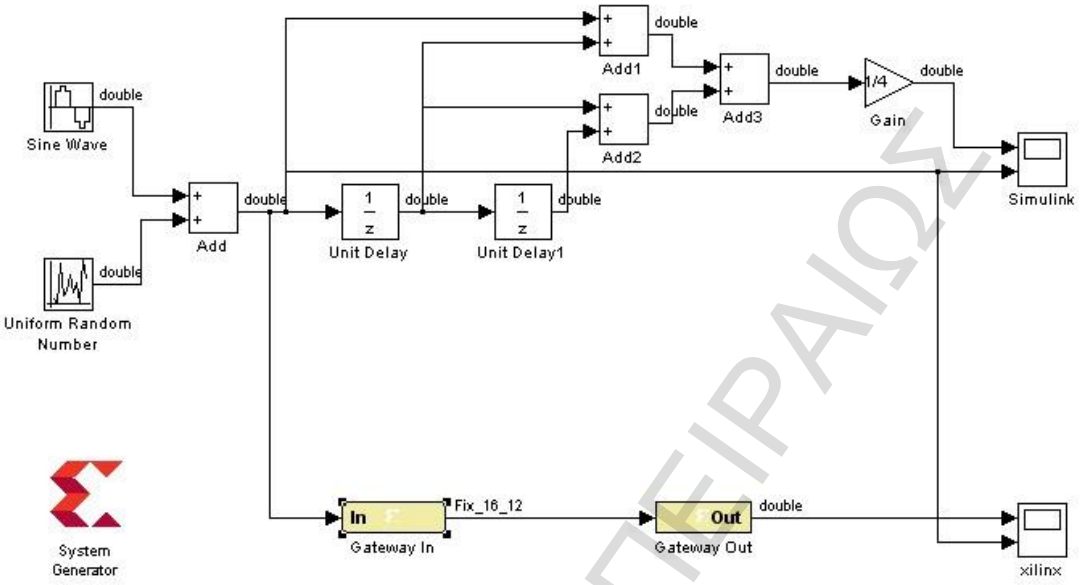
Κατ αρχάς δημιουργούμε ένα ημιτονοειδές σήμα στον οποίο προσθέτουμε θόρυβο. Χρειάζεται να ρυθμίσουμε το πλάτος του ημιτονοειδούς σήματος και του θορύβου. Κάνουμε διπλό κλικ στην γεννήτρια ημιτονοειδούς σήματος και ορίζουμε Amplitude : 4 και Sample time : 0.01. Κάνουμε διπλό κλικ στην γεννήτρια θορύβου και ορίζουμε Minimum : -1/2, Maximum : 1/2 και Sample time : 0.01. Μετονομάζουμε τον παλμογράφο σε Simulink

Κάνουμε διπλό κλικ πάνω στο εικονίδιο του παλμογράφου (scope) για να εμφανισθεί ο παλμογράφος. Πατάμε το κουμπάκι της έναρξης προσομοίωσης και βλέπουμε στον παλμογράφο την κυματομορφή της εικόνας 27. Στο κάτω παράθυρο είναι η είσοδος και στο πάνω η έξοδος. Βλέπουμε πως ο θόρυβος στο σήμα έχει περιορισθεί.

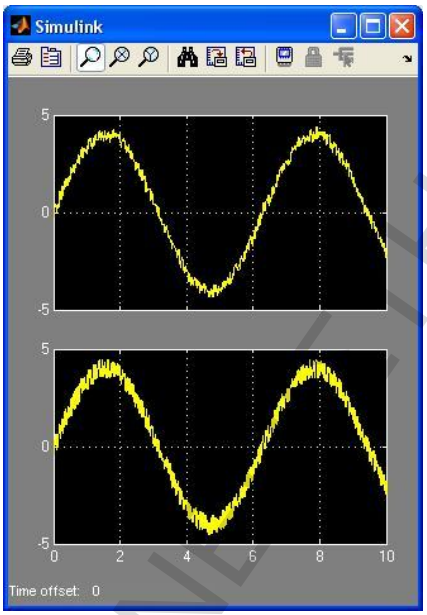
Τώρα θα δημιουργήσουμε το αντίστοιχο μοντέλο με μπλοκ της Xilinx. Τα μπλοκ του Simulink αναπαριστούν τους αριθμούς με μορφή κινητής υποδιαστολής. Αυτή η αναπαράσταση είναι δύσκολο να υλοποιηθεί με hardware και πολλές φορές δεν είναι και απαραίτητο. Γι' αυτό τα μπλοκ της Xilinx αναπαριστούν τους αριθμούς με μορφή σταθερής υποδιαστολής. Αφήνεται στον σχεδιαστή του συστήματος να αποφασίσει για το πλήθος των bit που θα χρησιμοποιηθούν για το ακέραιο και για το δεκαδικό μέρος του αριθμού. Με περισσότερα bit από τα απαιτούμενα έχουμε κόστος σε υλικό και σε χρόνο εκτέλεσης. Με λιγότερα bit έχουμε υπερχείλιση και λάθος αποτελέσματα.

Ξεκινάμε την δημιουργία του μοντέλου προσθέτοντας ένα μπλοκ gateway in, ένα μπλοκ gateway out και έναν παλμογράφο όπως στην εικόνα 26. Μετονομάζουμε τον παλμογράφο που προσθέσαμε σε «Xilinx». Προσθέτουμε ένα εικονίδιο System Generator και κάνουμε διπλό κλικ σ' αυτό. Ορίζουμε στην καρτέλα Compilation, στο πλαίσιο Part, το fpga της πλακέτας μας. Στην περίπτωση της εργασίας αυτής το εξάρτημα είναι το Virtex5 xc5vlx110t-1ff1136.

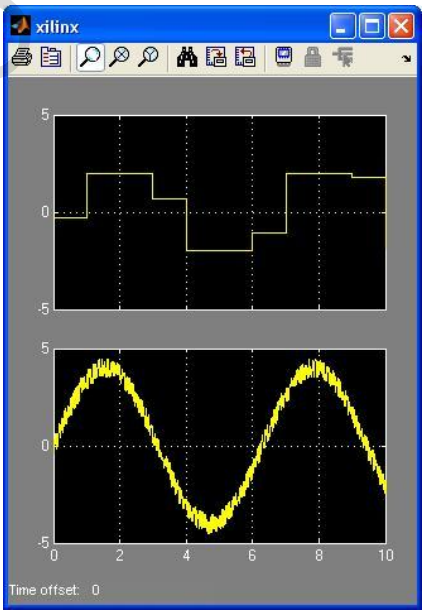
Κάνουμε διπλό κλικ πάνω στο εικονίδιο του παλμογράφου (scope) για να εμφανισθεί ο παλμογράφος. Πατάμε το κουμπάκι της έναρξης προσομοίωσης και βλέπουμε στον παλμογράφο την κυματομορφή της εικόνας 28.



Εικόνα 26 Προσθήκη μπλοκ της Xilinx στο μοντέλο

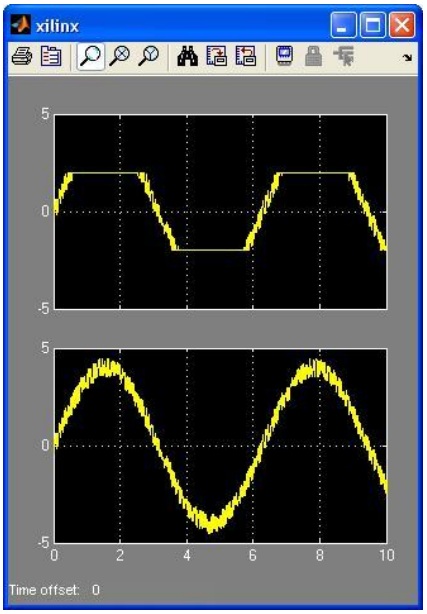


Εικόνα 27 Προσομοίωση Simulink

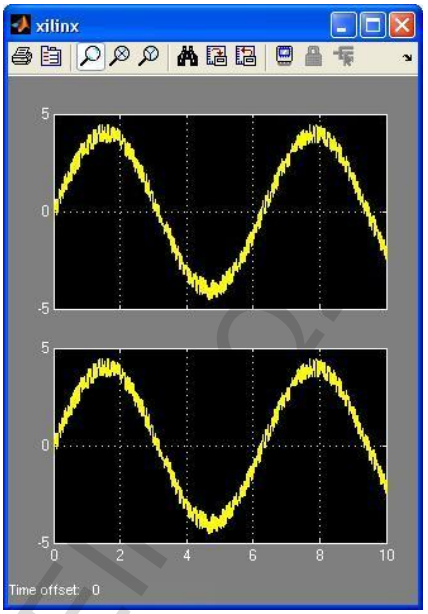


Εικόνα 28 Προσομοίωση Xilinx

Η πάνω εικόνα που δείχνει την έξοδο δεν μοιάζει καθόλου με αυτήν που δίνουμε στην είσοδο. Η κύρια αιτία είναι ο χαμηλός ρυθμός δειγματοληψίας. Για να το διορθώσουμε αυτό κάνουμε διπλό κλικ στο εικονίδιο του System Generator, επιλέγουμε την καρτέλα Clocking και ορίζουμε Simulink System Period : 0.01. Κάνουμε διπλό κλικ στο μπλοκ Gateway In και ορίζουμε Sample Period : 0.01. Δοκιμάζουμε ξανά την προσομοίωση πατώντας το κουμπάκι έναρξης προσομοίωσης. Βλέπουμε στον παλμογράφο την κυματομορφή της εικόνας 29. Παρατηρούμε πως το σήμα εξόδου έχει ψαλιδισθεί.



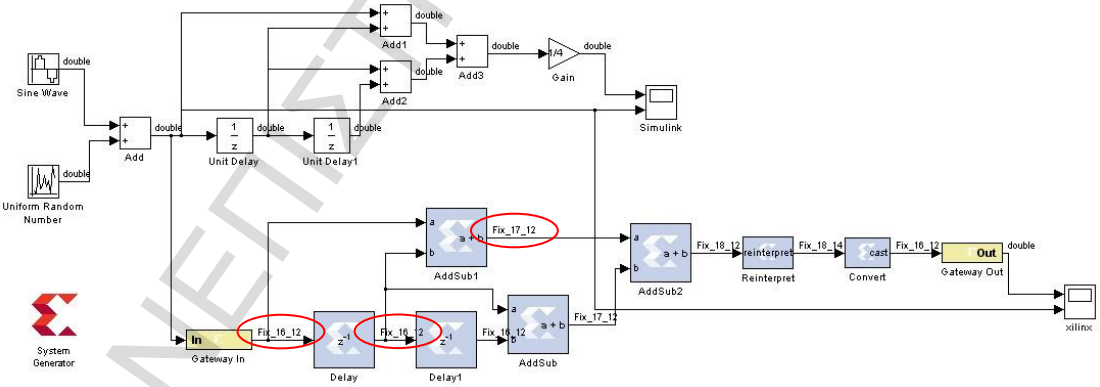
Εικόνα 29 Ψαλιδισμός του σήματος



Εικόνα 30 Σωστή επιλογή των bit αναπαράστασης της τιμής εισόδου

Το πρόβλημα βρίσκεται στο πλήθος των bit που χρησιμοποιούνται για την αναπαράσταση της τιμής εισόδου. Τα μπλοκ gateway in είναι αρχικά ρυθμισμένα να χρησιμοποιούν 16 bit με 14 bit για το δεκαδικό μέρος. Αυτή η ρύθμιση αφήνει μόνο 2 bit για το ακέραιο μέρος και καθώς αυτά δεν επαρκούν για την αναπαράσταση του σήματος, αυτό ψαλιδίζεται. Για να το διορθώσουμε αυτό κάνουμε διπλό κλικ στο μπλοκ gateway in και στο πλαίσιο Fixed-point Precision ορίζουμε Number of bits : 16 και Binary point 12. Τρέχουμε την προσομοίωση ξανά και βλέπουμε την κυματομορφή της εικόνας 30 στον παλμογράφο. Είμαστε έτοιμοι πια να δημιουργήσουμε το μοντέλο μας.

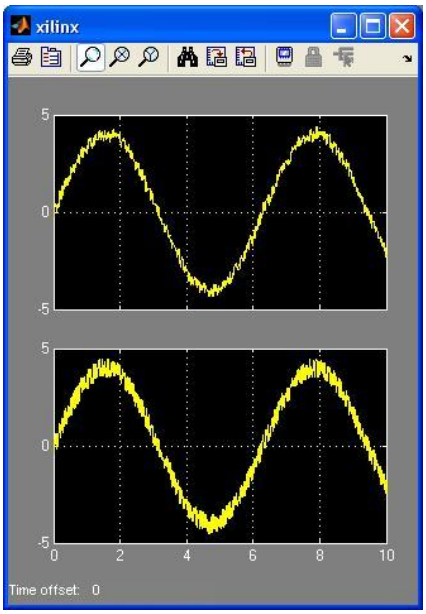
Χρησιμοποιώντας αποκλειστικά μπλοκ της Xilinx φτιάχνουμε το νέο μοντέλο παράλληλα στο προηγούμενο. Καταλήγουμε στο μοντέλο της εικόνας 31.



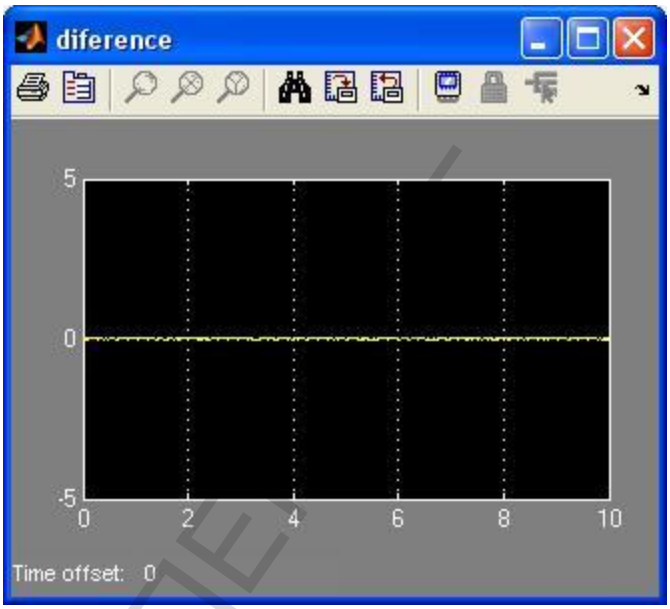
Εικόνα 31 Μοντέλο Simulink και το αντίστοιχο μοντέλο Xilinx

Στο μοντέλο Simulink όλοι οι αριθμοί αναπαριστώνται σαν διπλής ακριβείας δεκαδικοί κινητής υποδιαστολής. Στο μοντέλο Xilinx επιλέξαμε να αναπαριστώνται σαν fixed point με 16 bit συνολικά από τα οποία τα 12 bit χρησιμοποιούνται για το δεκαδικό μέρος του αριθμού. Θα πρέπει να φροντίζουμε εμείς ώστε να διατηρείται αυτή η αναπαράσταση στο μοντέλο μας. Όπως έχει επισημανθεί στην εικόνα 31, ένας αθροιστής που δέχεται δύο αριθμούς των 16 bit μας δίνει άθροισμα αριθμό των 17 bit. Κατόπιν ακολουθεί και άλλος αθροιστής και το νέο άθροισμα αποτελείται από 18 bit. Για να επαναφέρουμε την αρχική αναπαράσταση χρησιμοποιούμε πρώτα ένα μπλοκ «reinterpret» που κάνει μετατόπιση των bit δύο θέσεις δεξιά, κάνοντας ουσιαστικά διαίρεση με το τέσσερα. Έτσι γίνεται η διαίρεση που απαιτεί ο

υπολογισμός του μέσου όρου. Κατόπιν τοποθετούμε ένα μπλοκ «Convert» που αφαιρεί τα δύο lsb bit από τον αριθμό ώστε να αναπαρίσταται πάλι με 16 bit.

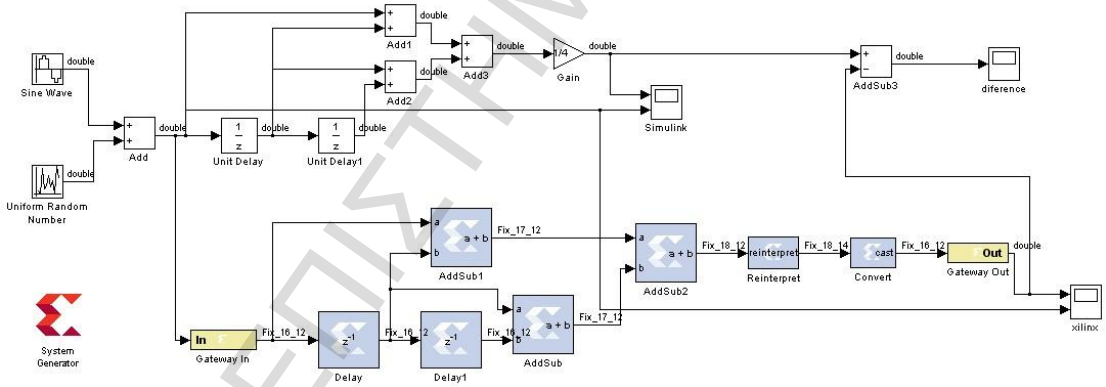


Εικόνα 32 Μοντέλο Xilinx



Εικόνα 33 Διαφορά των δύο μοντέλων

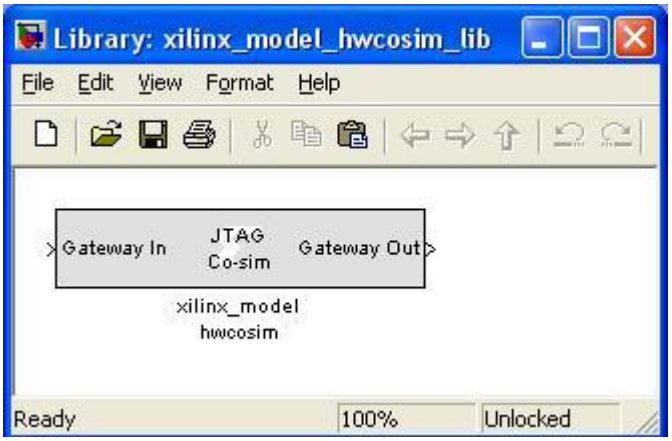
Τρέχοντας ξανά την προσομοίωση παίρνουμε την έξοδο της εικόνας 32. Αυτή μοιάζει πολύ με την έξοδο του μοντέλου Simulink της εικόνας 27. Για να είμαστε σίγουροι ότι τα δύο μοντέλα είναι απαράλλακτα στην λειτουργία τους προσθέτουμε στο μοντέλο μας ένα μπλοκ αφαίρεσης και ένα ακόμα μπλοκ παλμογράφου όπως φαίνεται στο μοντέλο της εικόνας 34. Μετονομάζουμε τον νέο παλμογράφο σε «Difference» μιας και θα μας δείξει την διαφορά των δύο μοντέλων.



Εικόνα 34 Πλήρες μοντέλο

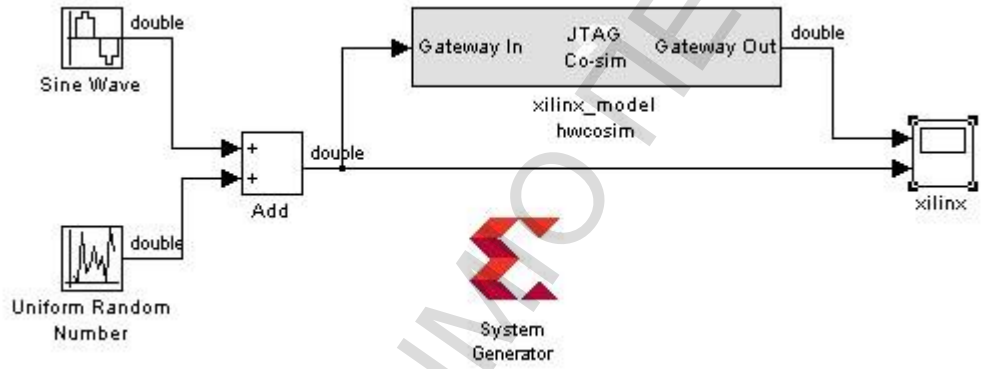
Βλέπουμε πως ο παλμογράφος «Difference» μας δείχνει ευθεία γραμμή (εικόνα 33). Άρα η λειτουργία των δύο μοντέλων είναι ακριβώς η ίδια. Η μικροσκοπικές διακυμάνσεις που παρατηρούνται, οφείλονται στον κβαντισμό που υφίσταται το σήμα στο μοντέλο Xilinx.

Τώρα είναι η ώρα να υλοποιήσουμε το μοντέλο Xilinx σε hardware που θα τρέξει στο FPGA. Κάνουμε διπλό κλικ στο εικονίδιο system generator. Στην καρτέλα Compilation και στο πλαίσιο Compilation επιλέγουμε hardware co-simulation και κατόπιν την πλακέτα που θα χρησιμοποιήσουμε. Για την εργασία αυτή επιλέγουμε Virtex5. Κατόπιν πατάμε το κουμπάκι Generate και ξεκινά η δημιουργία του hardware μοντέλου. Μετά από 6-7 λεπτά η διαδικασία ολοκληρώνεται και εμφανίζεται ένα μπλοκ που περιλαμβάνει όλα τα μπλοκ της Xilinx που βρίσκονταν ανάμεσα στο «gateway in» και «gateway out» (εικόνα 35)



Εικόνα 35 μπλοκ για HW Co-Sim

Χρησιμοποιούμε αυτό το μπλοκ για να αντικαταστήσουμε όλα τα μπλοκ της Xilinx συμπεριλαμβανομένων των gateway in και gateway out. Φτιάχνουμε ένα μοντέλο όπως αυτό που φαίνεται στην εικόνα 36.



Εικόνα 36 Μοντέλο HW CoSim

Το σήμα εισόδου παράγεται από το Simulink με τα μπλοκ sine wave, Uniform Random Number και Add. Η έξοδος καταγράφεται παράλληλα με το σήμα εισόδου σε ένα παλμογράφο του Simulink. Πατώντας το κουμπάκι της έναρξης της προσομοίωσης το κατάλληλο bitfile γράφεται στο fpga, και μέσω του αγωγού jtag στέλνεται στην πλακέτα το σήμα που παράγει το Simulink. Μέσω του αγωγού jtag πάλι, η πλακέτα στέλνει πίσω στον παλμογράφο του Simulink την έξοδο που έχει υπολογίσει. Είναι τώρα ώρα για να μετρήσουμε την απόδοση της κάθε μεθόδου προσομοίωσης.

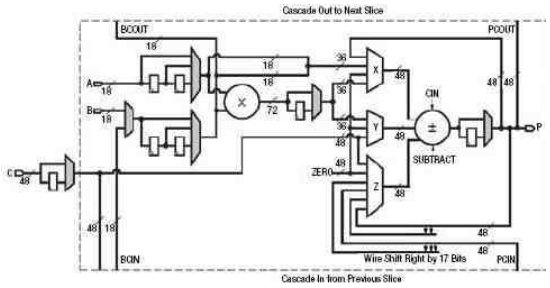
Οι δοκιμές έγιναν μετρώντας τον χρόνο που απαιτείται για να ολοκληρωθούν 1000 κύκλοι σε κάθε ένα από τα δύο μοντέλα.

Μοντέλο	Χρόνος
Simulink model	5.4 sec
HW Co-Simulation	180 sec

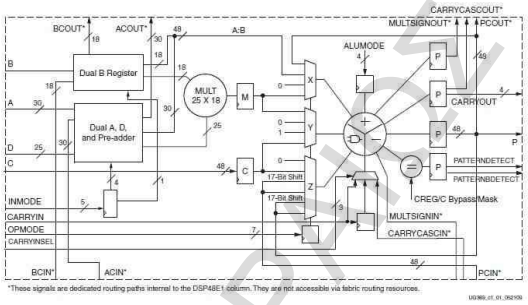
Στο απλό αυτό μοντέλο φαίνεται το σημαντικό πρόβλημα στην προσομοίωση με χρήση hardware. Η επικοινωνία ανάμεσα στο λογισμικό που τρέχει στον υπολογιστή και στο κύκλωμα που έχει υλοποιηθεί στο FPGA της πλακέτας απαιτεί χρόνο πολλαπλάσιο του χρόνου επεξεργασίας των δεδομένων. Το όποιο όφελος από την αυξημένη ταχύτητα των υπολογισμών στο FPGA χάνεται μπροστά στην καθυστέρηση από την μεταφορά των δεδομένων.

6.2. DSP48 slice

Στις σειρές πλακετών FPGA virtex4 και virtex5 η Xilinx έχει ενσωματώσει το τεμάχιο (slice) DSP48. Αυτό είναι μία αριθμητική μονάδα υψηλών επιδόσεων μαζί με έναν πολλαπλασιαστή. Το τεμάχιο αποτελείται από τέσσερα τμήματα: α) καταχωρητές εισόδου – εξόδου, β) πολλαπλασιαστής προσημασμένων αριθμών 25X18 bit, γ) αθροιστής - αφαιρέτης τριών εισόδων των 48 bit, δ) πολυπλέκτες.



Εικόνα 37 Virtex-4 DSP48 slice



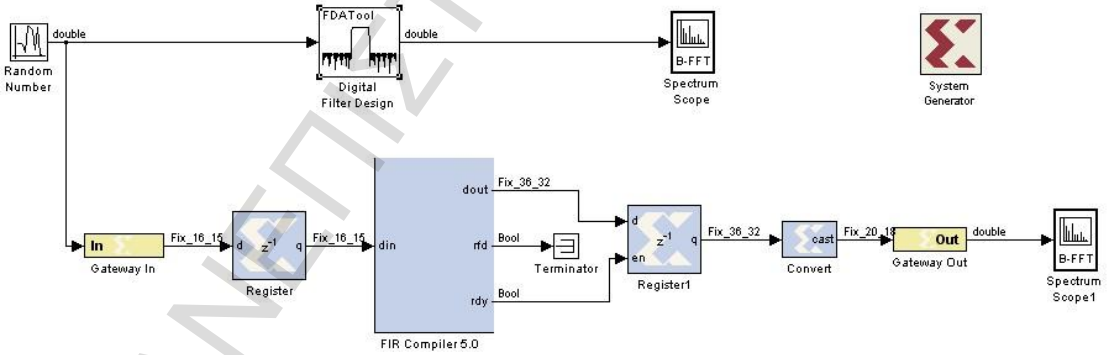
Εικόνα 38 Virtex-5 DSP48 slice

Στην πλακέτα Virtex5 της εργασίας υπάρχουν ενσωματωμένα 64 τεμάχια DSP48. Ο αριθμός αυτός σε άλλες πλακέτες της σειράς virtex-5 μπορεί να φθάσει και τα 550. Το τεμάχιο DSP48 μπορεί να εκτελέσει 48 διαφορετικές μαθηματικές πράξεις. Η βασική πράξη που εκτελεί είναι η p=a*b+(c+cin). Άλλες πράξεις μπορούν να επιλεγούν δυναμικά μέσω ενός κωδικού (opcode) των 11 bit. Το τεμάχιο DSP48 έχει μικρή κατανάλωση και υψηλή απόδοση. Μπορεί να λειτουργήσει σε συσκευές με ρολόι που ξεπερνά τους 500 MHZ. Είναι ιδιαίτερα χρήσιμο σε υλοποιήσεις φίλτρων FIR βασισμένων σε MAC (Multiply – Accumulate). Το μοντέλο φίλτρου που ακολουθεί απαιτεί FPGA που περιέχει το τεμάχιο DSP48.

6.2.1. Προσομοίωση φίλτρου FIR

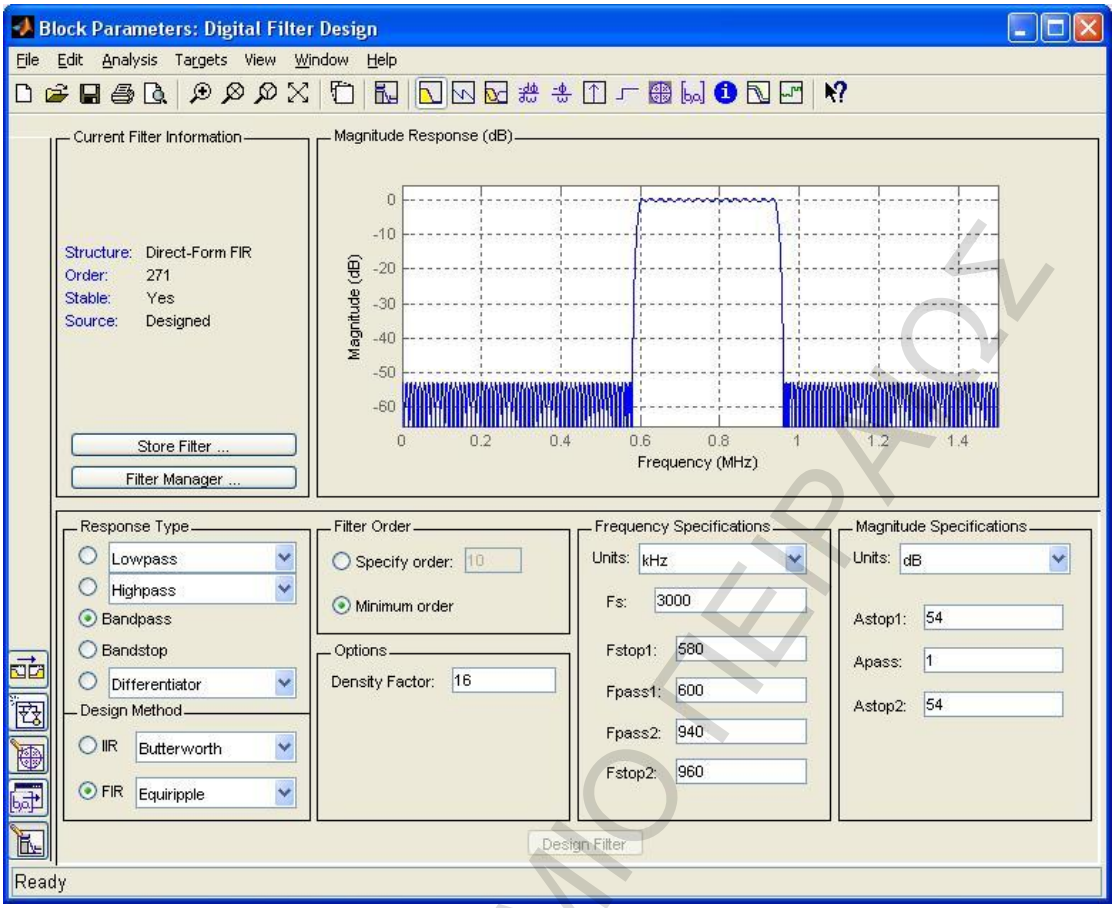
Το μοντέλο αυτό υπάρχει στα παραδείγματα που παρέχονται με το πακέτο ISE της XILINX και βρίσκεται στην θέση:

«C:\Xilinx\14.2\ISE_DS\ISE\sysgen\examples\getting_started_training\lab7\solution\» με το όνομα «lab7_solution.mdl»



Εικόνα 39 FIR Filter

Στο επάνω μέρος υπάρχει το φίλτρο της software προσομοίωσης του Simulink που ουσιαστικά υλοποιείται με το FDA Tool του Simulink. Στο κάτω μέρος το ίδιο φίλτρο είναι φτιαγμένο με μπλοκ της Xilinx. Αυτό θα υλοποιηθεί κατόπιν σε hardware στο FPGA. Τα δύο φίλτρα παίρνουν τις παραμέτρους τους από το «FDA Tool». Κάνοντας διπλό κλικ πάνω του ανοίγει η καρτέλα με τις ρυθμίσεις των παραμέτρων του φίλτρου (εικόνα 40).



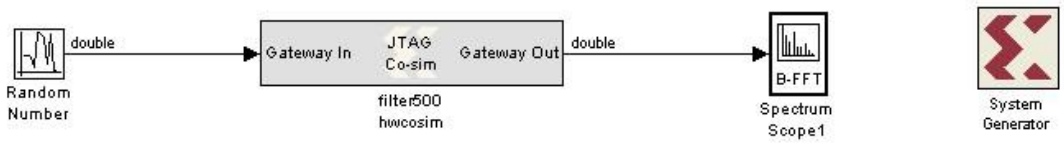
Εικόνα 40 Παράμετροι του FIR Filter

Επιλέγουμε τύπο φίλτρου Bandpass και αλλάζοντας τις τιμές Fstop1 και Fstop2 μπορούμε ρυθμίσουμε πόσο απότομες θα είναι οι πλαγιές του φίλτρου. Όσο πιο απότομες οι πλαγιές τόσο υψηλότερη η τάξη του φίλτρου και τόσο πολυπλοκότερη η υλοποίησή του. Πειραματικά βρήκαμε πως η πιο πολύπλοκη κατασκευή που μπορεί να υλοποιηθεί με την Virtex5 είναι με τιμές Fstop1=580 και Fstop2=960. Με αυτό τον συνδυασμό χρησιμοποιούνται 62 από τα 64 DSP48 μπλοκ που διαθέτει το FPGA. Αφού ρυθμίσουμε τις παραμέτρους όπως φαίνονται στην εικόνα 40, πατάμε το κουμπάκι «Design Filter» για να υπολογισθεί το φίλτρο. Για να μεταφερθούν οι παράμετροι της σχεδίασης και στο FIR Compiler επιλέγουμε File→Export…→Export to Workspace. Ελέγχουμε πως είναι επιλεγμένες οι ρυθμίσεις:

Export to Workspace, Export as Coefficients, Numerator: Num

Τώρα μπορούμε να κλείσουμε το παράθυρο σχεδίασης φίλτρων. Κάνουμε διπλό κλικ στο FIR Compiler και ελέγχουμε πως στο πλαίσιο Coefficients γράφει «Num» ώστε να διαβάσει τις τιμές που υπολόγισε το FDA tool.

Προκειμένου να δημιουργήσουμε το bitfile για το FPGA κάνουμε διπλό κλικ στο εικονίδιο του System Generator. Στην καρτέλα Compilation και στο πλαίσιο Compilation επιλέγουμε hardware co-simulation και κατόπιν επιλέγουμε το μοντέλο Virtex5. Στην συνέχεια πατάμε το κουμπάκι Generate και ξεκινά η δημιουργία του hardware μοντέλου. Μετά από 40 περίπου λεπτά η διαδικασία ολοκληρώνεται και εμφανίζεται ένα μπλοκ που περιλαμβάνει όλα τα μπλοκ της Xilinx που βρίσκονταν ανάμεσα στο gateway in και gateway out. Χρησιμοποιώντας αυτό το μπλοκ φτιάχνουμε το μοντέλο για την προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop (εικόνα 41).

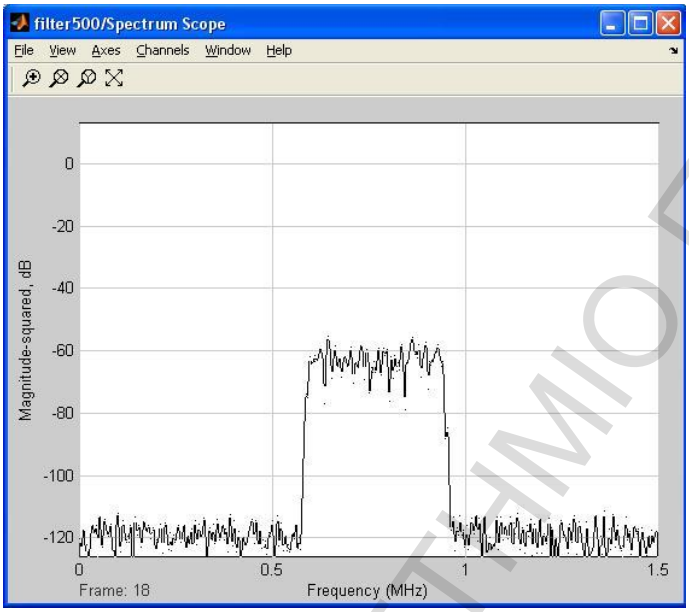


Εικόνα 41 Το μοντέλο φίλτρου FIR με χρήση της τεχνικής FPGA-In-the-Loop

Μπορούμε τώρα να κάνουμε μετρήσεις για τον χρόνο που απαιτείται για να προσομοιωθούν 1000 frames σε κάθε ένα από τα δύο μοντέλα.

Μοντέλο	Χρόνος
Simulink model	25 sec
HW Co-Simulation	1600 sec

Και στα δύο μοντέλα η απόκριση στο Spectrum Scope είναι ίδια με αυτή της εικόνας 42.



Εικόνα 42 Απόκριση φίλτρου

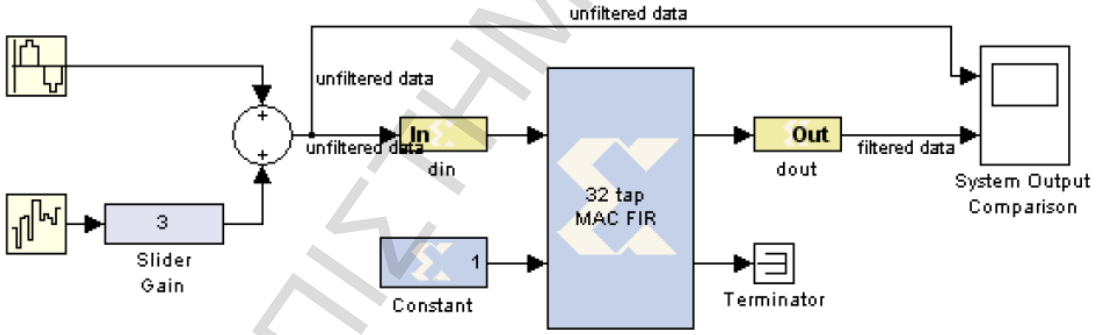
Στο μοντέλο αυτό οι υπολογισμοί απαιτούν πολύ περισσότερο χρόνο από αυτούς του πρώτου μοντέλου. Ο χρόνος που απαιτείται για χίλιους κύκλους προσομοίωσης στο software μοντέλο σχεδόν πενταπλασιάστηκε. Παραδόξως ο χρόνος για την hardware co-simulation σχεδόν δεκαπλασιάστηκε. Αυτό εξηγείται από το γεγονός ότι στο πρώτο μοντέλο χρειαζόταν η επεξεργασία μιας μόνο τιμής για την εξαγωγή ενός σημείου στην έξοδο. Τώρα απαιτείται η επεξεργασία πολύ περισσότερων τιμών για την δημιουργία ενός πλαισίου. Κατά συνέπεια πολύ περισσότερα δεδομένα χρειάζεται να μεταφερθούν σε κάθε κύκλο προσομοίωσης. Η μεταφορά αυτή απαιτεί πολλαπλάσιο χρόνο από αυτόν που κερδίζουμε με την εκτέλεση των υπολογισμών σε hardware. Έτσι η απόδοση της προσομοίωσης με χρήση hardware co-simulation μειώθηκε στο μισό σε σχέση με το προηγούμενο παράδειγμα.

7. Επικοινωνία υλικού-λογισμικού

Υπάρχουν πολλοί παράγοντες που καθορίζουν το ποσοστό της επιτάχυνσης που μπορεί να επιτευχθεί σε μια σχεδίαση κατά την προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop. Αυτοί περιλαμβάνουν το μέγεθος της σχεδίασης, το πλήθος των εισόδων-εξόδων και τον υπερχρονισμό (oversampling) του hardware. Ο υπερχρονισμός αφορά την ασύγχρονη λειτουργία του FPGA και του Simulink. Επιλέγοντας τον τρόπο χρονισμού «free running» στο FPGA τότε το ρολόι του FPGA δεν οδηγείται από το Simulink αλλά από το κύκλωμα χρονισμού της πλακέτας. Έτσι το hardware λειτουργεί σε μέγιστη ταχύτητα και μπορεί να εκτελέσει πολλούς κύκλους σε ένα κύκλο του Simulink. Η άλλη επιλογή είναι το Simulink να παράγει τους παλμούς χρονισμού του FPGA. Σ' αυτή την περίπτωση παράγεται ένας κύκλος ρολογιού για το FPGA σε κάθε κύκλο του Simulink.

Κάτω από κανονικές συνθήκες το PC επικοινωνεί με το FPGA σε κάθε κύκλο του Simulink. Αυτές οι συνεχείς δοσοληψίες μεταξύ hardware και software μπορεί να είναι ο περιοριστικός παράγοντας της απόδοσης της προσομοίωσης. Είναι δυνατό το επιπλέον φορτίο να κάνει την προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop να είναι πιο αργή από την προσομοίωση μόνο με software.

Στο Xilinx application 1031: «Decreasing Simulation Runtimes with System Generator for DSP Hardware Co-Simulation» προσομοιώνεται το μοντέλο της εικόνας 43 κάνοντας χρήση της τεχνικής FPGA-In-the-Loop. Η πλακέτα που χρησιμοποιείται είναι η ML405 η οποία υποστηρίζει τύπους διεπαφής JTAG και ETHERNET. Γίνονται μετρήσεις για τον χρόνο εκτέλεσης της προσομοίωσης με τον κάθε τρόπο σύνδεσης. Χρησιμοποιούνται δύο τύποι σύνδεσης για την διεπαφή JTAG. Στον πρώτο, στην πλευρά του υπολογιστή χρησιμοποιείται σύνδεση USB ενώ στον δεύτερο χρησιμοποιείται η παράλληλη θύρα. Όσο αφορά την σύνδεση τύπου ETHERNET χρησιμοποιήθηκαν τρεις διαφορετικοί τύποι, με ταχύτητες 10Mbps, 100Mbps και 1000Mbps. Σε κάθε περίπτωση, οι μετρήσεις αφορούν τον χρόνο εκτέλεσης 100.000 κύκλων προσομοίωσης.



Εικόνα 43 Μοντέλο φίλτρου για μετρήσεις επίδοσης διεπαφής

Ακολουθεί ο πίνακας σύγκρισης της ταχύτητας του κάθε μέσου σύνδεσης

Simulink simulation cycles = 100 000	JTAG: Platform USB (12 MHz)	JTAG: Parallel IV (5 MHz)	P2P Ethernet 10 Mbps	P2P Ethernet 100 Mbps	P2P Ethernet 1 Gbps
Simulink: Outside System Generator	29.88	30.42	127.74	28.99	28.57
Sysgen: Block Configuration	0.09	0.08	0.19	0.09	0.09
Sysgen: Compile	0.04	0.04	0.31	0.04	0.05
Sysgen: Initialize Simulation	3.66	4.98	0.10	9.08	8.94
Sysgen: Simulation	117.53	41.14	86.06	17.69	15.68
TOTAL	151.2	76.67	214.41	55.89	53.33

Εικόνα 44 Σύγκριση ταχύτητας συνδέσεων.

Όπως φαίνεται για την συγκεκριμένη δοκιμή με σύνδεση JTAG μέσω USB χρειάζονται 117.53 sec. που είναι και ο μεγαλύτερος χρόνος μεταξύ των διαφόρων συνδέσεων. Τον καλύτερο χρόνο ανάμεσα στις αναφερόμενες συνδέσεις επιτυγχάνει η gigabit Ethernet. Με μικρή διαφορά ακολουθεί η σύνδεση με Ethernet 100 Mbps. Η πλακέτα Virtex5 της εργασίας έχει διαθέσιμο τρόπο σύνδεσης μόνο το JTAG με USB. Ο χρόνος εκτέλεσης της προσομοίωσης με σύνδεση JTAG μέσω USB είναι 117,53 δευτερόλεπτα ενώ ο αντίστοιχος χρόνος με σύνδεση ETHERNET 100Mbps είναι μόλις 17,69 δευτερόλεπτα. Βλέπουμε δηλαδή επιτάχυνση της προσομοίωσης κατά έξι φορές αλλάζοντας μόνο τον τρόπο σύνδεσης. Και στις δύο περιπτώσεις υπάρχει ένας λανθάνων χρόνος αρχικοποίησης του συστήματος της τάξης των τριάντα δευτερολέπτων.

Ένας τρόπος να βελτιωθεί η απόδοση της προσομοίωσης είναι η ομαδοποίηση των δεδομένων και η μαζική μεταφορά τους (burst transfers). Αντί να στέλνονται και να λαμβάνονται δεδομένα σε κάθε κύκλο του Simulink, στέλνονται σαν διάνυσμα (vector), επεξεργάζονται στο FPGA και κατόπιν επιστρέφουν στο Simulink πάλι σαν διάνυσμα. Αυτή η τεχνική μπορεί να εξοικονομήσει χρόνο και να κάνει την προσομοίωση με hardware πιο αποδοτική.

Τα βήματα που εκτελούνται με αυτή την μέθοδο σε κάθε κύκλο του Simulink είναι τα παρακάτω:

- Αποθήκευση μιας σειράς αριθμητικών δεδομένων σε ένα διάνυσμα του Simulink.
- Αποστολή του διανύσματος σε μια προσωρινή μνήμη (buffer) στο FPGA χρησιμοποιώντας μαζική μεταφορά δεδομένων (burst transfer).
- Διαδοχική επεξεργασία όλων των δεδομένων της προσωρινής μνήμης επιλέγοντας τον τρόπο λειτουργίας «free running» στο FPGA.
- Αποθήκευση των δεδομένων από το FPGA στην προσωρινή μνήμη.
- Αποστολή των δεδομένων της προσωρινής μνήμης στο Simulink με την μορφή διανύσματος χρησιμοποιώντας μαζική μεταφορά δεδομένων.
- Μετατροπή των δεδομένων του διανύσματος σε μια σειρά από αριθμητικά δεδομένα.

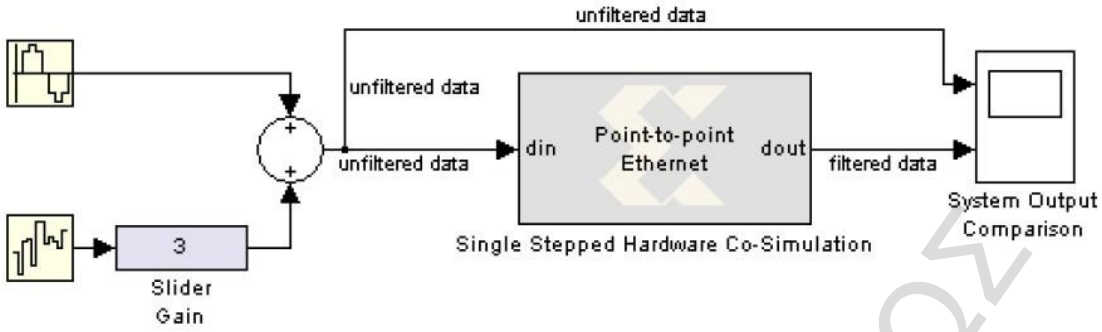
7.1. Κοινόχρηστες μνήμες

Για να μπορέσει να υλοποιηθεί η μαζική μεταφορά δεδομένων θα πρέπει να ενσωματωθούν μνήμες στο hardware. Αυτές συνήθως κατασκευάζονται από τις μνήμες BRAM του FPGA. Σε αυτές αποθηκεύονται τα δεδομένα που λαμβάνονται και αποστέλλονται στο FPGA. Είναι φανερό ότι το μέγεθός τους περιορίζεται από το μέγεθος των μνημών που διαθέτει η συσκευή.

Χρησιμοποιούνται δύο τύποι μνήμης: κοινόχρηστα FIFO (shared FIFO) και κοινόχρηστες μνήμες με αμοιβαίο αποκλεισμό (lockable shared memory). Οι μνήμες αυτές αν και μοιάζουν με τα μπλοκ που διαθέτει το toolkit της Xilinx έχουν μια σημαντική διαφορά. Ελέγχονται και από το PC και από το FPGA. Υλοποιούνται στο FPGA αλλά η μία τους πλευρά ελέγχεται από το software και η άλλη από το hardware. Στις μνήμες με αμοιβαίο αποκλεισμό το software και το hardware μπορούν να διαβάσουν και να γράψουν στις μνήμες αυτές. Όταν όμως χρησιμοποιούνται από το software, κλειδώνονται και δεν μπορεί να τις προσπελάσει το hardware. Αντίστοιχα όταν χρησιμοποιούνται από το hardware, το software αποκλείεται.

Στο Xilinx application 1031: «Decreasing Simulation Runtimes with System Generator for DSP Hardware Co-Simulation» γίνεται σύγκριση των τρόπων επικοινωνίας: χωρίς κοινόχρηστες μνήμες και μέσω κοινόχρηστων μνημών. Το μοντέλο ενός φίλτρου που προσομοιώνεται στην πρώτη περίπτωση φαίνεται στην εικόνα 45. Ουσιαστικά είναι το ίδιο με αυτό της εικόνας 43, αλλά το κύκλωμα που θα υλοποιηθεί στο FPGA έχει αντικατασταθεί από ένα μπλοκ.

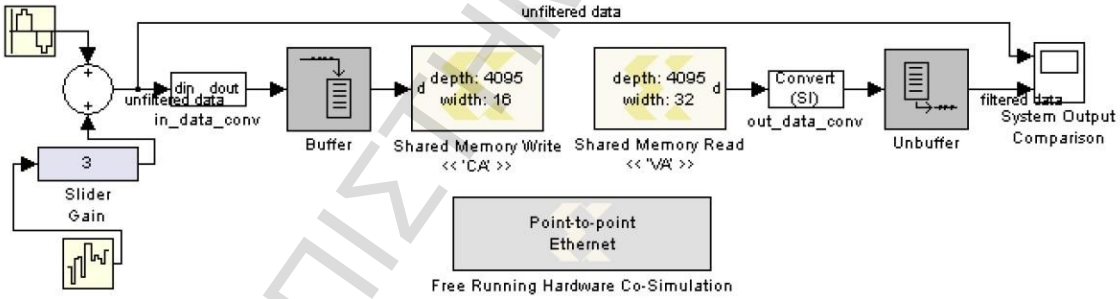
Στην εικόνα 47 φαίνεται το ίδιο μοντέλο τροποποιημένο ώστε να χρησιμοποιεί κοινόχρηστες μνήμες. Στο μοντέλο αυτό τα δεδομένα γράφονται μαζικά στην μνήμη εισόδου και κατόπιν επεξεργάζονται από το φίλτρο που είναι υλοποιημένο στο FPGA. Κατόπιν τα δεδομένα από την μνήμη εξόδου μεταφέρονται μαζικά στον υπολογιστή.



Εικόνα 45 Μοντέλο φίλτρου χωρίς χρήση της τεχνικής κοινόχρηστης μνήμης

Simulink Simulation Cycles	P2P Ethernet HW-Cosim (1 Gbps)		
	1,000	10,000	100,000
Simulink: Outside System Generator	0.33	3.02	30.90
Sysgen: Block Configuration	0.09	0.08	0.09
Sysgen: Compile	0.05	0.05	0.04
Sysgen: Initialize Simulation	9.22	9.02	9.06
Sysgen: Simulation	0.18	1.57	16.70
TOTAL	9.87	13.79	56.79

Εικόνα 46 Χρόνοι προσομοίωσης χωρίς χρήση της τεχνικής κοινόχρηστης μνήμης



Εικόνα 47 Μοντέλο φίλτρου με χρήση της τεχνικής κοινόχρηστης μνήμης

Simulink Simulation Cycles	P2P Ethernet HW-Cosim (1 Gbps)			
	1,000	10,000	100,000	1,000,000
Simulink: Outside System Generator	0.10	0.24	1.78	16.32
Sysgen: Block Configuration	0.09	0.09	0.09	0.09
Sysgen: Compile	0.04	0.04	0.04	0.04
Sysgen: Initialize Simulation	9.21	9.18	9.30	9.65
Sysgen: Simulation	3.25	0.00	0.00	0.00
TOTAL	9.44	9.55	11.20	26.09

Εικόνα 48 Χρόνοι προσομοίωσης με χρήση της τεχνικής κοινόχρηστης μνήμης

Στα αποτελέσματα των πινάκων 46 και 48 παρατηρούμε πως για μικρό πλήθος κύκλων προσομοίωσης(<10.000) δεν υπάρχει όφελος από την χρήση των κοινόχρηστων μνημών. Το όφελος αρχίζει να γίνεται σημαντικό για προσομοιώσεις με 100.000 κύκλους ή περισσότερους. Για προσομοίωση 100.000 κύκλων ο χρόνος εκτέλεσης στο μοντέλο χωρίς μνήμες είναι 56,79 δευτερόλεπτα ενώ στο μοντέλο με κοινόχρηστες μνήμες ο χρόνος αυτός πέφτει στα 11,20 δευτερόλεπτα. Έχουμε αύξηση της απόδοσης στο πενταπλάσιο με χρήση των κοινόχρηστων μνημών για προσομοίωση 100.000 κύκλων. Σε κάθε περίπτωση η διασύνδεση με τον υπολογιστή γίνεται με ETHERNET 1Gbps.

Μπλοκ κοινόχρηστων μνημών υπάρχουν στο toolkit της Xilinx. Στην εργασία αυτή, κοινόχρηστες μνήμες χρησιμοποιήθηκαν στο παράδειγμα «Επεξεργασία σήματος Video» που ακολουθεί.

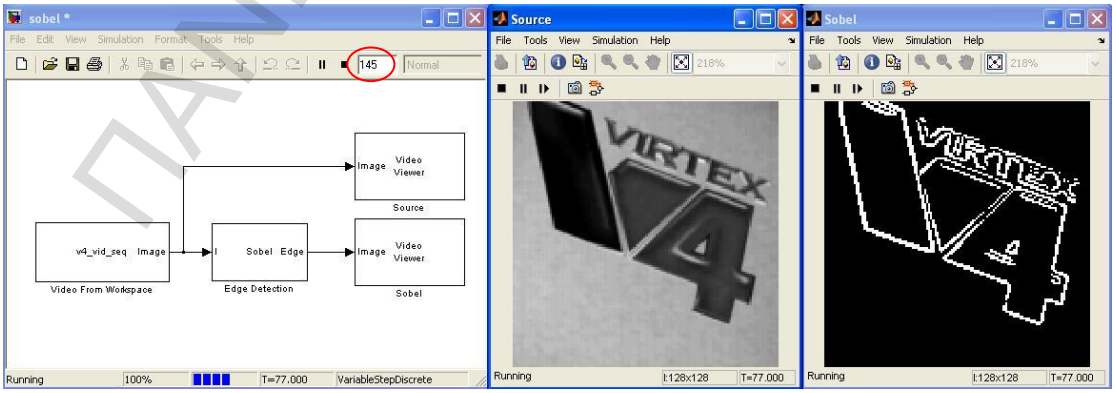
7.2. Επεξεργασία σήματος Video

Στην εφαρμογή αυτή θα δημιουργηθεί ένα μοντέλο που θα εφαρμόζει τον ανιχνευτή ακμών Sobel σε πλαίσια εικόνας ενός αρχείου πολυμέσων. Η ανίχνευση θα γίνει με επεξεργασία του κάθε εικονοστοιχείου της εικόνας ξεχωριστά. Για να μειωθεί ο χρόνος επεξεργασίας δεν θα ακολουθηθεί η διαδικασία: αποστολή του κάθε εικονοστοιχείου στην πλακέτα με το FPGA - επεξεργασία του εικονοστοιχείου - αποστολή του επεξεργασμένου εικονοστοιχείου στον υπολογιστή. Η καινοτομία είναι ότι θα αποστέλλεται ολόκληρο το πλαίσιο της εικόνας προς επεξεργασία στην πλακέτα. Εκεί θα γίνεται επεξεργασία κάθε ενός εικονοστοιχείου και στο τέλος ολόκληρο το πλαίσιο της επεξεργασμένης εικόνας θα αποστέλλεται στον υπολογιστή. Με αυτόν τον τρόπο ελαχιστοποιείται ο χρόνος που απαιτείται για την επικοινωνία πλακέτας – υπολογιστή.

Στο System Generator υπάρχει τρόπος επικοινωνίας ανάμεσα στο Simulink και στο FPGA μέσω κοινόχρηστης μνήμης. Ο τρόπος αυτός είναι κατάλληλος για τις εφαρμογές όπου μεγάλος όγκος δεδομένων μπορεί να μεταφερθεί στην πλακέτα με το FPGA και εκεί να γίνει η επεξεργασία του με πλήρη ταχύτητα. Ανάλογα με το μέσο επικοινωνίας της πλακέτας με τον υπολογιστή (Ethernet, usb, Jtag) είναι δυνατό να υπάρχει αρκετά μεγάλη ταχύτητα επεξεργασίας δεδομένων ώστε να γίνει επεξεργασία σημάτων σε πραγματικό χρόνο.

Ο τρόπος επικοινωνίας που εφαρμόζεται σε αυτό το παράδειγμα είναι μέσω κοινόχρηστης μνήμης (lockable shared memory). Η μνήμη αυτή υλοποιείται στο FPGA και έχουν πρόσβαση σ' αυτήν και το Simulink και το hardware του FPGA. Το Simulink έχει πρόσβαση μέσω της σύνδεσης Jtag. Τα απαραίτητα κυκλώματα που χειρίζονται τον αμοιβαίο αποκλεισμό των Simulink και hardware δημιουργούνται αυτόματα κατά την δημιουργία (compilation) της κοινόχρηστης μνήμης.

Το παράδειγμα που ακολουθεί είναι ένα από αυτά που παρέχονται με το ISE και υπάρχει στο
C:\Xilinx\14.2\ISE_DS\ISE\sysgen\examples\shared_memory\hardware_cosim\conv5x5_video.
Το αρχείο v4_vid_seq είναι μια ακολουθία από πλαίσια ασπρόμαυρης εικόνας με μέγεθος 128X128 στοιχεία και βάθος 8 bit σε κάθε εικονοστοιχείο. Στην εικόνα 49 στα αριστερά φαίνεται ένα κύκλωμα Simulink που εφαρμόζει έναν ανιχνευτή ακμών Sobel στο σήμα αυτό.

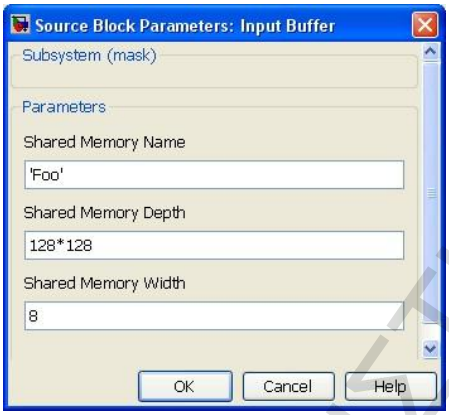


Εικόνα 49 Ανιχνευτής ακμών Sobel

Στην εικόνα 49 στο κέντρο, φαίνεται ένα στιγμιότυπο από το σήμα εισόδου και στα δεξιά φαίνεται το σήμα εξόδου κατά την ίδια χρονική στιγμή. Για να λειτουργήσει η προσομοίωση πρέπει πρώτα να μεταφέρουμε τον πίνακα με τα αποθηκευμένα πλαίσια του σήματος εισόδου στο χώρο εργασίας του Matlab. Για να γίνει αυτό κάνουμε διπλό κλικ πάνω στο αρχείο v4_vid_seq.mat που βρίσκεται στο παράθυρο current folder. Στο μπλοκ «video from workspace» κάνουμε διπλό κλικ και στο πλαίσιο «signal» γράφουμε το όνομα του αρχείου: «v4_vid_seq». Το σήμα εισόδου περιέχει 145 πλαίσια οπότε για να το επεξεργασθεί ολόκληρο, το Simulink πρέπει να τρέξει για 145 κύκλους. Εισάγουμε την τιμή αυτή στο πλαίσιο που επισημένεται με κόκκινο στην εικόνα 49. Στο υπολογιστή με επεξεργαστή P4 2.66GHZ χρειάστηκαν 40 sec για την επεξεργασία των 145 πλαισίων.

Το μοντέλο που θα χρησιμοποιήσουμε για την προσομοίωση με hardware βασίζεται στο μπλοκ «5X5filter» της Xilinx (εικόνα 51). Υπάρχουν 9 προκαθορισμένες μήτρες που καθορίζουν 9 διαφορετικά αποτελέσματα όπως ανίχνευση ακμών, αύξηση λεπτομέρειας (sharpen) και εξομάλυνση (soften).

Η επικοινωνία γίνεται μέσω κοινόχρηστων μνημών. Το μέγεθός τους προσδιορίζεται από το μέγεθος του κάθε πλαισίου. Οι διαστάσεις των πλαισίων είναι 128X128 με 8 bit σε κάθε εικονοστοιχείο. Κάνοντας διπλό κλικ πάνω στο μπλοκ μνήμης εισόδου (εικόνα 53) δίνουμε τις τιμές στο πλαίσιο των παραμέτρων της μνήμης την οποία ονομάζουμε «Foo» (εικόνα 50). Με αυτό τον τρόπο ένα ολόκληρο πλαίσιο (128X128=16K) θα μεταφέρεται στην μνήμη του FPGA και κατόπιν θα αρχίζει η επεξεργασία ολόκληρου του πλαισίου χωρίς άλλες μεταφορές δεδομένων μεταξύ hardware και software. Μετά την επεξεργασία τα δεδομένα αποθηκεύονται στην μνήμη εξόδου που έχει τα ίδια χαρακτηριστικά με την μνήμη εισόδου και ονομάζεται «bar».

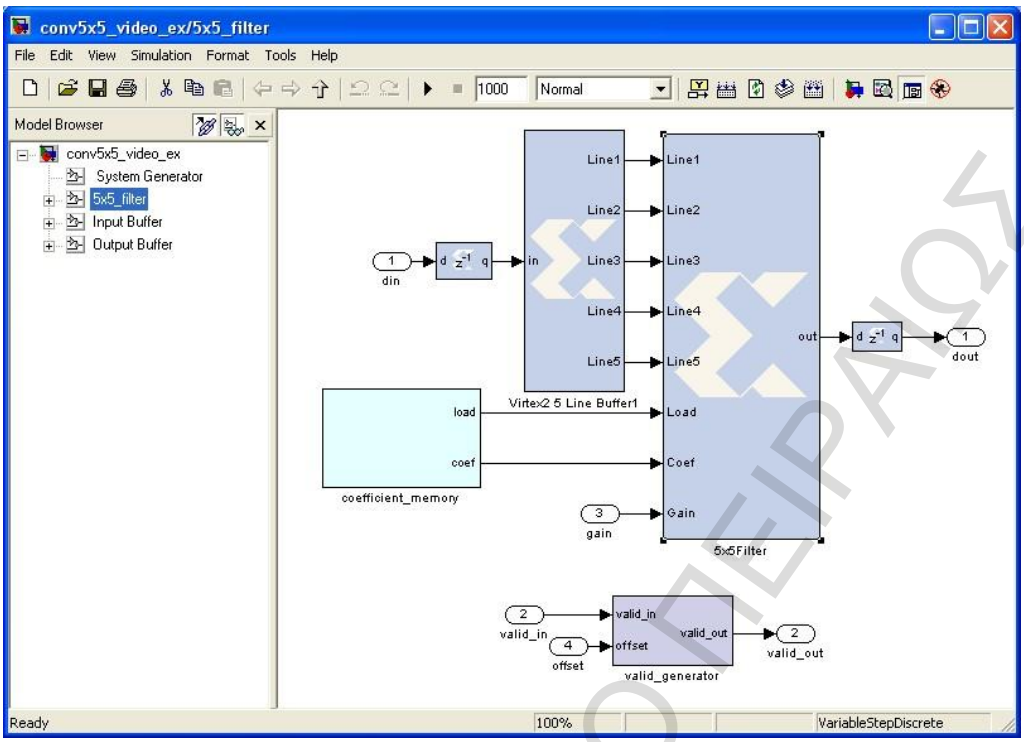


Εικόνα 50 Παράμετροι μνήμης

Τα δεδομένα από την μνήμη εισόδου προωθούνται σε μία προσωρινή μνήμη πέντε γραμμών (5 line buffer) - (εικόνα 51). Το μέγεθος της μνήμης μπορεί να καθοριστεί για να καλύψει τις ανάγκες διάφορων εφαρμογών. Σε αυτό το παράδειγμα το ορίζουμε ίσο με 128 που είναι το πλάτος της εικόνας. Στην έξοδο Line5 το σήμα είναι ίδιο με αυτό της εισόδου. Στην γραμμή Line4 το σήμα είναι ίδιο με αυτό της εισόδου αλλά καθυστερημένο κατά 128 κύκλους. Στην γραμμή Line3 το σήμα είναι καθυστερημένο κατά 256 κύκλους, στην γραμμή Line2 κατά 384 κύκλους και στην γραμμή 1 κατά 512 κύκλους. Το αποτέλεσμα είναι πέντε διαδοχικές γραμμές του πλαισίου να εμφανίζονται ταυτόχρονα στην έξοδο της προσωρινής μνήμης. Η έξοδος της προσωρινής μνήμης οδηγεί το μπλοκ 5X5filter.

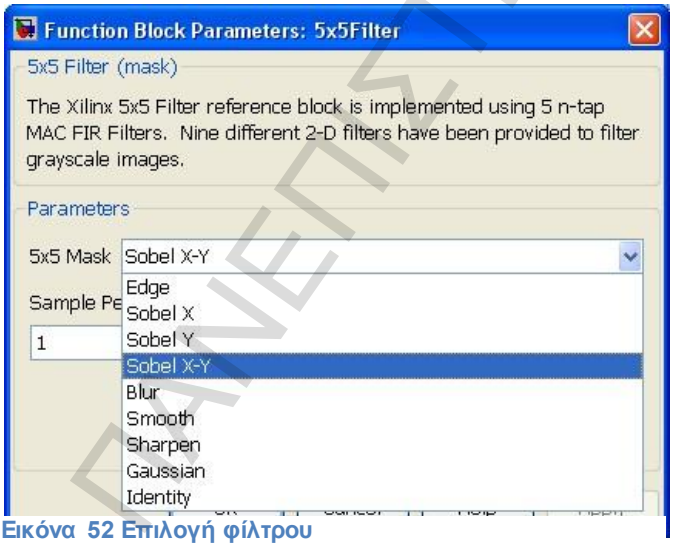
Το φίλτρο 5X5 χρειάζεται 5 κύκλους ρολογιού για να υπολογίσει την τιμή ενός εικονοστοιχείου. Άρα η πλακέτα Virtex5 που λειτουργεί στους 100MHZ με πλαίσιο εικόνας 128X128=16384 εικονοστοιχεία θα μπορεί να επεξεργαστεί $(100X10^6/5)/16384=1220$ πλαίσια/sec.

Στην συνέχεια φαίνεται το μοντέλο «conv5x5_video_ex.mdl» που θα υλοποιηθεί με hardware (εικόνα 51).



Εικόνα 51 Shared memory 5X5 filter

Κάνοντας διπλό κλικ πάνω στο μπλοκ 5X5_filter εμφανίζεται το παρακάτω πλαίσιο (εικόνα 52) επιλογής της λειτουργίας του φίλτρου. Όπως αναφέρθηκε παραπάνω, υπάρχουν 9 επιλογές. Η επιλογή που θα κάνουμε σε αυτό το σημείο είναι αυτή που θα έχει το φίλτρο μόλις φορτώνεται στο FPGA. Υπάρχει δυνατότητα να αλλάξουμε την λειτουργία του φίλτρου χωρίς να χρειαστεί να ξανασυνθέσουμε το hardware μοντέλο.



Εικόνα 52 Επιλογή φίλτρου

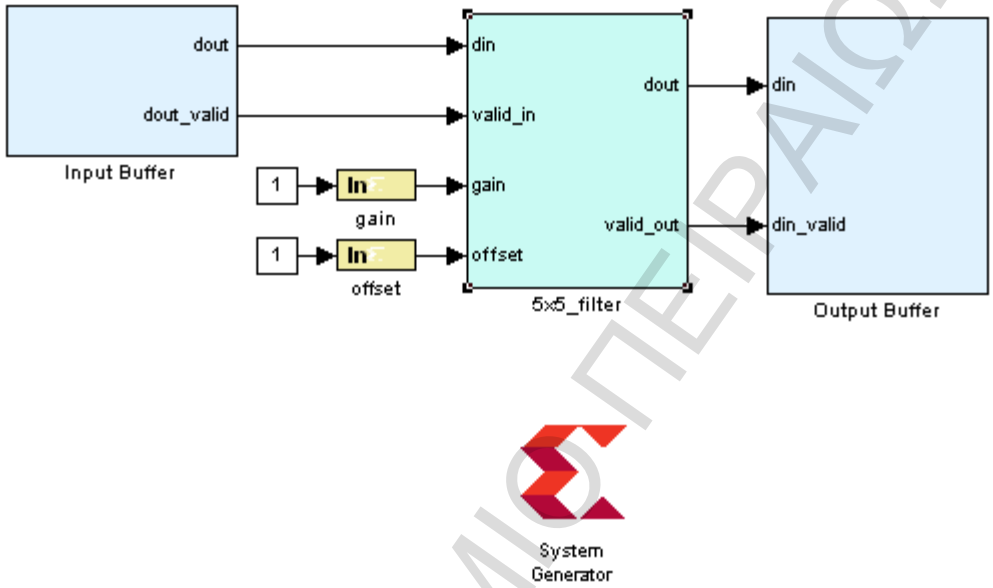
Στο μοντέλο της εικόνας 51 φαίνεται ένα μπλοκ με γαλάζιο χρώμα με το όνομα «coefficient memory». Αυτή είναι η μνήμη παραμέτρων. Σ' αυτήν μπορούμε να φορτώσουμε τις δικές μας παραμέτρους και να ορίσουμε δικές μας λειτουργίες στο φίλτρο. Το φόρτωμα των παραμέτρων μπορεί να γίνει κατά την διάρκεια εκτέλεσης της προσομοίωσης με την συνάρτηση

xlReloadFilterCoef. Π.χ. για να αλλάξουμε την λειτουργία σε ανιχνευτή ακμών Sobel εισάγουμε στο παράθυρο εντολών του Matlab την εντολή:

```
xlReloadFilterCoef('sobelxy')
```

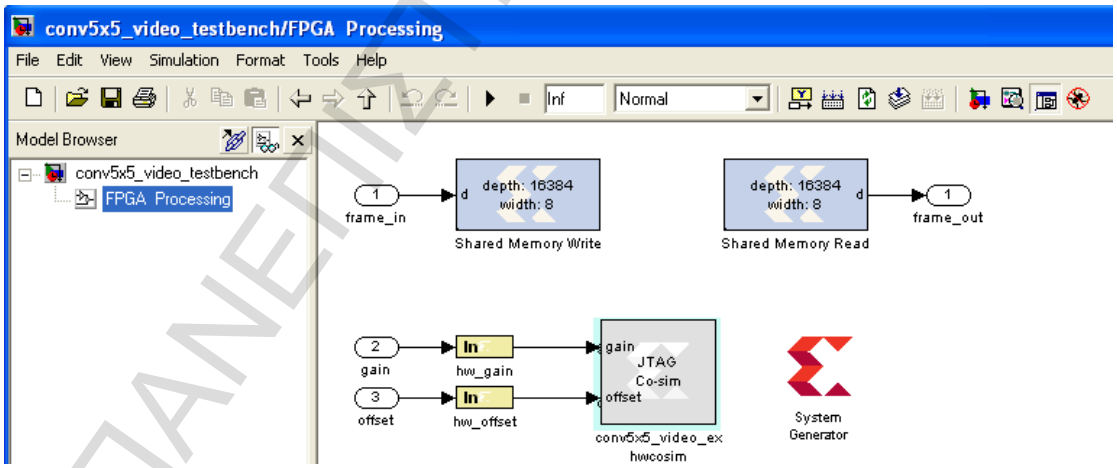
Οι μήτρες για τα διάφορα φίλτρα ορίζονται μέσα στην συνάρτηση xlReloadFilterCoef και περιγράφονται με σχόλια ώστε να μπορούμε εύκολα να προσθέσουμε δικές μας μήτρες.

Για να δημιουργήσουμε το μπλοκ για την προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop, κάνουμε διπλό κλικ στο μπλοκ της Xilinx στο μοντέλο «conv5x5_video_ex.mdl» το οποίο φαίνεται παρακάτω (εικόνα 53).



Εικόνα 53 Μοντέλο φίλτρου για το FPGA

Στην καρτέλα Compilation επιλέγουμε Compilation→Hardware Co-Simulation→Virtex5.

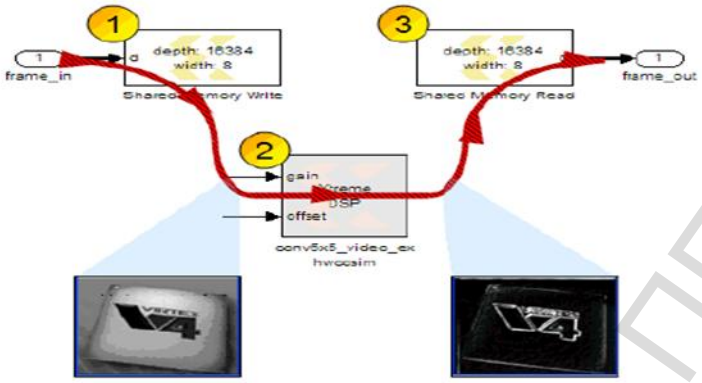


Εικόνα 54 Πλήρες Μοντέλο HW Co-Simulation

Κατόπιν πατάμε στο κουμπί Generate. Όταν δημιουργηθεί το μπλοκ για προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop ανοίγουμε το μοντέλο «conv5x5_video_testbench.mdl» και σέρνουμε το μπλοκ που δημιουργήθηκε πάνω στο γαλάζιο πλαίσιο του μοντέλου. Μόλις τοποθετηθεί ακριβώς στην θέση του συνδέεται αυτόματα (οι κόκκινες διακεκομμένες γραμμές γίνονται μαύρες – εικόνα 54).

Το μπλοκ hwcosim θα πρέπει να τεθεί σε «free running mode». Πάνω στο μπλοκ «conv5x5_video_ex_hwcosim» κάνουμε διπλό κλικ και στο πλαίσιο clocking της καρτέλας basic επιλέγουμε free running.

Στο hardware μοντέλο της εικόνας 54 υπάρχουν τρία επιμέρους κυκλώματα: η μνήμη εισόδου, το μπλοκ επεξεργασίας εικόνας και η μνήμη εξόδου. Η μνήμη εισόδου θα πρέπει να ενεργοποιηθεί πρώτη ώστε να μεταφερθούν σε αυτήν τα δεδομένα προς επεξεργασία. Κατόπιν θα ενεργοποιηθεί το μπλοκ επεξεργασίας εικόνας. Στο στάδιο αυτό θα διαβασθούν τα δεδομένα της μνήμης εισόδου για να γίνει η επεξεργασία και το επεξεργασμένο πλαίσιο εικόνας θα μεταφερθεί στην μνήμη εξόδου. Τέλος θα ενεργοποιηθεί η μνήμη εξόδου και τα επεξεργασμένα δεδομένα θα μεταφερθούν στον υπολογιστή. Ορίζουμε την σειρά των ενεργειών αυτών με την χρήση προτεραιοτήτων για κάθε μπλοκ.



Εικόνα 55 Προτεραιότητες των μπλοκ

Για να επιλέξουμε την κατάλληλη προτεραιότητα κάνουμε δεξί κλικ σε κάθε ένα από τα τρία μπλοκ και επιλέγουμε «block properties…». Στο πλαίσιο «priority» εισάγουμε «1» για το μπλοκ «Shared memory Write», «2» για το μπλοκ «hwcosim» και «3» για το μπλοκ «Shared memory Read».

Οι μετρήσεις αφορούν τον χρόνο που απαιτείται για την επεξεργασία των 145 πλαισίων του σήματος εισόδου. Έγιναν με την βοήθεια των script rt_sw_timer και rt_hw_timer. Αυτά χρησιμοποιούν το ζεύγος εντολών «tic - toc» του matlab για να ξεκινήσουν το χρονόμετρο και για να εμφανίσουν τον χρόνο που χρειάστηκε η προσομοίωση για να ολοκληρωθεί. Τα script αυτά έχουν την παρακάτω μορφή:

```
tic
        sim('conv5x5_video_testbench',  [0,145])
toc
```

Η εντολή «tic» μηδενίζει το χρονόμετρο. Η εντολή «sim» ξεκινά την προσομοίωση του μοντέλου 'conv5x5_video_testbench' για 145 κύκλους. Τέλος, η εντολή «toc» εμφανίζει στο παράθυρο εντολών του Matlab την τρέχουσα τιμή χρονομέτρου.

Οι μετρήσεις έδειξαν τα παρακάτω αποτελέσματα:

Simulink software: 40 sec

Simulink HW Co-Sim: 97 sec

Στο μοντέλο αυτό χρησιμοποιείται η μαζική μεταφορά δεδομένων (burst transfer). Κάθε πλαίσιο μεταφέρεται ολόκληρο στην μνήμη του FPGA και κατόπιν γίνεται η επεξεργασία του κάθε bit της εικόνας. Στην συνέχεια με τον ίδιο τρόπο το πλαίσιο μεταφέρεται πίσω στον υπολογιστή. Η μέθοδος αυτή βελτίωσε την απόδοση της προσομοίωσης με χρήση της τεχνικής FPGA-In-the-Loop, που και πάλι όμως είναι πιο αργή από την προσομοίωση με software. Η προσομοίωση με software χρειάζεται περίπου το μισό χρόνο από αυτόν της προσομοίωσης με χρήση της τεχνικής FPGA-In-the-Loop.

Το φίλτρο που έχει υλοποιηθεί στο FPGA χρειάζεται πέντε κύκλους ρολογιού για να επεξεργασθεί ένα εικονοστοιχείο. Άρα για τα 16.384 εικονοστοιχεία της εικόνας και με συχνότητα ρολογιού 100MHZ συμπεραίνουμε πως το hardware μοντέλο μπορεί να

επεξεργαστεί 1220 πλαίσια/sec. Ο λόγος της πολύ μικρής τελικής απόδοσης είναι η μικρή ταχύτητα μετάδοσης δεδομένων ανάμεσα στον υπολογιστή και το hardware. Για το μοντέλο αυτό χρειάζεται η μεταφορά ενός πλαισίου εικόνας με μέγεθος 16 KB, δύο φορές σε κάθε κύκλο προσομοίωσης. Εκτιμούμε, βάση των στοιχείων των πινάκων 46 και 48 με τα αποτελέσματα των μετρήσεων της Xilinx, πως αν η πλακέτα υποστήριζε επικοινωνία μέσω Ethernet τότε η προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop θα ήταν δύο ή τρεις φορές γρηγορότερη από την προσομοίωση με software.

8. Επιτάχυνση προσομοίωσης των μετρήσεων Bit Error Rate

Ένα πεδίο που απαιτεί την ταχεία επεξεργασία δεδομένων είναι η κωδικοποίηση σημάτων. Αλγόριθμοι κωδικοποίησης όπως ο Viterbi ή ο Turbo Code είναι ιδιαίτερα απαιτητικοί σε ταχύτητα και ταυτόχρονα έχουν και μεγάλη υπολογιστική πολυπλοκότητα. Οι δοκιμές σε κωδικοποιητές-αποκωδικοποιητές που εφαρμόζουν αυτούς τους αλγόριθμους απαιτούν ένα πολύ μεγάλο πλήθος κύκλων προσομοίωσης για να ολοκληρωθούν. Παράλληλα οι παράμετροι που επηρεάζουν την απόδοσή τους είναι τόσο πολλές και η λειτουργία των αποκωδικοποιητών τόσο πολύπλοκη που καθιστούν αδύνατο τον υπολογισμό της με αναλυτικές μεθόδους. Αυτές οι ιδιότητες κάνουν την κωδικοποίηση ιδανικό πεδίο για την εφαρμογή της προσομοίωσης με την τεχνική FPGA-In-the-Loop.

8.1. Σύντομη περιγραφή της εφαρμογής

Στην εφαρμογή αυτή θα δημιουργηθεί ένα μοντέλο που θα εκτιμά την απόδοση της κωδικοποίησης Turbo Code η οποία χρησιμοποιείται για την μετάδοση δεδομένων μέσα από περιβάλλον που υπόκειται σε θόρυβο. Η διαδικασία που ακολουθείται για την εκτίμηση της απόδοσης είναι η ακόλουθη:

- Ένα σετ με τυχαία δεδομένα δημιουργείται.
- Το σετ κωδικοποιείται με τον αλγόριθμο κωδικοποίησης Turbo Code
- Στο κωδικοποιημένο σετ δεδομένων εισάγονται σκόπιμα μερικά λάθη. Αυτή η ενέργεια προσομοιώνει την μετάδοση του σήματος μέσα από περιβάλλον που υπόκειται σε θόρυβο. Το πλήθος των σφαλμάτων που εισάγονται είναι ανάλογο του επιπέδου του θορύβου που θέλουμε να προσομοιώσουμε.
- Το τροποποιημένο σήμα αποκωδικοποιείται και ορισμένα λάθη εντοπίζονται και διορθώνονται.
- Το αρχικό σήμα συγκρίνεται με το αποκωδικοποιημένο και καταμετράται το πλήθος των σφαλμάτων.
- Τα παραπάνω βήματα επαναλαμβάνονται μέχρι ο όγκος των δεδομένων που μεταδόθηκαν ή το πλήθος των σφαλμάτων που καταμετρήθηκαν να φθάσει κάποιο προκαθορισμένο όριο.

Η διαδικασία αυτή επαναλαμβάνεται για κάθε σημείο του γραφήματος. Θα χρειαστεί να επαναληφθεί πολλές φορές για να δημιουργηθεί ένα πλήρες γράφημα με μετρήσεις για διάφορα επίπεδα θορύβου. Ένα πλήθος από παραμέτρους επηρεάζουν την απόδοση της κωδικοποίησης. Για να έχουμε πλήρη εικόνα θα πρέπει να γίνει μια σειρά δοκιμών με διάφορους συνδυασμούς στις παραμέτρους αυτές. Γίνεται σαφές πως η εκτίμηση της απόδοσης της κωδικοποίησης είναι μια ιδιαίτερα εκτενής διαδικασία.

Προκειμένου να εξοικειωθούμε με την ορολογία που χρησιμοποιείται και για να γίνει κατανοητός ο λόγος για τον οποίο η λειτουργία του μοντέλου κωδικοποιητή – αποκωδικοποιητή είναι απαιτητική υπολογιστικά, θα ακολουθήσει η παρουσίαση των βασικών αρχών λειτουργίας του αλγόριθμου κωδικοποίησης TURBO CODE.

8.2. Κωδικοποίηση

Κατά την μετάδοση ενός σήματος θέλουμε να ξέρουμε ότι το μήνυμα έφτασε στον παραλήπτη χωρίς λάθη. Για να το πετύχουμε αυτό προσθέτουμε στο σήμα επιπλέον bit ελέγχου, όπως το bit ισοτιμίας. Για κάθε byte προστίθεται και ένα bit που έχει τιμή τέτοια ώστε το πλήθος των bit με τιμή ίση με 1 μέσα στο byte μαζί με το bit ισοτιμίας να είναι άρτιο ή περιττό, αν έχουμε άρτια ή περιττή ισοτιμία αντίστοιχα. Αυτή η μέθοδος μπορεί να εντοπίσει λάθη μόνο αν το πλήθος τους είναι μονός αριθμός δηλ. 1, 3 ,5 ή 7. Χρησιμοποιείται στην σειριακή επικοινωνία (RS232) στα πληκτρολόγια PS2 και αλλού. Δεν προσφέρει πληροφορία ως προς το ποιο bit έχει λάθος ώστε να γίνει διόρθωση.

Άλλοι τρόποι εντοπισμού λαθών όπως ο κώδικας Hamming στον οποίο προστίθενται περισσότερα bit ελέγχου στο μήνυμα προσφέρουν δυνατότητα να εντοπισθούν τα λάθη όποιο και αν είναι το πλήθος τους και εάν το σφάλμα είναι μόνο σε ένα bit να καθορισθεί ποιο είναι

αυτό ώστε να διορθωθεί. Χρειάζονται όμως m bit ελέγχου για να διασφαλιστεί η ακεραιότητα των n bit του μηνύματος με $2^m \geq n$.

Ένα άλλο ζήτημα είναι το τι θα γίνει αφότου εντοπισθεί κάποιο σφάλμα. Μία δυνατότητα είναι να ζητηθεί επανάληψη της αποστολής του μηνύματος. Η μέθοδος αυτή ονομάζεται ARQ (**A**utomatic **R**epeat re**Q**uest). Αυτή χρησιμοποιείται στην ανταλλαγή αρχείων στους υπολογιστές όπου κάθε bit του μηνύματος είναι σημαντικό και η επανάληψη του μηνύματος εύκολη. Μία δεύτερη δυνατότητα είναι η απόρριψη του εσφαλμένου μηνύματος και στην συνέχεια επεξεργασία των επόμενων μηνυμάτων. Αυτή μπορεί να χρησιμοποιηθεί στην κινητή τηλεφωνία όπου είναι προτιμότερο να υπάρχει μια μικρή διακοπή στην ομιλία παρά καθυστέρηση λόγω της επανάληψης της μετάδοσης του μηνύματος.

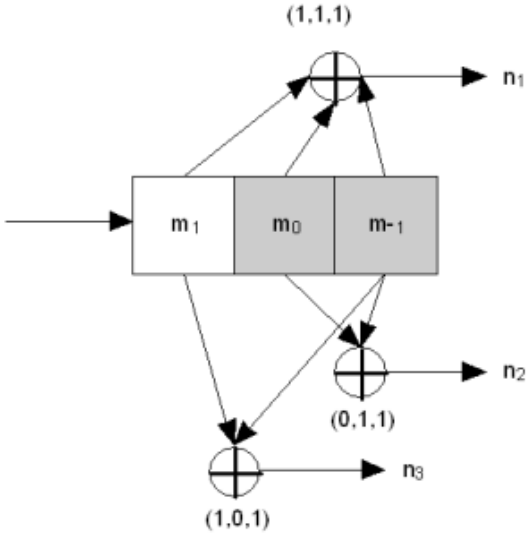
Υπάρχουν εφαρμογές όπου δεν είναι δυνατή η επανάληψη της εκπομπής του μηνύματος αλλά ταυτόχρονα είναι σημαντικό το μήνυμα να παραληφθεί σωστά. Ένα παράδειγμα είναι η ψηφιακή τηλεόραση. Η συσκευή της τηλεόρασης μπορεί να λάβει το σήμα, αλλά δεν μπορεί να ζητήσει από τον σταθμό την επανάληψη της εκπομπής. Η επικοινωνία δεν είναι αμφίδρομη. Άλλο παράδειγμα είναι η αποστολή δεδομένων από τα διαστημόπλοια που ταξιδεύουν στα άκρα του ηλιακού μας συστήματος. Το σήμα από αυτές τις αποστάσεις χρειάζεται αρκετά λεπτά για να φθάσει στην Γη οπότε δεν είναι δυνατόν να υπάρχει επιβεβαίωση για κάθε μπλοκ δεδομένων που στέλνεται, ούτε επανάληψη της αποστολής. Άλλωστε τα σήματα αυτά εκτίθενται σε ένα ιδιαίτερα θορυβώδες περιβάλλον και η επανάληψη της αποστολής μπορεί και πάλι να αποδώσει εσφαλμένα δεδομένα. Αυτή η επανάληψη θα μείωνε σημαντικά τον ρυθμό μετάδοσης χωρίς να αποδώσει σωστά δεδομένα.

Η λύση σε αυτές τις περιπτώσεις είναι η λεγόμενη FEC (**F**orward **E**rror **C**orrection). Ένας αρκετά μεγάλος αριθμός πλεοναζόντων bit προστίθενται στα δεδομένα που στέλνονται έτσι ώστε να είναι μπορεί να γίνει όχι μόνο ο εντοπισμός της ύπαρξης λαθών αλλά και η διόρθωσή τους. Ο αριθμός των πλεοναζόντων bit δεν είναι σταθερός σε κάθε εφαρμογή. Στον κωδικοποιητή «turbo code» που χρησιμοποιήθηκε σ' αυτή την εργασία ο λόγος είναι 1 προς 3 (για κάθε bit δεδομένων προστίθενται 2 bit ελέγχου οπότε στέλνονται 3 bit). Για αυξημένη αξιοπιστία υπάρχει δυνατότητα αποστολής επιπλέον bit με λόγο 1 προς 5. Όσο περισσότερα πλεονάζοντα bit χρησιμοποιούμε τόσο περισσότερο ανθεκτικά σε έκθεση σε θόρυβο θα είναι τα δεδομένα.

Ο «TURBO CODE» προτάθηκε για πρώτη φορά το 1993 από τους Berrou, Glavieux και Thitimajshima. Η απόδοσή του είναι τέτοια που ο ρυθμός εσφαλμένων δεδομένων (bit error rate) μέσα από ένα κανάλι που υπόκειται σε θόρυβο πλησιάζει το θεωρητικό όριο που περιέγραψε το 1948 ο Shannon. Η κωδικοποίηση και αποκωδικοποίηση που πρότεινε ο Berrou χρησιμοποιεί επιμέρους συστατικά τμήματα που προϋπήρχαν. Απαιτεί όμως πολύπλοκους υπολογισμούς ιδιαίτερα στην πλευρά του αποκωδικοποιητή. Αυτός είναι και ο λόγος που δεν χρησιμοποιήθηκαν αμέσως.

8.2.1. Περιγραφή του Turbo Code

Μπορούμε να χαρακτηρίσουμε τον «Turbo Code» σαν «μία αλληλουχία δύο παράλληλων ανατροφοδοτούμενων συστηματικών συνελικτικών κωδικοποιητών» (a Parallel concatenation of two recursive systematic convolutional codes). Τα επιμέρους στοιχεία του περιγράφονται και επεξηγούνται παρακάτω.



Εικόνα 56 Μη-Ανατροφοδοτούμενος Μη-Συστηματικός Συνελικτικός Κωδικοποιητής

Για την συνελικτική κωδικοποίηση χρειάζονται *k* καταχωρητές του ενός bit. Το πλήθος καταχωρητών *k* ονομάζεται «constraint length». Αρχικά όλοι οι καταχωρητές έχουν την τιμή «0». Ο κωδικοποιητής έχει *n* «modulo-2 αθροιστές». (Οι modulo-2 αθροιστές ουσιαστικά κάνουν XOR στις εισόδους τους και έχουν ένα bit σαν έξοδο). Στον κωδικοποιητή της εικόνας 56 υπάρχουν τρεις αθροιστές και για κάθε αθροιστή υπάρχει ένα πολυώνυμο που καθορίζει την έξοδο. Αυτά είναι: $G_1=(1,1,1)$, $G_2=(0,1,1)$, $G_3=(1,0,1)$. Το bit εισόδου τροφοδοτεί τον m_1 καταχωρητή. Χρησιμοποιώντας τα πολυώνυμα και τα τρία bit των καταχωρητών, τρία bit παράγονται και εξάγονται, τα n_1, n_2, n_3.

$n_1=m_1+m_0+m_{-1}$

$n_2=m_0+m_{-1}$

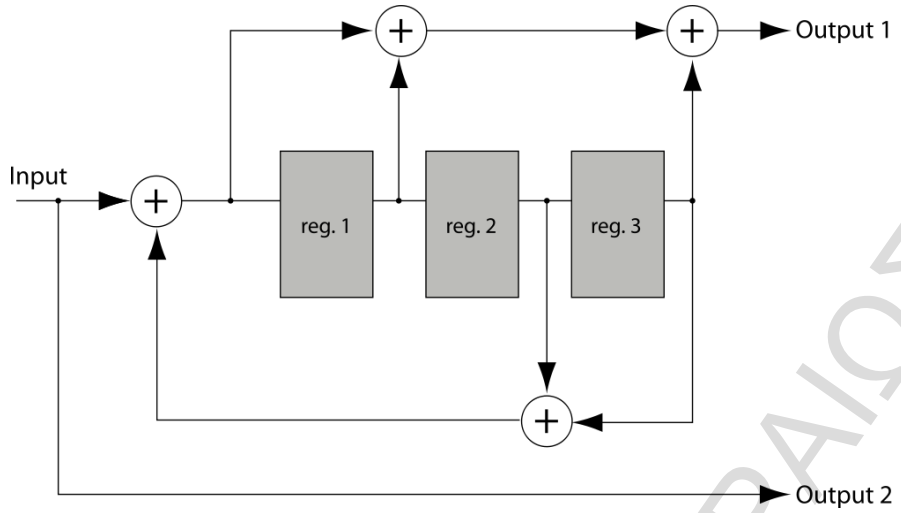
$n_3=m_1+m_{-1}$

Κατόπιν τα bit στους καταχωρητές ολισθαίνουν προς τα δεξιά και στην αριστερή θέση εισάγεται το νέο bit του σήματος εισόδου. Όταν τα bit από το μπλοκ δεδομένων εισόδου τελειώσουν τότε εισάγονται μηδενικά bit μέχρις ότου όλοι οι καταχωρητές φθάσουν στην τιμή «0».

Ο κωδικοποιητής αυτός παράγει τρία bit για κάθε bit εισόδου. Έχει δηλαδή λόγο (rate) 1/3.

8.2.2. Συστηματικός Ανατροφοδοτούμενος Συνελικτικός κωδικοποιητής

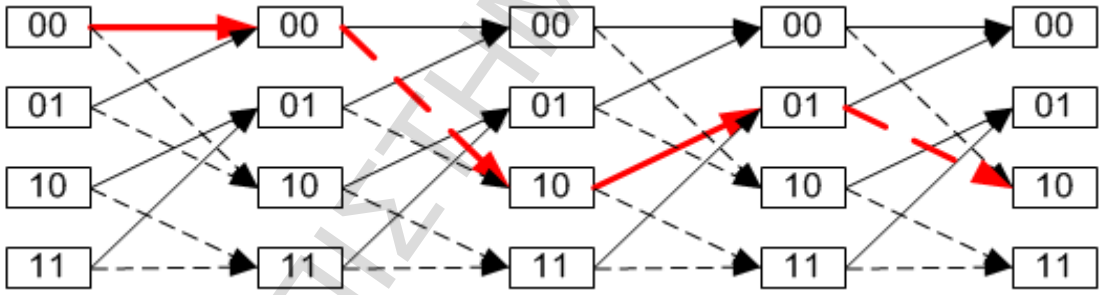
Όταν σε έναν κωδικοποιητή η έξοδος επιστρέφει στην είσοδο, τότε ο κωδικοποιητής χαρακτηρίζεται ως ανατροφοδουτούμενος (recursive). Εάν το σήμα εισόδου εμφανίζεται αυτούσιο στην έξοδο τότε ο κωδικοποιητής χαρακτηρίζεται ως συστηματικός (systematic). Στον κωδικοποιητή της εικόνας 56 δεν υπάρχει ανατροφοδότηση (non recursive) και το σήμα εισόδου δεν εμφανίζεται σε καμία έξοδο (non systematic). Στην εικόνα 57 φαίνεται ένας κωδικοποιητής όπου υπάρχει ανατροφοδότηση (recursive) και η είσοδος εμφανίζεται αυτούσια σε μία έξοδο (systematic) και έχει rate 1/2.



Εικόνα 57 Ανατροφοδοτούμενος Συστηματικός Συνελικτικός Κωδικοποιητής

8.2.3. Διάγραμμα Trellis

Ένας συνελικτικός κωδικοποιητής μπορεί να θεωρηθεί ως μια μηχανή καταστάσεων (finite state machine). Το πλήθος n των bit του κωδικοποιητή καθορίζει το πλήθος των καταστάσεων της μηχανής σε 2^n καταστάσεις. Έστω ότι ο κωδικοποιητής της εικόνας 56 έχει την τιμή «1» στο bit m_0 και την τιμή «0» στο bit m_{-1}. Το bit m_1 στην πραγματικότητα δεν είναι καταχωρητής αλλά αναπαριστά την τιμή του bit εισόδου. Τα δύο bit m_0 και m_{-1} καθορίζουν την κατάσταση «01». Τώρα ανάλογα με το bit εισόδου η επόμενη κατάσταση μπορεί να είναι είτε «00» είτε «10». Δεν είναι δυνατό η επόμενη κατάσταση να είναι «01» ή «11». Όλες οι δυνατές μεταβάσεις φαίνονται στο διάγραμμα trellis που ακολουθεί (εικόνα 58).



Εικόνα 58 Διάγραμμα Trellis

Μία ακολουθία από bit εισόδου μπορεί να παρουσιασθεί σαν μια διαδρομή στο διάγραμμα trellis. Στο διάγραμμα trellis φαίνεται με συνεχόμενη μαύρη γραμμή η μετάβαση της κατάστασης εάν το bit εισόδου είναι «0» και με διακεκομμένη μαύρη γραμμή η μετάβαση εάν το bit εισόδου είναι «1». Με κόκκινο χρώμα παρουσιάζεται η διαδρομή της ακολουθίας των bit «0101».

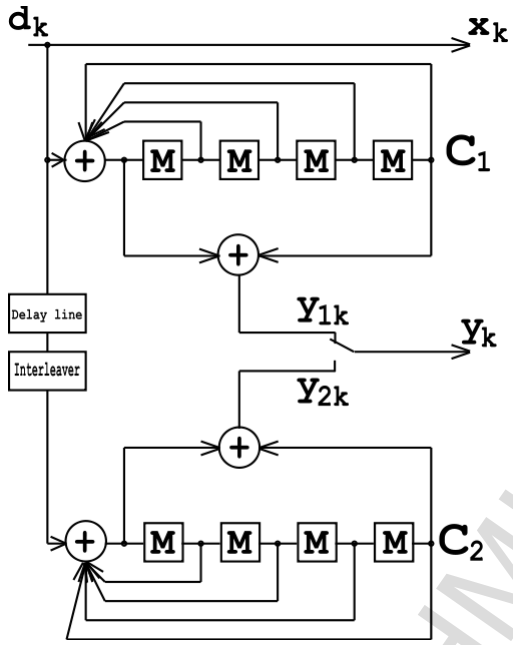
Το διάγραμμα trellis είναι γνωστό και στον αποκωδικοποιητή. Όταν ληφθεί μια ακολουθία από bit στον αποκωδικοποιητή τότε αυτή θα πρέπει να ταιριάζει στο διάγραμμα trellis. Διαφορετικά, θα έχει συμβεί κάποιο σφάλμα και θα πρέπει να διαλέξουμε την πλησιέστερη σωστή διαδρομή. Αυτή η γνώση των δυνατών διαδρομών αξιοποιείται κατά την αποκωδικοποίηση.

8.2.4. Ο κωδικοποιητής TURBO ENCODER

Ο Turbo encoder χρησιμοποιεί δύο όμοιους συνελικτικούς κωδικοποιητές συνδεμένους όπως φαίνεται στην εικόνα 59. Ο πρώτος κωδικοποιητής οδηγείται απ' ευθείας από το σήμα εισόδου ενώ ο δεύτερος οδηγείται διαμέσου ενός interleaver. Ο σκοπός του interleaver είναι να

αναδιατάξει την ακολουθία των bit μέσα στο πακέτο δεδομένων. Έτσι αν κάποια χρονική στιγμή επηρεαστεί το σήμα του πρώτου κωδικοποιητή από θόρυβο, στο σήμα του δεύτερου κωδικοποιητή θα επηρεαστούν κάποια διαφορετικά bit του πακέτου.

Μπορούμε να φανταστούμε τον interleaver σαν ένα πίνακα δύο διαστάσεων n,m με μέγεθος (n επί m) ίσο με το μέγεθος του μπλοκ. Υπάρχουν διάφοροι τρόποι λειτουργίας του interleaver. Ένας τρόπος είναι ο «row-column» όπου τα bit του πακέτου γράφονται σε ένα πίνακα σειρά-σειρά και κατόπιν διαβάζονται στήλη-στήλη. Άλλος τρόπος είναι ο «odd-even» όπου διαβάζονται πρώτα οι μονές γραμμές και κατόπιν οι ζυγές. Ένας τρίτος τρόπος είναι ο «helical» όπου τα bit διαβάζονται διαγώνια.



Εικόνα 59 Turbo Encoder

Στους συνελικτικούς κωδικοποιητές αυξάνουμε το πλήθος *k* των bit του constraint length προκειμένου να βελτιωθεί η απόδοσή τους (μπορεί να φθάσει και τα 16 bit). Στον turbo code, το *k* διατηρεί μια μικρή τιμή (τυπικά *k*=4) αλλά επιτυγχάνει καλή απόδοση εκτελώντας επαναλήψεις κατά την αποκωδικοποίηση.

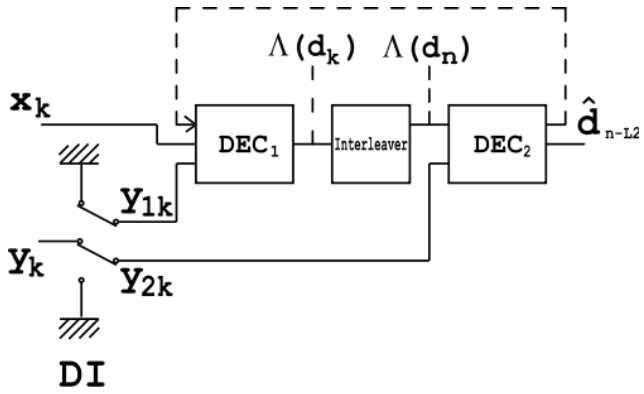
8.2.5. Η Τεχνική Puncturing

Το puncturing είναι μια τεχνική με τη οποία μπορεί να αυξηθεί ο ρυθμός του κωδικοποιητή. Χρησιμοποιώντας έναν πολυπλέκτη στις εξόδους Y1k και Y2k (εικόνα 59) ο ρυθμός ανεβαίνει από το 1/3 στο 1/2. Με αυτόν τον τρόπο επιλέγονται τα μονά bit από τον ένα κωδικοποιητή και τα ζυγά bit από τον άλλο. Σε πιο πολύπλοκα συστήματα μπορούν να χρησιμοποιηθούν πίνακες βάσει των οποίων θα αφαιρούνται bit από την έξοδο. Η σημαντικότερη εφαρμογή του puncturing είναι η ανόμοια κατανομή της προστασίας από σφάλματα ανάμεσα στα σημαντικότερα και τα λιγότερα σημαντικά bit των δεδομένων.

8.2.6. Ο αποκωδικοποιητής TURBO DECODER

Ο αποκωδικοποιητής έχει παρόμοια κατασκευή με τον κωδικοποιητή. Δύο όμοιοι αποκωδικοποιητές συνδέονται σε σειρά, και ένας interleaver αναλαμβάνει να επαναφέρει τα bit του πακέτου δεδομένων στην αρχική τους σειρά. Σε αντίθεση με προγενέστερους «hard decision» αποκωδικοποιητές όπως ο viterbi, ο TURBO αποκωδικοποιητής είναι «soft decision».

Δηλαδή η έξοδός $\Lambda(d_k)$ του αποκωδικοποιητή δεν παίρνει τις δυαδικές τιμές «0» ή «1» αλλά είναι ο λογάριθμος της πιθανότητας το bit εισόδου να είναι «0» ή «1».



Εικόνα 60 Turbo Decoder

Η πιθανότητα αυτή ονομάζεται LLR (Log Likelihood Ratio - λογάριθμος λόγου πιθανοτήτων) $\Lambda(d_k)= \log\frac{p(d_k=1)}{p(d_k=0)}$

Εάν ο αποκωδικοποιητής αναπαριστά το LLR σαν προσημασμένο αριθμό των 8 bit τότε:

$\Lambda(d_k)$= -127 σημαίνει «σίγουρα 0»

$\Lambda(d_k)$= -100 σημαίνει «πιθανότατα 0»

$\Lambda(d_k)$= 0 σημαίνει «μπορεί να είναι 1 ή 0»

$\Lambda(d_k)$= 100 σημαίνει «πιθανότατα 1»

$\Lambda(d_k)$= 127 σημαίνει «σίγουρα 1»

Κατά την αποκωδικοποίηση η εκτίμηση του πρώτου αποκωδικοποιητή οδηγείται στον δεύτερο αποκωδικοποιητή όπου λαμβάνεται υπ' όψη στην εκτίμηση του LLR του bit μαζί με το σήμα εισόδου.

Ένας τρίτος παράγοντας που συμπεριλαμβάνεται στον υπολογισμό του LLR του bit είναι πιθανότητα για την τιμή των bit που εξάγεται από το διάγραμμα trellis.

Η εκτίμηση του δεύτερου αποκωδικοποιητή οδηγείται ξανά πίσω στον πρώτο αποκωδικοποιητή όπου τώρα κάνει βελτιωμένες εκτιμήσεις λαμβάνοντας υπ' όψη τις πιθανότητες που του δόθηκαν.

Η παραπάνω διαδικασία υλοποιείται σε ένα τμήμα (module) που ονομάζεται SISO - Soft Input Soft Output.

Ακολουθεί επανάληψη της παραπάνω διαδικασίας είτε για προκαθορισμένο αριθμό επαναλήψεων είτε μέχρι οι έξοδοι των δύο αποκωδικοποιητών να παραμείνουν ίδιες για ορισμένο αριθμό επαναλήψεων. Ο αριθμός των επαναλήψεων είναι συνήθως 3-6 αλλά μπορεί να φθάσουν και τις 16 σε εφαρμογές που το κανάλι επικοινωνίας εκτίθεται σε υψηλό επίπεδο θορύβου. Η δυνατότητα να τερματισθούν οι επαναλήψεις όταν εκτιμηθεί ότι υπάρχει ασφαλής εκτίμηση λέγεται «fast termination» (γρήγορος τερματισμός) και χρησιμοποιείται για να επιταχυνθεί η διαδικασία της αποκωδικοποίησης. Όταν η εκτίμηση για την τιμή των bit ενός πακέτου παραμένει αναλλοίωτη για τόσες επαναλήψεις όσες το κατώφλι που έχουμε θέσει (fast termination threshold) τότε οι επαναλήψεις τερματίζονται και τα υπάρχοντα δεδομένα εξάγονται. Ο γρήγορος τερματισμός δεν εγγυάται την ορθότητα των δεδομένων. Είναι ένας συμβιβασμός ανάμεσα στην ταχύτητα και στην ακρίβεια.

Συνοψίζοντας, για να παρθεί απόφαση για την τιμή του κάθε bit λαμβάνονται υπ' όψη τρεις τιμές:

Α) Η τιμή του σήματος που έχει εξαχθεί από το κανάλι επικοινωνίας (extrinsic LLR).

Β) Η LLR πιθανότητα του bit που εκτίμησε ο έτερος αποκωδικοποιητής .

Γ) Η πιθανότητα LLR που προκύπτει από το διάγραμμα trellis του αποκωδικοποιητή. (a priori probability - APP)

Είναι φανερό πως απαιτούνται υπολογισμοί σημαντικής πολυπλοκότητας. Για κάθε bit πρέπει να υπολογισθεί ο λογάριθμος ενός αθροίσματος από λογάριθμους. Αυτό πρέπει να γίνει σε κάθε έναν από τους δύο αποκωδικοποιητές και για κάθε επανάληψη. Στην πράξη, στις hardware υλοποιήσεις του turbo decoder γίνονται απλοποιήσεις, παραδοχές και χρήση πινάκων (look up tables) ώστε η πολυπλοκότητα να κατέβει σε ρεαλιστικό επίπεδο.

Υπάρχουν πολλές παράμετροι που επηρεάζουν την απόδοση του turbo code. Αυτές είναι :

- Ο λόγος (rate) των bit του σήματος εισόδου/bit εξόδου του κωδικοποιητή. Αυξάνει τον ρυθμό μετάδοσης προκειμένου να βελτιώσει την απόδοση.
- Το πλήθος των καταχωρητών *k* (constraint length) του ζεύγους κωδικοποιητή αποκωδικοποιητή. Αυξάνει την πολυπλοκότητα προκειμένου να βελτιώσει την απόδοση.
- Ο τρόπος λειτουργίας του interleaver.
- Το μέγεθος του μπλοκ δεδομένων.
- Το Puncturing. Αυξάνει τον ρυθμό μετάδοσης παραλείποντας κάποια bit, βάση ενός συγκεκριμένου μοτίβου. Μειώνει την απόδοση του κωδικοποιητή.
- Η σχέση του σήματος προς τον θόρυβο.

Η άλγεβρα του turbo code παρουσιάζεται αναλυτικά στο «Fundamentals of Turbo Codes» του Bernard Sklar.

Το μεγάλο πλήθος παραμέτρων που επηρεάζουν το Bit Error Rate κατά την μετάδοση κάνουν πολύ δύσκολο τον υπολογισμό του με αναλυτικές μεθόδους. Μία λύση είναι η προσομοίωση του συστήματος που θα αποδώσει μετρήσεις πολύ κοντά στις πραγματικές τιμές. Ο μεγάλος αριθμός των bit που πρέπει να επεξεργασθούν (της τάξης των εκατοντάδων εκατομμυρίων) κάνει την προσομοίωση με σκοπό την εκτίμηση του BER μια ιδιαίτερα χρονοβόρα διαδικασία. Η προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop μπορεί να δώσει λύση σ' αυτό το πρόβλημα.

9. Μετρήσεις Bit Error Rate

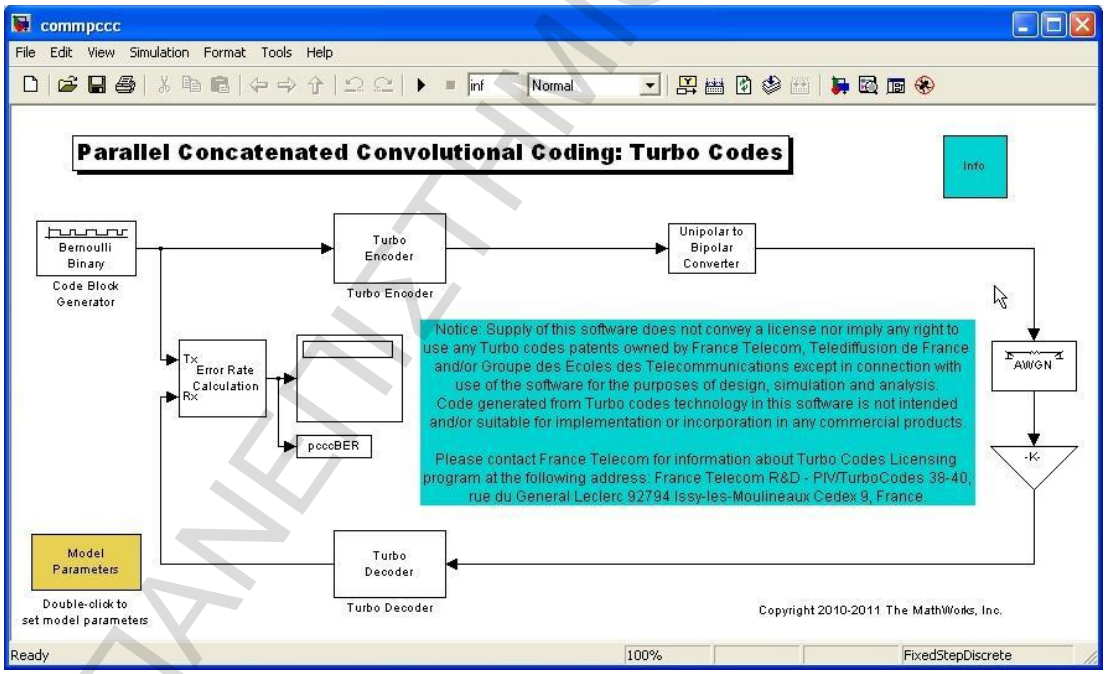
Η εκτίμηση του BER (Bit Error Rate) σε ένα σύστημα όπου τα δεδομένα κωδικοποιούνται, εκτίθενται σε θόρυβο και κατόπιν αποκωδικοποιούνται είναι πολύ δύσκολο να γίνει με αναλυτικές μεθόδους. Υπάρχουν πάρα πολλές παράμετροι που επηρεάζουν το BER όπως το μέγεθος του μπλοκ, ο λόγος (rate) του κωδικοποιητή, το επίπεδο θορύβου και ο αριθμός των επαναλήψεων στον αποκωδικοποιητή για να αναφέρουμε τις κυριότερες. Η πολυπλοκότητα των υπολογισμών είναι τέτοια που μας αποτρέπει από την χρήση αναλυτικών μεθόδων υπολογισμού και μας ωθεί προς την εκτίμηση του BER με προσομοίωση του συστήματος. Στην αρχή του κεφαλαίου αυτού παρουσιάζεται το μοντέλο προσομοίωσης του συστήματος κωδικοποίηση-μετάδοση-αποκωδικοποίηση με χρήση του Simulink.

Όπως παρουσιάστηκε στο προηγούμενο κεφάλαιο, ο κωδικοποιητής – αποκωδικοποιητής «TURBO CODE» είναι ιδιαίτερα απαιτητικός σε πλήθος υπολογισμών. Για κάθε bit που μεταδίδεται χρειάζεται ένας αρκετά μεγάλος αριθμός υπολογισμών στους οποίους περιλαμβάνονται και πολύπλοκες πράξεις όπως υπολογισμός λογάριθμων. Παράλληλα για την εκτίμηση του BER με προσομοίωση χρειάζεται η επεξεργασία ενός μεγάλου πλήθους bit, της τάξης των εκατομμυρίων. Το μεγάλο πλήθος των υπολογισμών και η πολυπλοκότητά τους κάνει την εκτίμηση του BER με προσομοίωση με software μια ιδιαίτερα χρονοβόρα διαδικασία.

Η λύση στο πρόβλημα είναι η τεχνική FPGA-In-the-Loop. Υλοποιώντας τμήματα του μοντέλου προσομοίωσης σε hardware, η διαδικασία μέτρησης του BER με προσομοίωση επιταχύνεται σημαντικά. Το hardware μοντέλο που χρησιμοποιήθηκε για την προσομοίωση με την τεχνική FPGA-In-the-Loop παρουσιάζεται στο δεύτερο μέρος του κεφαλαίου αυτού.

9.1. Προσομοίωση με Software

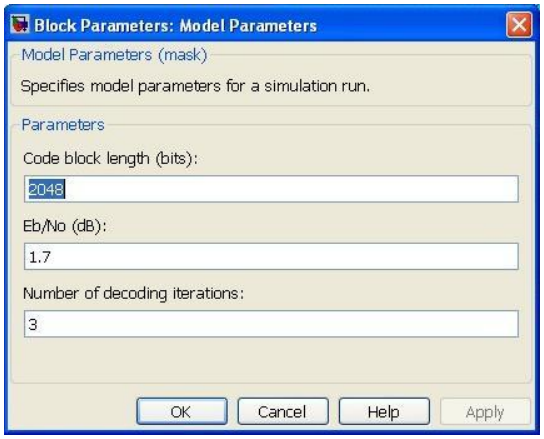
Το μοντέλο του Simulink με το όνομα «commpccc» είναι μια εφαρμογή του turbo codec.



Εικόνα 61 Turbo Codec

Το σήμα εισόδου παράγεται από μια γεννήτρια Bernoulli. Η γεννήτρια αυτή παράγει δυαδικούς αριθμούς με κατανομή Bernoulli. Δέχεται την παράμετρο p και παράγει «μηδέν» με πιθανότητα p και «ένα» με πιθανότητα (1-p). Η κατανομή Bernoulli έχει μέση τιμή (1-p) και διακύμανση (variance) ίση με p(1-p).

Μετά την γεννήτρια δεδομένων ακολουθεί ο κωδικοποιητής «Turbo Encoder». Οι παράμετροί του κανονίζονται από το μπλοκ «Model Parameters».



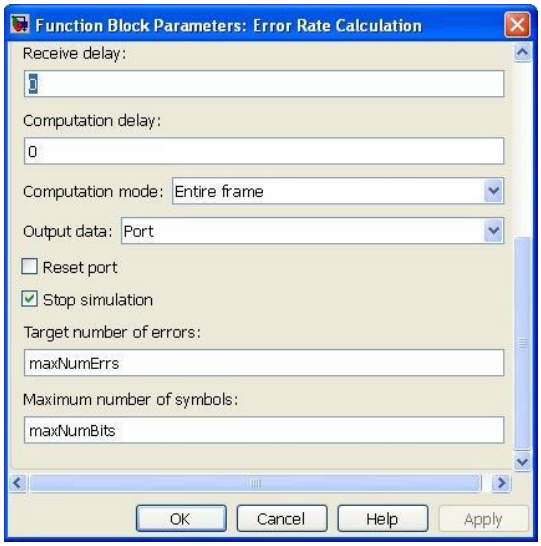
Εικόνα 62 Παράμετροι Codec

Το μέγεθος του μπλοκ μπορεί να πάρει οποιαδήποτε τιμή από αυτές που υποστηρίζονται από το πρότυπο 3gpp. Η παράμετρος Eb/No ρυθμίζει την σχέση σήματος προς θόρυβο του καναλιού. Τέλος ρυθμίζουμε τον πλήθος των επαναλήψεων κατά την αποκωδικοποίηση. Οι τιμές που φαίνονται στην εικόνα 62 είναι αυτές που χρησιμοποιήθηκαν κατά την σύγκριση της software και hardware co-simulation υλοποίησης της προσομοίωσης.

Ακολουθεί ο μετατροπέας Unipolar to Bipolar. Ο μετατροπέας αυτός μετατρέπει το δυαδικό σήμα εισόδου σε σήμα διπλής ακρίβειας. Κατόπιν το σήμα οδηγείται μέσα από ένα κανάλι που υπόκειται σε θόρυβο το οποίο προσομοιώνει τον θόρυβο που προστίθεται κατά την μετάδοση του σήματος. Το μπλοκ AWGN προσθέτει λευκό θόρυβο με κατανομή Gauss στο σήμα εισόδου. Η σχέση σήματος προς θόρυβο καθορίζεται από το μπλοκ «Model Parameters» (εικόνα 62).

Το σήμα αφού έχει εκτεθεί σε θόρυβο οδηγείται στον αποκωδικοποιητή. Τόσο στο κωδικοποιητή όσο και στον αποκωδικοποιητή το διάγραμμα trellis καθορίζεται κάνοντας διπλό κλικ πάνω στο εικονίδιό τους. Γίνεται χρήση της συνάρτησης poly2trellis που δημιουργεί το διάγραμμα trellis δίνοντας τις παραμέτρους. Οι τυπικές παράμετροι (4, [13 15], 13) που χρησιμοποιούνται αφορούν το constraint length, code generator (octal), και feedback connection (octal).

Το σήμα από τον αποκωδικοποιητή μαζί με το σήμα εισόδου οδηγούνται σε ένα μπλοκ «error rate calculation» (εικόνα 63). Το μπλοκ αυτό συγκρίνει τα δύο σήματα και δημιουργεί τα εξής στατιστικά στοιχεία: πλήθος bit που έχουν επεξεργασθεί, πλήθος λανθασμένων bit και ποσοστό λάθους κάνοντας διαίρεση των δύο παραπάνω αριθμών.



Εικόνα 63 Παράμετροι προσομοίωσης

Υπάρχει η δυνατότητα τα στοιχεία αυτά να χρησιμοποιηθούν για τον τερματισμό της προσομοίωσης. Όταν τσεκάρουμε το πλαίσιο «Stop Simulation» τότε η προσομοίωση τερματίζεται όταν το πλήθος των λανθασμένων bit φτάσει στο όριο που έχουμε ορίσει στο πλαίσιο «Target number of errors» ή όταν έχουν επεξεργασθεί συγκεκριμένο πλήθος bit όπως ορίζεται στο πλαίσιο «Maximum number of symbols». Στις μετρήσεις χρησιμοποιείται αυτή η μέθοδος τερματισμού της προσομοίωσης. Για μεγαλύτερη ευελιξία τα δύο αυτά όρια ορίζονται μέσω των μεταβλητών maxNumErrs και maxNumBits. Πρέπει πριν την εκτέλεση της προσομοίωσης να ορίσουμε τιμές για τις μεταβλητές αυτές. Δίνουμε στο παράθυρο εντολών του Matlab τις εντολές:

```
maxNumErrs=8000
```

και

```
maxNumBits=1000000
```

Τα στατιστικά του μπλοκ «error rate calculation» εμφανίζονται στην οθόνη με το μπλοκ «Display» και εξάγονται και στο workspace του Matlab με το μπλοκ «to workspace»

Ένας διαφορετικός τρόπος περιγραφής του software μοντέλου προσομοίωσης υπάρχει στα παραδείγματα του Matlab με το όνομα «commTurboCoding.m». Πρόκειται για το ίδιο μοντέλο αλλά η περιγραφή του γίνεται με script του Matlab και όχι με μπλοκ. Πλεονεκτεί έναντι του μοντέλου με μπλοκ στο ότι οι παράμετροι προσομοίωσης καθορίζονται μέσα στις γραμμές του κώδικα και είναι όλες μαζί στις γραμμές 22 έως 26 όπως φαίνεται παρακάτω.

```
numIter   = 6;          % Number of decoding iterations
blkLength = 2048;       % Block length
EbNo = [0.5 1];         % Eb/No values  to loop over
maxNumErrs  = 100;      % maximum number of errors per Eb/No value
maxNumBlks = 360;       % maximum number of blocks per Eb/No value
```

Επίσης στο τέλος της προσομοίωσης καλείται η λειτουργία γραφημάτων του Matlab, οπότε σχεδιάζεται και το αντίστοιχο γράφημα. Μειονεκτεί προφανώς στο ότι πρέπει κανείς να μελετήσει όλο τον κώδικα για να κατανοήσει την λειτουργία του, ενώ στο μοντέλο με μπλοκ η λειτουργία του κάθε μπλοκ στο μοντέλο είναι ξεκάθαρη.

Τα αποτελέσματα των μετρήσεων με αυτό το μοντέλο παρουσιάζονται παρακάτω, στο κεφάλαιο 9.6.2.

9.2. Προσομοίωση με την τεχνική Hardware In the Loop

Στην συνέχεια θα χρησιμοποιήσουμε την τεχνική FPGA-In-the-Loop για να επιταχύνουμε την διαδικασία μετρήσεων BER (Bit Error Rate).

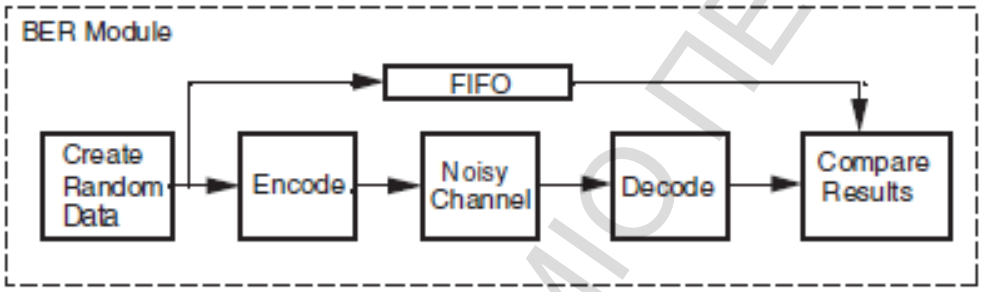
9.2.1. Περιγραφή της διαδικασίας μετρήσεων

Η διαδικασία για τις δοκιμές BER είναι απλή:

- Δημιουργούμε ένα τυχαίο σύνολο δεδομένων.
- Κωδικοποιούμε τα δεδομένα.
- Προσθέτουμε τυχαία σφάλματα ώστε να προσομοιώσουμε την μετάδοση του σήματος μέσα από ένα κανάλι που υπόκειται σε θόρυβο.
- Αποκωδικοποιούμε τα δεδομένα.
- Συγκρίνουμε με τα αρχικά δεδομένα και μετράμε τα σφάλματα που προέκυψαν.

Η υλοποίηση της παραπάνω διαδικασίας με software μπορεί να είναι εξαιρετικά χρονοβόρα. Γι' αυτό θα δημιουργήσουμε hardware εκδόσεις του κάθε μπλοκ ώστε να επιταχυνθεί η διαδικασία.

Το παρακάτω σχεδιάγραμμα δείχνει τα βήματα σε μορφή μπλοκ.



Εικόνα 64 Μπλοκ διάγραμμα δοκιμών BER

Το μπλοκ εισόδου χρειάζεται να δημιουργεί μπλοκ με τυχαία δεδομένα του ενός bit με μέγεθος από 40 έως 5114 bit σύμφωνα με το πρότυπο 3GPP. Ένας καταχωρητής ολίσθησης με ανάδραση (linear feedback shift register – LFSR) είναι κατάλληλος για αυτή την δουλειά. Ο καταχωρητής αυτού του τύπου διατίθεται από το περιβάλλον του system generator.

Ο κωδικοποιητής 3GPP διατίθεται από την Xilinx. Μία δοκιμαστική έκδοση εμπεριέχεται στο πακέτο της εφαρμογής xapp948.zip. Η δοκιμαστική έκδοση έχει περιορισμένο χρόνο λειτουργίας του αποκωδικοποιητή. Μετά από μερικές ώρες παύει να λειτουργεί και το σύστημα πρέπει να επανεκκινηθεί. Ο περιορισμός αυτός δεν επηρεάζει τις δοκιμές της εργασίας που διαρκούν μόλις μερικά λεπτά.

Το κανάλι που υπόκειται σε θόρυβο δημιουργείται από μπλοκ που διατίθενται στο system generator.

Ο αποκωδικοποιητής 3GPP επίσης διατίθεται από την Xilinx. Υπάρχει μια δοκιμαστική έκδοση μαζί με το πακέτο της εφαρμογής xapp948.zip. Ισχύουν οι ίδιοι περιορισμοί όπως και για τον κωδικοποιητή.

Η μνήμη FIFO διατίθεται από το περιβάλλον του system generator.

Το μπλοκ σύγκρισης έχει δημιουργηθεί από την Xilinx χρησιμοποιώντας μπλοκ του system generator.

Η λειτουργία του μπλοκ σύγκρισης είναι απλή. Όταν το αποκωδικοποιημένο σήμα είναι διαθέσιμο, η έξοδος RDY του αποκωδικοποιητή παίρνει την τιμή HIGH. Το μπλοκ σύγκρισης περιμένει το σήμα RDY του αποκωδικοποιητή και τότε συγκρίνει τα αποκωδικοποιημένα δεδομένα με τα αρχικά. Ένας μετρητής απαριθμεί τα λάθη σε κάθε μπλοκ. Ένας δεύτερος μετρητής απαριθμεί το πλήθος των μπλοκ που είχαν ένα ή περισσότερα λάθη. Ένας τρίτος μετρητής απαριθμεί το συνολικό πλήθος των bit που έχουν μεταδοθεί. Με αυτούς τους τρεις μετρητές είναι δυνατό να υπολογισθεί το bit error rate – BER και το frame error rate – FER.

Στο πακέτο xapp948.zip υπάρχουν έτοιμα bitfiles για συγκεκριμένες πλακέτες FPGA. Υπάρχουν σε φακέλους που το όνομα του καθενός είναι το όνομα της πλακέτας στην οποία απευθύνεται. Τα ονόματα των φακέλων και οι αντίστοιχες πλακέτες αυτές είναι:

ml402 ethernet ML402 με σύνδεση Ethernet και XC4VSX35 FPGA.

wc2 Annapolis wildcard II με σύνδεση pcmcia και XC2V3000 FPGA

wc4 Annapolis wildcard 4 με σύνδεση pcmcia και XC4VSX35 FPGA

xdsp Nallatech Extreme DSP με σύνδεση pcmcia και pci και XC2V3000 FPGA

Το όνομα του αρχείου που περιέχεται στους φακέλους έχει τη μορφή : tcc_3gpp_v1_0_1r3_2i3_6m3_w32_scale.x86 και ακολουθεί τις παρακάτω συμβάσεις:

- tcc_3gpp_v1_0 = tcc 3gpp σχεδίαση, έκδοση 1.0
- 1r3 = λόγος (rate) 1/3
- 2i3 = χρησιμοποιεί στο σήμα εισόδου 2 bit για το ακέραιο μέρος του αριθμού και 3 bit για το δεκαδικό μέρος.
- 6m3 = χρησιμοποιεί για τους εσωτερικούς υπολογισμούς 6 bit για το ακέραιο μέρος του αριθμού και 3 bit για το δεκαδικό μέρος.
- scale MAX_SCALE = ο αλγόριθμος κωδικοποίησης είναι ο MAX_SCALE.
- .x86 = είναι η κατάληξη του αρχείου για την πλακέτα wildcard.

Αν η δική μας πλακέτα δεν είναι μία από τις προαναφερθείσες τότε θα πρέπει να ακολουθήσουμε την παρακάτω διαδικασία για να δημιουργήσουμε το κατάλληλο bitfile.

9.2.2. Προετοιμασία αρχείων

Χρειαζόμαστε το υλικό για το «Xilinx application 948». Μπορούμε να το κατεβάσουμε από το site της Xilinx στην διεύθυνση: «http://www.xilinx.com/support/index.htm». Στο πλαίσιο «Search Support» εισάγουμε την λέξη «xapp948» και στα αποτελέσματα που εμφανίζονται επιλέγουμε το αρχείο zip: «xapp948.zip [ZIP, 3851 KB]». Για να μας επιτραπεί πρόσβαση στο αρχείο θα πρέπει να έχουμε δημιουργήσει λογαριασμό στην Xilinx και να έχουμε συνδεθεί με την επιλογή «sign in». Αν δεν έχουμε λογαριασμό μπορούμε να δημιουργήσουμε καταχωρώντας τα στοιχεία μας στο site επιλέγοντας «sign in» και κατόπιν «create account».

Όλα τα αρχεία είναι πακεταρισμένα σε ένα αρχείο zip με το όνομα XAPP948.ZIP. Αποσυμπιέζουμε όλα τα αρχεία σε έναν φάκελο. Στην εργασία αυτή χρησιμοποιήσαμε το όνομα φακέλου «3gpp» τον οποίο δημιουργήσαμε στην διαδρομή «C:\Xilinx\14.2\ISE_DS\ISE\sysgen\3gpp». Δεν είναι απαραίτητο να βρίσκεται σε αυτήν ακριβώς την διαδρομή. Απλά σε αυτό το σημείο υπάρχουν και άλλα παραδείγματα της Xilinx. Θα πρέπει όμως να προσέξουμε ότι το System Generator δεν δέχεται κενά μέσα στα ονόματα αρχείων και φακέλων. Τώρα ξεκινάμε το «System Generator». Ορίζουμε σαν τρέχοντα φάκελο στο πλαίσιο «current folder» το «C:\Xilinx\14.2\ISE_DS\ISE\sysgen\3gpp» ή την αντίστοιχη διαδρομή για τον φάκελο που έχουμε ορίσει.

Ανάμεσα στα αρχεία της εφαρμογής υπάρχει και ο φάκελος «results» με αρχεία τύπου text. Σε αυτά είναι αποθηκευμένα τα αποτελέσματα προηγούμενων δοκιμών.

Το όνομα των αρχείων αυτών έχει την εξής ερμηνεία:

Bs40 = μέγεθος μπλοκ ίσο με 40.

i3 = 3 επαναλήψεις.

ft3 = όριο για γρήγορο τερματισμό ίσο με 3.

rm0 = όχι rate matching. Είναι η μοναδική επιλογή στην παρούσα έκδοση. Ίσως επόμενες εκδόσεις να υποστηρίζουν rate matching (puncturing).

Οι υπόλοιπες παράμετροι ακολουθούν τις συμβάσεις που περιγράφονται στο όνομα του αρχείου bitfile.

Αν δεν αλλαχθεί το προεπιλεγμένο όνομα αποθήκευσης τα αποτελέσματα αυτά θα διαγραφούν και στην θέση τους θα αποθηκευθούν αυτά της νέας δοκιμής. Μπορούμε λοιπόν προτού προχωρήσουμε, να δούμε τα αποτελέσματα αυτά. Έτσι θα μπορούμε να ελέγξουμε ότι

η προσομοίωση που τρέχουμε στην δικιά μας πλακέτα βγάζει σωστά αποτελέσματα. Δεν χρειάζεται καμία προετοιμασία για να δούμε τα γραφήματα των αποθηκευμένων αποτελεσμάτων. Απλά εκτελούμε το script «plot_script.m» κα τα γραφήματα των αποτελεσμάτων εμφανίζονται στην οθόνη.

Μεταξύ των αρχείων της εφαρμογής υπάρχουν έτοιμα bitfile για πέντε διαφορετικές πλακέτες. Για να δημιουργήσουμε και να προσθέσουμε το bitfile για την δική μας πλακέτα ακολουθούμε την παρακάτω διαδικασία:

Κάνουμε διπλό κλικ στο μοντέλο «tcc_3gpp_v1_0_demo_synth.mdl» στο παράθυρο «Current folder». Αυτό είναι το μοντέλο που θα χρησιμοποιηθεί για την σύνθεση.

Πρώτα ορίζουμε το FPGA για το οποίο θέλουμε να δημιουργήσουμε το bitfile. Στο παράθυρο «Model Browser» που άνοιξε επιλέγουμε το «tcc_3gpp_v1_0_demo_synth» κάνοντας κλικ επάνω του. Κάνουμε διπλό κλικ στο σύμβολο του system generator και από το πλαίσιο Compilation επιλέγουμε hardware co-simulation και κατόπιν Virtex5 (ή όπως αλλιώς έχουμε ονομάσει την πλακέτα μας). Κλείνουμε αυτό το παράθυρο πατώντας «OK».

Τώρα ξεκινάμε την διαδικασία δημιουργίας του bitfile. Στο παράθυρο «Model Browser επιλέγουμε το «BER Module»» κάνοντας κλικ επάνω του. Κάνουμε διπλό κλικ στο σύμβολο του system generator και από το πλαίσιο Compilation επιλέγουμε hardware co-simulation και κατόπιν Virtex5. Πατάμε το κουμπί generate.

Εάν χρησιμοποιούμε ελληνικά windows θα μας εμφανιστεί το μήνυμα λάθους: «Failed to load library 'tcc_awgn_lib' referenced by 'tcc_3gpp_v1_0_demo_synth/BER Module/Channel with Rate Matching/Channel/multiple input channel/channel_1'». Αυτό οφείλεται στην κωδικοποίηση των χαρακτήρων στο μοντέλο. Μπορεί να ξεπεραστεί αλλάζοντας την κωδικοποίηση των χαρακτήρων, εισάγοντας στην κονσόλα του matlab τις παρακάτω εντολές:

```
bdclose all
slCharacterEncoding windows-1252
```

και ελέγχουμε ότι εκτελέστηκαν εισάγοντας:

```
slCharacterEncoding
```

Θα πρέπει να πάρουμε απάντηση:

```
ans =
windows-1252
```

Το system generator θα λειτουργήσει για 30-50 λεπτά ανάλογα με την ταχύτητα του υπολογιστή και στο τέλος θα μας ενημερώσει πως η διαδικασία ολοκληρώθηκε με επιτυχία. Ένα παράθυρο με όνομα «BER_Module_hwcosim_lib» θα ανοίξει με το μπλοκ του bitfile στο εσωτερικό του. Καλό είναι να μην κλείσουμε το παράθυρο αυτό. Κλείνουμε το παράθυρο του μοντέλου «tcc_3gpp_v1_0_demo_synth».

Τώρα πρέπει να αντιγράψουμε το bitfile που δημιουργήθηκε για την πλακέτα μας στον φάκελο όπου υπάρχουν τα bitfile των διαφόρων πλακετών. Είτε από την εξερεύνηση των Windows είτε από το παράθυρο «Current folder» δημιουργούμε μέσα στον φάκελο «sysgen_bit_files» όπου βρίσκονται τα έτοιμα bitfile της εφαρμογής, ένα φάκελο με το όνομα «Virtex5». Μέσα σ' αυτόν αντιγράφουμε το bitfile με όνομα «ber_module_cw.bit» που δημιουργήθηκε από το system generator και βρίσκεται μέσα στον φάκελο netlists.

Θα πρέπει τώρα να προσθέσουμε το bitfile αυτό στην βιβλιοθήκη της εφαρμογής. Κάνουμε διπλό κλικ στο «hwitl_lib.mdl» για να ανοίξουμε την βιβλιοθήκη. Για να επιτραπεί να κάνουμε την προσθήκη πρέπει πρώτα να την ξεκλειδώσουμε. Από το μενού edit επιλέγουμε Unlock Library. Καλό είναι σε αυτό το σημείο να επιλέξουμε από το μενού view της βιβλιοθήκης την επιλογή zoom out ώστε να βλέπουμε όλα τα υπάρχοντα μπλοκ και να έχουμε και κενό χώρο για να δημιουργήσουμε ένα ακόμη μπλοκ.

Από το παράθυρο της βιβλιοθήκης Library: hwitl_lib κάνουμε κλικ σε ένα υπάρχον μοντέλο και κρατώντας πατημένο το CTRL το σέρνουμε σε κενό χώρο στο παράθυρο για να δημιουργήσουμε ένα αντίγραφο. Κάνουμε κλικ στο όνομα στο κάτω μέρος του νέου μπλοκ και το μετονομάζουμε σε Virtex5.

Κάνουμε διπλό κλικ στο μπλοκ που μετονομάσαμε σε Virtex5 και στο εσωτερικό του βλέπουμε το μπλοκ «BER Module hwcosim». Αυτό πρέπει να αντικατασταθεί με το μπλοκ που δημιουργήσαμε. Κάνουμε κλικ επάνω του και πατάμε delete για να το διαγράψουμε. Στο παράθυρο επιβεβαίωσης που ανοίγει πατάμε «disable link» Οι συνδέσεις του μπλοκ παραμένουν σαν κόκκινες γραμμές. Από το παράθυρο «BER_Module_hwcosim_lib» που αφήσαμε ανοικτό προηγουμένως, κάνουμε κλικ πάνω στο μπλοκ και το σέρνουμε στην κενή θέση που δημιουργήσαμε στο παράθυρο της βιβλιοθήκης. Μετακινώντας το κατάλληλα και αλλάζοντας το μέγεθος του μπλοκ φροντίζουμε να συμπέσουν οι γραμμές σύνδεσης του μπλοκ με τις κόκκινες γραμμές. Μόλις ευθυγραμμιστούν θα συνδεθούν αυτόματα και το χρώμα των γραμμών θα γίνει μαύρο. Αν είχαμε κλείσει κατά λάθος το παράθυρο με το μπλοκ μπορούμε να το επαναφέρουμε κάνοντας διπλό κλικ στο «BER Module hwcosim.mdl» που έχει δημιουργηθεί μέσα στον φάκελο netlist.

Κάνουμε διπλό κλικ στο μπλοκ που συνδέσαμε και επιλέγουμε στο πλαίσιο clock source: Free running. Στο πλαίσιο Bitstream file εισάγουμε την συνάρτηση «select_bit_file(bit_file,hw_type,device)» (χωρίς τα εισαγωγικά) η οποία θα επιλέξει το όνομα του κατάλληλου bitfile για την πλακέτα μας. Κλείνουμε όλα τα παράθυρα αποθηκεύοντας.

Θα πρέπει να ενημερώσουμε τη συνάρτηση «select_bit_file.m» για την ύπαρξη του bitfile της πλακέτας μας. Ανοίγουμε το αρχείο με το ίδιο όνομα και εισάγουμε τις δύο παρακάτω γραμμές ανάμεσα στην γραμμή 48 και την γραμμή 49:

```
  elseif strcmp(type,'Virtex5')
  bit_file_name=['.\sysgen_bit_files\Virtex5\',name,'.x86']
```

Κάνουμε διπλό κλικ στο «hwitl_lib.mdl» για να ανοίξουμε την βιβλιοθήκη. Μέσα στο παράθυρο της βιβλιοθήκης κάνουμε διπλό κλικ στο template. Εμφανίζονται τα ονόματα των πλακετών για τις οποίες υπάρχουν διαθέσιμα bitfile. Υπάρχει και η πλακέτα μας μεταξύ των άλλων. Την ενεργοποιούμε επιλέγοντας το «τικ» στο αντίστοιχο κουτάκι. Αν μας εμφανιστεί μήνυμα λάθους «broken link» κάνουμε δεξί κλικ στο module της βιβλιοθήκης που έχει το πρόβλημα (θα έχει ανοίξει αυτόματα) και επιλέγουμε link options→Resolve link…

Τώρα θα πρέπει να επιλέξουμε την πλακέτα μας ως πλακέτα για τις δοκιμές. Κάνουμε διπλό κλικ στο «tcc_3gpp_v1_0_demo_hwitl.mdl» και ανοίγει το μοντέλο προσομοίωσης με χρήση της τεχνικής FPGA-In-the-Loop. Στο μπλοκ που βρίσκεται στο κέντρο κάνουμε δεξί κλικ και από την επιλογή «Block Choice» διαλέγω «Virtex5». Επίσης χρειάζεται να ορίσουμε το chip της πλακέτας μας. Στο εικονίδιο του «System Generator» κάνουμε διπλό κλικ και στο πλαίσιο «Part» επιλέγουμε Virtex5→xc5vlx110t→-1→ff1136. Πατάμε OK. Κλείνουμε το παράθυρο της βιβλιοθήκης αποθηκεύοντας τις αλλαγές.

Η επιλογή της πλακέτας που θα χρησιμοποιηθεί για την προσομοίωση γίνεται ορίζοντας τον κατάλληλο αριθμό μέσα στο script «demo_ctrl.m». Επιλέγουμε το αριθμό 7 για την Virtex5. Κάνουμε διπλό κλικ στο «demo_ctrl.m» και στην γραμμή 56 αλλάζουμε την επιλεγμένη πλακέτα σε 7:

```
    platform= 7; % vv=0-3
```

Αποθηκεύουμε τις αλλαγές και κλείνουμε τον editor.

Πρέπει τώρα να τροποποιήσουμε το script δοκιμών ώστε να λαμβάνει υπ' όψιν του την επιλογή 7 για την πλακέτα μας. Κάνουμε διπλό κλικ στο «run_tests.m» και ανάμεσα στις γραμμές 164 και 165 εισάγουμε τις παρακάτω εντολές:

```
elseif platform==7
hw_type='Virtex5'; pause_time=5; device='xc5vlx110t';
set_param('tcc_3gpp_v1_0_demo_hwitl/HWITL Config', 'BlockChoice', 'Virtex5')
```

Αποθηκεύουμε τις αλλαγές και κλείνουμε τον editor. Τώρα είμαστε έτοιμοι να τρέξουμε το script των δοκιμών.

Εκτελούμε το script «demo_ctrl.m».

9.2.3. Παράμετροι δοκιμών

Οι παράμετροι τις οποίες μπορούμε να αλλάξουμε χωρίς να χρειαστεί να αλλάξουμε τον κωδικοποιητή και τον αποκωδικοποιητή είναι :

- Το πλήθος των επαναλήψεων.
- Το κατώφλι τερματισμού των επαναλήψεων (fast termination threshold).
- Η ελάχιστη τιμή BER που απαιτείται προκειμένου να ολοκληρωθούν οι μετρήσεις που απαιτούνται για τον υπολογισμό ενός σημείου BER στο διάγραμμα.
- Ο μέγιστος αριθμός των bit που θα επεξεργασθούν προκειμένου να ολοκληρωθούν οι μετρήσεις που απαιτούνται για τον υπολογισμό ενός σημείου BER στο διάγραμμα.

Οι παράμετροι που δεν μπορούν να αλλάξουν από το software και απαιτούν νέα υλοποίηση του κωδικοποιητή και του αποκωδικοποιητή είναι :

- Το πλήθος των bit του σήματος εισόδου.
- Το πλήθος των bit που χρησιμοποιούνται εσωτερικά για τους υπολογισμούς.
- Η δυνατότητα τερματισμού των επαναλήψεων (fast termination).
- Προσθήκη προαιρετικών σημάτων όπως το sclr κ.α.

Οι παράμετροι που μπορούν να αλλαχτούν ορίζονται στο script demo_ctrl.m. Συγκεκριμένα οι αριθμοί των γραμμών και οι παράμετροι που αλλάζουν παρατίθενται παρακάτω:

```
73    max_bitcount    = 1e9;      % maximum number of bits per calculated point
74    max_errors      = 8000;     % maximum number of errors per calculated point
```

Αναφέρεται στον μέγιστο αριθμό των bit που θα επεξεργασθούν και στο μέγιστο αριθμό λαθών που θα σημειωθούν προκειμένου να υπολογιστεί ένα σημείο. Η προσομοίωση υπολογίζει συνεχώς νέα bit μέχρι μία από τις δύο παραπάνω συνθήκες να γίνει αληθής.

```
76    min_ber         = 1e-4;     % finish when BER betters this value
77    eb_n0_step      = 0.25;     % step between successive BER values
```

Η προσομοίωση υπολογίζει συνεχώς νέα σημεία αυξάνοντας κάθε φορά το λόγο σήματος προς θόρυβο eb/no κατά το βήμα που αναφέρεται στην γραμμή 77 μέχρι ο λόγος λαθών BER να γίνει μικρότερος από αυτόν που αναφέρεται στην γραμμή 76.

```
114   bs              = [ 378, 762];  % define the block size
115   iter            = [ 3,5];       % define the number of iterations
117   ft_vals         = [ 0,3 ];      % define the fast termination threshold value
```

Οι παραπάνω τιμές αναφέρονται στο μέγεθος του μπλοκ, στο πλήθος των επαναλήψεων του αποκωδικοποιητή και στο κατώφλι τερματισμού των επαναλήψεων. Η τιμή 0 σημαίνει ότι δεν χρησιμοποιείται η δυνατότητα τερματισμού ενώ η τιμή 3 σημαίνει ότι οι επαναλήψεις θα τερματισθούν όταν για 3 συνεχόμενες επαναλήψεις οι εκτιμήσεις για τα bit του μπλοκ δεδομένων δεν αλλάξουν. Όπου αναφέρονται δύο τιμές, θα γίνουν μετρήσεις και για τις δύο. Άρα με τις τιμές που εμφανίζονται πιο πάνω θα γίνουν 2X2X2=8 μετρήσεις.

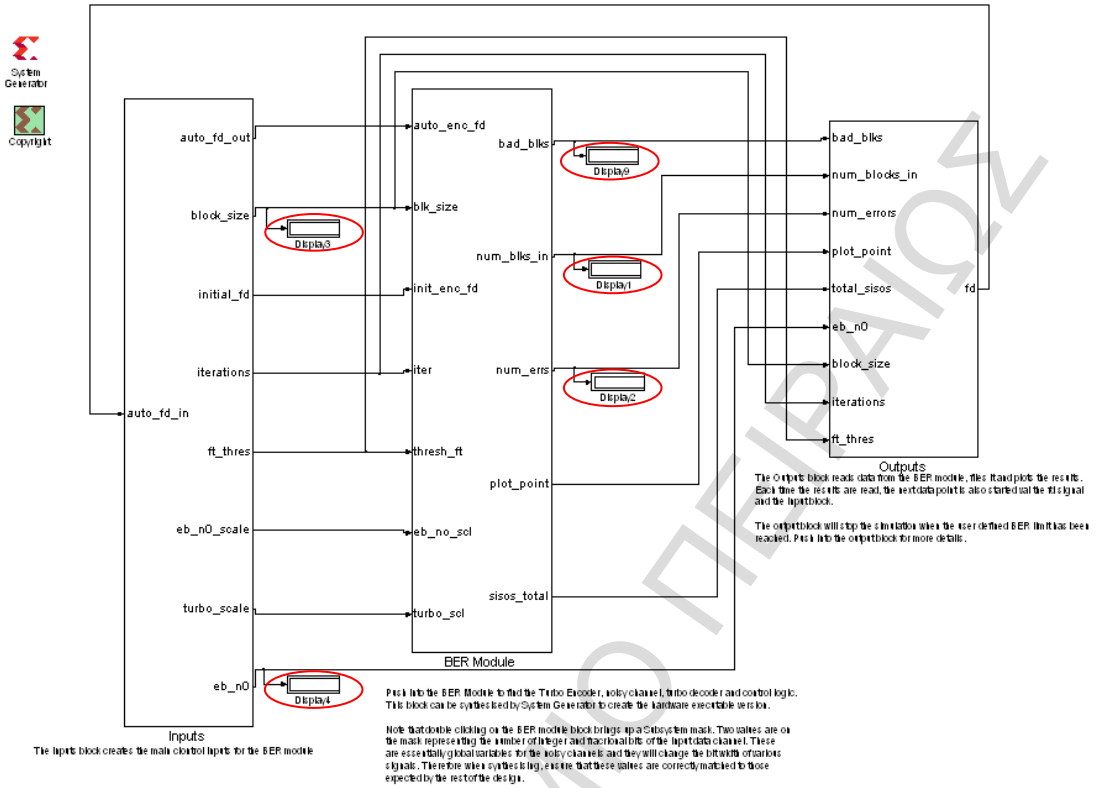
Οι παρακάτω παράμετροι επίσης αναφέρονται αλλά δεν αλλάζουν την προσομοίωση. Χρησιμοποιούνται μόνο για δοθούν κατάλληλοι τίτλοι στα διαγράμματα:

```
95    i_int           = [ 2 ];    % input integer bit width
96    i_frac          = [ 3 ];    % input fractional bit width
97    m_int           = [ 6 ];    % metric integer bit width
98    m_frac          = [ 3 ];    % metric fractional bit width
99    win             = [ 32 ];   %window size
```

Το i_int και το i_frac αφορούν το πλήθος των bit του σήματος εισόδου. Το m_int και το m_frac αφορούν το πλήθος των bit που χρησιμοποιούνται εσωτερικά για τους υπολογισμούς. Όταν η μνήμη RAM είναι περιορισμένη τότε γίνεται επεξεργασία πάνω σε ένα τμήμα του μπλοκ που λέγεται παράθυρο (window). Το win είναι το μέγεθος του παράθυρου πάνω στο οποίο γίνεται η αποκωδικοποίηση. Για να αλλάξουν οι παράμετροι αυτές πρέπει να γίνει νέα υλοποίηση του μπλοκ του κωδικοποιητή και του μπλοκ του αποκωδικοποιητή χρησιμοποιώντας το «core generator». Αν και το κάθε μπλοκ υλοποιείται ξεχωριστά είναι προφανές ότι θα πρέπει να έχουν τις ίδιες ακριβώς επιλογές παραμέτρων.

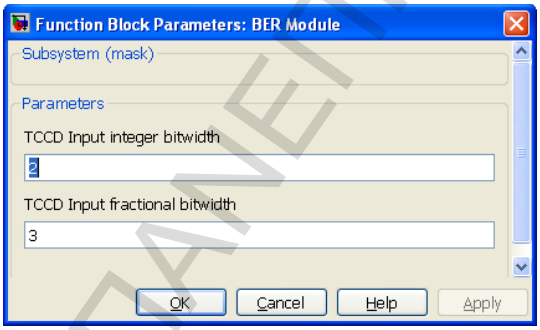
9.3. Περιγραφή του Hardware Μοντέλου

Το πλήρες μοντέλο υπάρχει με το όνομα «tcc_3gpp_v1_0_demo_synth.mdl».



Εικόνα 65 Πλήρες μοντέλο

Το μπλοκ με το όνομα «BER Module» έχει υλοποιηθεί αποκλειστικά με μπλοκ της Xilinx και είναι το αυτό που θα υλοποιηθεί σε hardware στο FPGA. Τα μπλοκ «Inputs» και «Outputs» αποτελούνται από στοιχεία του Simulink (όχι της Xilinx) και κάνουν την επικοινωνία ανάμεσα στο software (script του Matlab) και στο hardware. Με διπλό κλικ επάνω στο BER Module εμφανίζεται το παράθυρο παραμέτρων της εικόνας 66 στο οποίο μπορούμε να ορίσουμε το πλήθος των bit του καναλιού μετάδοσης. Οι τιμές αυτές επηρεάζουν ένα πλήθος από εξαρτήματα μέσα το κανάλι μετάδοσης. Προφανώς πρέπει να είναι ίδιες με αυτές του αποκωδικοποιητή.



Εικόνα 66 Παράμετροι καναλιού μετάδοσης

Τα σήματα που χρησιμοποιεί το μπλοκ εισόδου (αριστερά στην εικόνα 65) είναι τα παρακάτω:

- AUTO_FD_OUT – Automatic First Data. Κάθε φορά που έχει ολοκληρωθεί ο υπολογισμός ενός σημείου BER, το μπλοκ εξόδου σηματοδοτεί το ξεκίνημα του υπολογισμού ενός νέου σημείου μέσω του σήματος FD που οδηγεί την είσοδο αυτή.

- BLOCK_SIZE – Είναι το μέγεθος του μπλοκ που χρησιμοποιείται στην δοκιμή. Το πρότυπο 3GPP υποστηρίζει μέγεθος μπλοκ από 40 έως 5114 bit.
- INITIAL_FD - Το σήμα αυτό παράγεται από το software και είναι αυτό που ξεκινά την προσομοίωση.
- ITERATIONS – Είναι ο αριθμός των επαναλήψεων που θα γίνουν κατά την αποκωδικοποίηση.
- FT_THRESHOLD – Είναι το κατώφλι γρήγορου τερματισμού. Όταν οι εκτιμήσεις του αποκωδικοποιητή για τα bit του μπλοκ μείνουν σταθερές για τόσες επαναλήψεις όσες ορίζει αυτό το σήμα τότε οι επαναλήψεις σταματούν και ο αποκωδικοποιητής εξάγει τα δεδομένα. Αυτή η μέθοδος δεν εξασφαλίζει ότι τα δεδομένα είναι σωστά. Είναι ένας συμβιβασμός ανάμεσα στην ταχύτητα και στην ακρίβεια. Η λειτουργία γρήγορου τερματισμού έχει υλοποιηθεί στον αποκωδικοποιητή της εργασίας.
- EB_NO_SCALE – Είναι το επίπεδο θορύβου που θα τροφοδοτηθεί στο κανάλι που προσθέτει θόρυβο στο σήμα. Έχει τέτοια τιμή ώστε ο θόρυβος να αντιστοιχεί στον δοθέντα λόγο σήματος προς θόρυβο.
- TURBO_SCALE – Ο αποκωδικοποιητής χρειάζεται να γνωρίζει το ποσό του θορύβου που εισέρχεται στο σύστημα. Στα πραγματικά συστήματα κάνουμε κάποια εκτίμηση, στην προσομοίωση όμως το ποσό του θορύβου είναι γνωστό.
- EB_NO – Είναι ο λόγος Energy per Bit/Noise του σημείου που υπολογίζεται.

Τα σήματα που χρησιμοποιεί το μπλοκ εξόδου (δεξιά στην εικόνα 65) είναι τα παρακάτω:

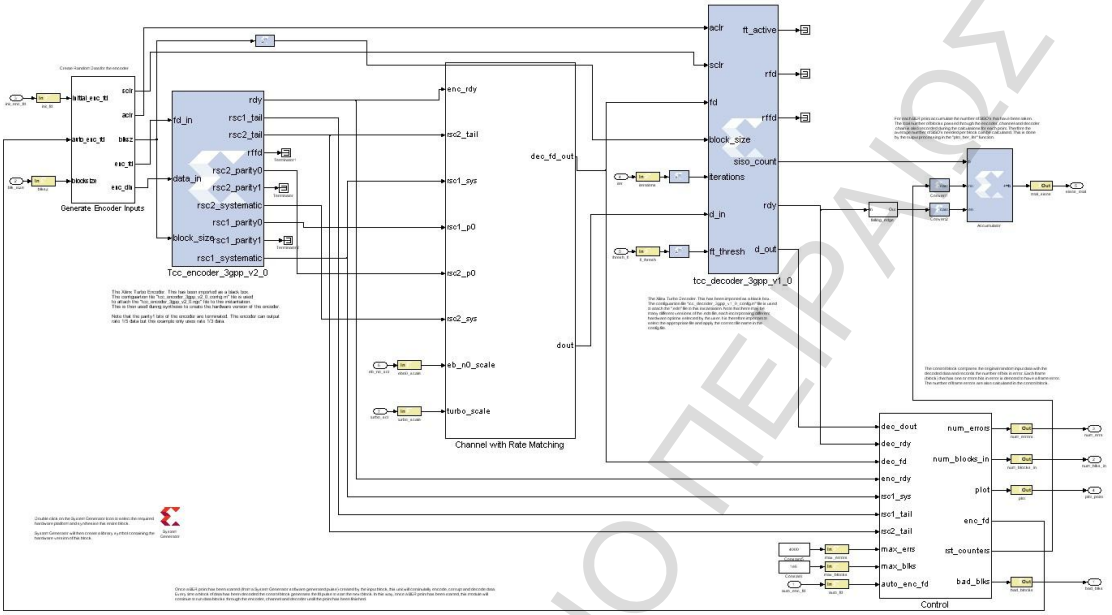
- BAD_BLKS – Είναι το πλήθος των μπλοκ στα οποία βρέθηκαν σφάλματα.
- NUM_BLOCKS_IN – Το πλήθος των μπλοκ που έχουν επεξεργασθεί.
- NUM_ERRORS – Το πλήθος των λανθασμένων bit που βρέθηκαν σε όλα τα μπλοκ που έχουν επεξεργασθεί.
- PLOT_POINT – Σηματοδοτεί την ολοκλήρωση του υπολογισμού ενός σημείου BER. Ενεργοποιεί τις λειτουργίες αποθήκευσης και τοποθέτησης του σημείου στο γράφημα.
- TOTAL_SISOS – Είναι το πλήθος των SISO's (Soft Input Soft Output) που έγιναν για την αποκωδικοποίηση όλων των μπλοκ. Χρησιμοποιείται για να υπολογισθεί ο μέσος όρος των SISO ανά μπλοκ. Φυσικά έχει νόημα εάν έχει ενεργοποιηθεί η λειτουργία γρήγορου τερματισμού. Διαφορετικά είναι γνωστό ότι το πλήθος των SISO θα ισούται με το διπλάσιο του αριθμού των επαναλήψεων.
- EB_NO - Είναι ο λόγος Energy per Bit/Noise του σημείου που υπολογίστηκε. Χρησιμοποιείται για να εμφανίσει το σημείο BER στην κατάλληλη θέση στο γράφημα.
- BLOCK_SIZE - Είναι το μέγεθος του μπλοκ που χρησιμοποιήθηκε στην δοκιμή. Χρησιμεύει για να μπει ο σωστός τίτλος στο γράφημα.
- ITERATIONS – Είναι ο μέγιστος αριθμός των επαναλήψεων. Χρησιμεύει για να μπει ο σωστός τίτλος στο γράφημα.
- FT_THRESH - Είναι το κατώφλι γρήγορου τερματισμού. Χρησιμεύει για να μπει ο σωστός τίτλος στο γράφημα.

Υπάρχουν ενδείξεις στην οθόνη που ανανεώνονται κατά την διάρκεια της προσομοίωσης. Συγκεκριμένα εμφανίζονται το πλήθος των μπλοκ που έχουν περάσει από το κανάλι (display 1), το πλήθος των λανθασμένων bit (display 2) και το πλήθος των λανθασμένων μπλοκ (display 9). Επίσης εμφανίζονται το μέγεθος του μπλοκ δεδομένων (display 3) και ο λόγος σήματος προς θόρυβο που έχουμε θέσει (display 4). Οι ενδείξεις αυτές φαίνονται επισημασμένες στην εικόνα 65.

Το μοντέλο «tcc_3gpp_v1_0_demo_hwitl.mdl» είναι ακριβώς ίδιο με αυτό της εικόνας 65 αλλά το BER Module έχει αντικατασταθεί με το μπλοκ που αντιστοιχεί στο hardware του FPGA. Είναι αυτό που θα τρέξουμε για τις δοκιμές. Απαιτείται φυσικά να έχουμε συνδέσει την πλακέτα FPGA στον υπολογιστή.

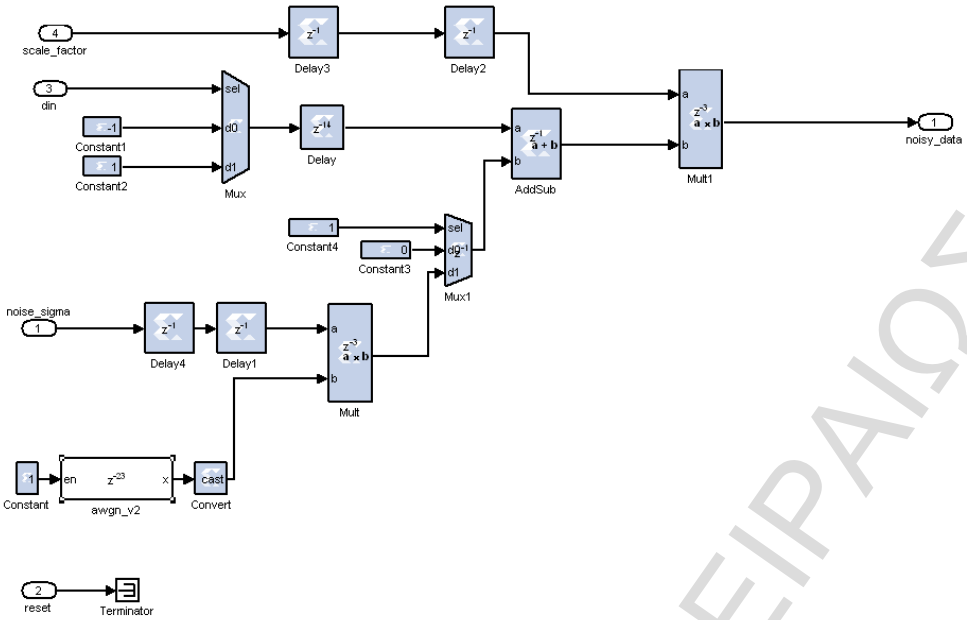
Κάνοντας δεξί κλικ πάνω στο BER module και επιλέγοντας look under mask εμφανίζεται η δομή του module (εικόνα 67). Στα αριστερά υπάρχει το μπλοκ εισόδου. Αυτό δημιουργεί μπλοκ δεδομένων με τυχαία bit. Κάθε μπλοκ έχει μέγεθος από 40 έως 5114 bit σύμφωνα με το πρότυπο 3GPP. Βασίζεται σε ένα καταχωρητή ολίσθησης με ανάδραση (linear feedback shift register – LFSR). Αυτά τα δεδομένα οδηγούνται στον κωδικοποιητή (γαλάζιο μπλοκ στα αριστερά). Η έξοδος του κωδικοποιητή περνά από ένα κανάλι που προσομοιώνει την μετάδοση

του σήματος μέσα από θορυβώδες περιβάλλον. Βασίζεται στο μπλοκ WGNG (White Gaussian Noise Generator) της Xilinx που είναι μία γεννήτρια λευκού θορύβου με κατανομή Gauss. Σε πραγματικό περιβάλλον τα τρία σήματα που παράγει ο κωδικοποιητής πολυπλέκονται και μεταδίδονται σειριακά. Στο μοντέλο της προσομοίωσης για λόγους ταχύτητας τα τρία σήματα δεν πολυπλέκονται και γι' αυτό χρησιμοποιούνται τρία κανάλια, ένα για κάθε σήμα. Αν είχε εφαρμοσθεί η πολύπλεξη τότε θα χρειαζόταν μόνο ένα κανάλι που υπόκειται σε θόρυβο. Ένα από τα τρία όμοια κανάλια φαίνεται στην εικόνα 68



Εικόνα 67 Δομή του BER module

Η έξοδος του καναλιού οδηγείται στον αποκωδικοποιητή (γαλάζιο μπλοκ στα δεξιά) και από εκεί στο μπλοκ ελέγχου. Το μπλοκ αυτό συγκρίνει τα αρχικά δεδομένα με αυτά που πέρασαν από κανάλι «κωδικοποιητής-μετάδοση-αποκωδικοποιητής». Κάθε μπλοκ δεδομένων που έχει ένα ή περισσότερα λάθη καταμετράται και το πλήθος λανθασμένων bit καταμετράται επίσης. Το σύνολο των SISO που χρειάστηκαν για να αποκωδικοποιηθεί κάθε μπλοκ δεδομένων μετράται από τον συσσωρευτή (accumulator) στο πάνω δεξί άκρο του διαγράμματος. Έτσι μπορεί να υπολογισθεί ο μέσος αριθμός των SISO που χρειάστηκαν για την αποκωδικοποίηση του κάθε μπλοκ.



Εικόνα 68 Κανάλι που υπόκειται σε θόρυβο

9.3.1. Λειτουργία του Hardware μοντέλου

Η λειτουργία του μοντέλου ελέγχεται από το script «demo_ctrl.m» του Matlab. Το script αυτό καθορίζει τις παραμέτρους της κάθε δοκιμής. Αποθηκεύει τις μεταβλητές στο workspace του Matlab απ' όπου τις διαβάζουν τα input module και output module του μοντέλου. Η δοκιμές ξεκινούν εκτελώντας το script «demo_ctrl.m». Αρχικές τιμές δίνονται σε μεταβλητές και ακολουθούν τα παρακάτω βήματα.

1. Το input module θέτει σε κατάσταση high το σήμα INITIAL_FD για να σηματοδοτήσει την έναρξη της προσομοίωσης. Μηδενίζεται η τιμή Eb/No και υπολογίζονται οι τιμές EB_NO_SCALE και TURBO_SCALE. Επίσης δίνονται τιμές στις μεταβλητές που καθορίζουν τα BLOCK_SIZE, ITERATIONS και FT_THRESHOLD.
2. Το BER_MODULE αρχίζει να λειτουργεί. Τυχαία δεδομένα παράγονται και τροφοδοτούν τον κωδικοποιητή. Μόλις τα δεδομένα κωδικοποιηθούν η έξοδος RDY παίρνει την τιμή high και στα δεδομένα προστίθεται θόρυβος το μέγεθος του οποίου καθορίζεται από την τιμή του EB_NO_SCALE.
3. Ο αποκωδικοποιητής δέχεται τα δεδομένα με τον θόρυβο και αρχίζει να αποκωδικοποιεί. Μόλις τελειώσει η αποκωδικοποίηση, η έξοδος RDY του αποκωδικοποιητή παίρνει την τιμή high. Ο αποκωδικοποιητής εξάγει τα αποκωδικοποιημένα δεδομένα και ανανεώνονται οι μετρητές των bit και των πλαισίων με σφάλματα.
4. Τα παραπάνω βήματα επαναλαμβάνονται αυτόματα. Ένα νέο μπλοκ δεδομένων δημιουργείται, τροφοδοτεί τον κωδικοποιητή, αποκωδικοποιείται και καταμετρούνται τα σφάλματα. Αυτή η διαδικασία επαναλαμβάνεται για πολλές χιλιάδες μπλοκ. Ο παλμός του σήματος INITIAL_FD ξεκίνησε όλη αυτή την διαδικασία. Το output module δεν έχει χρησιμοποιηθεί ακόμα.
5. Μπλοκ παράγονται συνεχώς μέχρι να ισχύσει μια από τις δύο συνθήκες τερματισμού των επαναλήψεων. Οι συνθήκες τερματισμού είναι α) ο αριθμός των μπλοκ που επεξεργάσθηκαν να φθάσει στο προκαθορισμένο όριο. β) ο αριθμός των εσφαλμένων bit να φθάσει στο προκαθορισμένο όριο. Οι τιμές αυτές είναι μεταβλητές που καθορίζονται από το script «demo_ctrl.m». Όταν μία συνθήκη ισχύσει τότε το σήμα PLOT_POINT παίρνει την τιμή high.
6. Το σήμα PLOT_POINT παρακολουθείται από το output module το οποίο σχεδιάζει το σημείο που υπολογίστηκε στο γράφημα και αποθηκεύει την τιμή του. Εάν η τρέχουσα τιμή του BER δεν έχει φτάσει την απαιτούμενη τιμή τότε απαιτείται και άλλο σημείο. Το output module ενημερώνει το input module για νέα επανάληψη μέσω του σήματος AUTO_FD_IN.

7. Μόλις το input module λάβει το σήμα AUTO_FD_IN αυξάνει την τιμή του Eb/No κατά την προκαθορισμένη τιμή βήματος. Οι κατάλληλες τιμές για τα σήματα TURBO_SCALE και EB_NO_SCALE υπολογίζονται και η προσομοίωση ξεκινά πάλι.
8. Όταν σταματήσει η προσομοίωση του μοντέλου ο έλεγχος περνά πάλι στο script «demo_ctrl.m» που υπολογίζει το νέο σετ παραμέτρων και ξεκινά πάλι την προσομοίωση μέχρι να δοκιμαστούν όλοι οι συνδυασμοί παραμέτρων που έχουν ορισθεί.

9.4. Core Generator

Το core generator θα πρέπει να χρησιμοποιηθεί αν θέλουμε να αλλάξουμε τις παραμέτρους του κωδικοποιητή και του αποκωδικοποιητή που ορίζονται στο hardware όπως ο αριθμός των bit που χρησιμοποιούνται για το σήμα εισόδου και για την επεξεργασία των σημάτων στο FPGA.

Το core generator είναι μια ξεχωριστή εφαρμογή του πακέτου ISE Design Suite. Προσφέρει στον σχεδιαστή την δυνατότητα να υλοποιήσει μπλοκ με λειτουργίες από ένα μεγάλο κατάλογο που περιλαμβάνει μαθηματικές πράξεις, ψηφιακή επεξεργασία σημάτων, μνήμες κ.α. Πρόκειται για τα λεγόμενα «Intellectual Properties (IP)». Είναι πλήρως παραμετροποιήσιμα και επιταχύνουν την διαδικασία σχεδίασης προσφέροντας έτοιμους πυρήνες για λειτουργίες που αφορούν ένα ευρύ φάσμα εφαρμογών.

Η πλειοψηφία των IP προορίζεται για επαγγελματικές – βιομηχανικές εφαρμογές και δεν προσφέρεται δωρεάν. Υπάρχει όμως δοκιμαστική έκδοση αυτών των πυρήνων που προσφέρεται δωρεάν κατόπιν συνεννόησης μέσω email με τον αρμόδιο της Xilinx. Η άδεια που παραχωρείται προσφέρει πρόσβαση στους συγκεκριμένους πυρήνες για ένα χρονικό διάστημα τεσσάρων μηνών και οι πυρήνες που θα δημιουργηθούν διακόπτουν την λειτουργία τους μετά από ένα χρονικό διάστημα λειτουργίας μερικών ωρών. Μετά θα πρέπει το FPGA να ξαναγραφεί για να επανέλθει η λειτουργία. Οι περιορισμοί αυτοί δεν δημιουργούν πρόβλημα στους σκοπούς αυτής της εργασίας.

9.4.1. Δημιουργία νέου Κωδικοποιητή-Αποκωδικοποιητή

Μέσα στο πακέτο XAPP948.zip περιλαμβάνονται πυρήνες του κωδικοποιητή και του αποκωδικοποιητή για δοκιμαστική χρήση με τα ονόματα : «tcc_decoder_3gpp_v1_0_1r3_2i3_6m3_w32_scale.edn» και «tcc_encoder_3gpp_v2_0.ngc»

Για να δημιουργήσουμε νέο ζεύγος κωδικοποιητή και αποκωδικοποιητή θα πρέπει προηγουμένως να έχουμε αποκτήσει άδεια δοκιμαστικής χρήσης από την Xilinx. Το σχετικό αρχείο πρέπει να τοποθετηθεί μέσα στον φάκελο «.xilinx» μαζί με τις άλλες άδειες που έχουμε (όπως έχει ήδη αναφερθεί, το system generator απαιτεί επίσης ξεχωριστή άδεια).

Κατόπιν ξεκινάμε το core generator το οποίο βρίσκεται στο στην διαδρομή «Xilinx design tools→ISE design suite14.2→ISE design tools→32 bit tools→CORE Generator». Η πρώτη κίνηση είναι να δημιουργήσουμε ένα νέο project από το μενού «File→New Project». Στην καρτέλα «Project options» και στο πλαίσιο «part» επιλέγουμε τα χαρακτηριστικά του FPGA για το οποίο θέλουμε να δημιουργηθεί ο πυρήνας. Για την Virtex5 της εργασίας επιλέγουμε:

Family	Virtex5
Device	xc5vlx110t
Package	ff1136
Speed Grade	-1

Κατόπιν από τον IP κατάλογο στα αριστερά της οθόνης επεκτείνουμε την κατηγορία «Communication & Networking» πατώντας στο «+» δίπλα στην κατηγορία. Μετά επεκτείνουμε την κατηγορία «Error Correction» και από τον κατάλογο που αναπτύσσεται επιλέγουμε «3GPP Turbo Decoder». Από τις επιλογές στα δεξιά επιλέγουμε «Customize and Generate». Εμφανίζεται ένα προειδοποιητικό μήνυμα που αναφέρει ότι ο πυρήνας που θα παραχθεί προορίζεται για εκτίμηση και έχει χρονικούς περιορισμούς στην λειτουργία του.

Κατόπιν εμφανίζεται η οθόνη με τις επιλογές για τον κωδικοποιητή (εικόνα 69) που αφορούν τις προαιρετικές εξόδους και τον αριθμό των bit του σήματος εισόδου και των

εσωτερικών υπολογισμών. Υπάρχει επίσης δυνατότητα να ενεργοποιηθεί η επιλογή γρήγορου τερματισμού (fast termination). Τέλος, ο λόγος κωδικοποίησης (rate) μπορεί να ορισθεί στο 1/5 αντί του 1/3. Ας σημειωθεί όμως πως η παρούσα σχεδίαση υποστηρίζει μόνο τον λόγο 1/3. Στην δεύτερη σελίδα επιλογών υπάρχει δυνατότητα να προσθέσουμε εξόδους ND (New Data), RD_OUTPUT, CE (Clock Enable), SCLR και SISO_COUNT. Τέλος πατάμε το κουμπάκι «Generate» και μετά από δεκαπέντε περίπου λεπτά δημιουργούνται μια σειρά από αρχεία με όνομα «tcc_decoder_3gpp_v4_0» και διάφορες καταλήξεις.

Εικόνα 69 Παραμετροποίηση Αποκωδικοποιητή

Για να μπορέσουμε να χρησιμοποιήσουμε τον αποκωδικοποιητή που συνθέσαμε θα πρέπει να τον εισάγουμε στο μοντέλο μας σαν μπλοκ. Αυτό γίνεται με την χρήση του μπλοκ «black box». Ανοίγουμε το system generator και δημιουργούμε ένα κενό πρότζεκτ. Σ' αυτό το σημείο είναι σημαντικό να αποθηκεύσουμε το κενό πρότζεκτ, διαφορετικά δεν θα μπορέσουμε να ολοκληρώσουμε την διαδικασία. Από το «Xilinx toolkit» επιλέγουμε και σέρνουμε μέσα στο κενό πρότζεκτ το μπλοκ «black box». Αυτόματα ανοίγει ένα παράθυρο διαλόγου και μας ζητά να ορίσουμε το αρχείο που περιγράφει την λειτουργία του μπλοκ. Το παράθυρο αυτό δεν ανοίγει αν δεν έχουμε προηγουμένως αποθηκεύσει το πρότζεκτ. Βρίσκουμε το αρχείο με την κατάληξη .vhd δηλαδή το «tcc_decoder_3gpp_v4_0.vhd» και πατάμε «OK».

Αυτόματα δημιουργείται ένα αρχείο με όνομα «tcc_decoder_3gpp_v4_0_config.m» που περιγράφει τις ιδιότητες του νέου μπλοκ. Ανοίγει μόνο του στον editor του Matlab. Στην γραμμή 91 έχει την εντολή :

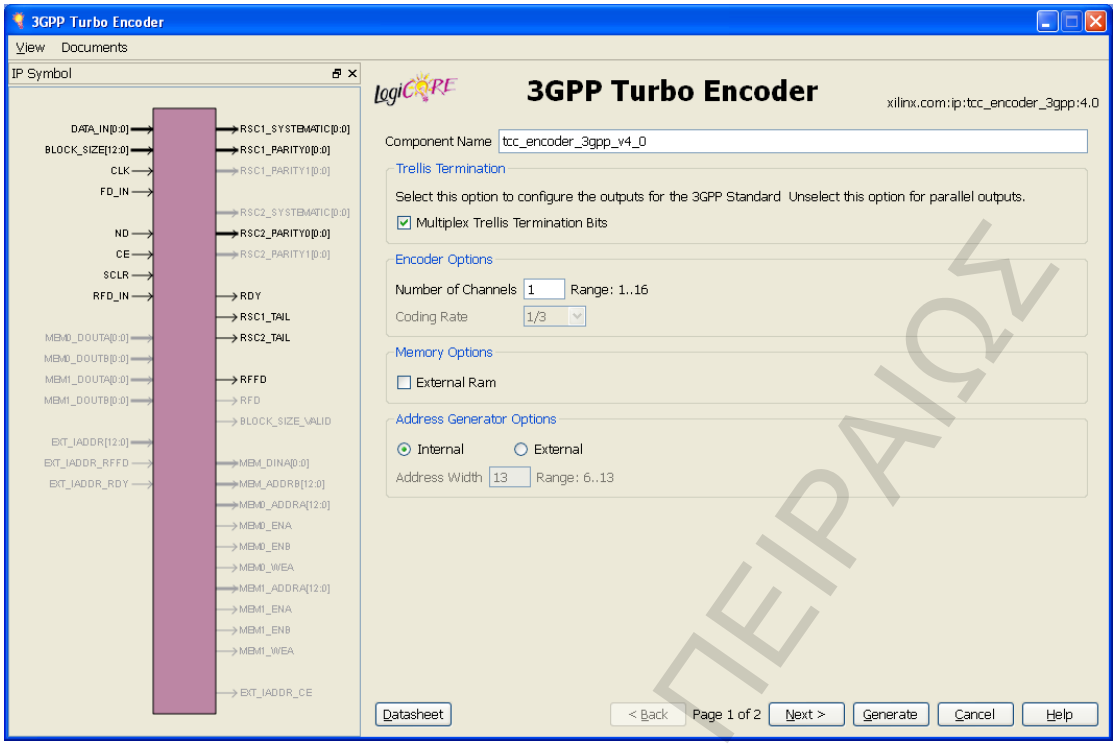
```
this_block.addFile('tcc_decoder_3gpp_v4_0.vhd');
```

Ακριβώς από κάτω προσθέτουμε την εντολή :

```
this_block.addFile('tcc_decoder_3gpp_v4_0.ngc');
```

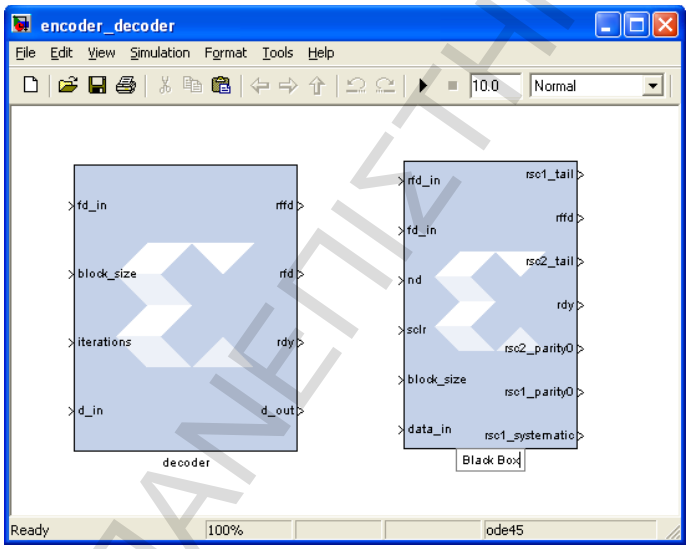
Αποθηκεύουμε το αρχείο και κλείνουμε τον editor. Μετονομάζουμε το νέο μπλοκ σε «Decoder».

Στην συνέχεια θα δημιουργήσουμε τον κωδικοποιητή. Επιστρέφουμε στο Xilinx Core Generator και επιλέγουμε 3GPP Turbo Encoder. Από τις επιλογές στα δεξιά επιλέγουμε «Customize and Generate». Εμφανίζεται το προειδοποιητικό μήνυμα που αναφέρει ότι ο πυρήνας που θα παραχθεί προορίζεται για εκτίμηση και έχει περιορισμούς στην λειτουργία του. Πατάμε OK και εμφανίζεται η καρτέλα της εικόνας 70 με τις επιλογές για τον κωδικοποιητή. Οι επιλογές αφορούν τις προαιρετικές εξόδους καθώς και την ύπαρξη εξωτερικής μνήμης. Εάν τσεκάρουμε την επιλογή «Multiplex Trellis Termination Bits» που απαιτείται από το πρότυπο 3GPP τότε αυτόματα επιλέγεται ο λόγος κωδικοποίησης στο 1/3 που επίσης ορίζεται από το πρότυπο. Ο κωδικοποιητής μπορεί να λειτουργήσει με από 1 έως 16 κανάλια εισόδου. Για την σχεδίαση της εργασίας επιλέγουμε «1». Προσέχουμε οι επιλογές να είναι οι ίδιες με αυτές που επιλέξαμε στον αποκωδικοποιητή. Πατάμε το κουμπί Generate.



Εικόνα 70 Παραμετροποίηση Κωδικοποιητή

Μετά από δέκα λεπτά περίπου η διαδικασία ολοκληρώνεται και δημιουργούνται μια σειρά από αρχεία με όνομα «tcc_encoder_3gpp_v4_0» και διάφορες καταλήξεις. Μπορούμε τώρα να κλείσουμε το «CORE Generator». Στη συνέχεια θα πρέπει να δημιουργήσουμε ένα μπλοκ για τον κωδικοποιητή. Ανοίγουμε το πρότζεκτ στο οποίο δημιουργήσαμε πριν το μπλοκ του αποκωδικοποιητή και εισάγουμε από το «Xilinx toolbox» ένα μπλοκ «black box».



Εικόνα 71 Δημιουργία Κωδικοποιητή-Αποκωδικοποιητή

Αυτόματα ανοίγει ένα παράθυρο διαλόγου και μας ζητά να ορίσουμε το αρχείο που περιγράφει την λειτουργία του μπλοκ. Βρίσκουμε το αρχείο με την κατάληξη .vhd δηλαδή το «tcc_encoder_3gpp_v4_0.vhd» και πατάμε «OK». Δημιουργείται ένα έγγραφο που περιγράφει τις ιδιότητες του νέου μπλοκ, με όνομα «tcc_encoder_3gpp_v4_0_config.m». Ανοίγει αυτόματα στον editor στον editor του Matlab. Στην γραμμή 117 έχει την εντολή :

```
this_block.addFile('tcc_encoder_3gpp_v4_0.vhd');
```

Ακριβώς από κάτω προσθέτουμε την εντολή :

```
this_block.addFile('tcc_encoder_3gpp_v4_0.ngc');
```

Αποθηκεύουμε το αρχείο και κλείνουμε τον editor. Μετονομάζουμε το νέο μπλοκ σε «encoder».

Για να τοποθετήσουμε τους νέους κωδικοποιητές – αποκωδικοποιητές ανοίγουμε το μοντέλο «tcc_3gpp_v1_0_demo_synth.mdl». Εμφανίζεται το μοντέλο της εικόνας 65. Πάνω στο «BER module» κάνουμε δεξί κλικ και επιλέγουμε «look under mask». Εμφανίζεται η εσωτερική δομή του module (εικόνα 67). Με γαλάζιο χρώμα φαίνονται ο κωδικοποιητής στα αριστερά και ο αποκωδικοποιητής στα δεξιά. Αφαιρούμε τα υπάρχοντα μπλοκ και τοποθετούμε στην θέση τους αυτά που δημιουργήσαμε.

9.4.2. Περιορισμοί στην σχεδίαση

Το core generator μας δίνει την δυνατότητα να αλλάξουμε όλες τις παραμέτρους του ζεύγους κωδικοποιητή-αποκωδικοποιητή. Δεν είναι όμως όλοι οι δυνατοί συνδυασμοί συμβατοί με την σχεδίαση. Υπάρχουν ορισμένοι περιορισμοί που θα πρέπει να τηρηθούν προκειμένου το μοντέλο που θα προκύψει να είναι συνθέσιμο.

Το μοντέλο έχει σχεδιασθεί ώστε να χρησιμοποιεί λόγο κωδικοποιητή (rate) 1/3. Στην σύνθεση των κωδικοποιητή-αποκωδικοποιητή υπάρχει επιλογή για λόγο κωδικοποιητή 1/3 και 1/5. Μόνο η επιλογή 1/3 είναι συμβατή με την σχεδίαση.

Όπως φαίνεται στο σχήμα της εικόνας 64, η έξοδος του κωδικοποιητή οδηγείται σε ένα κανάλι μετάδοσης και η έξοδος του καναλιού οδηγείται στον αποκωδικοποιητή. Το πλήθος των bit εισόδου του αποκωδικοποιητή επιλέγονται κατά την σύνθεσή του στο Core Generator (εικόνα 69) και θα πρέπει να είναι ίδιο με το πλήθος των bit του καναλιού. Αυτό επιλέγεται από το συνθέσιμο μοντέλο «tcc_3gpp_v1_0_demo_synth.mdl». Ανοίγοντας το μοντέλο και κάνοντας διπλό κλικ πάνω στο «BER module» ανοίγει το παράθυρο στο οποίο μπορούμε να εισάγουμε το πλήθος των bit του καναλιού μετάδοσης (εικόνα 66).

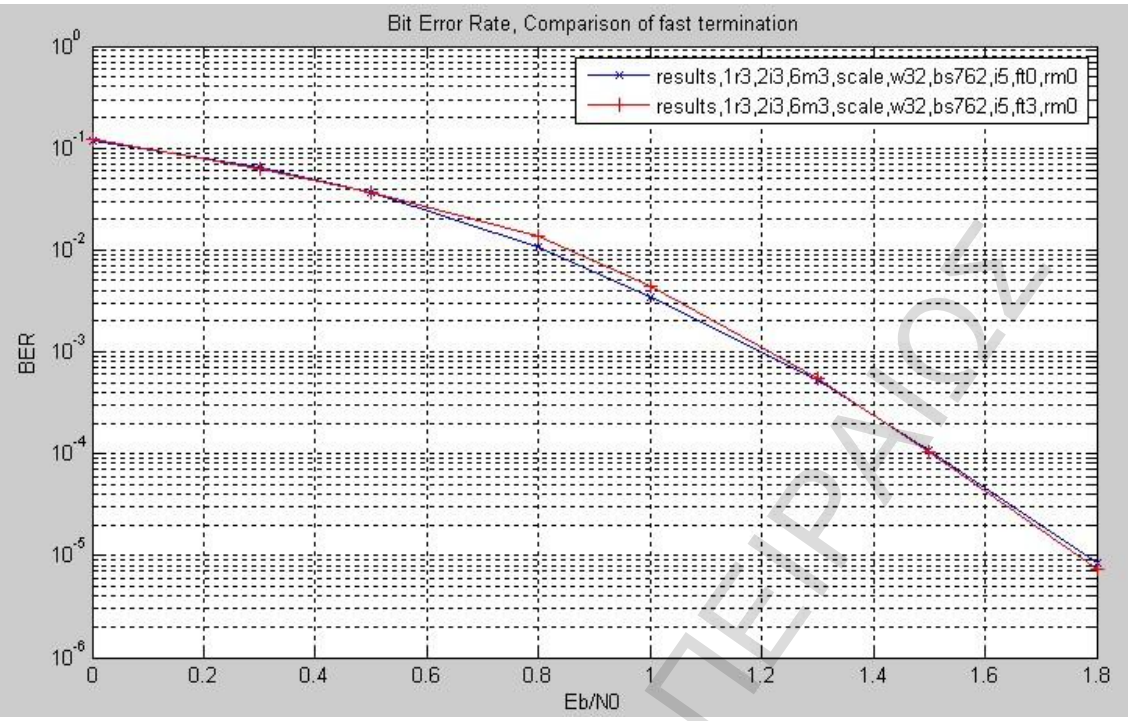
Το system Generator χρησιμοποιεί την είσοδο clock enable (CE) για να ελέγξει την λειτουργία των πυρήνων. Για παράδειγμα η εκτέλεση βήμα-βήμα γίνεται ελέγχοντας την είσοδο CE. Πρέπει λοιπόν όλοι οι πυρήνες που δημιουργούμε να περιλαμβάνουν την είσοδο CE.

9.5. Μετρήσεις

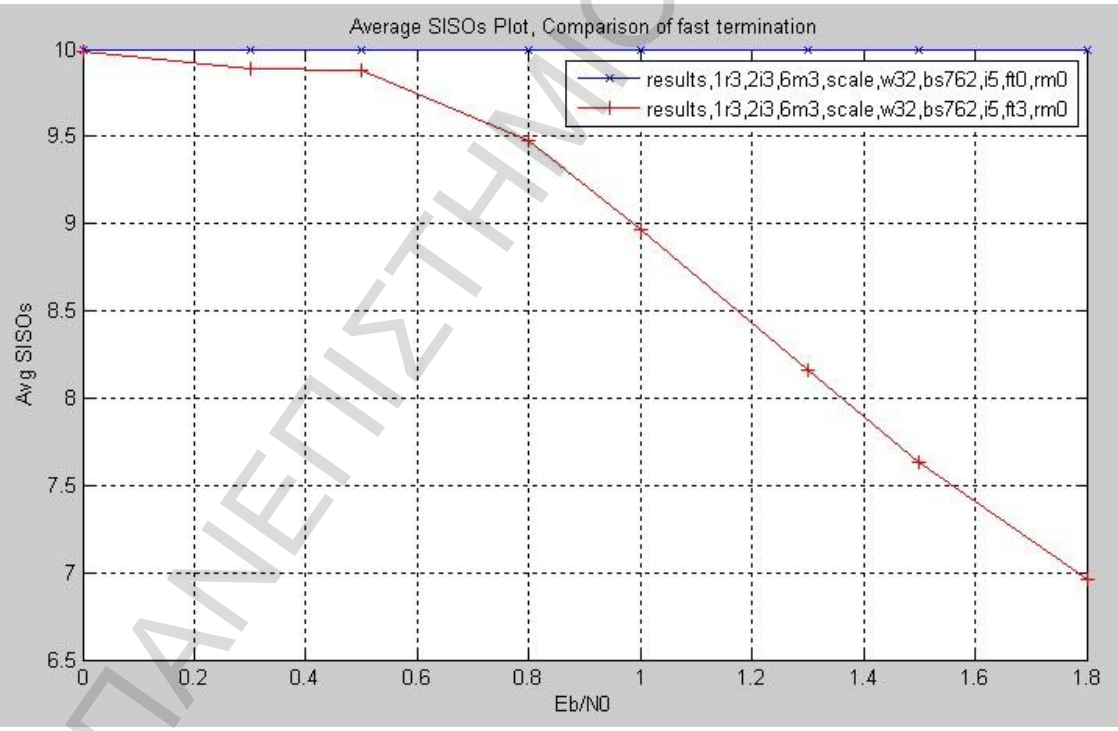
Στο κεφάλαιο αυτό παρουσιάζονται οι μετρήσεις που έγιναν στο μοντέλο «TURBO CODE» στην προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop και κατόπιν οι μετρήσεις του μοντέλου που προσομοιώνεται με software στο simulink.

9.5.1. Μετρήσεις με hardware co-simulation

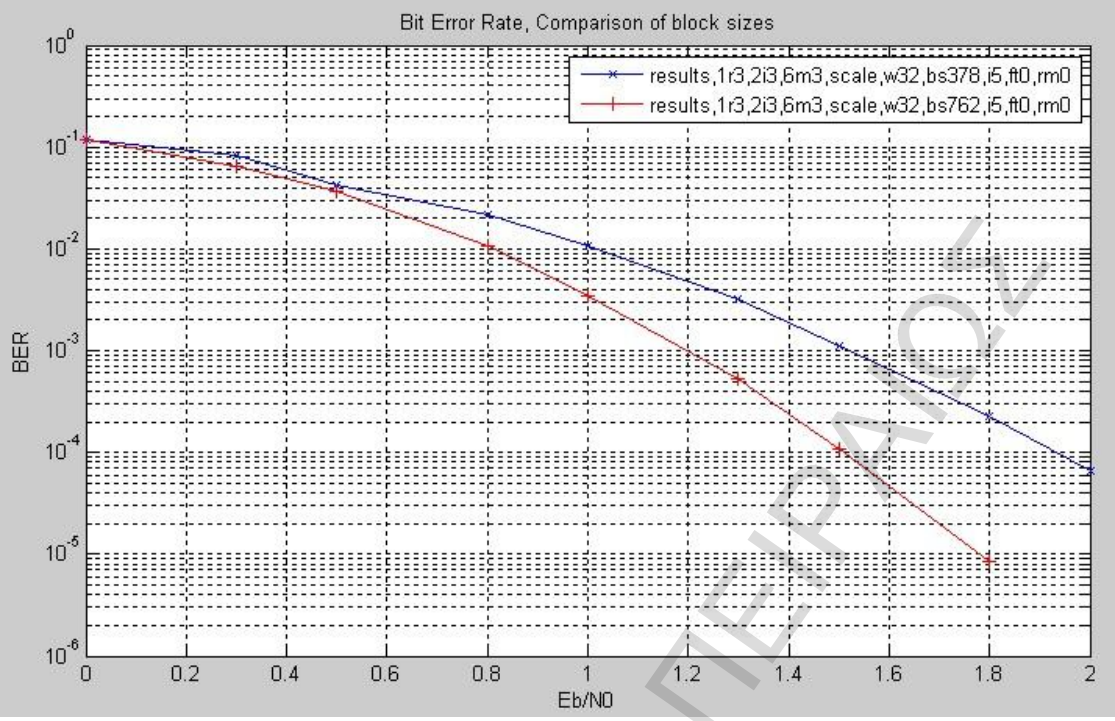
Η διαδικασία μετρήσεων ξεκινά εκτελώντας το script «demo_ctrl.m». Το script στέλνει στην πλακέτα τους συνδυασμούς παραμέτρων που έχουμε ορίσει και η πλακέτα δημιουργεί τις τιμές για τα γραφήματα. Με τις αρχικές παραμέτρους χρειάζεται περίπου 10 λεπτά για να δημιουργηθούν 9 γραφήματα. Μερικά από αυτά φαίνονται παρακάτω.



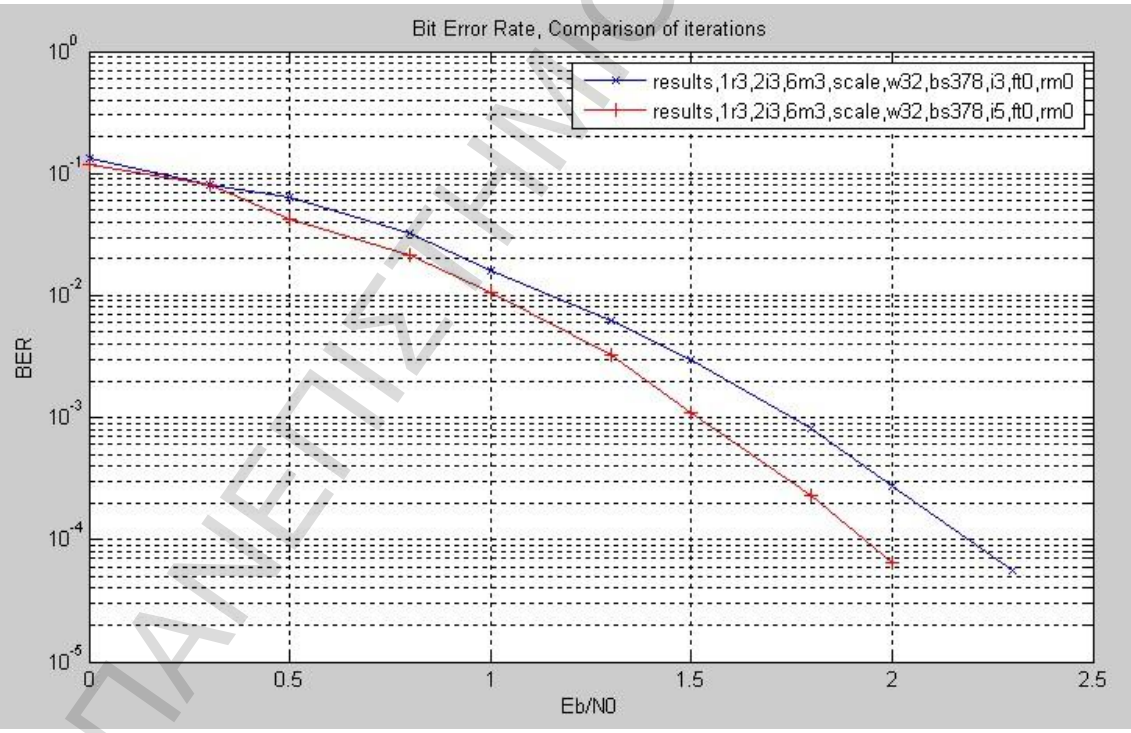
Εικόνα 72 Επίδραση του γρήγορου τερματισμού στο BER



Εικόνα 73 Σχέση θορύβου και γρήγορου τερματισμού



Εικόνα 74 Επίδραση του μεγέθους μπλοκ στο BER



Εικόνα 75 Επίδραση αριθμού επαναλήψεων στο BER

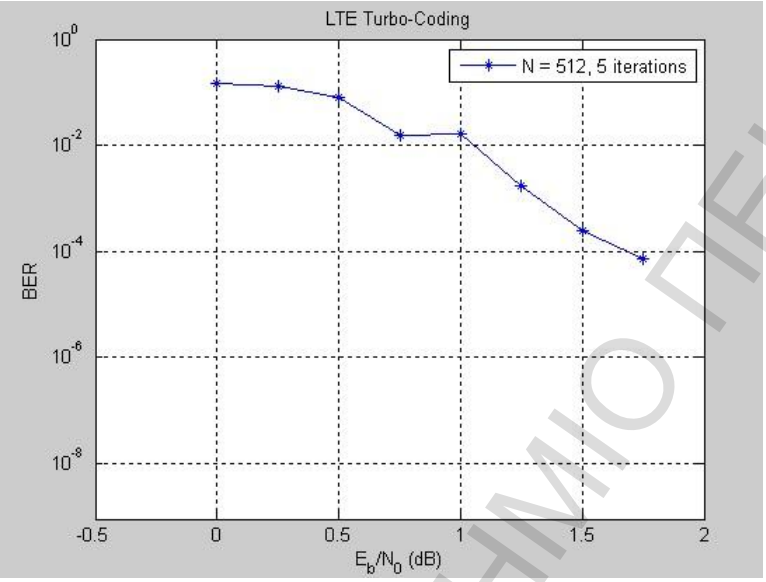
Όση ώρα «τρέχει» η προσομοίωση μπορούμε να παρακολουθούμε την εξέλιξη υπολογισμού κάθε σημείου του γραφήματος από τους μετρητές που υπάρχουν στο μοντέλο και που έχουν επισημανθεί με κόκκινο πλαίσιο στην εικόνα 65. Εμφανίζεται το μέγεθος του μπλοκ και το επίπεδο θορύβου που έχουμε επιλέξει, το πλήθος των μπλοκ που έχουν επεξεργασθεί, το πλήθος των λανθασμένων bit και το πλήθος των μπλοκ με λανθασμένα bit.

9.5.2. Μετρήσεις με software προσομοίωση

Χρησιμοποιούμε το script του matlab «commTurboCoding.m» στο οποίο είναι εύκολο να δώσουμε τιμές στις παραμέτρους και να φτιάξουμε γράφημα με τα αποτελέσματα. Συγκεκριμένα στις γραμμές 22 έως 26 βάζουμε τις παρακάτω τιμές οι οποίες θα πρέπει να μας δημιουργήσουν ένα γράφημα παρόμοιο με αυτό της εικόνας 74.

```
numIter     = 5; % Number of decoding iterations
blkLength   = 512; % Block length
EbNo        = [0 0.25 0.5 0.75 1 1.25 1.5 1.75 2 ];% Eb/No values to loop over
maxNumErrs = 100; % maximum number of errors per Eb/No value
maxNumBlks = 2000;% maximum number of blocks per Eb/No value
```

Το γράφημα που δημιουργήθηκε φαίνεται παρακάτω και χρειάστηκε οκτώ λεπτά.



Εικόνα 76 Γράφημα Simulink με μικρό πλήθος δειγμάτων

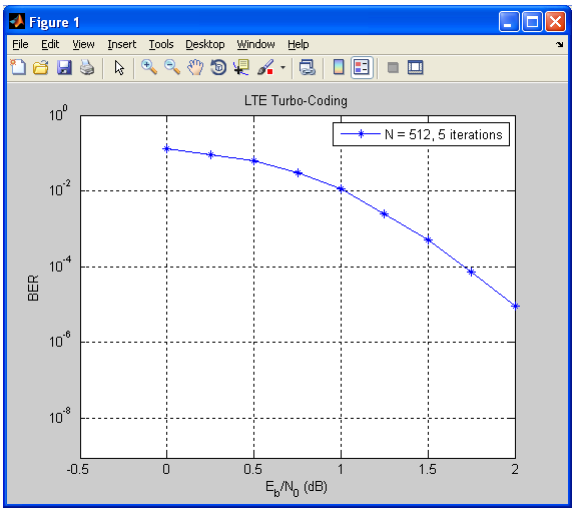
Το διάγραμμα αυτό διαφέρει αρκετά απ' αυτά της εικόνας 76. Η αιτία είναι ότι για λόγους ταχύτητας έχουμε επεξεργαστεί σημαντικά μικρότερο πλήθος bit. Συγκεκριμένα στην προσομοίωση «hardware co-simulation» έχουμε ορίσει ότι ένα σημείο έχει υπολογισθεί με αρκετή ακρίβεια όταν έχουν επεξεργασθεί 10^9 bit ή όταν έχουν συμβεί 8000 λάθη. Στην προσομοίωση «software simulation» έχουμε ορίσει το σημείο να υπολογίζεται μετά από 10^6 bit ή 100 λάθη. Παρόλο το σημαντικά μικρότερο αριθμό bit προς επεξεργασία, χρειάστηκαν οκτώ λεπτά για να σχεδιασθεί η μία από τις δύο καμπύλες του ενός γραφήματος, δηλαδή όσος χρόνος απαιτήθηκε για να υπολογισθούν όλα τα γραφήματα του hardware co-simulation. Ο μικρός αριθμός των μπλοκ που επεξεργάσθηκαν δεν επέτρεψε την κατασκευή γραφημάτων με αρκετή ακρίβεια.

Μπορούμε να θέσουμε ακριβώς τις ίδιες παραμέτρους στα δύο μοντέλα. Ορίζουμε πλήθος επαναλήψεων:5, bit ανά μπλοκ:512, λόγο σήματος προς θόρυβο: 0 έως 2 σε βήματα του 0.25. Επίσης θέτουμε μέγιστο πλήθος σφαλμάτων: 8000 και μέγιστο πλήθος μπλοκ για τον υπολογισμό του κάθε σημείου: 2.000.000 έτσι ώστε με μέγεθος μπλοκ ίσο με 512 το συνολικό πλήθος bit να είναι $1X10^9$.

Οι παράμετροι αυτές βρίσκονται στις γραμμές 22 έως 26 του script «commTurboCoding».

```
numIter     = 5;            % Number of decoding iterations
blkLength   = 512;          % Block length
EbNo        = [0 0.25 0.5 0.75 1 1.25 1.5 1.75 2 ];% Eb/No values to loop over
maxNumErrs = 8000;          % maximum number of errors per Eb/No value
maxNumBlks = 2000000;       % maximum number of blocks per Eb/No value
```

Έχουμε προσθέσει το ζεύγος εντολών tic-toc ώστε για κάθε σημείο να εμφανίζεται ο χρόνος που έχει απαιτηθεί για τον υπολογισμό του.




```
Processing EbNo = 0
Elapsed time is 12.045444 seconds.
Processing EbNo = 0.25
Elapsed time is 22.778600 seconds.
Processing EbNo = 0.5
Elapsed time is 38.720880 seconds.
Processing EbNo = 0.75
Elapsed time is 70.657970 seconds.
Processing EbNo = 1
Elapsed time is 156.700142 seconds.
Processing EbNo = 1.25
Elapsed time is 560.261686 seconds.
Processing EbNo = 1.5
Elapsed time is 2501.423684 seconds.
Processing EbNo = 1.75
Elapsed time is 16014.159751 seconds.
Processing EbNo = 2
Elapsed time is 123737.812091 seconds.
```

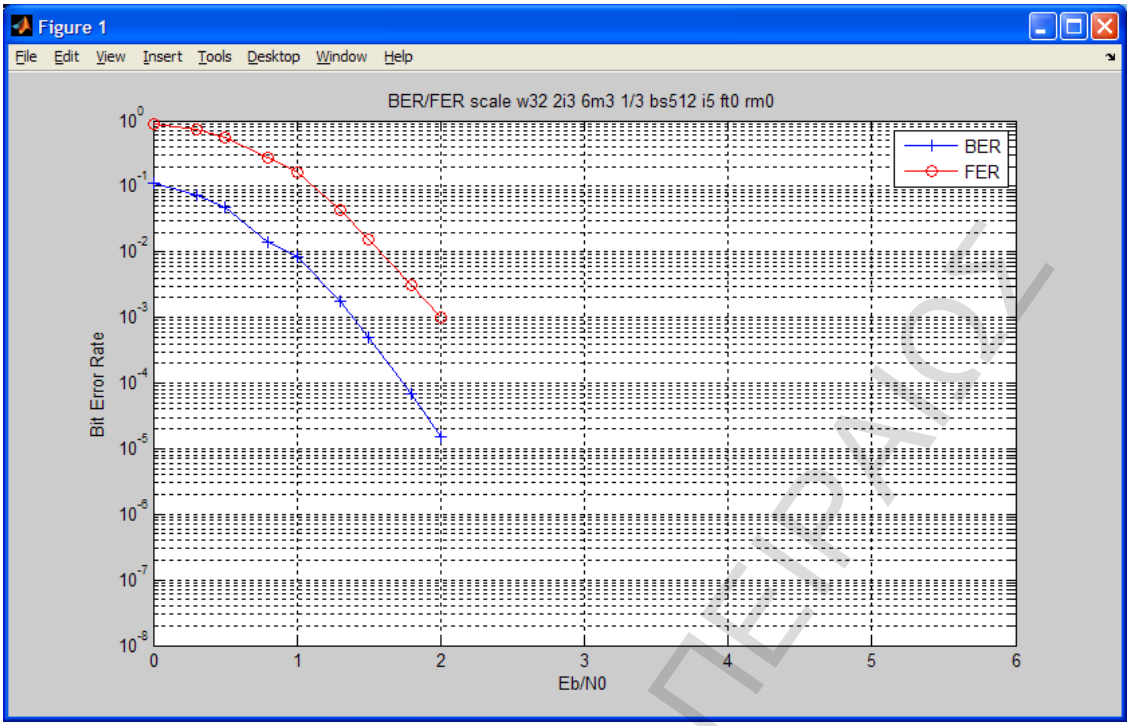
Εικόνα 77 Γράφημα Simulink και ο χρόνος που απαιτήθηκε για τον υπολογισμό των σημείων του

Τα πρώτα σημεία υπολογίζονται γρήγορα γιατί ο λόγος σήματος προς θόρυβο είναι πολύ χαμηλός. Όσο αυξάνει ο λόγος αυτός αυξάνει και ο χρόνος υπολογισμού γιατί χρειάζεται προσομοίωση της μετάδοσης πολύ περισσότερων bit μέχρι να σημειωθεί το απαραίτητο πλήθος σφαλμάτων για τον υπολογισμό ενός σημείου. Βλέπουμε πως ο χρόνος για τον υπολογισμό των δύο τελευταίων σημείων είναι περίπου 4 ώρες και 30 ώρες αντίστοιχα. Η προσομοίωση συνολικά χρειάστηκε 34½ ώρες, χρόνος που δεν αφήνει περιθώρια για πρακτική εφαρμογή της προσομοίωσης.

Στην συνέχεια δοκιμάζουμε την προσομοίωση με τις ίδιες παραμέτρους με χρήση της τεχνικής FPGA-In-the-Loop. Οι τιμές ορίζονται στις γραμμές 74 έως 78 στο script demo_ctrl.

```
max_bitcount = 1e9;          % maximum number of bits per calculated point
max_errors   = 8000;         % maximum number of errors per calculated point
%min_errors   = 10;          % minimum number of errors per calculated point
min_ber      = 5e-5;         % finish when BER betters this value
eb_n0_step   = 0.25;         % step between sucessive BER values
```

Στο μοντέλο της HW Co_Sim ο λόγος Eb/No ξεκινά πάντα από το 0 και αυξάνει σύμφωνα με το βήμα που έχουμε ορίσει μέχρι ο λόγος BER να πέσει χαμηλότερα από το όριο min_ber. Με δοκιμές βρήκαμε πως με min_ber ίσο με $5X10^{-5}$, η προσομοίωση σταματά μετά τον υπολογισμό της τιμής Eb/No = 2. Με το ζεύγος εντολών tic-toc μετράμε τον συνολικό χρόνο προσομοίωσης που είναι ίσος με 115 sec.



Εικόνα 78 Γράφημα προσομοίωσης με χρήση της τεχνική FPGA-In-the-Loop

Το γράφημα ολοκληρώνεται 1075 φορές ποιο γρήγορα από το αντίστοιχο μοντέλο Simulink.

Ένας σύντομος τρόπος για να συγκρίνουμε τα δύο μοντέλα είναι να θέσουμε λόγο σήματος προς θόρυβο ίσο με 2dB και όριο επεξεργασίας τα 10^9 bit για την hardware co-simulation και 10^6 bit για την software simulation. Με το χαμηλό επίπεδο θορύβου που θέσαμε θα χρειαστεί να εξαντληθεί όλο το πλήθος των bit προς επεξεργασία. Οπότε αυτό που μετράμε είναι ο χρόνος που απαιτείται για την επεξεργασία 10^9 bit και 10^6 bit αντίστοιχα για τις δύο υλοποιήσεις της προσομοίωσης. Μπορούμε να εκτελέσουμε τις μετρήσεις για αριθμό επαναλήψεων ίσο με τρία και για αριθμό επαναλήψεων ίσο με πέντε.

	software simulation επεξεργασία 10^6 bit		hardware co-simulation επεξεργασία 10^9 bit	
Επαναλήψεις αποκωδικοποιητή	3	5	3	5
Χρόνος (sec)	108	177	96	115

Βλέπουμε πως στην προσομοίωση με χρήση της τεχνικής FPGA-In-the-Loop για την επεξεργασία 10^9 bit απαιτείται παραπλήσιος (μάλιστα μικρότερος) χρόνος επεξεργασίας με αυτόν της προσομοίωσης με software για την επεξεργασία 10^6 bit. Σε αυτήν την απλή δοκιμή φαίνεται ότι με χρήση της τεχνικής FPGA-In-the-Loop η προσομοίωση επιταχύνεται κατά μέσο όρο κατά 1332 φορές.

10. Συμπεράσματα – Περίληψη

Κατά την διάρκεια αυτής της εργασίας δοκιμάστηκαν προσομοιώσεις διαφόρων συστημάτων. Υπήρξαν συστήματα απλά και άλλα με μεγάλη πολυπλοκότητα. Στην εργασία παρουσιάζονται τα τέσσερα πιο αντιπροσωπευτικά από αυτά. Παρακάτω φαίνεται ο πίνακας με συγκεντρωμένες τις μετρήσεις απόδοσης των τεσσάρων μοντέλων που δοκιμάστηκαν.

$^{A}/_{A}$		Χρόνος Software Simulation	Χρόνος Hardware Co-Simulation	Λόγος επιτάχυνσης
1	Απλό φίλτρο, 1000 κύκλοι	5.4	180	0,03
2	Φίλτρο FIR 1000 κύκλοι	25	1600	0,015
3	Επεξεργασία video, 145 πλαίσια	40	95	0,42
4	Γράφημα BER	123737	115	1075
5	Μέτρηση BER με τρεις επαναλήψεις	108 sec για 10^6 bit	96 sec για 10^9 bit	1125
6	Μέτρηση BER με πέντε επαναλήψεις	177 sec για 10^6 bit	115 sec για 10^9 bit	1540

Στο πρώτο μοντέλο με το απλό φίλτρο φαίνεται ξεκάθαρα το σημαντικό πρόβλημα στην προσομοίωση με χρήση hardware. Η επικοινωνία ανάμεσα στο λογισμικό που τρέχει στον υπολογιστή και στο κύκλωμα που έχει υλοποιηθεί στο FPGA της πλακέτας απαιτεί χρόνο πολλαπλάσιο του χρόνου επεξεργασίας των δεδομένων. Το όποιο όφελος από την αυξημένη ταχύτητα των υπολογισμών στο FPGA χάνεται μπροστά στην καθυστέρηση από την μεταφορά των δεδομένων.

Στο δεύτερο μοντέλο με τα φίλτρο FIR η κατάσταση για την απόδοση της hardware co-simulation χειροτερεύει. Οι υπολογισμοί απαιτούν πολύ περισσότερο χρόνο από αυτούς του πρώτου μοντέλου και αυτό φαίνεται στον χρόνο που απαιτούνται 1000 κύκλοι προσομοίωσης. Στο software μοντέλο ο χρόνος σχεδόν πενταπλασιάστηκε. Παραδόξως ο χρόνος για την hardware co-simulation σχεδόν δεκαπλασιάστηκε. Αυτό εξηγείται από το γεγονός ότι ενώ στο πρώτο μοντέλο χρειαζόταν η επεξεργασία μιας μόνο τιμής για την εξαγωγή ενός σημείου, τώρα απαιτείται η επεξεργασία πολύ περισσότερων τιμών για την δημιουργία ενός πλαισίου. Έτσι ο λόγος επιτάχυνσης με χρήση hardware co-simulation μειώθηκε στο μισό σε σχέση με το προηγούμενο παράδειγμα.

Στο τρίτο μοντέλο χρησιμοποιείται η μαζική μεταφορά δεδομένων (burst transfer). Κάθε πλαίσιο μεταφέρεται ολόκληρο στην μνήμη του FPGA και κατόπιν γίνεται η επεξεργασία του κάθε bit της εικόνας. Στην συνέχεια με τον ίδιο τρόπο το πλαίσιο μεταφέρεται πίσω στον υπολογιστή. Η μέθοδος αυτή βελτίωσε την απόδοση της προσομοίωσης με FPGA που και πάλι όμως είναι πιο αργή από την προσομοίωση με software. Οι χρόνοι στις δύο προσομοιώσεις είναι συγκρίσιμοι. Η προσομοίωση με software χρειάζεται περίπου το μισό χρόνο από αυτόν της προσομοίωσης με FPGA. Θεωρείται βέβαιο πως αν η πλακέτα υποστήριζε επικοινωνία μέσω Ethernet τότε η προσομοίωση με χρήση υλικού θα ήταν δύο ή τρεις φορές γρηγορότερη από την προσομοίωση με software.

Στο τέταρτο μοντέλο η προσομοίωση με FPGA πετυχαίνει ταχύτητες ασύλληπτες για την προσομοίωση με software. Οι χρόνοι προσομοίωσης μειώνονται με συντελεστή μεγαλύτερο από 1000. Σε συγκεκριμένες δοκιμές, όπως φαίνεται στον παραπάνω πίνακα, όπου οι υπολογισμοί γίνονταν πιο πολύπλοκοι, ο λόγος επιτάχυνσης ξεπέρασε το 1500. Θα μπορούσαμε να πούμε ότι το μοντέλο αυτό αναδεικνύει την χρήση της τεχνικής FPGA-In-the-Loop για την προσομοίωση συστημάτων. Ο λόγος που απέδωσε πολύ καλά είναι ότι ο όγκος των δεδομένων που μεταφέρονται μεταξύ υπολογιστή και πλακέτας είναι μηδαμινός. Ο υπολογιστής στέλνει μόνο τις παραμέτρους της συγκεκριμένης μέτρησης. Τα δεδομένα παράγονται από το ίδιο το FPGA και κατόπιν επεξεργάζονται. Το hardware λειτουργεί αδιάκοπα

μέχρι να εκπληρωθούν οι συνθήκες που έχουν δοθεί. Τότε μόνο υπάρχει ξανά επικοινωνία με τον υπολογιστή για να αποσταλούν σε αυτόν τα αποτελέσματα της μέτρησης.

Βλέπουμε ότι το FPGA μπορεί να επεξεργαστεί δεδομένα σε ταχύτητες πολλαπλάσιες αυτών που μπορεί να πετύχει ένας υπολογιστής. Αυτό οφείλεται στην παραλληλία των FPGA. Πολλοί υπολογισμοί εκτελούνται παράλληλα σε διάφορα τμήματα του υλικού. Αν και τα FPGA λειτουργούν σε συχνότητες πολύ μικρότερες των σύγχρονων επεξεργαστών (η πλακέτα της εργασίας εργάζεται στους 100MHZ) η απόδοσή τους είναι πολύ μεγάλη γιατί οι υπολογισμοί γίνονται με υλικό φτιαγμένο για αυτό τον σκοπό. Μία πρόσθεση για παράδειγμα θα εκτελεσθεί σε ένα κύκλο ρολογιού στο FPGA. Στον υπολογιστή θα χρειαστεί πολύ περισσότερους κύκλους γιατί η πρόσθεση αυτή θα περιγραφεί από κάποια γλώσσα υψηλού επιπέδου και θα αντιστοιχηθεί στην εκτέλεση μίας σειράς εντολών. Κάθε μία από αυτές τις εντολές επίσης θα χρειάζεται ένα πλήθος από κύκλους ρολογιού του υπολογιστή για να ολοκληρωθεί.

Το αδύνατο σημείο της προσομοίωσης με χρήση της τεχνικής FPGA-In-the-Loop βρίσκεται στον χρόνο που απαιτείται για την επικοινωνία με τον υπολογιστή. Αν ο όγκος των δεδομένων που μεταφέρονται σε κάθε κύκλο προσομοίωσης είναι μεγάλος τότε ο χρόνος μεταφοράς θα υπερκαλύπτει το όποιο όφελος από την αυξημένη ταχύτητα των υπολογισμών. Είναι λοιπόν σημαντικό να χρησιμοποιούμε μέσο σύνδεσης που επιτρέπει μεγάλη ταχύτητα μεταφοράς δεδομένων. Από μόνη της η αλλαγή του μέσου επικοινωνίας από JTAG σε Ethernet είναι αρκετή για να μειώσει τον χρόνο προσομοίωσης με χρήση της τεχνικής FPGA-in-the-Loop κατά μία τάξη μεγέθους.

Σημαντικό ρόλο παίζει και η πολυπλοκότητα των υπολογισμών. Θα πρέπει στο FPGA να εκτελούνται πολλοί υπολογισμοί σε κάθε κύκλο προσομοίωσης ώστε να ελπίζουμε σε κάποιο όφελος από την χρήση της τεχνικής FPGA-In-the-Loop. Η μαζική μεταφορά δεδομένων με κοινόχρηστες μνήμες βοηθά σε αυτή την κατεύθυνση. Με αυτό τον τρόπο επικοινωνίας μεγάλο πλήθος δεδομένων μεταφέρεται και κατόπιν γίνεται η επεξεργασία τους σε ένα μόνο κύκλο προσομοίωσης.

Τέλος πρέπει να ληφθεί υπ' όψη ότι η δημιουργία μοντέλων με τα μπλοκ της Xilinx παρουσιάζει αυξημένη δυσκολία παρόλο που γίνεται σε ένα αρκετά υψηλό επίπεδο αφαίρεσης. Η διαδικασία συνολικά είναι ιδιαίτερα χρονοβόρα. Πολλά μπλοκ του Simulink δεν έχουν το αντίστοιχο της Xilinx και χρειάζεται να αντικατασταθούν από ένα σύνολο μπλοκ για να προσομοιωθεί η ίδια λειτουργία. Επίσης χρειάζεται αρκετός χρόνος για να συντεθεί το μοντέλο (μισή έως μία ώρα) και χρειάζεται και ένα μικρό χρονικό διάστημα (10-20 δευτερόλεπτα) για να προγραμματισθεί το FPGA και να αρχίσει η προσομοίωση. Θα πρέπει λοιπόν η εφαρμογή που θα επιλέξουμε να προσομοιώσουμε με την τεχνική FPGA-In-the-Loop να απαιτεί ένα πλήθος δοκιμών τέτοιο ώστε η αυξημένη ταχύτητα προσομοίωσης να μας αποζημιώσει για τον μεγάλο χρόνο δημιουργίας του hardware μοντέλου.

Βιβλιογραφία

1. Ψαράκης Μιχάλης, Κρανίτης Νεκτάριος, Γκιζόπουλος Δημήτρης: Ψηφιακή σχεδίαση: ενσωματωμένα συστήματα με VHDL υπό Ashenden Peter J.
2. System Generator for DSP Getting Started Guide.
3. System Generator for DSP User Guide.
4. Xilinx Application 1031: Decreasing Simulation Runtimes with System Generator for DSP Hardware Co-Simulation.
5. Xilinx Application 948: Hardware Acceleration of 3GPP Turbo Encoder/Decoder BER Measurements Using System Generator.
6. Xilinx Data Sheet 318: 3GPP Turbo Decoder v4.0
7. Xilinx Data Sheet 319: 3GPP Turbo Encoder v4.0
8. Xilinx Data Sheet 100: Virtex-5 Family Overview
9. Sayantan Choudhury, Modeling and Simulation of a Turbo Encoder and Decoder for Wireless Communication Systems.
10. Bernard Sklar, Fundamentals of Turbo Codes.
11. Claude Berrou, Alain Glavieux and Punya Thitimajshima, Near Shannon limit error – correcting coding and decoding: Turbo - Codes
12. Convolutional code, wikipedia, http://en.wikipedia.org/wiki/Convolutional_code
13. Turbo code, wikipedia,
14. Video Processing Acceleration Using FPGA-in-the-Loop, Mathworks, http://www.mathworks.com/help/hdlverifier/examples/video-processing-acceleration-using-fpga-in-the-loop.html
15. ISE Simulator (ISim), Xilinx, http://www.xilinx.com/tools/isim.htm
16. LabVIEW, National Instruments, http://www.ni.com/labview/