ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ:

ΣΧΕΔΙΑΣΗ & ΠΡΟΣΟΜΟΙΩΣΗ ΕΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΟΥ ΑΜΕΣΟΥ ΨΗΦΙΑΚΟΥ ΣΥΝΘΕΤΗ 12-bit, 2.8 GHz ΣΕ ΤΕΧΝΟΛΟΓΙΑ 65 nm

---

DESIGN & IMPLEMENTATION OF A PROGRAMMABLE 12-bit 2.8 GHz DIRECT DIGITAL SYNTHESIZER IN 65 nm TECHNOLOGY

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΑΛΚΙΒΙΑΔΗΣ ΧΑΤΖΟΠΟΥΛΟΣ

ΘΕΟΦΙΛΟΣ ΣΠΥΡΟΥ 8583

ΘΕΣΣΑΛΟΝΙΚΗ 2019

Πίνακας περιεχομένων

Ευρετήριο εικόνων

Ευρετήριο πινάκων

Επιτομή

Η παρούσα διπλωματική εργασία αφορά τη σχεδίαση και προσομοίωση ενός προγραμματιζόμενου Άμεσου Ψηφιακού Συνθέτη (Direct Digital Synthesizer) με μέγιστη συχνότητα χρονισμού στα $2{,}8\ GHz$. Η λειτουργία του έγκειται στην αναδημιουργία ενός αρμονικού σήματος ημιτόνου με ανάλυση συχνότητας έως και λίγο μεγαλύτερη από $10\ \mu Hz$, η οποία ισοδυναμεί με ακρίβεια 5 δεκαδικών ψηφίων της συχνότητας του σήματος που παράγεται. Η ακρίβεια εξόδου είναι $12\ bit$ και η ανάλυση φάσης $14\ bit$, που σημαίνει ότι το αποθηκευμένο σήμα αποτελείται από $2^{14} = 16.384$ διακριτές τιμές. Η τεχνολογία που χρησιμοποιείται είναι η $TSMC\ 65\ nm$.

Ο εν λόγω Άμεσος Ψηφιακός Συνθέτης μπορεί στη συνέχεια να τροφοδοτήσει έναν D/A Converter (Ψηφιακό σε Αναλογικό Μετατροπέα) για τη δημιουργία αναλογικού ημιτονοειδούς σήματος. Οι χρήσεις του είναι πολλαπλές με την πιο χαρακτηριστική να είναι η σύνθεση ήχου. Γενικά, αποτελεί πολύ καλή επιλογή ως πηγή συχνοτήτων χαμηλού θορύβου σε όλες τις εφαρμογές που απαιτείται η συχνή και γρήγορη μεταβολή της συχνότητας ενός σήματος.

Σκοπός της εργασίας είναι η σχεδίαση του συνολικού κυκλώματος, η προσομοίωση, σύνθεση και εξαγωγή του τελικού σχεδίου σε φυσικό επίπεδο, πληρώντας πάντα όλες τις προδιαγραφές. Μέσα από τη διαδικασία αυτή, αναλύονται όλα τα στάδια σχεδίασης ενός ολοκληρωμένου κυκλώματος, χρησιμοποιώντας τις πιο σύγχρονες τεχνικές.

Αφού πρώτα γίνει μια εισαγωγή των εργαλείων που χρησιμοποιούνται και παρουσιαστεί η θεωρία της Άμεσης Ψηφιακής Σύνθεσης, στη συνέχεια πραγματοποιείται ανάλυση του κυκλώματος και της αρχιτεκτονικής αυτού. Έπειτα, παρουσιάζονται τα βήματα και τα αποτελέσματα της σχεδίασης και της προσομοίωσης των επιμέρους κυκλωμάτων μαζί με αναφορές σχετικά με το χρόνο εκτέλεσης, την καταναλισκόμενη ενέργεια και την επιφάνεια ολοκλήρωσης.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή μου και επιβλέποντα της διπλωματικής μου εργασίας, κ. Χατζόπουλο Αλκιβιάδη, για την εμπιστοσύνη που μου έδειξε καθ' όλη τη διάρκεια εκπόνησης της εργασίας, για τη βοήθεια που μου έδωσε σε κρίσιμα σημεία, αλλά και την ευκαιρία που μου έδωσε να εντρυφήσω πάνω στο αντικείμενο που μου αρέσει περισσότερο. Οι γνώσεις που απέκτησα σε συνδυασμό με την καθοδήγησή του, θεωρώ πως αποτελούν εφόδιο όχι μόνο για την επιτυχή ολοκλήρωση της εργασίας, αλλά και για τη μετέπειτα σταδιοδρομία μου στο χώρο της σχεδίασης κυκλωμάτων.

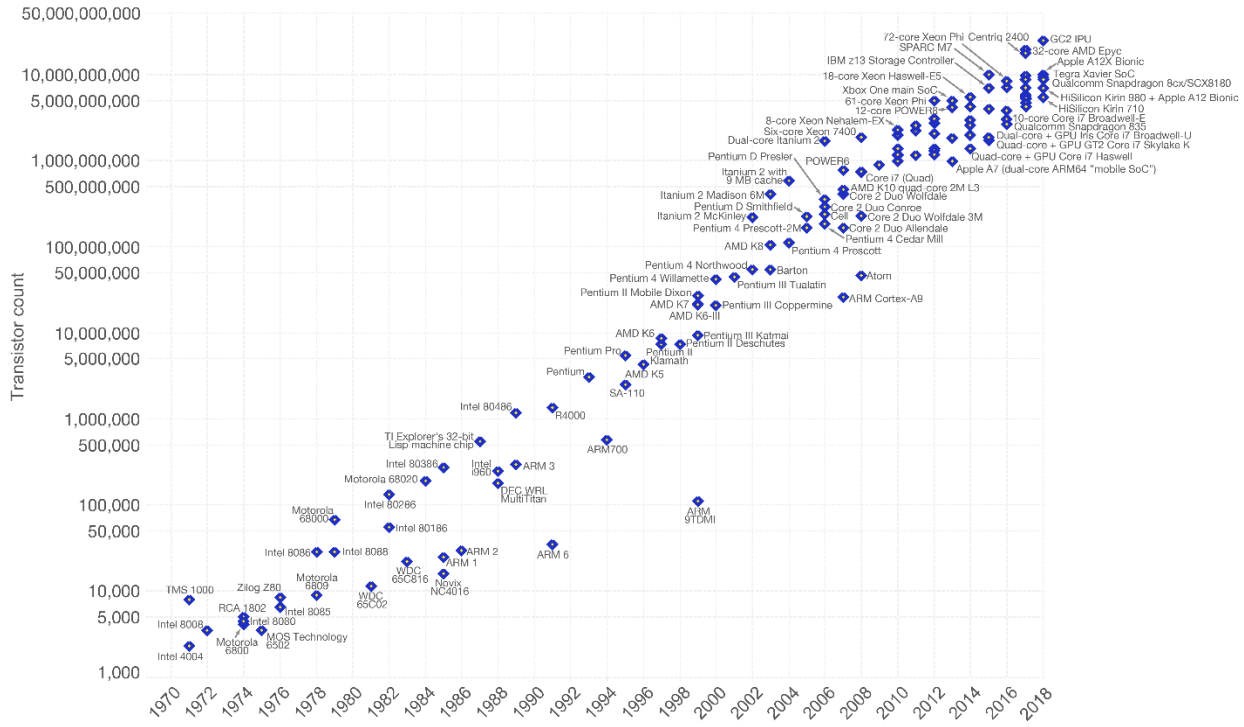
Θα ήθελα ακόμη να ευχαριστήσω τον μεταπτυχιακό φοιτητή – υποψήφιο διδάκτορα του τμήματός μας, Κατσέλα Λεωνίδα, του οποίου η συμβολή και βοήθεια ήταν ιδιαίτερα καθοριστική σε κρίσιμα σημεία της εργασίας. Η εμπειρία του στη σχεδίαση κυκλωμάτων βοήθησε ώστε να αντιμετωπιστούν διάφορα προβλήματα που προέκυψαν κατά την εκπόνηση της εργασίας.

1 Ψηφιακή σχεδίαση

1.1 Εισαγωγή

Το 1958, ο Jack Kilby κατασκεύασε το πρώτο ολοκληρωμένο κύκλωμα flip-flop με δύο τρανζίστορ στην Texas Instruments. Έξι δεκαετίες αργότερα, ο μικροεπεξεργαστής EPYC Rome της AMD περιέχει περίπου 32 δισεκατομμύρια τρανζίστορ, το FPGA Versal / Everest της Xilinx 50 δισεκατομμύρια τρανζίστορ, ενώ μία μνήμη flash των 8 Tb της Samsung περιέχει πάνω από 2 τρισεκατομμύρια τρανζίστορ. Εκτός από το πλήθος των τρανζίστορ, εντυπωσιακό είναι και το μέγεθος αυτών, το οποίο έχει φτάσει μόλις τα 5 nm, αλλά και η επιφάνεια ολοκλήρωσης, η οποία παραμένει επίσης πολύ μικρή. Χαρακτηριστικό παράδειγμα αποτελεί ο Snapdragon 855 της Qualcomm, ο οποίος κυκλοφόρησε το 2018 με πάνω από 5 δισεκατομμύρια τρανζίστορ σε μία επιφάνεια περίπου 73 mm^2 και κατανάλωση μόλις μερικών watt.

Παρατηρείται λοιπόν μία εντυπωσιακή ανάπτυξη, η οποία οφείλεται τόσο στη συνεχή σμίκρυνση των τρανζίστορ όσο και στις βελτιώσεις στην τεχνολογία κατασκευής, όπως η υιοθέτηση τεχνικών σχεδίασης 3D ολοκληρωμένων κυκλωμάτων και χρησιμοποιώντας έως και 96 επίπεδα ολοκλήρωσης. Σε αντίθεση με άλλους κλάδους της μηχανικής όπου συχνά γίνονται συμβιβασμοί, στην μικροηλεκτρονική καθώς τα τρανζίστορ μικραίνουν σε μέγεθος, γίνονται επίσης ταχύτερα, καταναλώνουν λιγότερη ισχύ και έχουν μικρότερο κόστος κατασκευής. Η σύμπραξη όλων αυτών των τεχνολογιών και των χαρακτηριστικών δεν έχει επιφέρει επανάσταση μόνο στην ηλεκτρονική, αλλά και στην κοινωνία γενικότερα.



Εικόνα 1-1 Πλήθος τρανζίστορ μικροεπεξεργαστών (λογαριθμική κλίμακα) [Δ1]
Κάθε περίπου 2 χρόνια παρατηρείται διπλασιασμός των ολοκληρωμένων τρανζίστορ.

Η παραπάνω εξέλιξη είναι χαρακτηριστικό των ψηφιακών ολοκληρωμένων κυκλωμάτων και οφείλεται κυρίως σε διάφορους αυτοματισμούς, ομαδοποιήσεις και επαναχρησιμοποιήσεις κυκλωμάτων. Το πλεονέκτημα αυτό του ψηφιακού κόσμου έγινε αντιληπτό από πολύ νωρίς στην επιστημονική κοινότητα και έτσι ο κλάδος γνώρισε τεράστια άνθιση σε μόλις λίγες δεκαετίες. Εν αντιθέσει, η αναλογική σχεδίαση δεν περιέχει πολλά περιθώρια αυτοματοποίησης και ο ρόλος του μηχανικού ξεκινά από πολύ χαμηλό επίπεδο, ενώ πρέπει να έχει εμπειρία ώστε να ορίζει και να μεταβάλει αντίστοιχα τις παραμέτρους του υπό σχεδίαση κυκλώματος.

Για να γίνουν όλα αυτά πραγματικότητα, τεράστιο ρόλο έπαιξαν οι ηλεκτρονικοί υπολογιστές και τα προγράμματα σχεδίασης και ελέγχου που υλοποιήθηκαν για αυτούς. Σε συνδυασμό με τα λογισμικά αυτά, αναπτύχθηκαν και νέες γλώσσες περιγραφής υλικού (hardware description languages), οι οποίες επέτρεψαν τη δημιουργία κυκλωμάτων μέσω λεκτικής περιγραφής. Έτσι, είτε πρόκειται για το πιο απλό κύκλωμα που αποτελείται από μερικές δεκάδες τρανζίστορ, είτε πρόκειται για έναν σύνθετο μικροεπεξεργαστή, η λογική σχεδίασης είναι παρόμοια. Ξεκινώντας με τη συγγραφή του κώδικα που περιγράφει το κύκλωμα, ο μηχανικός στη συνέχεια είναι σε θέση να υλοποιήσει, να επαληθεύσει και να σχεδιάσει σε φυσικό επίπεδο το ζητούμενο κύκλωμα.

1.2 Γλώσσες περιγραφής υλικού (Hardware Description Languages, HDL)

Οι γλώσσες περιγραφής υλικού (Hardware Description Languages, HDL) αποτελούν μία οικογένεια γλωσσών προγραμματισμού, οι οποίες χρησιμοποιούνται για την περιγραφή και σχεδίαση ηλεκτρονικών κυκλωμάτων και συνηθέστερα ψηφιακών. Επιτρέπουν την περιγραφή της λειτουργίας, της σχεδίασης και της οργάνωσης του κυκλώματος, μαζί με δοκιμές που επαληθεύουν τη λειτουργία του μέσω προσομοίωσης.

Αντιθέτως με τις συμβατικές γλώσσες προγραμματισμού, οι οποίες ορίζουν μία ακολουθία εντολών για την επίτευξη ενός αλγορίθμου, οι HDL γλώσσες δεν υλοποιούν αλγορίθμους, δηλαδή δεν «τρέχουν» σε κάποιο σύστημα, αλλά δημιουργούν ένα σύστημα. Μπορεί κανείς να πει ότι οι συμβατικές γλώσσες προγραμματισμού περιγράφουν τις ενέργειες που πρέπει να κάνει ένα σύστημα (π.χ. μία CPU) για να πετύχει τα επιθυμητά αποτελέσματα. Από την άλλη, οι γλώσσες περιγραφής υλικού περιγράφουν το ίδιο το σύστημα, δηλαδή τα επιμέρους κυκλώματα που πρέπει να υλοποιηθούν, ώστε να επιτευχθούν τα επιθυμητά αποτελέσματα.

Η λειτουργία του υλικού (hardware) μπορεί να αναπαρασταθεί και με παραδοσιακές γλώσσες προγραμματισμού σε συνδυασμό πάντα με εκτεταμένες και δύσχρηστες βιβλιοθήκες. Το βασικό πρόβλημα είναι ότι οι κλασικές γλώσσες προγραμματισμού δεν περιέχουν δυνατότητες έκφρασης του χρόνου κι έτσι ακόμα και με την εισαγωγή επιπρόσθετων βιβλιοθηκών, δεν μπορούν να λειτουργήσουν αποτελεσματικά ως γλώσσες περιγραφής υλικού. Επίσης, καλό είναι να αποφεύγονται τέτοιου είδους πρακτικές, καθώς η λογική ενός κώδικα γλώσσας προγραμματισμού είναι τελείως διαφορετική από αυτή ενός που έχει γραφεί με HDL, διότι οι δύο κατηγορίες γλωσσών έχουν δημιουργηθεί για να εξυπηρετούν διαφορετικούς σκοπούς και έτσι μπορεί να προκύψουν διάφορα λάθη και δυσνόητο περιεχόμενο.

Οι κύριες εφαρμογές των γλωσσών περιγραφής υλικού είναι η δημιουργία κυκλωμάτων σε ASIC (Application Specific Integrated Circuit) και η αναδιάταξη των πινάκων λογικών πυλών και των υπόλοιπων ψηφιακών τμημάτων ενός FPGA (Field Programmable Gate Array). Η κύρια διαφορά μεταξύ των ASIC και των FPGA έγκειται στη η δυνατότητα επαναχρησιμοποίησης του υλικού. Με άλλα λόγια, σε ένα FPGA μπορούμε να «σβήσουμε» ένα παλιό κύκλωμα που σχεδιάσαμε και στη θέση του να υλοποιήσουμε ένα νέο, χρησιμοποιώντας πάντα το ίδιο ολοκληρωμένο. Τα ASIC δεν παρέχουν δυνατότητα «διαγραφής» και επομένως πρέπει να είμαστε πολύ προσεκτικοί πριν προβούμε στη δημιουργία τους, καθώς το κόστος τους είναι ιδιαίτερα υψηλό.

Η ιστορία των γλωσσών περιγραφής υλικού ξεκινά το 1977 με τις ISP και KARL. Στην πορεία αναπτύχθηκαν και άλλες γλώσσες, αλλά δεν ήταν μέχρι το 1984 όταν και δημιουργήθηκε η Verilog, η οποία ήταν η πρώτη HDL όπως τις γνωρίζουμε σήμερα. Στο πέρασμα των χρόνων έγιναν πολλές προσπάθειες με νέες γλώσσες που χρησιμοποιούσαν νέες τεχνικές και προσπαθούσαν να καλύψουν όσα κενά άφηναν οι προηγούμενές τους. Έχουν ακόμη δημιουργηθεί και HDL γλώσσες που αφορούν στην αναλογική ή μεικτή σχεδίαση ολοκληρωμένων κυκλωμάτων, αλλά βρίσκονται σε αρχικά στάδια και δεν χρησιμοποιούνται ευρέως. Οι δύο σύγχρονες γλώσσες περιγραφής υλικού που χρησιμοποιούνται μέχρι σήμερα είναι οι VHDL και Verilog.

1.2.1 VHDL

Η VHDL ή VHSIC (Very High-Speed Integrated Circuit Hardware Description Language), όπως είναι η πλήρης της ονομασία, είναι μία γλώσσα η οποία αναπτύχθηκε στις αρχές της δεκαετίας του ’80 από το υπουργείο αμύνης των Η.Π.Α. Ο σκοπός δημιουργίας της ήταν η καθολική τεκμηρίωση των ASIC κυκλωμάτων που προμηθευόταν οι Η.Π.Α. για τον εξοπλισμό του αμυντικού τους συστήματος. Έτσι, αντικατέστησε τα μακροσκελή και πολύπλοκα εγχειρίδια των κατασκευαστών, οι οποίοι μέχρι τότε δεν είχαν κοινό τρόπο περιγραφής και βασίζονταν στις λεπτομέρειες της εκάστοτε υλοποίησης.

Η καθιέρωσή της ως πρότυπο από την IEEE έγινε από την πρώτη κιόλας έκδοση της γλώσσας (IEEE 1076-1987). Η έκδοση αυτή περιείχε αρκετούς τύπους δεδομένων, όπως ακεραίους (integer) και πραγματικούς (real) αριθμούς, λογικούς (bit και Boolean), χαρακτήρες (character) και χρόνο (time). Υποστήριζε ακόμη και πίνακες bit (bit_vector) και χαρακτήρων (string) για την εύκολη αναπαράσταση διανυσμάτων bit ή data bus και αλφαριθμητικών ακολουθιών, αντίστοιχα. Ένα συχνά εμφανιζόμενο πρόβλημα στην ψηφιακή λογική, που όμως παρέμεινε άλυτο στην πρώτη έκδοση, ήταν αυτό της «λογικής πολλαπλών τιμών», όπου εκτός από τις καταστάσεις 0 και 1, υπάρχουν περισσότερες (π.χ. για το σθένος οδήγησης ενός σήματος, άγνωστες τιμές κ.α.).

Έτσι, 6 χρόνια αργότερα, δημιουργήθηκε το ανανεωμένο IEEE 1076-1993, όπου προστέθηκαν τα παραπάνω χαρακτηριστικά και επιπλέον υποστήριζε τον βαθμωτό τύπο std_ulogic μαζί με την ανυσματική του έκδοση std_ulogic_vector. Επίσης, η σύνταξη έγινε πιο συνεπής και υπήρχε μεγαλύτερη ευελιξία στη χρήση ονομάτων. Μέσα στην επόμενη δεκαετία έλαβαν χώρα διάφορες μικρότερες αλλαγές (το 2000 και 2002), προσθέτοντας την ιδέα των προστατευμένων

τύπων (παρόμοια με τη C++) και αφαιρώντας κάποιους περιορισμούς από τους κανόνες αντιστοίχισης θυρών (port mapping rules).

Τον Ιούνιο του 2006, η γλώσσα αναθεωρήθηκε, διατηρώντας όμως πλήρη συμβατότητα με όλες τις παλαιότερες εκδόσεις. Το νέο πρότυπο παρείχε διάφορες επεκτάσεις που διευκόλυναν τη συγγραφή και τη διαχείριση κώδικα σε VHDL. Οι πιο βασικές αλλαγές έγιναν με την ενσωμάτωση θυγατρικών προτύπων (1164, 1076.2, 1076.3) στο βασικό πρότυπο 1076. Μερικές από αυτές ήταν η διεύρυνση του συνόλου τελεστών, πιο ευέλικτη χρήση των εντολών case και generate, ενσωμάτωση του VHPI (διασύνδεση προς τις γλώσσες C και C++) και ένα υποσύνολο της PSL (Property Specification Language). Οι αλλαγές αυτές βελτίωσαν την ποιότητα του παραγόμενου κώδικα VHDL, έκαναν τα testbenches πιο ευέλικτα και επέτρεψαν τη χρήση της VHDL σε ένα ευρύτερο πεδίο.

Μέσα από την έκδοση του 2006 εντοπίστηκαν αρκετά προβλήματα, η πλειοψηφία των οποίων και διορθώθηκε δύο χρόνια αργότερα με την αναθεώρηση σε VHDL 2008, της οποίας η προτυποποίηση έγινε έναν χρόνο μετά, τον Ιανουάριο του 2009, με το όνομα IEEE 1076-2008. Έκτοτε λίγες αναθεωρήσεις έχουν προκύψει με την πιο πρόσφατη να είναι το 2011. Εντός του έτους αναμένεται νέα έκδοση VHDL-2019, η οποία και ξεκίνησε να υλοποιείται από το 2017. Οι αλλαγές της τελευταίας έκδοσης θα αφορούν κυρίως προσθήκες εμπνευσμένες από γλώσσες προγραμματισμού πολύ υψηλού επιπέδου, ώστε να διευκολυνθεί ο προγραμματιστής. Μερικές από αυτές περιλαμβάνουν την προσθήκη “Garbage Collector” για την απόρριψη μη χρησιμοποιούμενων στοιχείων και δυνατότητα κοινής χρήσης μεταβλητών μεταξύ διαφορετικών οντοτήτων.

Η συνήθης χρήση της VHDL αφορά τη συγγραφή μοντέλων σε κείμενο που περιγράφουν ένα ψηφιακό ή μεικτό κύκλωμα. Προτού ένα πρόγραμμα σύνθεσης επεξεργαστεί το μοντέλο που περιγράφεται, χρησιμοποιείται ένα πρόγραμμα προσομοίωσης για να δοκιμαστεί η λογική σχεδίαση και να επαληθευτεί η ορθή, επιθυμητή λειτουργία της. Για το σκοπό αυτό χρησιμοποιούνται μοντέλα προσομοίωσης, τα λεγόμενα testbenches, τα οποία αναπαριστούν τα αντίστοιχα λογικά κυκλώματα και ορίζουν διάφορα σενάρια χρήσης, ώστε να αναλυθεί η συμπεριφορά του κυκλώματος.

Η VHDL υποστηρίζεται από όλα τα μεγάλα περιβάλλοντα ανάπτυξης. Μερικά από αυτά είναι το Xilinx ISE για υλοποίηση σε FPGA, το Altera Quartus, το Synopsys Synplify και το Mentor Graphics HDL Designer. Έπειτα, το παραγόμενο σχηματικό διάγραμμα μπορεί να επαληθευτεί με χρήση λογισμικού προσομοίωσης, όπου παρουσιάζονται οι κυματομορφές εισόδων και εξόδων του κυκλώματος. Για να επιτευχθεί αυτό, είναι αναγκαία η δημιουργία του σωστού testbench.

Πλέον η VHDL έχει επικρατήσει στην βιομηχανία για την περιγραφή, μοντελοποίηση και προσομοίωση ψηφιακών κυκλωμάτων. Ορισμένα από τα πλεονεκτήματα της VHDL είναι τα εξής:

1. **Προτυποποίηση:** Η VHDL, όπως αναφέρθηκε και παραπάνω, είναι πρότυπο της IEEE. Αυτό σημαίνει ότι μειώνεται η πιθανότητα να υπάρχει ασυμβατότητα μεταξύ διαφορετικών εκδόσεων.
2. **Δομημένη περιγραφή:** Οι περιγραφές που γίνονται σε VHDL είναι δομημένες. Επομένως ο προγραμματιστής μπορεί να επικεντρωθεί σε κάθε τμήμα της σχεδίασης ξεχωριστά και στη συνέχεια να ασχοληθεί με τον τρόπο διασύνδεσης των επιμέρους τμημάτων.
3. **Δυνατότητα σχεδίασης σε πολλά επίπεδα:** Η VHDL μπορεί να μοντελοποιήσει το υλικό σε πολλά ιεραρχικά επίπεδα ξεκινώντας από αυτό των λογικών πυλών έως το αλγοριθμικό επίπεδο. Τα επίπεδα μπορούν να συνδυαστούν, δίνοντας έτσι μεγαλύτερη ευελιξία.

1.2.2 Verilog

Η Verilog είναι ακόμη μία γλώσσα περιγραφής υλικού η οποία χρησιμοποιείται ευρέως από τους σχεδιαστές κυκλωμάτων. Όπως και η VHDL, χρησιμοποιείται για τη σχεδίαση ψηφιακών κυκλωμάτων σε επίπεδο RTL (Register Transfer Level), επίπεδο συμπεριφοράς (αλγοριθμικό) και επίπεδο πύλης. Ακόμη αποτελεί επιλογή για την επαλήθευση αναλογικών και μεικτών κυκλωμάτων, καθώς επίσης και σχεδιασμού γενικών κυκλωμάτων.

Αυτό που έλειπε από την οικογένεια των HDL γλωσσών μέχρι και το 1984 που εισήχθη η Verilog, ήταν μία γλώσσα που να μοιάζει συντακτικά με τη διάσημη και ευρέως χρησιμοποιούμενη εκείνη την εποχή, C. Ο λόγος που οδήγησε στην ανάγκη αυτή, ήταν η διευκόλυνση που παρείχε στους σχεδιαστές κυκλωμάτων, οι οποίοι ήδη χρησιμοποιούσαν λογισμικό σχεδίασης με γραφικό περιβάλλον και προγράμματα προσομοίωσης ηλεκτρονικών κυκλωμάτων. Έτσι, αμέσως υπήρξε μία εντυπωσιακή βελτίωση της παραγωγικότητας των σχεδιαστών από την πρώτη κιόλας στιγμή εμφάνισης της Verilog.

Η σύνταξη λοιπόν της Verilog μοιάζει πολύ με αυτή της C, έχοντας δανειστεί πολλές ιδέες από αυτή, όπως η χρήση των preprocessors και οι δομές ελέγχου και επανάληψης (if/else, for, while, case, κτλ.), ενώ οι τελεστές της ακολουθούν την ίδια προτεραιότητα. Υπάρχουν βέβαια και διαφορές μεταξύ των δύο, όπως είναι η χρήση των λέξεων begin/end αντί για τις παρενθέσεις { } στην C κατά τον προσδιορισμό των ορίων των διαφόρων τμημάτων (blocks). Η Verilog επίσης απαιτεί οι μεταβλητές να έχουν ένα ορισμένο μέγεθος, σε αντίθεση με τη C όπου συνήθως το μέγεθος ορίζεται από το σύστημα στο οποίο τρέχει το πρόγραμμα. Το συγκεκριμένο χαρακτηριστικό ενισχύει την ανάγκη διαχωρισμού των γλωσσών περιγραφής υλικού από τις κλασικές γλώσσες προγραμματισμού, μιας και για παράδειγμα οι είσοδοι και οι έξοδοι ενός κυκλώματος είναι πάντα σαφώς ορισμένα μεγέθη και εξαρτώνται από την εκάστοτε εφαρμογή.

Ένα σχέδιο σε Verilog συνίσταται από μία ιεραρχία μονάδων (modules). Οι μονάδες υπακούν στην ιεραρχία του συστήματος και επικοινωνούν μεταξύ τους μέσω ενός συνόλου καθορισμένων εισόδων, εξόδων ή αμφίδρομων θυρών. Στο εσωτερικό αυτών μπορεί να υπάρχει οποιοσδήποτε συνδυασμός από συνδέσεις, καταχωρητές, μεταβλητές (wire, reg, integer κ.α.), παράλληλα και ακολουθιακά τμήματα κώδικα, καθώς και άλλες μονάδες κατώτερης ιεραρχίας. Όλα τα τμήματα κώδικα στη Verilog τρέχουν παράλληλα μεταξύ τους, γεγονός που την καθιστά ως γλώσσα ροής δεδομένων (dataflow language).

Ο αγωγός (wire) της Verilog μπορεί να λαμβάνει οποιαδήποτε από τις πιθανές λογικές τιμές (0, 1, Z και X) και ταυτόχρονα να χαρακτηρίζει και το σθένος οδήγησης σήματος (ισχυρό, αδύναμο κτλ.). Με τον τρόπο αυτό, επιτρέπεται η δυνατότητα μοντελοποίησης κοινών γραμμών, στις οποίες συνδέονται πολλαπλές πηγές που οδηγούν μία σύνδεση (net).

Σημειώνεται πως μόνο ένα υποσύνολο των εντολών της Verilog είναι συνθέσιμο, δηλαδή αντιστοιχεί και μπορεί να δημιουργήσει πραγματικές οντότητες κυκλωμάτων. Μόνο οι μονάδες που έχουν σχεδιαστεί με βάση το επίπεδο RTL μπορούν να υλοποιηθούν σε φυσικό σχέδιο. Το λογισμικό μετατρέπει τον πηγαίο κώδικα της Verilog σε αρχείο netlist, μια λογικά ισοδύναμη περιγραφή αποτελούμενη μόνο από στοιχειώδη λογικά τμήματα (AND, OR, NOT, flip-flops, κτλ.), που είναι διαθέσιμα από την τεχνολογία FPGA ή VLSI που χρησιμοποιούμε. Περαιτέρω χειρισμός στο netlist τελικά οδηγεί σε ένα σχέδιο κατασκευής κυκλώματος (όπως μια μάσκα για ASIC ή ένα αρχείο bitstream για ένα FPGA).

Η Verilog, έχοντας δημιουργηθεί στις αρχές του 1984, ήταν η πρώτη σύγχρονη γλώσσα περιγραφής υλικού που εφευρέθηκε. Η αρχική ονομασία της ήταν "Automated Integrated Design Systems", η οποία ένα χρόνο αργότερα μετονομάστηκε σε “Gateway Design Automation”. Το Gateway Design Automation αγοράστηκε από την Cadence Design Systems το 1990. Η Cadence έχει τα πλήρη δικαιώματα των Gateway Verilog και Verilog-XL, καθώς και του HDL προσομοιωτή που θα γινόταν αργότερα ο πιο διαδεδομένος προσομοιωτής της δεκαετίας. Αρχικά, η Verilog ξεκίνησε για την αποκλειστική περιγραφή και προσομοίωση κυκλωμάτων, απ’ όπου και πήρε την ονομασία της από τις λέξεις “verification” και “logic”. Η δυνατότητα σύνθεσης προστέθηκε αργότερα.

Λίγα χρόνια μετά, και ενώ η VHDL γνώριζε όλο και μεγαλύτερη επιτυχία, η Cadence αποφάσισε να ανοίξει τη Verilog για προτυποποίηση. Έτσι, την έκανε ευρέως διαθέσιμη υπό τον Ανοιχτό Διεθνή Οργανισμό Verilog (Open Verilog International Organization), ο οποίος είναι πλέον γνωστός ως Accellera. Το 1995 η Verilog υπεβλήθη στην IEEE και δημιουργήθηκε το πρότυπο IEEE 1364 ή αλλιώς Verilog-95.

Το 2001 προέκυψε μία σημαντική αναβάθμιση στην Verilog-95 για να καλύψει όλες τις αδυναμίες που είχαν παρατηρηθεί. Αρχικά, προστέθηκε η υποστήριξη προσημασμένων (συμπλήρωμα του 2) συνδέσεων και μεταβλητών. Μέχρι πρότινος, οι σχεδιαστές έπρεπε να μεταβάλλουν τα bit των μεταβλητών, ώστε να πραγματοποιήσουν προσημασμένες πράξεις. Μία άλλη σημαντική προσθήκη, ήταν οι δεσμευμένες λέξεις για τον έλεγχο της ροής του προγράμματος (case, if, else). Επιπλέον, μερικές προσθήκες έγιναν με γνώμονα την καλύτερη αναγνωσιμότητα του κώδικα. Η Verilog-2001 είναι η κυρίαρχη έκδοση και υποστηρίζεται από την πλειονότητα των πακέτων λογισμικού σχεδίασης EDA.

Η τελευταία αναθεώρηση της Verilog, προτού προκύψει ο απόγονός της SystemVerilog, έγινε το 2005 και αποτελούταν από μικρές διορθώσεις και προσθήκες όπως η εισαγωγή της δεσμευμένης λέξης “uwire”. Ταυτόχρονα παρουσιάζεται ένα νέο πρότυπο, το Verilog-AMS, το

οποίο αποτελεί μία προσπάθεια μοντελοποίησης αναλογικών και μεικτών κυκλωμάτων, χρησιμοποιώντας την κλασική Verilog.

Η άνθιση νέων γλωσσών επαλήθευσης υλικού (διαφορετικές από τις «περιγραφής υλικού») οδήγησε στη δημιουργία μίας νέας γλώσσας, της SystemVerilog. Η τελευταία αποτελεί ένα υπερσύνολο της Verilog-2005 με πολλά νέα χαρακτηριστικά και δυνατότητες που ενισχύουν την επαλήθευση και μοντελοποίηση του κυκλώματος. Από το 2009, η SystemVerilog και η Verilog συγχωνεύθηκαν σε ένα κοινό πρότυπο, το SystemVerilog 2009 (IEEE 1800-2009). Η πιο πρόσφατη έκδοση έχει δημιουργηθεί το 2017.

1.2.3 Επιλογή γλώσσας

Τόσο η VHDL όσο και η Verilog αποτελούν τις δύο πιο συχνές επιλογές των σχεδιαστών κυκλωμάτων σήμερα. Και οι δύο παρέχουν αντίστοιχες δυνατότητες στους προγραμματιστές, ενώ έχουν κοινή λογική για την αντίληψη και παρουσίαση ενός κυκλώματος. Οι διαφορές τους λοιπόν, εκτός από τη σύνταξη και τις δεσμευμένες λέξεις (π.χ. το entity της VHDL με το module της Verilog που είναι αντίστοιχα), έγκεινται πιο πολύ σε λεπτομέρειες και κυρίως στη φιλοσοφία που υιοθετούν. Για παράδειγμα, η Verilog είναι πολύ πιο συνοπτική σε σχέση με τη VHDL, η οποία περιέχει πολλές δεσμευμένες λέξεις και συνήθως χρειάζεται περισσότερο κώδικα σε σχέση με τη Verilog για την περιγραφή του ίδιου κυκλώματος.

Η γλώσσα που επιλέχθηκε για την εκπόνηση της εργασίας αυτής είναι η Verilog για διάφορους λόγους. Αρχικά, κατά την έναρξη της εργασίας, γνώριζα ήδη VHDL την οποία είχα χρησιμοποιήσει σε παλαιότερες εργασίες. Έτσι, θεώρησα πως ήταν μία πολύ καλή ευκαιρία να μάθω και την «αντίπαλη» γλώσσα και ταυτόχρονα να δω πόσο καλά θα μπορούσα να ανταπεξέλθω στις απαιτήσεις της. Από την πρώτη στιγμή που ξεκίνησα να τη μαθαίνω, μου φάνηκε αρκετά πιο εύκολη τόσο στην εκμάθησή της, όσο και στη χρήση της. Αυτό διότι το συντακτικό της είναι εμπνευσμένο από τη C, με την οποία είμαι πολύ εξοικειωμένος, και επίσης γιατί είναι πιο λιτή και δεν απαιτεί την απομνημόνευση μακροσκελών κανόνων.

Επίσης, η Verilog είναι προϊόν της Cadence και επομένως τα εργαλεία της εταιρείας περιστρέφονται γύρω από αυτή και παρέχουν μεγαλύτερη υποστήριξη σε σχέση με τη VHDL. Χαρακτηριστικό παράδειγμα είναι ότι δεν υποστηρίζονται όλες οι λειτουργίες των πιο πρόσφατων εκδόσεων της VHDL, ενώ κάποιες βιβλιοθήκες (ακόμα και της IEEE) δεν είναι πλήρως συνθέσιμες στο Cadence.

Έτσι, ξεκινώντας σαν πρόκληση, διαπίστωσα πολλά σημεία υπεροχής της Verilog σε σχέση με τη VHDL. Επομένως τη θεωρώ την καλύτερη επιλογή για τη διπλωματική μου εργασία, χωρίς πάντα να υποβαθμίζεται ο ρόλος και η αξία της VHDL.

1.3 Εργαλεία

1.3.1 Incisive Simulator

Το Incisive είναι ένα σύνολο εργαλείων από την Cadence που εξυπηρετεί στη σχεδίαση ενός κυκλώματος και την προσομοίωση για την επαλήθευση (verification) της επιθυμητής

λειτουργίας του. Συνδυάζει υψηλή απόδοση κατά την προσομοίωση του κώδικα, ακρίβεια και ευελιξία, ενώ παρέχει δυνατότητες εντοπισμού λογικών σφαλμάτων που μπορεί να υπάρχουν στη σχεδίαση.

Ανάλογα με τις ανάγκες της εκάστοτε σχεδίασης, το Incisive παρέχει μια πληθώρα εργαλείων, τα οποία παρουσιάζονται συνοπτικά στον παρακάτω πίνακα:

Εργαλείο	Περιγραφή
NC Verilog	Μεταγλωττιστής για Verilog 95, Verilog 2001, SystemVerilog και Verilog-AMS.
NC VHDL	Μεταγλωττιστής για VHDL 87 και VHDL 93.
NC SystemC	Μεταγλωττιστής για SystemC.
NC Elaborator	Ενιαίος linker και elaborator για βιβλιοθήκες των Verilog, VHDL και SystemC.
NC Sim	Ενιαίος προσομοιωτής για Verilog, VHDL και SystemC.
Irun	Συνδυαστική χρήση των εργαλείων. Πρώτα καλείται ο κατάλληλος μεταγλωττιστής και έπειτα το NC Elaboration και το NC Sim.
Sim Vision	Πρόγραμμα με γραφικό περιβάλλον για την προβολή κυματομορφών και παρακολούθηση λογικών σημάτων.

Πίνακας 1-1 Εργαλεία Incisive

Δεδομένου πως η σχεδίαση του κυκλώματος της εργασίας γίνεται σε γλώσσα Verilog, θα χρησιμοποιηθούν τα εργαλεία NC Verilog, NC Elaborator, NC Sim για τη σχεδίαση και προσομοίωση, ενώ τα αποτελέσματα και η λογική ανάλυση θα παρουσιαστούν με το Sim Vision.

1.3.2 Genus Synthesis Solution

Το Cadence Genus είναι ένα γρήγορο, υψηλής χωρητικότητας εργαλείο, που προσφέρεται για τη σύνθεση απαιτητικών σχεδιάσεων ολοκληρωμένων κυκλωμάτων. Η καινοτόμος τεχνολογία που χρησιμοποιεί, παράγει ανώτερη λογική και δομές διασύνδεσης για φυσικά σχέδια κλίμακας νανομέτρων. Με το Genus παράγονται σχέδια επεξεργαστών, γραφικών και δικτυακών εφαρμογών. Η σύνθεση που πραγματοποιεί, εκτελείται αρκετά γρήγορα. Επιπροσθέτως, βελτιώνει την παραγωγικότητα του σχεδιαστή, αφού απλοποιεί τον προσδιορισμό των περιορισμών και επιτρέπει τη χρήση scripts.

Το στάδιο της σύνθεσης του κυκλώματος, που πραγματοποιεί το Genus, αποτελεί αναπόσπαστο κομμάτι για την κατασκευή τού τελικού ολοκληρωμένου κυκλώματος. Η γενική ροή της διαδικασίας της σύνθεσης ξεκινά με την εισαγωγή των βιβλιοθηκών της τεχνολογίας που

χρησιμοποιείται, ώστε να μπορεί το κύκλωμα να αντιστοιχιστεί σε πραγματικά στοιχεία (λογικές πύλες, flip-flop κ.α.) τα οποία έχουν πεπερασμένες καθυστερήσεις, χωρητικότητες και γενικά δεν είναι ιδανικά.

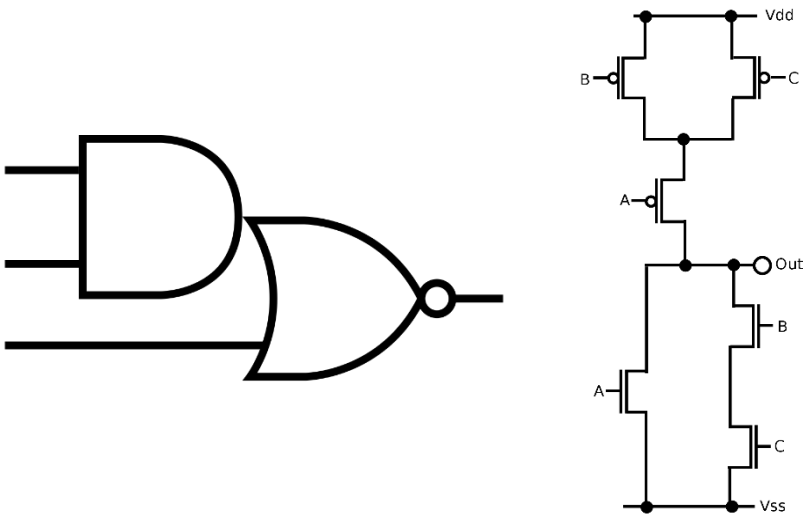
Στη συνέχεια ορίζονται όλοι οι περιορισμοί που πρέπει να πληροί το κύκλωμα. Χαρακτηριστικό παράδειγμα περιορισμού είναι η συχνότητα, ή αντίστοιχα η περίοδος, του ρολογιού σε ένα ακολουθιακό κύκλωμα. Μετά τη δήλωση των περιορισμών, το επόμενο βήμα είναι η φόρτωση των αρχείων HDL.

Αφού το Genus ελέγξει τον κώδικα για συντακτικά λάθη, ξεκινά η προεπεξεργασία του, ή αλλιώς το λεγόμενο elaboration. Κατά το elaboration, το οποίο είναι απαραίτητο στάδιο πριν τη σύνθεση, λαμβάνουν χώρα βελτιστοποιήσεις και διάφοροι έλεγχοι της σχεδίασης. Οι διεργασίες που εκτελεί το elaboration μεταξύ άλλων είναι ο συμπερασμός των registers του σχεδίου (δηλαδή σε ποια σημεία χρειάζεται ένα register)[1], βελτιστοποιήσεις υψηλού επιπέδου στον κώδικα (π.χ. εύρεση dead code, δηλαδή σημείων που δεν αξιοποιούνται ποτέ), έλεγχος ότι όλα τα απαιτούμενα δομικά στοιχεία και οι λειτουργίες των modules μπορούν να υλοποιηθούν με τη βιβλιοθήκη που χρησιμοποιείται (π.χ. εάν η σχεδίαση περιέχει μνήμη, θα πρέπει να υπάρχουν κελιά αντίστοιχης μνήμης στη βιβλιοθήκη). Έτσι, μετά το elaboration έχει δημιουργηθεί μία δομή για όλο το κύκλωμα και ο σχεδιαστής λαμβάνει μία πρώτη εκτίμηση.

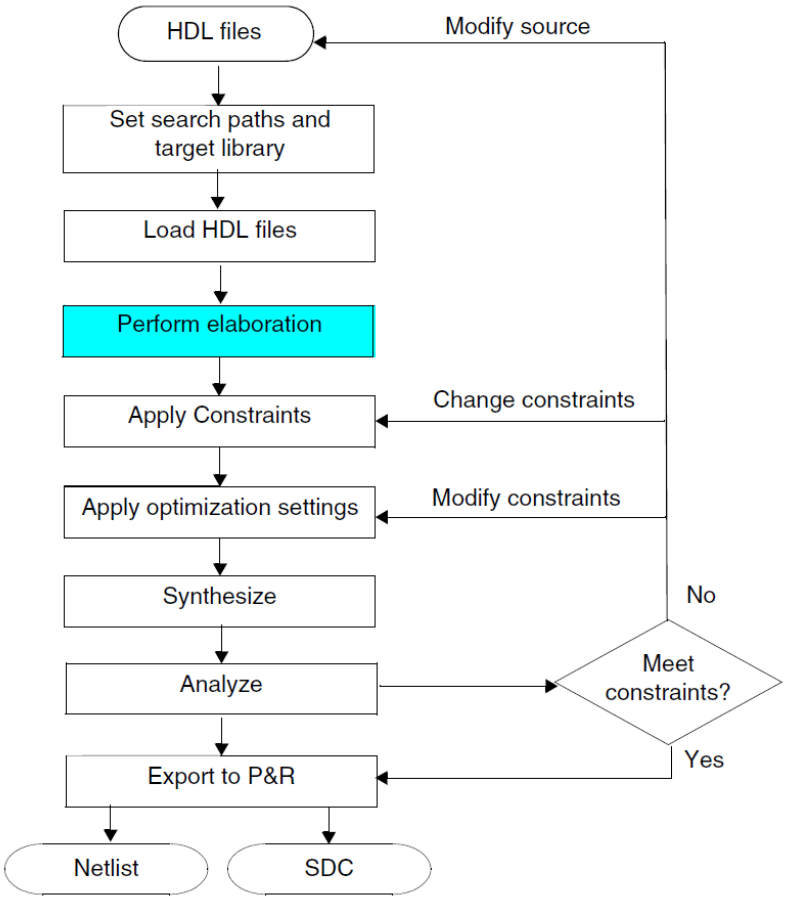
Ακολουθεί η σύνθεση η οποία διαχωρίζεται σε τρία στάδια: i) γενική (generic), ii) αντιστοιχισμένη (mapped) και iii) βελτιστοποιημένη (optimized). Στην πρώτη γίνεται η σύνθεση με γενικά και ιδανικά στοιχεία. Κατά την αντιστοιχισμένη σύνθεση, τα στοιχεία της βιβλιοθήκης παίρνουν τη θέση των γενικών. Αξίζει να αναφερθεί πως κατά την αντικατάσταση με στοιχεία της βιβλιοθήκης, ο αριθμός των συνολικών στοιχείων του κυκλώματος μειώνεται αρκετά, διότι συχνά στις βιβλιοθήκες υλοποιούνται στοιχεία πιο σύνθετης λογικής (βλ. Εικόνα 1-2). Με τον τρόπο αυτό μειώνονται οι καθυστερήσεις, η επιφάνεια και η κατανάλωση, αφού τα διάφορα στοιχεία που αποτελούν το πιο σύνθετο έχουν κοινά μέρη και δεν απαιτούνται εξωτερικές συνδέσεις, ενώ συνήθως χρειάζονται λιγότερα τρανζίστορ. Τέλος, κατά τη βελτιστοποιημένη σύνθεση τροποποιείται το κύκλωμα, ώστε να τηρούνται οι χρονικοί περιορισμοί και ταυτόχρονα να ελαχιστοποιηθεί η κατανάλωση ενέργειας.

Ολοκληρώνοντας τη σύνθεση, το Genus παρέχει ποικίλες αναφορές σχετικά με χαρακτηριστικά και μετρικές της σχεδίασης. Για παράδειγμα, δημιουργείται χρονική ανάλυση για όλες τις πιθανές διαδρομές (paths) στο εσωτερικό του κυκλώματος, αναφορά κατανάλωσης ενέργειας, αναφορά επιφάνειας ολοκλήρωσης κ.α.

[1] Οι μεταβλητές τύπου reg της Verilog δεν είναι απαραίτητο να αντιστοιχούν πάντα σε πραγματικά registers. Είναι πιθανό, έπειτα από βελτιστοποιήσεις, να αντικατασταθεί ένα reg ή να υλοποιηθεί για παράδειγμα με μία διασύνδεση σε άλλον καταχωρητή.



Εικόνα 1-2 Σύνθετη λογική πύλη AND-OR-Invert (AOI) 2-1 [Δ6]
(α) Σύμβολο (β) Κύκλωμα
Υλοποίηση με 6 τρανζίστορ έναντι των 10 που χρειάζεται το ισοδύναμο κύκλωμα με 2 πύλες NAND 2 εισόδων (4 τρανζίστορ), 1 inverter (2 τρανζίστορ) και 1 NOR 2 εισόδων (4 τρανζίστορ).



Εικόνα 1-3 Διάγραμμα ροής ενεργειών Genus [B7]

Στην Εικόνα 1-3 παρουσιάζεται το διάγραμμα ροής, όπως περιγράφεται στις προηγούμενες παραγράφους. Συνοπτικά, τα βήματα που ακολουθούνται από το Genus είναι τα παρακάτω:

i. Φόρτωση βιβλιοθηκών τεχνολογίας
ii. Δήλωση περιορισμών
iii. Ανάγνωση κώδικα HDL
iv. Προεπεξεργασία (elaboration)
v. Προσαρμογή ρυθμίσεων για βελτιστοποίηση (αλλαγή περιορισμών σε περίπτωση που είναι αδύνατο να ικανοποιηθούν)
vi. Σύνθεση
vii. Ανάλυση και εξαγωγή αποτελεσμάτων

1.3.3 Innovus Implementation System

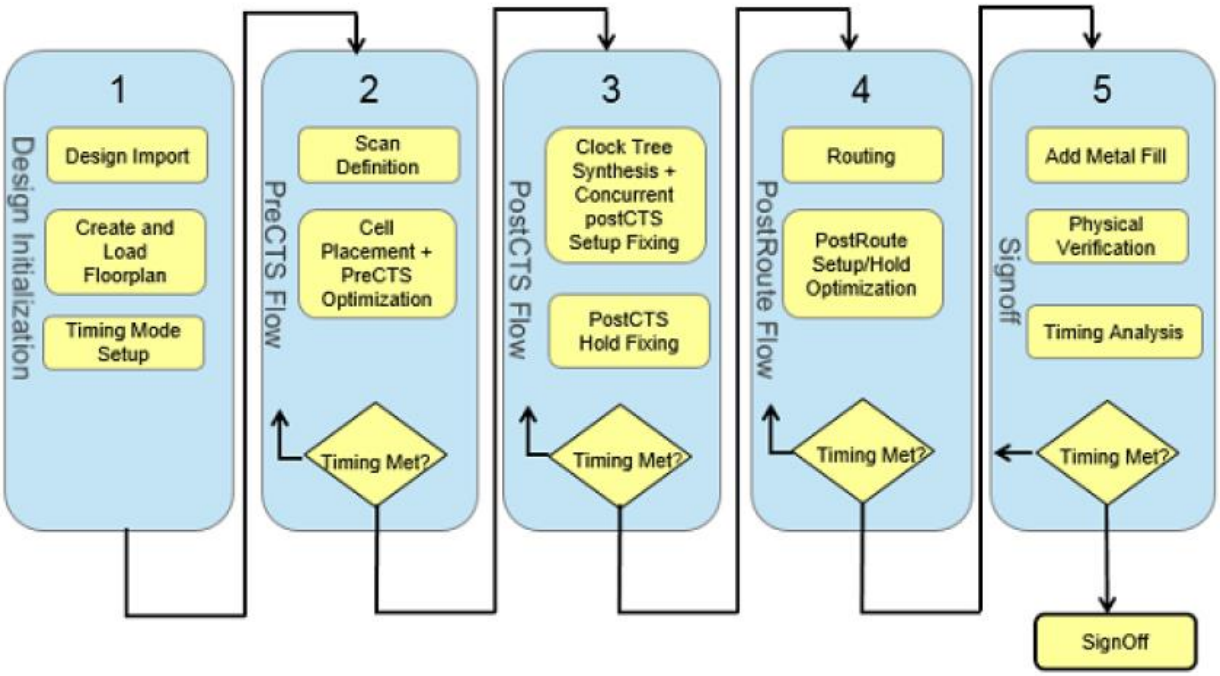
Το Innovus αποτελεί το τελικό στάδιο της σχεδίασης. Έχοντας ολοκληρώσει τη σύνθεση του κυκλώματος, πλέον μένει η μεταφορά του σχεδίου σε φυσικό επίπεδο (physical layout), δηλαδή η υλοποίηση και η διασύνδεση των στοιχείων στο ολοκληρωμένο κύκλωμα. Στο σημείο αυτό ουσιαστικά σχεδιάζεται το chip με πλήρη λεπτομέρεια για όλες τις στρώσεις (layers), την τροφοδοσία κτλ. Για να είναι όμως έτοιμο για κατασκευή, δηλαδή να γίνει το τελικό Sign Off, χρειάζεται διαρκώς να ελέγχουμε αν τηρούνται όλοι οι περιορισμοί που έχουμε θέσει και αντίστοιχα να αλλάζουμε την τοποθέτηση και διασύνδεση των στοιχείων.

Για να επιτευχθεί η κλειστότητα ως προς τον χρόνο (timing closure) ενός σχεδίου, πρέπει το εν λόγω σύστημα που υλοποιείται να μην παρουσιάζει λογικές ή σχεδιαστικές παραβάσεις, σφάλματα σε φυσικό επίπεδο και προπαντός να τηρεί τους χρονικούς περιορισμούς που έχουν τεθεί κατά τη σύνθεση. Προτού θεωρηθεί πως έχει επιτευχθεί το timing closure ενός chip, πρέπει να έχουν ληφθεί υπόψη όλες οι επιδράσεις που συμβαίνουν σε φυσικό επίπεδο, όπως είναι το metal fill (για την ομοιογένεια των μετάλλων ανά στρώση) και το coupling.

Παρά την ονομασία του, το timing closure δεν αφορά μόνο τη χρονική βελτιστοποίηση. Είναι μία πλήρης ροή διάφορων ενεργειών, οι οποίες πρέπει να συγκλίνουν στο επιθυμητό αποτέλεσμα. Μερικές από τις διαδικασίες που εμπεριέχονται στο timing closure είναι η τοποθέτηση (placement) των στοιχείων στην επιφάνεια του chip, η χρονική βελτιστοποίηση, η δημιουργία δέντρου ρολογιού (Clock Tree Synthesis, CTS), η διασύνδεση (routing) των στοιχείων κ.α. Κάθε βήμα πρέπει να ικανοποιεί τους στόχους που έχουν τεθεί, αλλιώς είναι πιθανό να μην επιτευχθεί το timing closure.

Καθώς προχωρά η ροή (βλ. Εικόνα 1-4), είναι σημαντικό να επικυρώνεται κάθε βήμα προτού ξεκινήσει το επόμενο. Ο στόχος είναι να υπάρχουν συνεπή και προβλέψιμα αποτελέσματα, στην πορεία της διαδικασίας, ώστε να είναι ευκολότερη η αποσφαλμάτωση και η επίλυση διάφορων θεμάτων όσο το δυνατόν νωρίτερα. Με τον τρόπο αυτό, εάν χρειαστεί να πραγματοποιηθούν αλλαγές για την ικανοποίηση των περιορισμών, τότε όσο νωρίτερα γίνουν, τόσο μικρότερο αντίκτυπο θα έχουν στη φυσική σχεδίαση. Είναι ακόμη καλή πρακτική να τρέχουμε

παράλληλα όλες τις διαδικασίες, ώστε να εντοπιστούν εμπόδια που ενδέχεται να προκύψουν αργότερα στα επόμενα βήματα.



Εικόνα 1-4 Διάγραμμα Timing Closure κατά τη διαδικασία της υλοποίησης [B8]

Συνοπτικά οι ενέργειες που λαμβάνουν χώρα κατά την παραπάνω ροή χωρίζονται σε πέντε μεγάλες κατηγορίες:

i. Αρχικοποίηση σχεδίασης (prePlace)
ii. Σχεδίαση πριν τη δημιουργία του δέντρου ρολογιού (preCTS)
iii. Σχεδίαση μετά τη δημιουργία του δέντρου ρολογιού (postCTS)
iv. Σχεδίαση μετά τη διασύνδεση του κυκλώματος (postRoute)
v. Οριστικοποίηση σχεδίασης (SignOff)

Στο αρχικό στάδιο φορτώνονται όλες οι απαραίτητες βιβλιοθήκες της τεχνολογίας, οι περιορισμοί και το σχέδιο μετά τη σύνθεση. Προτού ξεκινήσει η τοποθέτηση των στοιχείων, κάνουμε το floorplan, δηλαδή έναν πρώτο σχεδιασμό για την τοποθέτηση των επιμέρους μονάδων του κυκλώματος. Ακόμη, δημιουργούμε τις διασυνδέσεις για την τροφοδοσία των στοιχείων, η οποία συνήθως γίνεται με ένα «δαχτυλίδι» (power ring) γύρω από τον πυρήνα του chip και με «ραβδώσεις» (stripes) στο εσωτερικό αυτού, ώστε να τροφοδοτηθούν εύκολα όλα τα στοιχεία.

Στη συνέχεια λαμβάνει χώρα η τοποθέτηση των στοιχείων του κυκλώματος, σύμφωνα πάντα με το floorplan που έχουμε υποδείξει. Μπορούμε να ζητήσουμε τη βελτιστοποίηση της τοποθέτησης με γνώμονα τους χρονικούς περιορισμούς και την κατανάλωση ενέργειας. Πλέον είμαστε έτοιμοι να δημιουργήσουμε το δέντρο ρολογιού (clock tree), ώστε οι καθυστερήσεις στα clock pins των στοιχείων να είναι όσο το δυνατόν ομοιόμορφες και ελάχιστες. Στο

συγκεκριμένο σημείο βελτιστοποιούμε το κύκλωμα, ώστε να χρησιμοποιείται το clock tree με τον καλύτερο δυνατό τρόπο.

Μετά το clock tree, το κύκλωμα είναι πλέον έτοιμο για τη διασύνδεση όλων στοιχείων του. Σημειώνεται πως κατά τη διαδικασία της τοποθέτησης έχει γίνει ήδη μία πρώτη, γενική διασύνδεση. Στο postCTS στάδιο όμως, η διασύνδεση είναι λεπτομερής και γίνεται με τον βέλτιστο δυνατό τρόπο.

Στο τελικό βήμα και πριν η σχεδίαση είναι έτοιμη για υλοποίηση και παραγωγή του chip, χρειάζεται να γίνουν μερικά ακόμη βήματα. Έτσι, επειδή ενδέχεται μερικά layers να χρησιμοποιούνται σε μικρό ποσοστό, γεγονός το οποίο μπορεί να δημιουργήσει προβλήματα και πρέπει να αποφεύγεται, συμπληρώνουμε τα layers με επιπλέον στοιχεία, τα λεγόμενα fillers. Τα στοιχεία αυτά δεν έχουν κάποια λειτουργικότητα. Η χρησιμότητά τους είναι η συμπλήρωση των στρωμάτων του chip, ώστε να περιέχουν ένα συγκεκριμένο ποσοστό μετάλλου το καθένα. Τέλος, χρειάζεται να επαληθεύσουμε την ορθότητα του φυσικού σχεδίου σε ό,τι αφορά τη γεωμετρία του, τη διασύνδεση, το ποσοστό μετάλλου ανά στρώση κ.α. Εάν όλοι οι έλεγχοι είναι επιτυχείς, τότε το ολοκληρωμένο κύκλωμα είναι έτοιμο για κατασκευή.

Όπως φαίνεται και στην Εικόνα 1-4, είναι σημαντικό να ελέγχουμε σε κάθε βήμα εάν πληρούνται οι χρονικοί περιορισμοί. Έτσι, οι εκάστοτε βελτιστοποιήσεις έχουν σκοπό να ικανοποιήσουν τους χρόνους που απαιτούνται, πέραν της ελαχιστοποίησης της ενέργειας που καταναλώνεται. Βέβαια σε περίπτωση που σε κάποιο σημείο οι καθυστερήσεις είναι μεγαλύτερες των επιτρεπτών, δε σημαίνει απαραίτητα ότι δε θα μικρύνουν σε επόμενο βήμα για να βρεθούν εντός των επιτρεπτών ορίων. Για παράδειγμα, μερικά μονοπάτια που πριν το CTS είχαν μεγαλύτερες από τις επιτρεπτές καθυστερήσεις, είναι πιθανό μετά το CTS και τη βελτιστοποίηση που το συνοδεύει να τηρούν τους περιορισμούς συρρικνώνοντας τις καθυστερήσεις. Για είναι όμως βάσιμο το προηγούμενο παράδειγμα, οι καθυστερήσεις πριν το CTS θα πρέπει να είναι ελάχιστα εκτός των ορίων που έχουμε θέσει και να μην αποκλίνουν ιδιαίτερα από αυτά.

Στο πακέτο του Innovus περιλαμβάνονται διάφορα λογισμικά, τα οποία χρησιμοποιούνται κατά τσ στάδια της υλοποίησης και στόχο έχουν να βελτιώσουν τη διαδικασία του timing closure. Τα πιο σημαντικά από αυτά παρουσιάζονται στον πίνακα που ακολουθεί:

Εργαλείο	Χρήση
GigaPlace	Επιτρέπει την τοποθέτηση με βάση τις καθυστερήσεις (slack-driven) και τον διαμοιρασμό ανά τις στρώσεις (layers) πριν το CTS για μικρότερη συμφόρηση και χρονική βελτιστοποίηση.
GigaOpt	Ένα multi-threaded εργαλείο βελτιστοποίησης που χρησιμοποιείται στις preCTS, postCTS και postRoute βελτιστοποιήσεις.
Clock Concurrent Optimization (CCOpt)	Συνδυάζει το CTS με τη βελτιστοποίηση των μονοπατιών για την επίτευξη καλύτερων χρόνων, μικρότερης κατανάλωσης ενέργειας και χρησιμοποιούμενης επιφάνειας.
Layer-Aware Optimization	Ελέγχει την preRoute και postRoute βελτιστοποίηση ανά layer κα βελτιώνει τους χρόνους, ορίζοντας περιορισμούς για την κατώτατη επιτρεπτή στρώση κρίσιμων διασυνδέσεων (nets).

Πίνακας 1-2 Εργαλεία Innovus

2 Άμεση Ψηφιακή Σύνθεση (Direct Digital Synthesis)

2.1 Ορισμός και πλεονεκτήματα

Η άμεση ψηφιακή σύνθεση είναι μία μέθοδος παραγωγής αναλογικών κυματομορφών (συνήθως ημιτονοειδών) δημιουργώντας ένα μεταβαλλόμενο στο χρόνο ψηφιακό σήμα το οποίο εν συνεχεία μετατρέπεται σε αναλογικό μέσω ενός D/A Converter (Ψηφιακού σε Αναλογικό Μετατροπέα). Δεδομένου ότι οι λειτουργίες στο εσωτερικό ενός DDS είναι ως επί των πλείστων ψηφιακές, επιτρέπεται η γρήγορη εναλλαγή μεταξύ διαφορετικών συχνοτήτων εξόδου, καλή ανάλυση συχνότητας, καθώς και λειτουργία σε μεγάλο εύρος συχνοτήτων. Με την εξέλιξη της τεχνολογίας, οι σύγχρονοι DDS χαρακτηρίζονται από πολύ μικρό μέγεθος (compactness) και καταναλώνουν πολύ λίγη ενέργεια.

Η ικανότητα παραγωγής με ακρίβεια διαφόρων ειδών κυματομορφών σε ποικίλες συχνότητες είναι απαραίτητα στοιχεία σε μια πληθώρα τομέων της αγοράς. Είτε πρόκειται για πηγές χαμηλού θορύβου που χρειάζεται να αλλάζουν συχνά και γρήγορα συχνότητα, είτε απλώς για μία ακουστική συχνότητα, η ευκολία, το μικρό μέγεθος και το χαμηλό κόστος αποτελούν πρωταρχικούς παράγοντες.

Στην ιστορία των ηλεκτρονικών κυκλωμάτων υπήρξαν πολλές λύσεις για την παραγωγή συχνοτήτων. Μία από τις παλαιότερες τεχνικές, που επινοήθηκε από τον H. de Bellescize το 1932, είναι το Phase Locked Loop και βρίσκει ευρεία εφαρμογή ακόμα και σήμερα υλοποιημένο σε αναλογική, ψηφιακή ή και υβριδική μορφή. Μία άλλη τεχνική αφορά τη χρήση δυναμικά προγραμματιζόμενων D/A Converter, οι οποίοι συνδυάζουν τις εξόδους τους για την παραγωγή χαμηλότερων συχνοτήτων.

Ωστόσο, η τεχνική του DDS, παρ' όλο που είναι από τις πιο νέες, έχει γίνει αποδεκτή στην τεχνολογική κοινότητα για την ικανοποίηση απαιτήσεων παραγωγής συχνοτήτων σε πολλούς τομείς, όπως για παράδειγμα στο χώρο των τηλεπικοινωνιών. Ο κύριος λόγος είναι ότι μόλις με ένα ολοκληρωμένο κύκλωμα επιτρέπεται η δημιουργία οποιωνδήποτε αναλογικών σημάτων σε προγραμματιζόμενες συχνότητες. Μάλιστα η ακρίβεια και η ανάλυση συχνότητας που παρουσιάζουν οι DDS είναι πολύ μεγάλες, χωρίς να περιορίζεται το φάσμα. Όλα αυτά καταναλώνοντας πολύ μικρά ποσά ενέργειας ακόμα και σε υψηλές συχνότητες λειτουργίας.

Οι DDS λοιπόν, αποτελούν μία από τις καλύτερες λύσεις όταν πρόκειται για την παραγωγή κυματομορφών και είναι από τις πρώτες επιλογές των σχεδιαστών κυκλωμάτων. Παρακάτω συνοψίζονται τα πλεονεκτήματα αυτών:

- Ψηφιακά ελεγχόμενη συχνότητα σε πολύ μικρά υποπολλαπλάσια του Hz (π.χ. μHz) και φάσεις με τάξη μεγέθους πολύ μικρότερου της μοίρας (π.χ. εκατοστά της μοίρας),
- Πολύ γρήγορη εναλλαγή μεταξύ διαφορετικών συχνοτήτων εξόδου χωρίς να παρουσιάζονται ασυνέχειες,
- Προγραμματιζόμενος έλεγχος της συχνότητας εξόδου χωρίς εξωτερική επέμβαση,
- Απομακρυσμένος έλεγχος και βελτιστοποίηση μέσω ενός υπολογιστικού συστήματος.

2.2 Εφαρμογές

Οι εφαρμογές που βασίζονται στη χρήση ενός DDS για την παραγωγή κυματομορφών, χωρίζονται σε δύο μεγάλες κατηγορίες:

i. Τηλεπικοινωνιακά συστήματα
ii. Βιομηχανικές και ιατρικές εφαρμογές

2.2.1 Τηλεπικοινωνιακά συστήματα

Μία από τις κυριότερες εφαρμογές των DDS είναι τα τηλεπικοινωνιακά συστήματα. Συχνά απαιτούνται μεταβολές στις συχνότητες, οι οποίες πρέπει να γίνονται εύκολα και γρήγορα χωρίς την ανάγκη εξωτερικής παρέμβασης. Ένα άλλο σημαντικό χαρακτηριστικό που παίζει κυρίαρχο ρόλο στις τηλεπικοινωνίες είναι ο θόρυβος φάσης και η απόδοση λάθους (spurious performance), τα οποία πρέπει να διατηρούνται σε όσο το δυνατόν χαμηλότερα επίπεδα.

Γίνεται επομένως αντιληπτό πως ένας DDS αποτελεί ιδανική λύση σε αυτές τις περιπτώσεις, καθώς παρέχει ικανοποιητικές λύσεις στα παραπάνω προβλήματα. Το μεγάλο εύρος λειτουργίας του σε συνδυασμό με την καλή ανάλυση συχνότητας, ενισχύουν την αποτελεσματικότητά του και το κάνουν να υπερτερεί έναντι άλλων λύσεων και κυρίως των καθαρά αναλογικών.

Μερικά παραδείγματα χρήσης ενός DDS στις τηλεπικοινωνίες αφορούν τη διαμόρφωση, η οποία μπορεί να γίνεται και σε συνδυασμό με ένα PLL για την ενίσχυση της ολικής προσαρμοστικότητας στις απαιτούμενες συχνότητες. Ο DDS μπορεί επίσης να χρησιμοποιηθεί ως τοπικός ταλαντωτής σε ένα τηλεπικοινωνιακό σύστημα ή ακόμα και για την απευθείας μετάδοση σημάτων σε ραδιοσυχνότητες.

2.2.2 Βιομηχανικές και ιατρικές εφαρμογές

Ένα άλλο σημαντικό πεδίο εφαρμογής αποτελούν διάφορες βιομηχανικές και ιατρικές περιπτώσεις στις οποίες χρειάζεται ένας DDS για προγραμματιζόμενη παραγωγή κυματομορφών. Επειδή ένας DDS προγραμματίζεται ψηφιακά, η φάση και η συχνότητα μιας κυματομορφής μπορούν εύκολα να προσαρμοστούν χωρίς την ανάγκη εξωτερικής επέμβασης στα στοιχεία του κυκλώματος, όπως θα γινόταν σε έναν κλασικό αναλογικό παραγωγό κυματομορφών. Έτσι, παρέχεται η δυνατότητα αυτοματοποίησης, η οποία συχνά είναι απαιτούμενη σε τέτοιου είδους εφαρμογές.

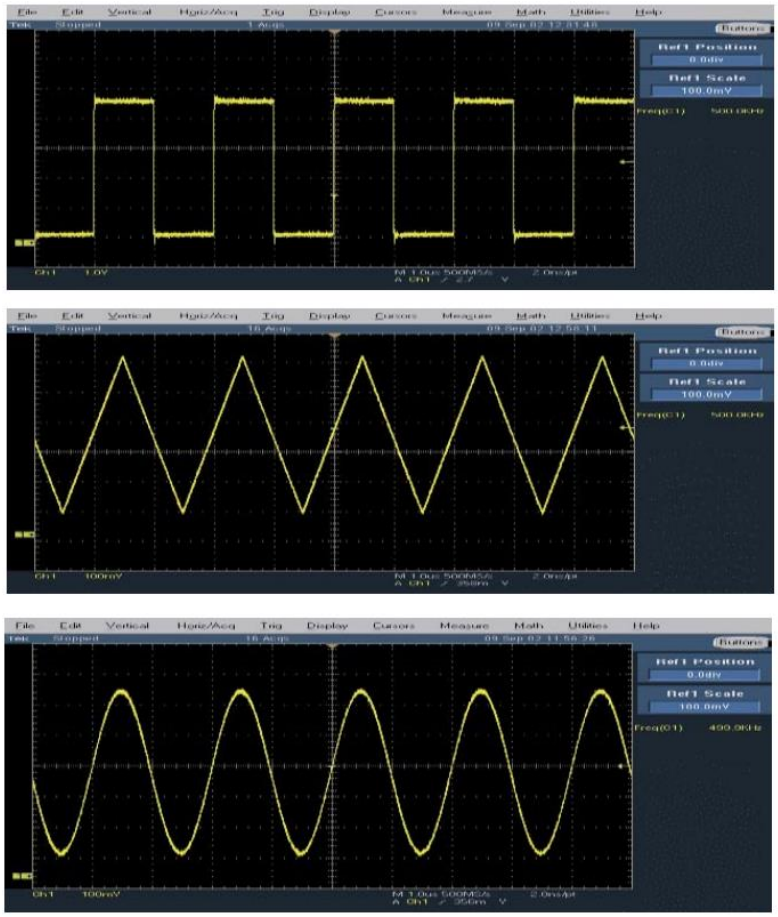
Επίσης, επιτρέπεται η εναλλαγή μεταξύ διαφόρων συχνοτήτων σε πραγματικό χρόνο, επομένως υπάρχει πολύ χαμηλή απόκριση και άμεση προσαρμογή στις διαφορετικές καταστάσεις που μπορεί να προκύπτουν για μία δεδομένη εφαρμογή. Σημειώνεται ακόμη πως λόγω της ψηφιακής του φύσης, ο DDS λειτουργεί ανεξάρτητα με τις μεταβολές τις θερμοκρασίας στο περιβάλλον του, πράγμα που είναι σύνηθες φαινόμενο σε βιομηχανικούς χώρους και συχνά μεταβάλει τις τιμές αναλογικών στοιχείων, όπως για παράδειγμα των αντιστάσεων.

Μερικές από τις πιο διαδομένες εφαρμογές ενός DDS για βιομηχανικούς σκοπούς, είναι οι προσαρμοζόμενες πηγές συχνοτήτων για τη μέτρηση μιας αντίστασης (impedance), όπως για παράδειγμα σε έναν αισθητήρα που βασίζεται στην αντίσταση. Μία άλλη πολύ συχνή του

χρήση, είναι στον χώρο της μουσικής, όπου χρησιμοποιείται για την παραγωγή ακουστικών συχνοτήτων.

2.3 Υποστηριζόμενες λειτουργίες

Οι άμεσοι ψηφιακοί συνθέτες δε χρησιμοποιούνται μόνο για την παραγωγή ημιτονοειδών σημάτων. Αντιθέτως, μπορούν να κατασκευάσουν οποιοδήποτε περιοδικό σήμα. Αρκεί η δειγματοληψία και αποθήκευση του σήματος στη μνήμη του DDS. Έπειτα το DDS αναλαμβάνει την επαναδημιουργία του σήματος σε οποιαδήποτε επιθυμητή συχνότητα. Οι πιο συνηθισμένες μορφές εξόδων, εκτός της ημιτονοειδούς, είναι η τετραγωνική (παλμός) και η τριγωνική. Η παραγωγή βέβαια των δύο τελευταίων, δε χρειάζεται κάποια μνήμη σε αντίθεση με τα υπόλοιπα περιοδικά σήματα.



Εικόνα 2-1 Τετραγωνική, τριγωνική και ημιτονοειδής έξοδος ενός DDS

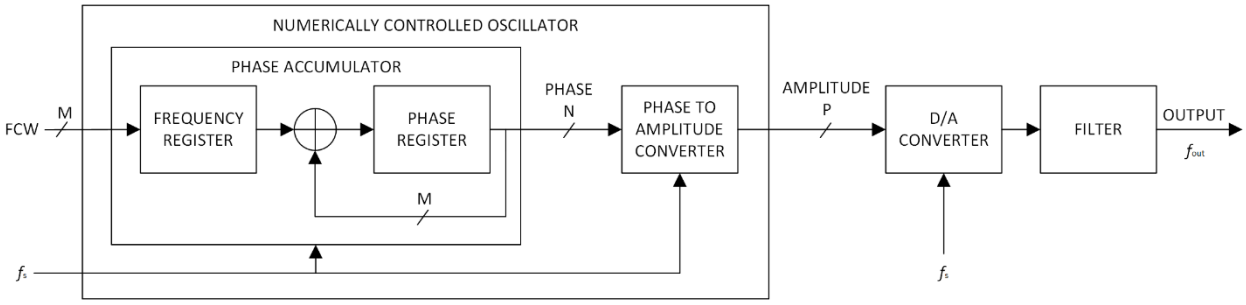
Στη συνέχεια γίνεται μια παρουσίαση της αρχιτεκτονικής ενός DDS κατά τη διαδικασία παραγωγής ενός ημιτονοειδούς σήματος, το οποίο αποτελεί το κύριο αντικείμενο της παρούσας εργασίας και επομένως είναι αναπόσπαστο κομμάτι της θεωρίας που εφαρμόζεται. Σημειώνεται πως παρόμοια λογική ακολουθείται για την κατασκευή οποιουδήποτε άλλου περιοδικού

σήματος εκτός των τριγωνικών και τετραγωνικών, τα οποία λόγω της απλούστερης φύσης τους, χρειάζονται λιγότερο πολύπλοκα συστήματα.

2.4 Αρχιτεκτονική συστήματος

Τα κύρια συστατικά μέρη ενός DDS, όπως παρουσιάζονται στην Εικόνα 2-2, είναι τα παρακάτω:

- Phase Accumulator (Συσσωρευτής Φάσης)
- Phase to Amplitude Converter (Μετατροπέας Φάσης σε Πλάτος)
- Numerically Controlled Oscillator (Αριθμητικώς Ελεγχόμενος Ταλαντωτής)
- D/A Converter (Μετατροπέας Ψηφιακού σε Αναλογικό)



Εικόνα 2-2 Μπλοκ διάγραμμα ενός άμεσου ψηφιακού συνθέτη

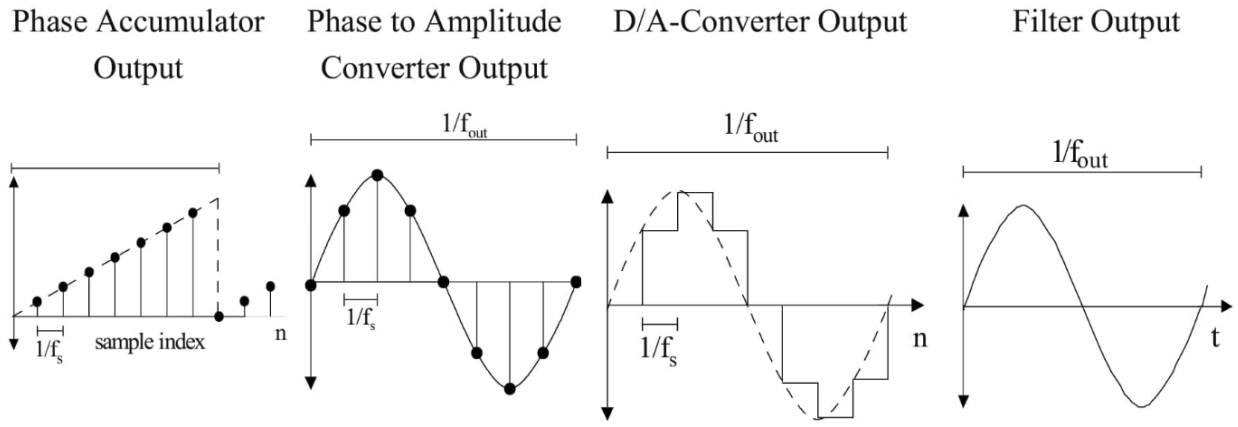
Η παραγόμενη συχνότητα ενός DDS εξαρτάται από δύο παραμέτρους: τη συχνότητα του ρολογιού f_S και την τιμή του καταχωρητή συχνότητας (frequency register) ή αλλιώς λέξη ελέγχου συχνότητας FCW (Frequency Control Word).

Η λέξη ελέγχου συχνότητας FCW μήκους M bit αποτελεί την κύρια είσοδο στο phase accumulator, καθώς είναι το βήμα με το οποίο αυξάνεται η τιμή του phase register σε κάθε ανοδικό παλμό του ρολογιού. Η έξοδος του phase accumulator είναι μία πριονωτή (sawtooth) κυματομορφή της φάσης.

Το phase to amplitude converter συχνά είναι ένα look-up table (LUT) 2^N θέσεων, στο οποίο βρίσκονται αποθηκευμένα τα δείγματα του υπό κατασκευή σήματος. Έτσι, η φάση που δίνει το phase accumulator ουσιαστικά αποτελεί τη διεύθυνση στο LUT, όπου εμπεριέχεται η αντίστοιχη τιμή πλάτους με ακρίβεια P bit. Το πλάτος αυτό μεταφέρεται στον D/A Converter όπου και γίνεται η μετατροπή του σε αναλογικό εντός του επιθυμητού εύρους τιμών (π.χ. $\pm 5\ Volt$). Τέλος, προαιρετικά και εξωτερικά του DDS, μπορεί να γίνει εξομάλυνση της εξόδου χρησιμοποιώντας ένα φίλτρο.

Όπως γίνεται εύκολα κατανοητό, εάν το βήμα αύξησης της φάσης είναι μεγάλο, δηλαδή χρησιμοποιείται μεγάλο FCW, τότε το περιεχόμενο του phase accumulator θα αυξάνεται γρήγορα και επομένως οι διευθύνσεις στον phase to amplitude converter θα αλλάζουν με μεγάλο ρυθμό. Έτσι, η συχνότητα του παραγόμενου σήματος θα είναι υψηλή. Αντίστοιχα, για να

δημιουργηθεί χαμηλή συχνότητα στην έξοδο του DDS, χρειάζεται να εφαρμοστεί μία μικρή λέξη FCW.

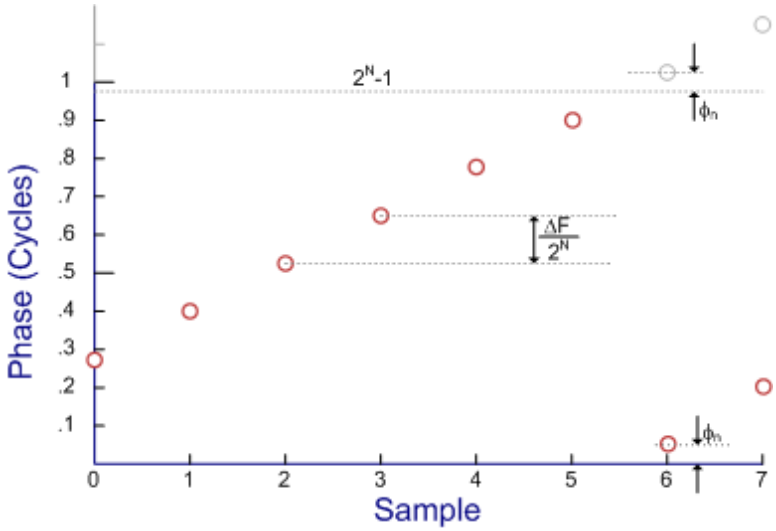


Εικόνα 2-3 Έξοδοι των επιμέρους τμημάτων ενός DDS

2.4.1 Συσσωρευτής Φάσης (Phase Accumulator, PA)

Το phase accumulator αποτελείται από το frequency register, το οποίο αποθηκεύει τη λέξη συχνότητας ελέγχου (FCW), έναν αθροιστή μήκους M bit και το phase register, το οποίο περιέχει την τρέχουσα φάση. Σε κάθε κύκλο του ρολογιού η τιμή της φάσης ανανεώνεται με την προηγούμενη τιμή της αυξημένη κατά FCW. Η νέα φάση αποθηκεύεται εκ νέου στον phase register και αποτελεί την έξοδο του PA, η οποία μοιάζει με τα «δόντια» ενός πριονιού ή με σκαλοπάτια ύψους $\Delta F = FCW$ το καθένα. Σημειώνεται πως για την παραγωγή μίας συγκεκριμένης και σταθερής συχνότητας, η λέξη FCW πρέπει να παραμένει επίσης σταθερή.

Όταν το άθροισμα του FCW με την τρέχουσα φάση ξεπερνά τη μέγιστη τιμή του phase register $(2^M - 1)$, τότε γίνεται υπερχείλιση και το επιπλέον overflow bit απορρίπτεται από τον αθροιστή, ώστε οι προσθετέοι να έχουν σταθερό μήκος και πάντα ίσο με M. Μετά την υπερχείλιση, το phase register περιέχει το υπόλοιπο φ_n της προηγούμενης πράξης που προκύπτει από την αφαίρεση του overflow bit. Τα παραπάνω παρουσιάζονται γραφικά στην Εικόνα 2-4. Μετά από κάποιες επαναλήψεις K, το υπόλοιπο θα επιστρέψει στην αρχική του τιμή (συνήθως 0), λόγω της φύσης του phase accumulator ως μηχανή πεπερασμένης κατάστασης (finite-state machine). Το διάστημα K



Εικόνα 2-4 Κανονικοποιημένη έξοδος του phase accumulator [Δ7]

αναφέρεται ως ο κύριος ρυθμός επανάληψης (grand repetition rate, GRR) και δίνεται από την παρακάτω σχέση:

$$GRR = \frac{2^M}{MK\Delta(\Delta F, 2^M)} \qquad (1)$$

όπου ΜΚΔ είναι η συνάρτηση εύρεσης του μέγιστου κοινού διαιρέτη. Το GRR αναπαριστά την πραγματική περιοδικότητα για δοσμένο ΔF, η οποία σε υψηλής ανάλυσης DDS μπορεί να γίνει αρκετά μεγάλη.

Συνήθως όμως, ενδιαφερόμαστε περισσότερο για τη συχνότητα εξόδου η οποία καθορίζεται από το μέσο όρο υπερχειλίσεων:

$$f_{out} = \frac{\Delta F}{2^M} f_s \qquad (2)$$

Η συχνότητα εξόδου ακόμη μεταβάλλεται κάθε φορά που αλλάζει το FCW, όπως περιγράφεται στην προηγούμενη ενότητα. Οι μεταβολές αυτές είναι άμεσες και συνεχείς στη φάση. Σε αντίθεση με τα PLL, δεν παρουσιάζεται κάποια καθυστέρηση (loop settling time) όταν αλλάζει η συχνότητα.

Καθώς η συχνότητα αυξάνεται, το πλήθος των δειγμάτων που χρησιμοποιούνται ανά κύκλο μειώνεται. Σύμφωνα με τη θεωρία δειγματοληψίας (sampling theory), απαιτούνται τουλάχιστον δύο δείγματα ανά περίοδο ώστε να ανακατασκευάσουν ένα σήμα. Έτσι, η μέγιστη συχνότητα εξόδου ενός DDS περιορίζεται στα $\frac{f_s}{2}$, δηλαδή στο μισό της συχνότητας του εξωτερικού ρολογιού. Ωστόσο, πολλές φορές για πρακτικούς λόγους, η συχνότητα εξόδου περιορίζεται σε κάτι λιγότερο από τη μέγιστη επιτρεπτή, βελτιώνοντας έτσι την ποιότητα του παραγόμενου σήματος και επιτρέποντας την εφαρμογή φίλτρου στην έξοδο.

Η ανάλυση συχνότητας καθορίζεται από την μικρότερη δυνατή αύξηση στην συχνότητα και δίνεται από τη σχέση:

$$f_{res} = \frac{f_s}{2^M} \qquad (3)$$

Από την εξίσωση (2) μπορούμε να θεωρήσουμε το phase accumulator και ως έναν προγραμματιζόμενο μη ακέραιο διαιρέτη συχνότητας με λόγο διαίρεσης $\frac{\Delta F}{2^M}$.

2.4.2 Μετατροπέας Φάσης σε Πλάτος (Phase to Amplitude Converter, PAC)

Μετά το PA, η φάση στην έξοδό του, περικόβεται στα N πιο σημαντικά bit (MSB) προτού αποτελέσει είσοδο στο phase to amplitude converter. Συχνά η ακρίβεια του PA είναι πολύ μεγαλύτερη της χωρητικότητας του PAC, δηλαδή $M \gg N$, χωρίς όμως αυτό να επηρεάζει τη συνολική ακρίβεια του συστήματος στη συχνότητα που παράγει. Ωστόσο, η περικοπή αυτή δημιουργεί ένα σφάλμα φάσης το οποίο είναι σημαντική πηγή δημιουργίας ψευδών τιμών (spurious

products). Μία άλλη πηγή τέτοιων σφαλμάτων αποτελεί το πεπερασμένο μήκος της εξόδου του PAC. Τα σφάλματα αυτά αναλύονται σε επόμενη ενότητα.

Το PAC μετατρέπει τη διακριτή φάση σε αντίστοιχες τιμές πλάτους της υπό κατασκευή κυματομορφής. Συνήθως η μνήμη που περιέχει για την αποθήκευση των τιμών πλάτους υλοποιείται με μία ROM (Read Only Memory). Με άλλα λόγια, αποθηκεύει 2^N διαδοχικά δείγματα της συνάρτησης που περιγράφει την κυματομορφή που επιθυμούμε να παράγουμε. Εάν για παράδειγμα, όπως συχνά συμβαίνει, η εν λόγω συνάρτηση είναι ένα ημίτονο, τότε το PAC περιέχει τις τιμές που δίνονται από την παρακάτω συνάρτηση:

$$\sin\left(2\pi\frac{\Phi(n)}{2^M}\right) \qquad (4)$$

όπου $\Phi(n)$ είναι μία τιμή (μήκους M bit) του phase register στην n - οστή περίοδο του ρολογιού. Είναι φανερό πως το n παίρνει τιμές στο σύνολο ακεραίων $[0, 2^N)$. Επίσης, η ακρίβεια P των τιμών που αποθηκεύονται στον PAC είναι συνήθως μερικά bit μικρότερη από την έξοδο N του PA (π.χ. $P = N - 2$).

Συχνά υιοθετούνται διάφορες τεχνικές για τη μείωση του μεγέθους μνήμης που απαιτείται, αξιοποιώντας διάφορες συμμετρίες όπου είναι διαθέσιμες, όπως τριγωνομετρικές επεκτάσεις, τριγωνομετρικές προσεγγίσεις και μεθόδους που λαμβάνουν υπόψη τις συμμετρίες των αρμονικών συναρτήσεων.

Εναλλακτικές υλοποιήσεις θέλουν το PAC να αποτελείται από μνήμες τύπου RAM (Random Access Memory), οι οποίες παρέχουν το πλεονέκτημα ότι το περιεχόμενό τους μπορεί κάθε φορά να φορτωθεί με διαφορετική συνάρτηση για οποιασδήποτε αυθαίρετη περιοδική κυματομορφή. Βέβαια, η αποθήκευση των τιμών της κυματομορφής που επρόκειτο να αναδημιουργηθεί, είναι απαραίτητο να γίνεται κάθε φορά που ενεργοποιείται το DDS, καθώς οι μνήμες RAM δε διατηρούν μόνιμα τα δεδομένα τους.

2.4.3 Αριθμητικώς Ελεγχόμενος Ταλαντωτής (Numerically Controlled Oscillator, NCO)

Όπως φαίνεται στην Εικόνα 2-2, το PA και το PAC μαζί αποτελούν τον numerically controlled oscillator ή στα ελληνικά, αριθμητικώς ελεγχόμενο ταλαντωτή. Πρόκειται απλά για μία ομαδοποίηση και ενιαία θεώρηση του καθαρά ψηφιακού μέρους ενός DDS. Το NCO λοιπόν δεν είναι κάτι επιπρόσθετο, αλλά η «καρδιά» του DDS.

Συχνά με τον όρο DDS αναφερόμαστε στο NCO, ενώ το συνολικό κύκλωμα αναφέρεται ως ένα «πλήρες DDS». Η διαφορά τους έγκειται αμιγώς στην ενσωμάτωση ή όχι του A/D Converter (το φίλτρο σχεδόν πάντα είναι εξωτερικό του ολοκληρωμένου κυκλώματος). Από το σημείο αυτό και στο υπόλοιπο της εργασίας, με τον όρο DDS θα εννοείται μόνο το ψηφιακό τμήμα χωρίς τον A/D Converter, δηλαδή ένα NCO.

2.5 Ανεπιθύμητα φαινόμενα

Κατά τη λειτουργία ενός DDS, υπάρχουν μερικά σφάλματα που εισάγονται στο σήμα παράγεται. Τα λεγόμενα “spurious products” είναι το αποτέλεσμα αρμονικών, ή και μη, παρεμβολών στη δημιουργία της κυματομορφής εξόδου, λόγω διαφόρων μη γραμμικών επιδράσεων στην πορεία δημιουργίας / επεξεργασίας του σήματος.

2.5.1 Σφάλμα εκ περικοπής φάσης

Ο αριθμός των bit M του phase accumulator συνήθως κυμαίνεται μεταξύ 16 και 64. Εάν επομένως χρησιμοποιούταν ολόκληρη η έξοδος του PA χωρίς περικοπή στα N πιο σημαντικά ψηφία, τότε το PAC θα απαιτούσε μία τεράστια χωρητικότητα μνήμης, αφού $2^M \gg 2^N$. Για να αποφευχθεί μία τέτοια περίπτωση περικόβουμε την έξοδο του PA, ώστε ο PAC να χρειάζεται μικρότερη μνήμη με χωρητικότητα σε λογικά πλαίσια. Η περικοπή αυτή όμως της εξόδου, επηρεάζει τη φάση και εισάγει μη αρμονικές παρεμβολές, ανάλογες με τα bit που κόβονται. Ο αριθμός των παρεμβολών αυτών μπορεί να βρεθεί από τον παρακάτω τύπο:

$$n_W = \frac{2^W}{MK\Delta(\Delta F, 2^W)} - 1 \qquad (5)$$

όπου W είναι ο αριθμός των bit που περικόβονται.

Για να γίνει πιο εύκολα αντιληπτό το εν λόγω σφάλμα, ας θεωρήσουμε μία ημιτονοειδή κυματομορφή για την οποία διαθέτουμε 2^N διακριτά σημεία για την περιγραφή της. Λόγω του μεγάλου εύρους συχνοτήτων που παρέχει ένα DDS, κάθε φορά χρησιμοποιούμε είτε λιγότερα, είτε περισσότερα από τα διαθέσιμα σημεία, ανάλογα με την παραγόμενη συχνότητα. Έτσι, διακρίνουμε δύο μεγάλες περιπτώσεις:

i. χρήση λιγότερων από τα διαθέσιμα σημεία
ii. επανάληψη διαδοχικών σημείων

2.5.1.1 Χρήση λιγότερων από τα διαθέσιμα σημεία

Η περίπτωση αυτή προκαλείται στις μεγάλες συχνότητες (συγκριτικά με το ρολόι αναφοράς), όταν δηλαδή έχουμε μεγάλο FCW. Έτσι, το περιεχόμενο του phase register αυξάνεται με μεγάλο ρυθμό και χρειάζεται λιγότερα βήματα έως ότου συμβεί υπερχείλιση. Επομένως, χρησιμοποιούνται λιγότερες τιμές από αυτές που περιέχει ο PAC, λόγω του ότι δεν αξιοποιούνται όλες οι διευθύνσεις της μνήμης του. Αποτέλεσμα είναι η έξοδος να κατασκευάζεται με λιγότερες τιμές ανά περίοδο, οπότε και η παραγόμενη κυματομορφή να παρουσιάζει μικρότερη ακρίβεια. Έτσι, το σήμα εξόδου χαρακτηρίζεται από μη αρμονικές παρεμβολές.

2.5.1.2 Επανάληψη διαδοχικών σημείων

Στην αντίθετη περίπτωση όπου η επιθυμητή συχνότητα είναι μικρή και έχουμε μικρή λέξη FCW, το περιεχόμενο του phase register αυξάνεται με αργό ρυθμό. Έτσι, μετά την περικοπή των bit, είναι πολύ πιθανό δύο διαδοχικές έξοδοι του PA να είναι ίδιες και επομένως να αξιοποιηθούν οι ίδιες τιμές από τη μνήμη του PAC. Με τον τρόπο αυτό λοιπόν, επειδή τα σημεία δεν

επαρκούν να περιγράψουν πλήρως το χαμηλής συχνότητας σήμα, παρατηρείται και πάλι μία μη αρμονική παρεμβολή που κάνει την έξοδο να αλλάζει βηματικά σε «σκαλοπάτια».

2.5.1.3 Απαλοιφή σφαλμάτων εκ περικοπής φάσης

Δυστυχώς τα παραπάνω σφάλματα δεν μπορούν να εξαλειφθούν πλήρως από τη λειτουργία ενός DDS. Ωστόσο, στην ειδική περίπτωση όπου το FCW ισούται ακριβώς με τη χωρητικότητα του PAC, δηλαδή $FCW = 2^N$, τότε θα χρησιμοποιούνται ακριβώς όλες οι τιμές της μνήμης χωρίς την επανάληψή τους. Ένα ισοδύναμο αυτού θα ήταν εάν είχαμε $FCW = 1$ και $M = N$, δηλαδή δε συνέβαινε περικοπή. Στην περίπτωση αυτή δεν υπάρχει σφάλμα από περικοπή.

Μία άλλη μέθοδος που χρησιμοποιείται για την εξάλειψη του σφάλματος εκ περικοπής φάσης και παρουσιάζει ουσιαστικές βελτιώσεις, είναι η εισαγωγή λευκού θορύβου πριν την περικοπή. Ο επονομαζόμενος και “dither noise” προστίθεται στα $W + 1$ χαμηλότερα bit της εξόδου του PA, ώστε να γραμμικοποιήσει τη διαδικασία περικοπής. Συνήθως, η διαδικασία αυτή έρχεται χωρίς κόστος στην απόδοση του συστήματος, καθώς η τελευταία καθορίζεται κυρίως από τον D/A Converter.

2.5.2 Σφάλμα εκ περικοπής πλάτους

Μία άλλη πηγή μη αρμονικών παρεμβολών είναι αυτή που οφείλεται στην κβαντική φύση των τιμών πλάτους της υπό κατασκευή κυματομορφής. Δεδομένου ότι ο D/A Converter έχει συγκεκριμένη ακρίβεια P bit, αναγκαστικά θα πρέπει και οι τιμές που είναι αποθηκευμένες στο PAC να αποτελούνται από P bit. Έτσι, μειώνεται η ακρίβεια κατά την ανακατασκευή του σήματος. Το εν λόγω σφάλμα είναι αναπόφευκτο, αλλά μπορεί να μειωθεί με την αύξηση των bit του D/A Converter. Βέβαια μία ενδεχόμενη αύξηση του P, θα περιορίσει την ταχύτητα του DDS, λόγω των δυνατοτήτων του D/A.

3 Σχεδίαση κυκλώματος

3.1 Προδιαγραφές συστήματος

3.1.1 Χαρακτηριστικά

Παρακάτω μελετάται και αναλύεται αρχικά η σχεδίαση και στη συνέχεια η υλοποίηση ενός Direct Digital Synthesizer με ακρίβεια εξόδου 12-bit και χρονισμένου στα 2,8 GHz. Με την προσθήκη ενός D/A Converter στην έξοδό του, μπορεί να δημιουργήσει έως και 2,8 δισεκατομμύρια δείγματα από την αποθηκευμένη κυματομορφή το δευτερόλεπτο (2,8 GSPS). Λόγω του θεωρήματος δειγματοληψίας του Nyquist, η μέγιστη παραγόμενη συχνότητα δεν μπορεί να ξεπερνά τα 1,4 GHz. Παρακάτω παρατίθενται συνοπτικά τα χαρακτηριστικά του εν λόγω DDS:

Ταχύτητα	$2{,}8\ GSPS$
Ανάλυση συχνότητας	$10\ \mu Hz$
Ανάλυση φάσης	$14\ bits$
Ακρίβεια πλάτους	$12\ bits$
Τάση τροφοδοσίας	$1 - 1{,}1\ V$
Παραγόμενο σήμα	$sinusoidal$
Δυνατότητα προγραμματισμού	$serial$

Πίνακας 3-1 Χαρακτηριστικά του υπό σχεδίαση DDS

Όπως αναγράφεται παραπάνω, υπάρχει η δυνατότητα προγραμματισμού του DDS σειριακά μέσω ενός Serial Peripheral Interface (SPI). Έτσι, μπορούν να οριστούν προγραμματιστικά η λέξη ελέγχου συχνότητας (FCW) και η αρχική φάση (phase offset). Δίνεται επομένως η δυνατότητα ελέγχου τόσο της παραγόμενης συχνότητας, όσο και της μετατόπισης του σήματος.

3.1.2 Εφαρμογές & Χρήσεις

Ένας γρήγορος DDS όπως αυτός, δημιουργεί τα βέλτιστα σήματα σε συχνότητες από μερικά KHz έως και αρκετά MHz. Σε αυτές τις συχνότητες χρησιμοποιεί έναν ικανοποιητικό αριθμό δειγμάτων από τα $2^{14} = 16.384$ που διαθέτει στη μνήμη του και επομένως δημιουργούνται τα σήματα με τη μεγαλύτερη ακρίβεια, πετυχαίνοντας αποτελέσματα πολύ κοντά σε ένα πραγματικό, μη διακριτό ημίτονο. Έτσι, οι συνήθεις χρήσεις του είναι οι παρακάτω:

- Πηγή για συστήματα σάρωσης και radar
- Διαμόρφωση / αποδιαμόρφωση FM
- Εξοπλισμός ελέγχου και μετρητικών διατάξεων
- Οπτικοακουστικές εφαρμογές
- Ταχεία εναλλαγή συχνότητας

3.1.3 Υποκυκλώματα

Για τη σχεδίαση του παραπάνω DDS, το συνολικό κύκλωμα διασπάστηκε σε διάφορα υποτμήματα, ή αλλιώς, μονάδες (modules). Οι μονάδες στη συνέχεια συνδέονται κατάλληλα μεταξύ τους συνδυάζοντας έτσι τις λειτουργίες τους με σκοπό την επίτευξη του επιθυμητού αποτελέσματος. Οι μονάδες που υλοποιήθηκαν είναι οι εξής:

i. Serial Peripheral Interface (SPI)
ii. Phase Accumulator (PA)
iii. Operation Profile Configurator (OPC)
iv. Phase to Amplitude Converter (PAC)
v. Numerically Controller Oscillator (NCO)

Στις ενότητες που ακολουθούν αναλύεται αρχικά η δομή κάθε μονάδας, όπου παρουσιάζεται ένα ενδεικτικό σχεδιάγραμμα με συνοπτικές περιγραφές όλων των εισόδων / εξόδων της. Στη συνέχεια παρουσιάζεται η λειτουργία της μονάδας και οι εσωτερικές διαδικασίες, ώστε να προκύψουν οι επιθυμητές έξοδοι από τις αντίστοιχες εισόδους που εφαρμόζονται. Τέλος, ακολουθεί η προσομοίωση και η επαλήθευση της σωστής λειτουργίας της μονάδας.

Για την επαλήθευση χρησιμοποιείται ένα σενάριο χρήσης, γνωστό ως testbench (βλ. Γλώσσες περιγραφής υλικού (Hardware Description Languages, HDL)), όπου γίνεται μια προσπάθεια να επιβεβαιωθεί η σωστή λειτουργία της μονάδας για όσο το δυνατόν μεγαλύτερο εύρος. Όπως είναι κατανοητό, δεν είναι δυνατός ο έλεγχος για κάθε πιθανή είσοδο, καθώς ο αριθμός τους αυξάνει εκθετικά με την αύξηση του πλήθους και του μεγέθους των εισόδων. Έτσι, γίνεται μια προσέγγιση με βάση μια υποθετική περίπτωση χρήσης, η οποία προσπαθεί να καλύψει διαφορετικές περιστάσεις.

3.2 Σειριακή Περιφερειακή Διασύνδεση (Serial Peripheral Interface, SPI)

3.2.1 Εισαγωγή

Το Serial Peripheral Interface (Σειριακή Περιφερειακή Διασύνδεση) είναι μία σύγχρονη σειριακή διασύνδεση για κοντινής απόστασης επικοινωνία, η οποία χρησιμοποιείται κυρίως σε ενσωματωμένα συστήματα (embedded systems). Η διασύνδεση αναπτύχθηκε από τη Motorola στα μέσα της δεκαετίας του ’80 και από τότε χρησιμοποιείται συχνά σε τέτοιου είδους εφαρμογές με τις πιο διάσημες από αυτές να είναι οι κάρτες μνήμης SD και οι οθόνες υγρού κρυστάλλου LCD.

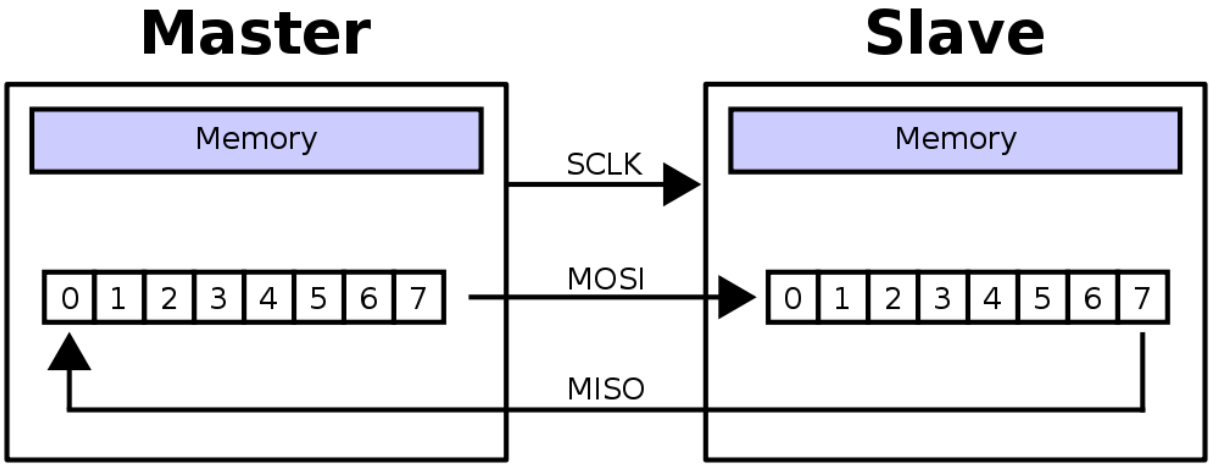
Η επικοινωνία μεταξύ των SPI συσκευών είναι full duplex και χρησιμοποιεί την αρχιτεκτονική master – slave. Ο master είναι πάντα μοναδικός, ενώ οι slaves μπορεί να είναι περισσότεροι του ενός. Συνήθως, οι τελευταίοι συνδέονται είτε σε τοπολογία δακτυλίου, είτε παράλληλα με τον master και επομένως είναι ανεξάρτητοι μεταξύ τους. Ο master επίσης καθορίζει και εναλλάσσει τις λειτουργίες εγγραφής και ανάγνωσης και είναι αυτός που ορίζει τους συμμετέχοντες slaves μέσω της θύρας slave select (ss).

Η λειτουργία του SPI καθορίζεται από 4 λογικά σήματα:

- SCLK: Serial Clock (έξοδος του master)
- MOSI: Master Output Slave Input (έξοδος δεδομένων του master)
- MISO: Master Input Slave Output (έξοδος δεδομένων του slave)
- SS: Slave Select (επιλογή slave από τον master)

Για την έναρξη της επικοινωνίας, ο master καθορίζει το ρολόι συγχρονισμού και επιλέγει τον slave με τον οποίο θα γίνει η επικοινωνία. Σε κάθε κύκλο του ρολογιού, ο master στέλνει ένα bit στην έξοδό του, MOSI, την οποία διαβάζει ο slave. Ταυτόχρονα ο slave στέλνει ένα bit για να διαβάσει ο master μέσω της εξόδου του, MISO. Η ακολουθία αυτή επαναλαμβάνεται συνεχώς και διατηρείται ακόμα και στην περίπτωση μονομερούς επικοινωνίας.

Για να υλοποιηθεί η παραπάνω λειτουργία, υπάρχουν δύο καταχωρητές ολίσθησης (shift registers), ένας στον master και ένας στον slave. Το μέγεθός τους καθορίζει και τη λέξη επικοινωνίας. Οι καταχωρητές μπορεί να θεωρηθεί πως συνδέονται σε μία νοητή τοπολογία δακτυλίου, όπως φαίνεται και στην Εικόνα 3-1. Έτσι, σε έναν κύκλο ρολογιού οι καταχωρητές τόσο του master, όσο και του slave, ολισθαίνουν κατά ένα bit και βγάζουν στην έξοδό τους το ψηφίο που είναι στη μία άκρη. Στον αμέσως επόμενο κύκλο, το bit που εξάγει ο κάθε register γράφεται στην άλλη άκρη του άλλου register. Συνήθως εξάγεται το MSB του κάθε register και γράφεται στο LSB του άλλου register.



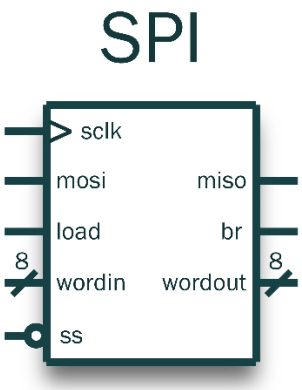
Εικόνα 3-1 Υλοποίηση SPI με 2 shift registers

Όταν όλα τα bit των registers έχουν μεταφερθεί, τότε οι δύο registers έχουν ανταλλάξει τιμές και έχει μεταφερθεί μία πλήρης λέξη από τον master στον slave και το αντίστροφο. Συνήθως απαιτείται η μετάδοση περισσότερων της μίας λέξης και επομένως η επικοινωνία συνεχίζει με τον ίδιο τρόπο έως ότου μεταδοθούν όλα τα απαιτούμενα bit.

Ο λόγος που επιλέχθηκε η σειριακή επικοινωνία για τον προγραμματισμό του DDS ήταν κυρίως η ελαχιστοποίηση του αριθμού των εισόδων / εξόδων του ολοκληρωμένου, καθώς το μέγεθος των registers στον phase accumulator είναι αρκετά μεγάλο (48 bit). Επίσης, τα συνολικά δεδομένα που χρειάζεται να μεταδοθούν έχουν μέγεθος μερικών bytes και επομένως σε συχνότητες τάξης MHz, που μπορεί να λειτουργεί το σειριακό ρολόι, ο χρόνος μετάδοσης είναι πάρα πολύ μικρός και δεν προσδίδει σημαντική καθυστέρηση στον χρόνο που απαιτείται για τον προγραμματισμό του DDS.

Σε ό,τι αφορά την επιλογή του SPI ως σειριακή διασύνδεση, είναι από τα πιο καθιερωμένα συστήματα που χρησιμοποιούνται για τον σκοπό αυτό. Είναι ακόμη συμβατό με μια πληθώρα μικροελεγκτών, μικροεπεξεργαστών και γενικότερα κυκλωμάτων που ενδείκνυνται για τον προγραμματισμό και έλεγχο του DDS. Τέλος, είναι ένα αρκετά απλό και αποδοτικό κύκλωμα.

3.2.2 Δομή



Εικόνα 3-2 Μονάδα SPI

I/O Pin	I/O Name	I/O Type	I/O Length	I/O Logic	Triggered On
sclk	Serial Clock	input	1	positive	positive edge
mosi	Master Out Slave In	input	1	positive	-
load	Load parallel enable	input	1	positive	-
wordin	Word In parallel	input	8	positive	-
ss	Slave Select	input	1	negative	positive edge
br	Buffer Ready	output	1	positive	-
miso	Master In Slave Out	output	1	positive	-
wordout	Word Out parallel	output	8	positive	-

Πίνακας 3-2 Είσοδοι / Έξοδοι SPI

Η είσοδος sclk αποτελεί το σειριακό ρολόι του SPI, το οποίο επιλέχθηκε να έχει ως μέγιστη συχνότητα τα 25 MHz, καθώς συνήθως οι master συγχρονίζονται σε μερικά MHz για τη σειριακή επικοινωνία με SPI. Οι λειτουργίες της μονάδας λαμβάνουν χώρα με τον θετικό παλμό του ρολογιού.

Η είσοδος ss, η οποία (όπως και το sclk) ρυθμίζεται από τον master της διασύνδεσης, χρησιμοποιεί αρνητική λογική. Όταν λοιπόν, βρίσκεται σε λογικό μηδέν, τότε το SPI τίθεται σε λειτουργία. Αντίθετα, με τον ανοδικό παλμό του σήματος ss η μονάδα αρχικοποιείται. Σημειώνεται ακόμη πως σε υψηλό δυναμικό του ss, όλοι οι έξοδοι είναι σε υψηλή αντίσταση (κατάσταση Z) και αποκόπτονται από το υπόλοιπο κύκλωμα.

Το mosi είναι η είσοδος όπως έρχεται από τον master και το miso η έξοδος του SPI που πηγαίνει προς εγγραφή στον master.

Μόλις μεταδοθεί μία πλήρης λέξη, η έξοδος br μεταφέρεται σε υψηλό δυναμικό για τον επόμενο κύκλο του ρολογιού, σηματοδοτώντας έτσι την ετοιμότητα για ανάγνωση ή εγγραφή της τιμής του εσωτερικού buffer.

Τα wordin και wordout χρησιμοποιούνται για την παράλληλη εισαγωγή και εξαγωγή της τρέχουσας λέξης προς και από το εσωτερικό του SPI αντίστοιχα. Σε περίπτωση λοιπόν, που το SPI χρησιμοποιείται για την ανάγνωση λέξης από τον master, το wordout προσφέρει τη λέξη που έχει διαβαστεί. Για να μεταδοθεί μία λέξη προς τον master, παρέχεται η δυνατότητα παράλληλης φόρτωσής της μέσω της εισόδου wordin.

Για να διαχωριστούν οι δύο παραπάνω καταστάσεις, χρησιμοποιείται η είσοδος load, η οποία όταν βρίσκεται σε υψηλό δυναμικό φορτώνεται η τιμή του wordin στο εσωτερικό register. Η τιμή του register είναι διαθέσιμη μέσω της εξόδου wordout ανεξαρτήτως της κατάστασης του load.

3.2.3 Λειτουργία

Το SPI που υλοποιήθηκε για τη σειριακή επικοινωνία του DDS ακολουθεί όλες τις αρχές και την αρχιτεκτονική ενός κλασικού SPI, όπως περιγράφεται στην Εισαγωγή. Επιπλέον των λειτουργιών αυτών, δίνεται η δυνατότητα παράλληλης ανάγνωσης και εγγραφής του εσωτερικού register (το οποίο στη συνέχεια θα αποκαλείται buffer). Η δυνατότητα αυτή είναι αναγκαία για τη μεταφορά των λέξεων από και προς τους εσωτερικούς καταχωρητές του συστήματος, ενώ επίσης είναι διαθέσιμη μόνο στο εσωτερικό του DDS και δεν είναι προσπελάσιμη από εξωτερικά pins του ολοκληρωμένου κυκλώματος.

Εκτός των εισόδων και εξόδων της μονάδας, χρησιμοποιούνται ακόμη δύο registers για την επίτευξη της επιθυμητής λειτουργίας του SPI. Το πρώτο είναι το buffer όπου αποθηκεύεται η λέξη που μεταδίδεται ή / και λαμβάνεται ολισθαίνοντας με το bit εισόδου του master. Το δεύτερο είναι το counter που χρησιμοποιείται για τη μέτρηση των bit που ανταλλάσσονται και σηματοδοτεί την ολοκλήρωση της μεταφοράς μίας πλήρους λέξης.

Όσο η είσοδος ss βρίσκεται σε χαμηλό επίπεδο, το SPI λειτουργεί κανονικά. Όταν όμως το ss βρεθεί σε υψηλό επίπεδο, το counter μηδενίζεται, ώστε μόλις τεθεί ξανά σε λειτουργία, το SPI να είναι πλήρως συγχρονισμένο με τον master. Το buffer δε χρειάζεται να μηδενιστεί, καθώς ακόμα και αν περιέχει παλιές τιμές (garbage), μπορούν να αγνοηθούν από τον master κατά την αρχή της επικοινωνίας. Ακόμη, όσο το ss βρίσκεται σε υψηλό δυναμικό, όλες οι έξοδοι είναι σε κατάσταση υψηλής αντίστασης Z, ώστε να είναι αποκομμένες από το υπόλοιπο κύκλωμα, μιας και όσο δε λειτουργεί το SPI δε μεταδίδουν κάποια πληροφορία και επομένως δεν πρέπει να επηρεάζουν το υπόλοιπο κύκλωμα.

Σχετικά με τις εξόδους, το br υποδεικνύει τον μηδενισμό του counter και βρίσκεται σε υψηλή κατάσταση για 1 κάθε 8 κύκλους, όποτε και όσο δηλαδή το counter είναι μηδέν. Με τον

τρόπο αυτό, δηλώνεται πως το buffer είναι έτοιμο είτε επειδή περιέχει μία ολόκληρη λέξη προς ανάγνωση, είτε για να του εγγραφεί μία νέα τιμή προς μετάδοση. Το wordout είναι συνεχώς συνδεδεμένο με το buffer, καθιστώντας το περιεχόμενό του διαθέσιμο για ανάγνωση. Τέλος, η έξοδος miso είναι διαρκώς συνδεδεμένη με το LSB του buffer.

Στον θετικό παλμό του sclk ή του ss, αν το ss είναι υψηλό, η κατάσταση του SPI αρχικοποιείται. Αντίθετα, εάν ξεκινάει η μετάδοση (counter = 0) και το load είναι υψηλό, τότε το περιεχόμενο του wordin αντιγράφεται στο buffer. Στον ίδιο κύκλο, η έξοδος miso παίρνει την τιμή του LSB της νέας λέξης που μόλις διαβάστηκε και επομένως η μετάδοση της λέξης ξεκινά απευθείας. Σε περίπτωση που το load είναι χαμηλό ή βρισκόμαστε στο μέσο της μετάδοσης (counter ≠ 0), το buffer ολισθαίνει κατά ένα bit δεξιά και στη θέση του MSB του γράφεται η τιμή του mosi. Μετά την ολίσθηση, η έξοδος miso αλλάζει δίνοντας το επόμενο bit του buffer. Το counter τέλος, αυξάνεται κατά ένα σε κάθε παλμό όταν το SPI βρίσκεται σε λειτουργία (ss = 0).

3.2.4 Προσομοίωση

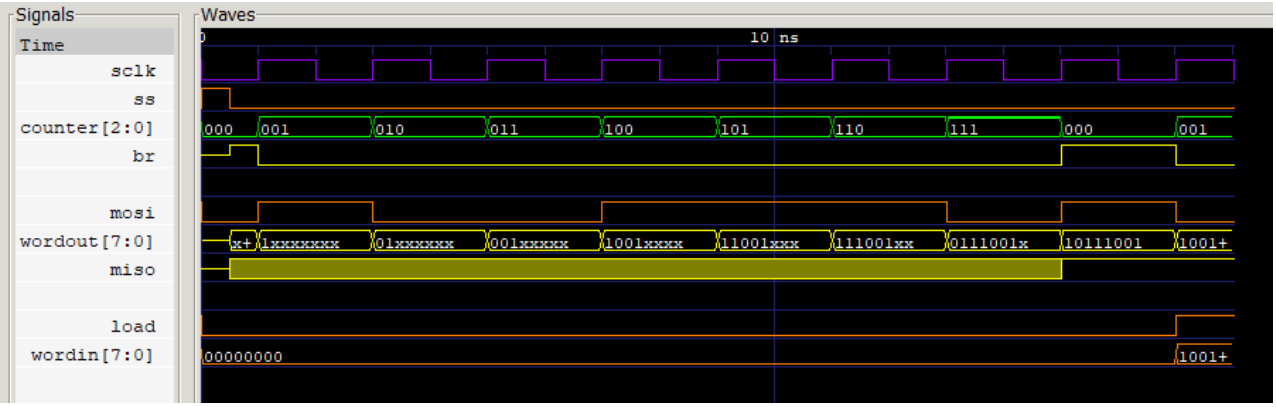
Προτού ξεκινήσει η ανάλυση των testbenches, παρατίθενται μερικές σημειώσεις για την ευκολότερη ανάγνωση των σημάτων. Σε όλα τα διαγράμματα που ακολουθούν στην παρούσα εργασία, το ρολόι του κυκλώματος θα σημειώνεται με μωβ χρώμα, οι είσοδοι με πορτοκαλί, οι έξοδοι με κίτρινο και οι θύρες διπλής κατεύθυνσης και τα λοιπά εσωτερικά σήματα με πράσινο χρώμα.

Για την υλοποίηση του testbench, δημιουργείται ένα νέο module χωρίς εισόδους και εξόδους. Περιέχει όμως έναν register αντίστοιχου μήκους για κάθε είσοδο του SPI και ένα wire για κάθε έξοδο. Με τον τρόπο αυτό, ελέγχουμε τις τιμές των εισόδων του SPI και είμαστε σε θέση να παρακολουθήσουμε τις εξόδους του μέσω των καλωδίων (wire) που συνδέονται σε αυτές.

Κατά την έναρξη του testbench, αρχικοποιούνται όλοι οι καταχωρητές, ώστε οι τιμές των εισόδων του SPI να έχουν γνωστές τιμές και να μη βρίσκονται σε κατάσταση X. Οι αρχικές τιμές είναι μηδέν, εκτός του ss που αρχικοποιείται σε υψηλό δυναμικό. Ορίζεται επίσης το αρχείο εξόδου της λογικής ανάλυσης με την εντολή $dumpfile, η έναρξη της καταγραφής των σημάτων με την $dumpvar και ο καθορισμός της ολοκλήρωσης της ανάλυσης με την $finish. Σημειώνεται πως είναι απαραίτητο να δηλώσουμε το πέρας της ανάλυσης, καθώς διαφορετικά ο προσομοιωτής θα τρέχει για πάντα.

Η χρονική μονάδα έχει οριστεί να είναι το 1 ns με ανάλυση χρόνου 100 ps. Με άλλα λόγια έχουμε ακρίβεια έως και ένα δεκαδικό ψηφίο της στοιχειώδους μονάδας χρόνου. Ρυθμίζουμε το ρολόι, ώστε να έχει περίοδο 2 ns, ενώ διαρκώς και κάθε μισό της περιόδου η τιμή του αλλάζει στην αρνητική της, δημιουργώντας έτσι τετραγωνικούς παλμούς. Σημειώνεται πως η ανάλυση είναι ιδανική και δε λαμβάνει υπόψη μεταβατικά φαινόμενα, όπως οι καθυστερήσεις των πυλών και επομένως η μονάδα μέτρησης χρόνου είναι ενδεικτική χωρίς να επηρεάζει τη συγκεκριμένη ανάλυση.

Για την επαλήθευση της σωστής λειτουργίας του SPI, χρειάζεται η ανάλυση των δύο διαφορετικών περιπτώσεων που μπορεί να βρεθεί. Αρχικά λοιπόν, μελετάται η περίπτωση όπου θέλουμε να διαβάσουμε μία λέξη από τον master και στη συνέχεια η περίπτωση όπου μεταδίδεται μία λέξη προς τον master.



Εικόνα 3-3 Προσομοίωση SPI: λειτουργία ανάγνωσης λέξης

Κατά την έναρξη της ανάλυσης, παρατηρούμε πως μέχρι να μεταβάλουμε το ss στη χαμηλή κατάσταση, όλες οι έξοδοι βρίσκονται σε κατάσταση υψηλής αντίστασης (το σήμα σημειώνεται στο ενδιάμεσο από τη χαμηλή και την υψηλή κατάσταση), όπως ήταν επιθυμητό. Επίσης, το buffer και επομένως οι έξοδοι wordout και miso που εξαρτώνται από αυτό, περιέχουν bit σε άγνωστη κατάσταση X. Όπως είπαμε, η κατάσταση αυτή δεν επηρεάζει σε κάτι την επικοινωνία, καθώς εφόσον διαβάζουμε από τον master και δεν επιθυμούμε να του στείλουμε κάποια πληροφορία, το miso και το wordout μπορούν να έχουν οποιαδήποτε τιμή, αφού δεν παίζει κανέναν ρόλο.

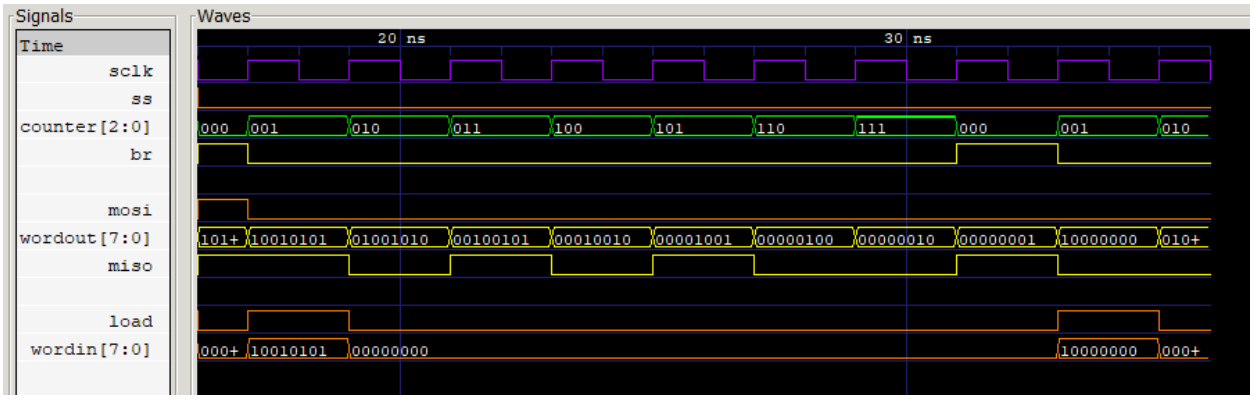
Με μια πρώτη ματιά, παρατηρούμε ακόμη τη σωστή λειτουργία του counter και του br. Το πρώτο μετράει σωστά με βήμα 1 και μετά τη μέγιστη τιμή του ξεκινάει και πάλι από το μηδέν. Το br επίσης είναι σε λογική κατάσταση 1 ακριβώς όσο ο counter είναι μηδενισμένος, υποδεικνύοντας ότι το buffer είναι έτοιμο.

Θα επιχειρήσουμε τώρα να μεταδώσουμε (από τον master) στο SPI τη λέξη 10111001 ξεκινώντας από το λιγότερο σημαντικό ψηφίο και προχωρώντας μέχρι το πιο σημαντικό. Έτσι, σε κάθε περίοδο θέτουμε το mosi στην αντίστοιχη κατάσταση με το bit που έχει σειρά να μεταδοθεί. Ανάλογα μεταβάλλεται και το MSB του εσωτερικού buffer, το οποίο παίρνει την τρέχουσα τιμή του mosi. Είναι εμφανής η δεξιά ολίσθηση που συμβαίνει στον buffer σε κάθε θετικό παλμό του ρολογιού και ταυτόχρονα η εμφάνιση του LSB στην έξοδο miso. Το τελευταίο γίνεται ευκολότερα κατανοητό τη χρονική στιγμή 15 ns, όπου ο πρώτος άσος που διαβάστηκε γίνεται πλέον το LSB του buffer και εμφανίζεται στην έξοδο miso.

Στα 15 ns επίσης, ολοκληρώνεται η μετάδοση της λέξης. Το περιεχόμενο του buffer και ισοδύναμα του wordout είναι ακριβώς αυτό που περιμέναμε. Το br μεταβαίνει σε υψηλό

δυναμικό για να επισημάνει την ολοκλήρωση εισαγωγής της λέξης και ότι η τιμή της είναι έτοιμη για ανάγνωση από την παράλληλη θύρα wordout.

Στη συνέχεια προχωρούμε στην ανάλυση για την επαλήθευση της μετάδοσης λέξης από το SPI προς τον master. Στον επόμενο κύκλο από την υπόδειξη πως το buffer είναι έτοιμο, θέτουμε το load σε υψηλό δυναμικό. Ταυτόχρονα δίνουμε στην είσοδο wordin (μέσω του register temp του testbench) την τιμή 10010101, η οποία πρέπει να μεταδοθεί προς τον master. Η τιμή αυτή αντιγράφεται αρχικά στο buffer για να ξεκινήσει η διαδικασία.



Εικόνα 3-4 Προσομοίωση SPI: λειτουργία μετάδοσης λέξης

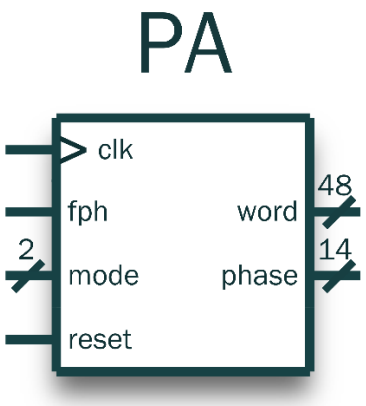
Όπως στην ανάγνωση λέξης, έτσι και στη μετάδοση, η διαδικασία ξεκινά όταν το counter είναι ένα και ολοκληρώνεται όταν αυτό μηδενιστεί. Με άλλα λόγια, το πρώτο bit εμφανίζεται στην έξοδο όταν το counter είναι ένα και το τελευταίο όταν ξαναγίνει μηδέν. Επίσης, παρατηρούμε την ορθή μετάδοση της λέξης, μιας και το miso λαμβάνει τις σωστές τιμές.

Οι δύο λειτουργίες (ανάγνωση και μετάδοση) είναι ανεξάρτητες και ομοιόμορφες, αφού ορίζουν κοινή διάρκεια και συγκεκριμένη αρχή και τέλος. Έτσι, είναι δυνατή η ανάγνωση λέξης αμέσως μετά από μία μετάδοση ή και αντίστροφα χωρίς να υπάρχει καμία καθυστέρηση ή κάποιο σφάλμα.

Στο testbench επιλέχθηκε η μετάδοση νέας λέξης ακριβώς μετά το πέρας της 10010101. Η νέα λέξη είναι η 10000000 και τη βλέπουμε να φορτώνεται στο buffer τη χρονική στιγμή 33 ns. Έτσι, στέλνονται δύο διαδοχικά byte χωρίς κανένα πρόβλημα και καθυστέρηση. Σημειώνεται πως λόγω της full duplex φύσης της SPI επικοινωνίας, κατά τη μετάδοση ενός byte είναι δυνατή και η ταυτόχρονη ανάγνωση ενός άλλου από τον master, μειώνοντας τον απαιτούμενο χρόνο στο μισό.

3.3 Συσσωρευτής Φάσης (Phase Accumulator, PA)

3.3.1 Δομή



Εικόνα 3-5 Μονάδα PA

I/O Pin	I/O Name	I/O Type	I/O Length	I/O Logic	Triggered On
clk	Clock	input	1	positive	positive edge
fph	Frequency / Phase select	input	1	positive	-
mode	function Mode	input	2	positive	-
reset	Reset	input	1	positive	positive edge
word	Word in / out	bidirectional	48	positive	-
phase	Phase	output	14	positive	-

Πίνακας 3-3 Είσοδοι / Έξοδοι PA

Το PA είναι και αυτό ένα σύγχρονο κύκλωμα. Το ρολόι του όμως, είναι διαφορετικό από το σειριακό ρολόι του SPI. Στο clk συνδέεται το κύριο ρολόι του συνολικού συστήματος, που είναι χρονισμένο στα 2,8 GHz.

Η είσοδος mode χρησιμοποιείται για την εναλλαγή μεταξύ της κανονικής λειτουργίας του PA και των λειτουργιών ανάγνωσης και εγγραφής των register του. Με το fph επιλέγεται είτε το register της συχνότητας, είτε αυτό της φάσης όταν είμαστε σε mode ανάγνωσης ή εγγραφής. Οι διαφορετικές λειτουργίες που υποστηρίζονται φαίνονται στον παρακάτω πίνακα:

mode	
1	**0**

00	Κανονική λειτουργία
01	Λειτουργία αναμονής / αδράνειας
10	Λειτουργία ανάγνωσης του καταχωρητή που δείχνει το fph
11	Λειτουργία εγγραφής στον καταχωρητή που δείχνει το fph

Πίνακας 3-4 Καταστάσεις λειτουργίας PA

Συνοπτικά για τις διάφορες λειτουργίες, όταν το MSB του mode είναι 1, το PA εκτελεί read / write διαδικασίες ανάλογα με το LSB του mode. Το 00 υποδεικνύει την κανονική του

λειτουργία, ενώ το 01 την αδράνεια της μονάδας. Στην τελευταία περίπτωση το PA δεν εκτελεί καμία ενέργεια έως ότου αλλάξει ξανά το mode. Η αναγκαιότητα της λειτουργίας αυτής θα γίνει σαφής σε επόμενες ενότητες με την περιγραφή των μονάδων OPC και NCO.

Η θύρα word είναι δύο κατευθύνσεων και χρησιμοποιείται αποκλειστικά όταν το MSB του mode είναι 1 και για την μεταφορά των δεδομένων των εσωτερικών registers από και προς τη μονάδα. Κατά τη λειτουργία ανάγνωσης λοιπόν, το word γίνεται έξοδος, ενώ κατά την εγγραφή είσοδος.

Η έξοδος phase δίνει την τρέχουσα φάση του συσσωρευτή (accumulator) με ακρίβεια 14 bit.

Τέλος, το PA υποστηρίζει δυνατότητα μηδενισμού των καταχωρητών του μέσω του reset pin, το οποίο ενεργοποιείται με θετικό παλμό. Το reset μπορεί να συμβεί ασύγχρονα από το ρολόι.

3.3.2 Λειτουργία

Το Phase Accumulator που υλοποιήθηκε παρέχει όλες τις δυνατότητες, όπως περιγράφονται στην Συσσωρευτής Φάσης (Phase Accumulator, PA) και επιπλέον δίνει την επιλογή για απευθείας πρόσβαση στους καταχωρητές του. Η τελευταία δυνατότητα είναι διαθέσιμη μόνο στο εσωτερικό του συνολικού συστήματος του DDS, καθώς η επεξεργασία των τιμών των registers του PA επιτρέπεται μόνο με σειριακό τρόπο μέσω της διασύνδεσης SPI.

Αρχικά, ορίζονται οι παράμετροι που αφορούν τα χαρακτηριστικά της μονάδας. Η πρώτη παράμετρος είναι το μέγεθος του accumulator M και η δεύτερη η ανάλυση φάσης N στην έξοδο του PA. Ο λόγος παραμετροποίησης των εν λόγω τιμών, αφορά τη δυνατότητα επαναχρησιμοποίησης της ίδιας σχεδίασης για διαφορετικών χαρακτηριστικών PA χωρίς την αλλαγή του κώδικα. Οι προκαθορισμένες τιμές, οι οποίες χρησιμοποιούνται και στη μετέπειτα υλοποίηση, είναι $M = 48\ bit$ και $N = 14\ bit$. Έτσι, σύμφωνα με την εξίσωση (3) η ανάλυση συχνότητας είναι ίση με:

$$f_{res} = \frac{f_s}{2^M} = \frac{(2{,}8 \cdot 10^9)}{2^{48}} \cong 10^{-5}\ Hz = 10\ \mu Hz$$

Η ανάλυση φάσης των $N = 14\ bit$, επιτρέπει τη χρήση έως και $2^{14} = 16.384$ διαφορετικών σημείων για την ανακατασκευή του σήματος.

Στη συνέχεια δηλώνονται οι είσοδοι και οι έξοδοι του κυκλώματος, όπως υποδεικνύονται στην Εικόνα 3-5. Επιπλέον, δημιουργούνται δύο καταχωρητές που είναι απαραίτητοι και αποτελούν την «καρδιά» του phase accumulator. Ο ένας είναι ο freq (frequency register), όπου αποθηκεύεται η λέξη ελέγχου συχνότητας FCW, και ο άλλος είναι ο acc (phase register), ο οποίος χρησιμοποιείται ως accumulator για την τρέχουσα φάση.

Ακολουθεί ο ορισμός των σημάτων εξόδου. Αρχικά, η έξοδος phase αντιστοιχίζεται στα N πιο σημαντικά ψηφία του accumulator ανεξαρτήτως του mode που βρίσκεται ο PA. Αντίθετα,

η θύρα διπλής κατεύθυνσης word εξαρτάται από την τρέχουσα λειτουργία. Έτσι, αν ζητείται η ανάγνωση ενός εσωτερικού register (mode = 10), το word μετατρέπεται σε έξοδο και δίνει την τιμή του επιλεγμένου register με βάση το fph. Η άλλη περίπτωση όπου χρησιμοποιείται το word είναι κατά τη λειτουργία εγγραφής (mode = 11), όπου λειτουργεί σαν είσοδος για να λάβει την τιμή προς εγγραφή. Για να επιτευχθεί αυτό, το word αποσυνδέεται από τα εσωτερικά registers για να μη δημιουργηθεί βραχυκύκλωμα. Αντίστοιχα και στις υπόλοιπες λειτουργίες, το word επίσης αποκόπτεται από τα εσωτερικά registers μιας και δε χρησιμοποιείται.

Στο κυρίως κομμάτι του PA, όταν το reset τεθεί σε υψηλό δυναμικό, οι εσωτερικοί καταχωρητές μηδενίζονται. Στην περίπτωση αυτή, η έξοδος phase θα γίνει επίσης μηδέν, ενώ αν το mode είναι 10, τότε και το word θα δίνει έξοδο μηδέν.

Εάν από την άλλη το reset βρίσκεται σε χαμηλό δυναμικό, τότε με τον θετικό παλμό του ρολογιού και όταν το mode είναι ρυθμισμένο στην κανονική λειτουργία, ο συσσωρευτής acc αυξάνεται κατά βήμα FCW και αυτομάτως ανανεώνεται η έξοδος phase. Υπάρχει περίπτωση η phase να μην αλλάξει όταν τα N πιο σημαντικά ψηφία του acc παραμείνουν ίδια μετά την πρόσθεση. Για να συμβεί αυτό, πρέπει να ισχύει $acc + freq < 2^{M-N} = 2^{34}$.

Τέλος, με τον θετικό παλμό του ρολογιού, όταν δε βρισκόμαστε σε κατάσταση reset και η λειτουργία έχει οριστεί σε αυτή της εγγραφής (mode = 11), το word λειτουργεί ως είσοδος και μεταφέρει την τιμή του στα εσωτερικά registers. Εάν $fph = 1$, επιλέγεται ο freq register για εγγραφή, αλλιώς επιλέγεται ο acc. Με τον τρόπο αυτό μπορούμε να ελέγχουμε πλήρως τη συχνότητα εξόδου και τη μετατόπιση φάσης του DDS.

3.3.3 Προσομοίωση

Αφού πρώτα γίνει ο ορισμός της χρονικής κλίμακας στο 1 ns / 100 ps, στη συνέχεια ξεκινά ο ορισμός της μονάδας του testbench όπου θα δημιουργηθούν όλα τα σενάρια ελέγχου για την επαλήθευση της σωστής λειτουργίας του phase accumulator. Όπως σε όλα τα testbenches, το module που δημιουργείται δεν έχει εισόδους και εξόδους, παρά μόνο εσωτερικά registers και wires που συνδέονται με τις εισόδους και τις εξόδους αντίστοιχα της ελεγχόμενης μονάδας.

Σε ό,τι αφορά τη θύρα δύο κατευθύνσεων word, δημιουργείται ένα register και ένα wire ίδιου μήκους. Έτσι, όταν συμπεριφέρεται ως έξοδος (mode = 10) το απομονώνουμε, διότι θα συνδεθεί στο εσωτερικό της μονάδας με το αντίστοιχο register, ενώ σε αντίθετη περίπτωση το συνδέουμε με το temp του testbench[2].Το wire word λοιπόν του testbench, ορίζεται ακριβώς αντίστροφα (συνθήκη της εντολής assign) από τη θύρα word εντός του PA, ώστε να μην υπάρξει βραχυκύκλωμα.

Συνεχίζοντας, δημιουργούμε ένα PA και συνδέουμε κατάλληλα τις εισόδους και εξόδους του με τα registers και wires του testbench, αντίστοιχα. Ένα από τα πλεονεκτήματα των

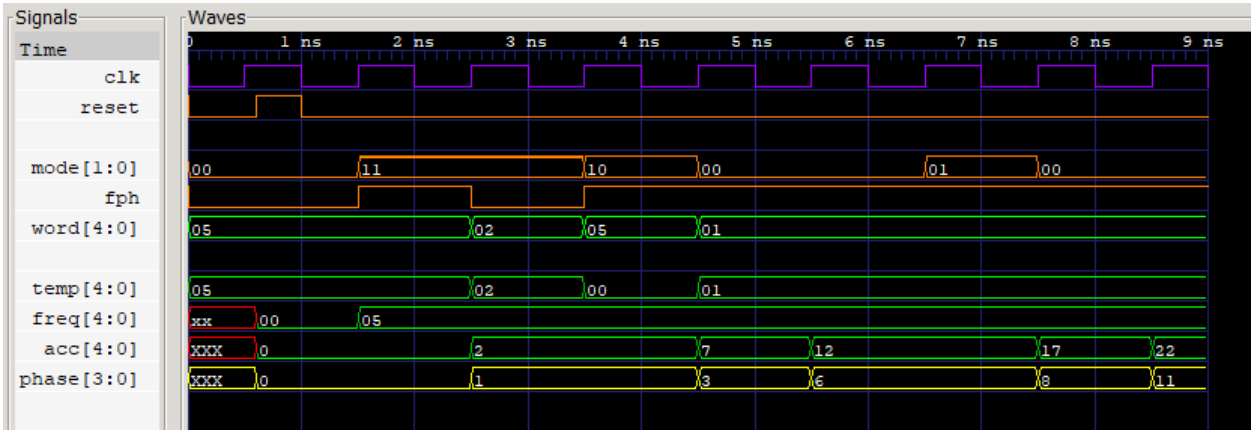
[2] Το word αποτελεί είσοδο για κάθε λειτουργία mode ≠ 10. Έτσι, το word του testbench είναι συνδεδεμένο με το temp, όσο το mode ≠ 10. Όπως έχει αναφερθεί, εφόσον το word δε χρησιμοποιείται στα modes 00 και 01, η τιμή του temp δεν επηρεάζει σε κάτι.

παραμέτρων που χρησιμοποιήσαμε στη σχεδίαση της μονάδας του PA, είναι ότι μπορούμε να προσομοιώσουμε και να επαληθεύσουμε την ομαλή λειτουργία του με μικρότερα μεγέθη. Έτσι, χωρίς βλάβη της γενικότητας η μονάδα που θα ελέγξουμε αποτελείται από frequency και phase registers μεγέθους $M = 5\ bit$ και ανάλυση συχνότητας $N = 4\ bit$. Με τον τρόπο αυτό, γίνεται πολύ ευκολότερη η διαχείριση των αποτελεσμάτων της ανάλυσης και η εξαγωγή των τελικών συμπερασμάτων.

Στη συνέχεια γίνεται η αρχικοποίηση των σημάτων που θα χρησιμοποιηθούν και ο ορισμός των λεπτομερειών της ανάλυσης (αρχείο εξόδου αποτελεσμάτων, έναρξη και πέρας προσομοίωσης). Το ρολόι ορίζεται να αλλάζει την κατάστασή του στην αρνητική του κάθε 0.5 ns, δημιουργώντας έναν τετραγωνικό παλμό περιόδου 1 ns.

Για να έχει νόημα η ανάλυση, θα πρέπει να δοκιμάσουμε όλες τις διαφορετικές καταστάσεις λειτουργίας (modes) που μπορεί να βρεθεί ο phase accumulator. Οι τιμές που θα χρησιμοποιήσουμε για τα FCW και phase offset είναι ενδεικτικές και δεν παίζουν κάποιο ρόλο στη συνολική ανάλυση.

Το πρώτο που ελέγχουμε στο κύκλωμα είναι η σωστή συμπεριφορά του σήματος reset. Έτσι, όπως φαίνεται στην Εικόνα 3-6, μέχρι τη στιγμή που το reset μεταβαίνει σε υψηλό δυναμικό, οι τιμές των εσωτερικών register και επομένως και της εξόδου phase είναι άγνωστες. Επίσης, επαληθεύεται η ασύγχρονη δυνατότητα του reset, το οποίο προκύπτει ανεξαρτήτως της κατάστασης του ρολογιού και με τον θετικό παλμό του οποίου, τα εσωτερικά registers μηδενίζονται αμέσως. Στη συνέχεια πρέπει να μηδενίσουμε το σήμα reset, διότι διαφορετικά δεν μπορεί να αλλάξει η κατάσταση του PA, ανεξαρτήτως του mode και των λοιπών εισόδων.



Εικόνα 3-6 Προσομοίωση λειτουργιών PA

Η προσομοίωση ξεκινά με το mode = 11, δηλαδή τη λειτουργία εγγραφής. Αρχικά μαζί με την εκχώρηση της τιμής 11 στην είσοδο mode, θέτουμε το fph = 1 για να επιλέξουμε το freq register για εγγραφή. Η τιμή που του δίνουμε είναι η FCW = 5 μέσω του temp, το οποίο συνδέεται με το word, όπως περιγράφεται προηγουμένως.

Έπειτα, θέλουμε η αρχική φάση να είναι 2, ώστε το παραγόμενο σήμα να είναι μετατοπισμένο κατά 2 μονάδες δεξιά[3]. Έτσι, κρατώντας το mode σε λειτουργία εγγραφής, αλλάζουμε το fph σε 0 και το temp σε 2, ώστε να αλλάξει η τιμή του acc.

Εφόσον πλέον έχουν γραφεί οι επιθυμητές τιμές στα registers freq και acc, όπως διακρίνεται στην Εικόνα 3-6, θα επαληθεύσουμε τη λειτουργία ανάγνωσης. Αλλάζουμε λοιπόν το mode σε 10 και θέτουμε το fph = 1, για να διαβάσουμε την τιμή του freq. Το word μετατρέπεται σε έξοδος και σωστά δίνει την τιμή 5 που είναι αποθηκευμένη στο freq.

Πλέον το PA είναι έτοιμο για κανονική λειτουργία. Θέτουμε το mode = 00, ενώ πλέον η τιμή των fph και word είναι αδιάφορη. Σε κάθε εκθετικό παλμό και όσο το mode είναι στην κανονική λειτουργία, ο accumulator αυξάνεται με βήμα 5, όσο δηλαδή είναι το FCW. Έτσι, ξεκινώντας από την αρχική τιμή 2 (phase offset), το accumulator μεταβαίνει στις τιμές 7, 12, 17 και ούτω καθ' εξής.

Τη χρονική στιγμή 6.5 ns ελέγχεται και η τελευταία λειτουργία που έχει μείνει, αυτή της αναμονής / αδράνειας. Όπως ήταν αναμενόμενο, όσο βρισκόμαστε στη συγκεκριμένη λειτουργία το PA παραμένει αδρανές και δεν εκτελεί καμία πράξη. Έτσι, η έξοδος phase παραμένει σταθερή για όσο ισχύει mode = 01.

Κλείνοντας την ανάλυση του PA, μπορούμε να κάνουμε μερικές ακόμα παρατηρήσεις. Αρχικά, βλέπουμε πως η έξοδος phase ισούται με το πηλίκο της ακέραιας διαίρεσης του καταχωρητή acc με το 2: $phase = acc\ div\ 2$. Το γεγονός αυτό είναι αναμενόμενο, καθώς η phase αποτελεί τα $N = 4$ σημαντικότερα ψηφία του acc. Έτσι, απορρίπτεται το $M - N = 5 - 4 = 1$ τελευταίο ψηφίο του acc. Η πράξη αυτή ισοδυναμεί με δεξιά ολίσθηση μίας θέσης ή αλλιώς με ακέραια διαίρεση του acc με το 2. Γενικεύοντας για οποιεσδήποτε παραμέτρους M, N, η περικομμένη έξοδος phase θα ισούται με: $phase = acc\ div\ 2^{M-N}$.

Τέλος, παρατηρούμε πως όταν το mode είναι 10 (χρονική στιγμή 3,5 ns), το temp αγνοείται όπως ήταν αναμενόμενο, ενώ όταν mode = 00 ή 01 (χρονική στιγμή 4,5 ns και έπειτα) επίσης το temp δεν επηρεάζει σε κάτι τη σωστή λειτουργία του PA.

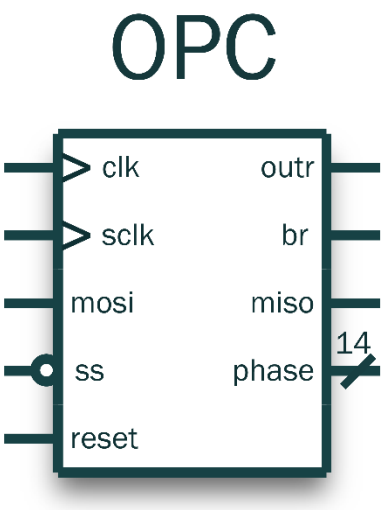
[3] Η φάση στο PA δεν έχει μονάδα μέτρησης. Για να την μετατρέψουμε σε μονάδα μέτρησης γωνίας, πρέπει να κάνουμε την αντίστοιχη μετατροπή. Έτσι, αν λέμε ότι έχουμε μετατόπιση φάσης κατά $\varphi_0 = 2$ μονάδες, τότε το ισοδύναμο σε μοίρες είναι:

$$\frac{\varphi_0}{2^M} = \frac{\alpha}{360°} \Leftrightarrow \alpha = 360° \cdot \frac{2}{2^5} \Leftrightarrow \boxed{\alpha = 22{,}5°}$$

όπου α είναι η φάση εκφρασμένη σε μοίρες.

3.4 Ρυθμιστής Προφίλ Λειτουργίας (Operation Profile Configurator, OPC)

3.4.1 Δομή



Εικόνα 3-7 Μονάδα OPC

I/O Pin	I/O Name	I/O Type	I/O Length	I/O Logic	Triggered On
clk	Clock	input	1	positive	positive edge
sclk	Serial Clock	input	1	positive	positive edge
mosi	Master Out Slave In	input	1	positive	-
ss	Slave Select	input	1	negative	-
reset	Reset	input	1	positive	positive edge
outr	Output Ready	output	1	positive	-
br	Buffer Ready	output	1	positive	-
miso	Master In Slave Out	output	1	positive	-
phase	Phase	output	14	positive	-

Πίνακας 3-5 Είσοδοι / Έξοδοι OPC

Το OPC αποτελεί έναν συνδυασμό των SPI και PA και για τον λόγο αυτό οι είσοδοι και έξοδοί του είναι ένα υποσύνολο των θυρών των μονάδων που το συνιστούν.

Παρατηρούμε πως το OPC λειτουργεί με δύο ρολόγια, το clk και το sclk. Το clk είναι το κυρίως ρολόι που χρησιμοποιείται στο σύστημα και το sclk είναι το σειριακό ρολόι του SPI. Και τα δύο είναι απαραίτητα για τη λειτουργία του OPC.

Συνοπτικά, οι είσοδοι mosi, ss και οι έξοδοι miso, br είναι οι αντίστοιχες του SPI, ενώ η είσοδος reset και η έξοδος phase είναι ίδιες με αυτές που περιγράφονται στη δομή του PA.

Η έξοδος outr είναι η μοναδική που δεν αντιστοιχεί σε κάποια από τις συνιστώσες μονάδες και χρησιμοποιείται για την υπόδειξη της ετοιμότητας της εξόδου. Δηλαδή, επισημαίνει ότι

το PA βρίσκεται σε κανονική λειτουργία και επομένως η έξοδος του DDS είναι έτοιμη να δώσει το ζητούμενο σήμα. Η χρησιμότητα του outr θα γίνει καλύτερα κατανοητή κατά την περιγραφή του NCO.

Στο OPC δεν έχουν προστεθεί οι θύρες που αφορούν την παράλληλη μεταφορά δεδομένων, διότι οι αντίστοιχες των SPI και PA χρησιμοποιούνται μόνο στο εσωτερικό του OPC και συνδέονται κατάλληλα για την εκτέλεση των λειτουργιών του.

3.4.2 Προγραμματισμός

Με τον όρο operation profile ορίζουμε μία δεδομένη ρύθμιση των frequency και phase register. Ανάλογα με τις τιμές των δύο καταχωρητών, αλλάζει η έξοδος του DDS και επομένως λέμε ότι ένα ζεύγος τιμών FCW και phase offset ορίζουν ένα προφίλ λειτουργίας (operation profile) του DDS. Το operation profile configurator αναλαμβάνει τη ρύθμιση του περιεχομένου των δύο καταχωρητών του PA μέσω της σειριακής διασύνδεσης. Διαχειρίζεται δηλαδή τα δεδομένα που εισάγονται προς εγγραφή ή εξάγονται για ανάγνωση.

Τα δεδομένα διαβάζονται σειριακά από το SPI και γίνονται διαθέσιμα σε λέξεις των 8 bit. Τα μεγέθη των καταχωρητών του phase accumulator όμως είναι 48 bit, δηλαδή 6 φορές μεγαλύτερα. Γίνεται αντιληπτή λοιπόν, η ανάγκη για ενδιάμεσες διαδικασίες που πρέπει να εκτελεστούν ώστε να μεταφερθούν τα δεδομένα από το SPI στο PA ή αντίστροφα.

Η μονάδα OPC είναι υπεύθυνη για τη ρύθμιση της λειτουργίας του DDS και επομένως για τον προγραμματισμό της. Έτσι, εδώ γίνεται η επεξεργασία των εντολών προγραμματισμού και στη συνέχεια η διαχείριση των δεδομένων και ο έλεγχος του PA. Για τον προγραμματισμό χρησιμοποιούνται λέξεις των 8 bit για τις εντολές και στη συνέχεια (εάν υπάρχουν) τα απαραίτητα δεδομένα έχουν μήκος 6 λέξεων (48 bit). Ανάλογα με την εντολή, αλλάζει και η διαδικασία που ακολουθείται.

Οι λειτουργίες που υποστηρίζονται από τον OPC είναι οι εξής:

i. Εγγραφή δεδομένων στα registers του PA (frequency, phase)
ii. Ανάγνωση δεδομένων από τα registers του PA (frequency, phase)
iii. Εκκίνηση κανονικής λειτουργίας DDS
iv. Παύση κυματομορφής εξόδου

Κάθε φορά μπορεί να εκτελείται μία και μόνο λειτουργία από τις παραπάνω, ενώ κάθε μία από αυτές ορίζεται με διαφορετική εντολή και απαιτεί διαφορετική διαδικασία για τη μεταφορά των δεδομένων, εάν τη συνοδεύουν. Η εγγραφή και ανάγνωση των δεδομένων (λειτουργίες i, ii) χρειάζονται 8 byte για την ολοκλήρωσή τους, ενώ οι δύο τελευταίες (λειτουργίες iii, iv) χρειάζονται μόλις 1 byte. Σημειώνεται πως για τον προγραμματισμό του DDS χρησιμοποιείται το σειριακό ρολόι, καθώς με αυτό λειτουργεί το SPI, το οποίο είναι υπεύθυνο για τη μεταφορά των εντολών και δεδομένων, ενώ τα δεδομένα εγγράφονται ή εξάγονται από το PA με το κυρίως ρολόι, το οποίο χρησιμοποιείται από το PA.

Προτού αναλύσουμε τις διάφορες λειτουργίες του OPC, θα αναλύσουμε το instruction byte το οποίο είναι απαραίτητο και πάντα το πρώτο byte και των τεσσάρων λειτουργιών. Παρακάτω παρουσιάζεται η χρήση των bit του instruction byte:

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
$\overline{OP}$	$\bar{R}$ / W	$\overline{PH}$ / F	H	X	X	X	X

Πίνακας 3-6 Instruction Byte

Bit #	**Bit usage**	**Bit status**
7	Operation bit	0: Normal / Halt mode 1: Read / Write mode
6	Read / Write select bit (used only when $\overline{OP} = 1$)	0: Read 1: Write
5	Frequency / Phase register select bit (used only when $\overline{OP} = 1$)	0: Phase select 1: Frequency select
4	Halt bit (used only when $\overline{OP} = 0$)	0: Normal mode 1: Halt
3 - 0	Not used	-

Πίνακας 3-7 Καταστάσεις Instruction Byte

3.4.2.1 Εγγραφή operation profile

Η διαδικασία εγγραφής χρησιμοποιείται για να εισάγουμε ένα νέο operation profile στο DDS με σκοπό να αλλάξουμε τη συχνότητα του παραγόμενου σήματος ή να το μετατοπίσουμε στο πεδίο της φάσης. Για να θέσουμε το DDS σε αυτή τη λειτουργία, αρχικά στέλνουμε το instruction byte. Εάν επιθυμούμε την αλλαγή της λέξης ελέγχου συχνότητας FCW, τότε το instruction byte γίνεται $111XXXXX$, αλλιώς για την αλλαγή του phase offset $110XXXXX$, όπου με το X ορίζεται η «αδιαφορία» για την κατάσταση των αντίστοιχων bit.

Αμέσως μετά το instruction byte, εισάγουμε τα 6 byte που αντιστοιχούν στη νέα τιμή που θέλουμε να γράψουμε στο register που επιλέχθηκε. Η εισαγωγή όλων των byte (συμπεριλαμβανομένου και του instruction) γίνεται πάντα από το LSB προς το MSB και αντίστοιχα η σειρά των byte δεδομένων αρχίζει πάντα με το λιγότερο σημαντικό byte της τιμής 48 bit προς εγγραφή.

Για την ολοκλήρωση της εγγραφής, πρέπει μετά τα δεδομένα να ακολουθεί πάντα ένα κενό μήκους 1 byte, το οποίο θα αποκαλούμε synchronization byte. Το synchronization byte είναι απαραίτητο, ώστε να εγγραφεί και το τελευταίο byte στο buffer του OPC και στη συνέχεια να μεταφερθεί η τιμή του στο επιλεγμένο register του PA.

3.4.2.2 Ανάγνωση operation profile

Για την ανάγνωση του τρέχοντος operation profile του DDS, ακολουθείται παρόμοια διαδικασία με αυτή της εγγραφής, αλλά με λίγο διαφορετική σειρά των byte. Όπως και πριν, το πρώτο byte είναι και πάλι το instruction το οποίο διαφέρει στο 2ο bit σε σχέση με αυτό της

εγγραφής. Έτσι οι πιθανές επιλογές είναι οι $101XXXXX$ και $100XXXXX$ για την ανάγνωση του frequency ή phase register, αντίστοιχα.

Αυτή τη φορά το synchronization byte έπεται του instruction, καθώς προτού ξεκινήσει η εξαγωγή των δεδομένων από το PA, πρέπει πρώτα να μεταφερθεί το περιεχόμενο τού προς ανάγνωση register στο buffer του OPC. Έτσι, το byte που διαβάζεται αμέσως μετά το instruction δεν περιέχει κάποια πληροφορία, αλλά τυχαία τιμή από παλαιότερη χρήση του buffer (garbage).

Μόλις ολοκληρωθεί και η μετάδοση του synchronization byte, ξεκινά η εξαγωγή της πληροφορίας που έχουμε ζητήσει. Και σε αυτήν την περίπτωση, η σειρά των bit είναι από το LSB προς το MSB και τα διαδοχικά byte από το λιγότερο προς το περισσότερο σημαντικό.

3.4.2.3 Έναρξη / Παύση DDS

Για τις δύο τελευταίες λειτουργίες που υποστηρίζει το OPC, χρειάζεται η μετάδοση ενός και μόνο byte, καθώς δεν υπάρχουν δεδομένα που πρέπει να μεταφερθούν. Έτσι, ανάλογα με το αν ζητείται η μετάβαση στην κανονική λειτουργία του DDS ή η παύση αυτής, τα instruction byte διαμορφώνονται είτε ως $0XX0XXXX$, είτε ως $0XX1XXXX$, αντίστοιχα.

Αφού εισαχθεί ένα από τα δύο παραπάνω instruction bytes, η μετάβαση στην αντίστοιχη λειτουργία του DDS ξεκινά με τον αμέσως επόμενο θετικό παλμό του κυρίως ρολογιού clk. Το operation profile πρέπει να έχει ήδη διαμορφωθεί με βάση τη διαδικασία της Εγγραφή operation profile.

Σημειώνεται πως όσο το DDS δε βρίσκεται σε καταστάσεις εγγραφής ή ανάγνωσης και επομένως λειτουργεί κανονικά ή βρίσκεται σε αδράνεια, είναι καλό να απενεργοποιείται η είσοδος ss του OPC, δηλαδή ss = 1, ώστε να αποφεύγεται η εισαγωγή απρόσκοπτης εντολής. Αντίστροφα, για την αλλαγή της κατάστασης πρέπει να τεθεί το ss σε χαμηλό δυναμικό και στη συνέχεια να δοθεί η επιθυμητή εντολή.

3.4.3 Λειτουργία

Όπως αναφέρθηκε, το OPC αποτελείται από το SPI, το PA και την επιπλέον απαραίτητη λογική για τη διαχείριση των δεδομένων και τον προγραμματισμό του DDS. Για την υλοποίηση αυτών, είναι απαραίτητη η κατάλληλη διασύνδεση των SPI και PA, καθώς και των επιπρόσθετων σημάτων (register και wire) που χρειάζονται.

Αρχικά λοιπόν, ορίζουμε τις παραμέτρους και τις εισόδους / εξόδους της μονάδας. Οι επιπλέον καταχωρητές που θα χρειαστούν είναι ο fph για την επιλογή register του PA και το mode για την επιλογή της λειτουργίας του PA. Ορίζουμε ακόμη ένα buffer μεγέθους M bit για τη διαχείριση και προσωρινή αποθήκευση των δεδομένων, καθώς και έναν μετρητή (counter) για την αρίθμηση των byte που εισάγονται ή εξάγονται.

Το word_read χρησιμοποιείται για την ανάγνωση μιας λέξης από το SPI και επομένως συνδέεται με το wordout του SPI. Το pa_word μεταφέρει τις λέξεις για εγγραφή ή ανάγνωση προς και από το PA, αντίστοιχα, και επομένως συνδέεται με το word. Όταν το word λειτουργεί ως έξοδος, δηλαδή διαβάζουμε μία τιμή από το εσωτερικό του PA, το pa_word λαμβάνει την

τιμή αυτή και γι’ αυτό το αποκόβουμε από το OPC (κατάσταση Z). Αντίθετα αν γίνεται εγγραφή σε κάποιον καταχωρητή του PA, το word μετατρέπεται σε είσοδος και το pa_word συνδέεται με το buffer του OPC, μεταφέροντας το περιεχόμενό του προς το PA. Το pa_word τέλος, αποκόπτεται και στην περίπτωση που ξεκινά η εισαγωγή του instruction byte, ώστε να αποφευχθούν απρόσκοπτες ενέργειες ή αλλοίωση του operation profile.

Στο σημείο αυτό ορίζεται και η τιμή της εξόδου outr, η οποία βρίσκεται σε υψηλό δυναμικό όσο το mode είναι 00 και επομένως το OPC βρίσκεται στην κανονική λειτουργία. Με τον τρόπο αυτό, υποδεικνύεται ότι το DDS έχει ξεκινήσει την παραγωγή του σήματος και επομένως η έξοδός του είναι έτοιμη.

Στη συνέχεια δημιουργούνται οι μονάδες SPI και PA και ορίζονται οι διασυνδέσεις των εισόδων / εξόδων τους. Στο σημείο αυτό σημασία έχουν οι συνδέσεις των θυρών που αφορούν την παράλληλη μεταφορά δεδομένων και γενικά όσων σημάτων δεν είναι διαθέσιμα εξωτερικά του OPC και συνεπώς και του DDS. Έτσι, το wordin συνδέεται με το byte μικρότερης σημασίας του buffer, ενώ το load ενεργοποιείται όσο το mode έχει την τιμή 10 για να ξεκινήσει η μεταφορά της τιμής του wordin προς το εξωτερικό της μονάδας.

Αξιοσημείωτη είναι επίσης η είσοδος mode του PA, στην οποία συνδέεται το register mode του OPC, αφού πρώτα περάσει από μία πύλη buffer: ~(~mode). Εάν παραληφθεί το buffer στην προκειμένη σύνδεση, τότε όταν τεθεί σε κανονική λειτουργία το DDS, η φάση θα αυξηθεί κατά FCW και επομένως θα ξεκινήσει από το σημείο phase offset + FCW, αντί του phase offset. Με το buffer καθυστερούμε την αλλαγή του mode στο PA, η οποία συμβαίνει στον επόμενο κύκλο clk από την αλλαγή του mode στο OPC. Επειδή όμως το outr ενεργοποιείται κατευθείαν και μαζί του και η έξοδος του DDS, το πρώτο πλάτος που εξάγεται είναι αυτό που αντιστοιχεί στο phase offset, καθώς το phase register δεν προλαβαίνει να αυξηθεί.

Ακολουθεί η κυρίως λειτουργία του OPC, η οποία χρησιμοποιεί τον θετικό παλμό του σειριακού ρολογιού sclk. Υποστηρίζεται και η λειτουργία ασύγχρονου reset, κατά την οποία όλοι οι καταχωρητές του OPC μεταβαίνουν στις αρχικές τους τιμές. Έχει σημασία να σημειώσουμε ότι η default λειτουργία ορίζεται η αδράνεια (mode = 01), ώστε μετά από reset το DDS να βρίσκεται σε αναμονή για να το προγραμματίσουμε. Επίσης, το counter αρχικοποιείται με την τιμή 111, διότι η αύξησή του γίνεται στην αρχή των διαδικασιών του OPC και επομένως κατά την πρώτη λέξη θα μηδενιστεί για να υποδείξει ότι πρέπει να διαβαστεί το instruction byte.

Είναι απαραίτητο το ss να βρίσκεται σε χαμηλό δυναμικό για όσο προγραμματίζουμε το DDS και για όσο μεταφέρονται τα instruction, synchronization και data bytes. Επίσης, σχεδόν όλες οι ενέργειες εκτελούνται κάθε φορά που το buffer του SPI είναι έτοιμο (br = 1), διότι μόνο τότε έχει νόημα να διαβάσουμε ή να στείλουμε λέξη από και προς το SPI αντίστοιχα. Η μόνη ενέργεια που λαμβάνει χώρα ανεξάρτητα του br, είναι η μεταφορά του περιεχομένου ενός register του PA προς το buffer του OPC. Αυτό συμβαίνει αμέσως μετά το instruction byte (counter = 1) και όταν είμαστε σε λειτουργία ανάγνωσης (mode = 10). Εδώ γίνεται κατανοητή η ανάγκη

για την τοποθέτηση του synchronization byte πριν την εξαγωγή των δεδομένων, αφού η τιμή θα γίνει διαθέσιμη για μετάδοση έναν κύκλο μετά, δηλαδή όταν counter = 2.

Η υπόλοιπη λειτουργία του OPC χωρίζεται σε τρία τμήματα. Αρχικά και όταν ξεκινά η εισαγωγή του instruction byte (counter = 0), το DDS μεταβαίνει σε κατάσταση αδράνειας αναμένοντας την ολοκλήρωση του instruction. Μόλις διαβαστεί η εντολή (counter = 1), εκχωρούνται οι αντίστοιχες τιμές στους καταχωρητές του OPC ανάλογα με την κατάσταση των bit του instruction byte. Το τρίτο και τελευταίο τμήμα θα εκτελεστεί μόνο εάν το OPC βρίσκεται σε λειτουργία εγγραφής / ανάγνωσης και για 6 διαδοχικά byte, που είναι το μέγεθος των δεδομένων. Σε κάθε μία από τις δύο περιπτώσεις, πρέπει να γίνει ολίσθηση του buffer κατά ένα byte δεξιά για την εισαγωγή και εγγραφή ενός νέου byte ή την ανανέωση της τιμής του SPI, αντίστοιχα. Για την πρώτη περίπτωση, μετά την ολίσθηση εγγράφουμε το byte που διαβάζεται από το SPI στα 8 σημαντικότερα bit του buffer.

Όταν το ss μεταβεί σε υψηλό δυναμικό και επομένως το DDS δεν προγραμματίζεται, το counter αρχικοποιείται στη μέγιστη τιμή του, ώστε μόλις το ss μηδενιστεί να ξεκινήσει η αρίθμηση από το μηδέν. Έτσι, το OPC διατηρείται συνεχώς συγχρονισμένο με το SPI και επομένως και με τον master της επικοινωνίας, ο οποίος ελέγχει τον προγραμματισμό.

Στο σημείο αυτό πρέπει να αναφέρουμε, ότι παρά την παραμετροποίηση του OPC, για να είναι πλήρως ανεξάρτητο του M, είναι απαραίτητο να γίνουν μικρές αλλαγές στη λογική του κυκλώματος. Το M επηρεάζει το μήκος των δεδομένων και επομένως το πλήθος των byte τους. Έτσι, το counter είναι απαραίτητο να έχει μήκος τουλάχιστον $\left\lceil \log_2 \left(\frac{M}{B} + 2 \right) \right\rceil$ bit [4]. Για παράδειγμα, για $M = 48$ και $B = 8$ το μέγεθος του μετρητή είναι $\left\lceil \log_2 \left(\frac{48}{8} + 2 \right) \right\rceil = \lceil 3 \rceil = 3$ bit, ενώ για $M = 70$ και $B = 8$ είναι $\left\lceil \log_2 \left(\frac{70}{8} + 2 \right) \right\rceil = \lceil \log_2 8{,}75 \rceil = \lceil 3.1293 \rceil = 4$. Στο τελευταίο παράδειγμα ο counter μπορεί να μετρήσει έως την τιμή $2^4 = 16$, ενώ τα συνολικά byte που χρειάζονται είναι μόλις $\left\lceil \frac{M}{B} + 2 \right\rceil = 9$. Επομένως, είτε θα πρέπει να υπάρξουν επιπλέον 7 synchronization bytes στην κάθε εγγραφή / ανάγνωση, ή ο counter να μηδενίζεται μετά την τιμή 9.

3.4.4 Προσομοίωση

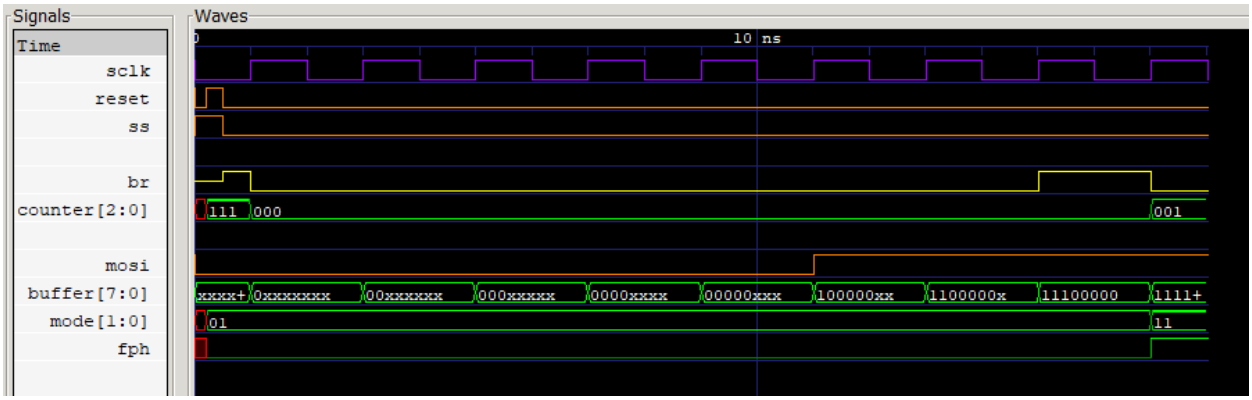
Αρχικά δημιουργούμε το testbench και αρχικοποιούμε όλα τα απαραίτητα σήματα με ίδιο τρόπο με τα προηγούμενα modules. Η συχνότητα του sclk επιλέχθηκε ως η μισή του clk για λόγους συντομίας και ευκολίας στην αναπαράσταση των χρονικών αναλύσεων. Στην πραγματικότητα και για τις μέγιστες υποστηριζόμενες συχνότητες των δύο ρολογιών, μπορεί να ισχύει $\frac{f_{clk}}{f_{sclk}} = \frac{2{,}8\ GHz}{25\ MHz} = 112$, ενώ <u>δεν</u> είναι απαραίτητο η f_{sclk} να είναι ακέραιο υποπολλαπλάσιο της f_{clk}.

Για την επαλήθευση του OPC θα περιοριστούμε στις λειτουργίες εγγραφής και ανάγνωσης τιμών. Οι υπόλοιπες λειτουργίες θα εξεταστούν στην ενότητα 3.6 του NCO. Ξεκινώντας την

[4] Το $+2$ αφορά τα απαραίτητα instruction και synchronization bytes.

ανάλυση θα επαληθεύσουμε την ορθή ανάγνωση του instruction byte και στη συνέχεια μία διαδικασία εγγραφής και μία ανάγνωσης του FCW για ένα operation profile.

3.4.4.1 Επαλήθευση ανάγνωσης instruction byte



Εικόνα 3-8 Προσομοίωση OPC: ανάγνωση instruction byte

Προτού εισάγουμε το instruction byte, κάνουμε reset το OPC για να πάρουν όλοι οι καταχωρητές τις αρχικές τιμές τους. Με την ολοκλήρωση του reset, θέτουμε το ss σε χαμηλό δυναμικό δηλώνοντας έτσι ότι επιθυμούμε να προγραμματίσουμε το DDS. Εάν ss = 1, τότε δε θα διαβαστεί ποτέ κανένα byte ούτε από το SPI και προφανώς ούτε από το OPC. Παρατηρούμε ακόμη ότι με τον πρώτο θετικό παλμό του sclk, το counter μηδενίζεται, όπως περιγράψαμε νωρίτερα.

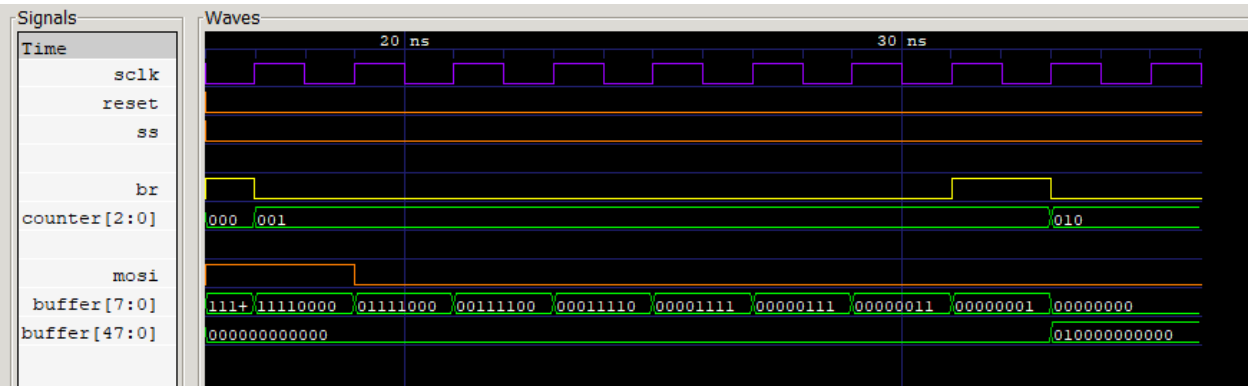
Με τον μηδενισμό του counter, ξεκινά η εισαγωγή των bits του instruction byte. Εφόσον η επικοινωνία είναι σειριακή, τα bit στέλνονται ένα - ένα μέσω της εισόδου mosi. Μόλις το buffer του SPI δηλώνεται ως έτοιμο, βλέπουμε πως έχει σχηματιστεί η πλήρης λέξη του instruction, η οποία είναι η 11100000 και υποδεικνύει πως επιθυμούμε να αλλάξουμε την τιμή του frequency register.

Αμέσως λοιπόν, όταν το counter γίνεται 1, μεταφέρονται οι αντίστοιχες τιμές στα mode και fph. Στο σημείο αυτό χρειάζεται προσοχή το γεγονός ότι πρώτα διαβάζουμε την τιμή του buffer του SPI και μετά αυτή μεταβάλλεται. Διαβάζουμε δηλαδή την τιμή 11100000 και αμέσως μετά επειδή το mosi είναι 1, το buffer γίνεται 11110000.

3.4.4.2 Επαλήθευση εγγραφής δεδομένων

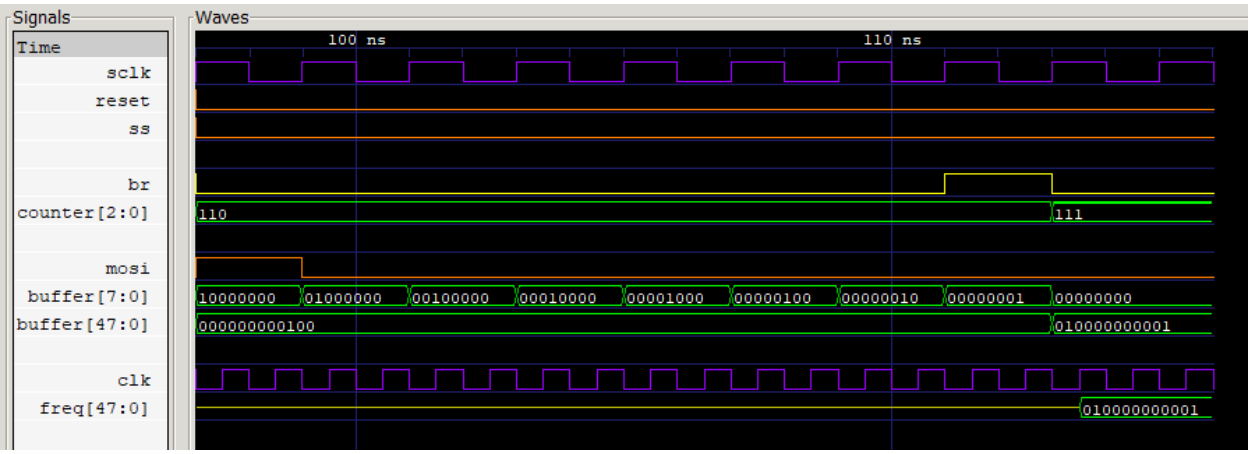
Ακολουθώντας το instruction byte, ξεκινά η εισαγωγή των δεδομένων. Στην Εικόνα 3-9 φαίνεται η εισαγωγή του πρώτου εκ των έξι byte. Παρατηρούμε πως μόλις ολοκληρωθεί η ανάγνωση του byte, τότε αυτό μεταφέρεται από το buffer του SPI στο πιο σημαντικό byte του buffer του OPC.

Η διαδικασία αυτή συνεχίζεται διαδοχικά για ακόμη 5 byte μέχρι να σχηματιστεί ολόκληρη η τιμή προς εγγραφή στο frequency register. Το τελευταίο byte, καθώς και η τελική μεταφορά των δεδομένων στο εσωτερικό του PA, παρουσιάζονται στην Εικόνα 3-10.



Εικόνα 3-9 Προσομοίωση OPC: εγγραφή δεδομένων - 1ο data byte

Στην αρχή της Εικόνα 3-10, το περιεχόμενο του buffer του OPC έχει ήδη υποστεί 4 από τις 5 συνολικές ολισθήσεις. Μόλις το counter πάρει τη μέγιστη τιμή αρίθμησης, καταγράφεται και το τελευταίο byte στο buffer. Η τιμή του όμως θα εγγραφεί στο frequency register του PA με τον αμέσως επόμενο θετικό παλμό του clk, με το οποίο λειτουργεί το PA.



Εικόνα 3-10 Προσομοίωση OPC: εγγραφή δεδομένων - 6ο data byte & synchronization

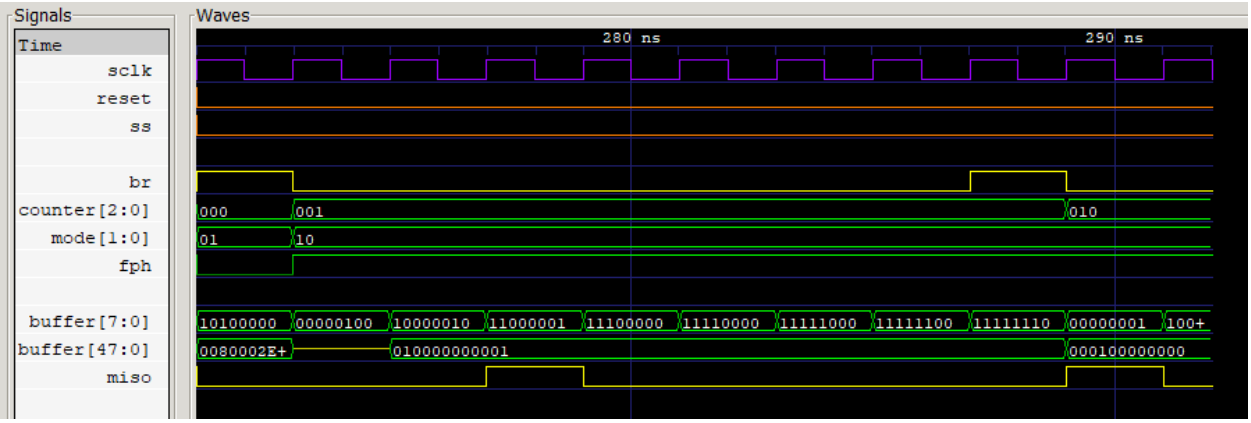
Εδώ φαίνεται η αναγκαιότητα του synchronization byte, το οποίο (για τη λειτουργία εγγραφής) διαρκεί όσο το counter έχει τη μέγιστη τιμή. Εάν δε δεσμεύσουμε ένα byte για τον συγχρονισμό των δεδομένων, το buffer θα χάσει το τελευταίο byte και θα γραφεί λανθασμένη τιμή στο frequency register.

3.4.4.3 Επαλήθευση ανάγνωσης δεδομένων

Ολοκληρώνοντας την επαλήθευση του OPC, θα ελέγξουμε τη διαδικασία ανάγνωσης μίας τιμής του τρέχοντος operation profile. Στην Εικόνα 3-11 παρουσιάζεται αρχικά η ολοκλήρωση του instruction byte, το οποίο είναι ρυθμισμένο για ανάγνωση (mode = 10) του frequency register (fph = 1).

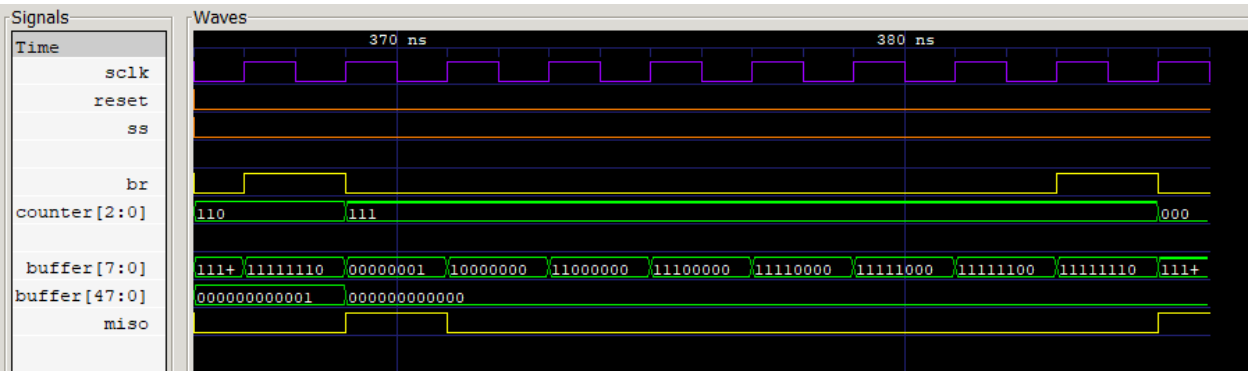
Όσο το counter είναι 1, βρισκόμαστε στο synchronization byte, το οποίο προηγείται των δεδομένων, όπως ήταν αναμενόμενο. Κατά το synchronization byte γίνεται η μεταφορά της τιμής του frequency register στο buffer του OPC. Εάν παραλειπόταν το συγκεκριμένο byte, τότε το

buffer δε θα άλλαζε τιμή εγκαίρως και θα περιείχε παλαιά τιμή από προηγούμενη λειτουργία, γεγονός που θα ήταν ανεπιθύμητο.



Εικόνα 3-11 Προσομοίωση OPC: ανάγνωση δεδομένων – Synchronization byte & έναρξη 1ου data byte

Για να αποφευχθεί λοιπόν η εμφάνιση λανθασμένων τιμών στην έξοδο, χρειάζεται μία περίοδος του sclk, ώστε να μεταφερθεί η αναμενόμενη τιμή στο buffer του OPC. Επειδή όμως πρέπει να είμαστε σε πλήρη συγχρονισμό με το SPI, αναγκαστικά περιμένουμε για τη μετάδοση ενός ολόκληρου byte, του οποίου το περιεχόμενο προφανώς δεν είναι χρήσιμο.



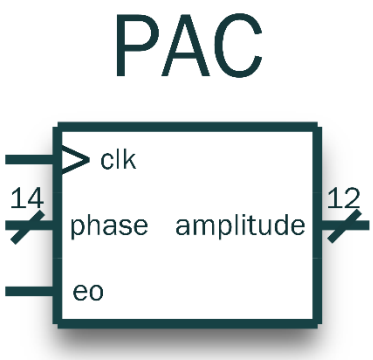
Εικόνα 3-12 Προσομοίωση OPC: ανάγνωση δεδομένων - 6ο data byte

Ολοκληρώνοντας την ανάλυση για το OPC, παρουσιάζεται η τελευταία ολίσθηση του buffer του OPC για την εξαγωγή του 6ου και τελευταίου byte της τιμής που ζητήθηκε. Σημειώνεται πως η εμφάνιση του τελευταίου byte γίνεται κατά τη διάρκεια που το counter έχει τη μέγιστη τιμή του, εν αντιθέσει με τη λειτουργία εγγραφής, όπου στην αντίστοιχη θέση υπήρχε το synchronization byte.

Τέλος, δεδομένης της σειριακής φύσης της επικοινωνίας του SPI, η έξοδος των τιμών γίνεται πάντα μέσω της εξόδου miso, της οποίας η σωστή λειτουργία επαληθεύεται στην Εικόνα 3-11 και την Εικόνα 3-12. Με την εισαγωγή ενός νέου byte στο SPI, το miso λαμβάνει το LSB του buffer και έπειτα από διαδοχικές ολισθήσεις προς τα δεξιά εμφανίζεται και το τελευταίο bit, ενώ ταυτόχρονα υποδεικνύεται η ετοιμότητα του buffer, όπως περιγράφεται στην Λειτουργία.

3.5 Μετατροπέας Φάσης σε Πλάτος (Phase to Amplitude Converter, PAC)

3.5.1 Δομή



Εικόνα 3-13 Μονάδα PAC

I/O Pin	I/O Name	I/O Type	I/O Length	I/O Logic	Triggered On
clk	Clock	input	1	positive	positive edge
phase	Phase	input	14	positive	-
eo	Enable Output	input	1	positive	-
amplitude	Amplitude	output	12	positive	-

Πίνακας 3-8 Είσοδοι / Έξοδοι PAC

Το PAC είναι ένα ακόμη σύγχρονο module που τρέχει με το κυρίως ρολόι του συστήματος συνδεδεμένο στην είσοδο clk.

Η είσοδος phase είναι η συνέχεια της εξόδου phase του PA και επομένως του OPC. Μπορεί να θεωρηθεί και σαν διεύθυνση για τη μνήμη του PAC όπου είναι αποθηκευμένες οι τιμές του ημιτονοειδούς σήματος που κατασκευάζεται.

Η μοναδική έξοδος του PAC είναι η amplitude, η οποία έχει μήκος 12 bit και αποτελεί την τιμή του πλάτους που αντιστοιχεί στη φάση εισόδου phase.

Υπάρχει ακόμη η δυνατότητα απενεργοποίησης της εξόδου amplitude του PAC, μεταβαίνοντας σε κατάσταση υψηλής αντίστασης Z. Το χαρακτηριστικό αυτό ελέγχεται από την είσοδο eo και η χρησιμότητά του θα γίνει καλύτερα κατανοητή με την περιγραφή της μονάδας NCO στην Αριθμητικώς Ελεγχόμενος Ταλαντωτής (Numerically Controlled Oscillator, NCO).

Σημειώνεται τέλος, πως το PAC δεν παρέχει δυνατότητα reset, καθώς δε χρησιμεύει σε κάτι, διότι όλα τα εσωτερικά registers ανανεώνουν την τιμή τους σε κάθε περίοδο, ανάλογα με την είσοδο του PAC. Έτσι, στην αρχή παρ' όλο που οι τιμές τους είναι άγνωστες (X), μόλις γίνει reset το υπόλοιπο σύστημα, τότε έμμεσα θα αρχικοποιηθεί και το PAC.

3.5.2 Μνήμη

Ένα από τα δομικά στοιχεία και μάλιστα το σημαντικότερο κομμάτι του PAC, είναι η μνήμη του. Δεδομένου πως οι τιμές του αποθηκευμένου ημιτόνου δεν αλλάζουν, εύκολα οδηγούμαστε στο συμπέρασμα ότι η μνήμη δε χρειάζεται να υποστηρίζει δυνατότητες εγγραφής. Έτσι, η πρώτη σκέψη είναι μία ROM, η οποία κρατάει μόνιμα τα δεδομένα της και δε χρειάζεται επανεγγραφή.

Γνωρίζοντας όμως πως το DDS μπορεί να λειτουργήσει σε τεράστιες συχνότητες, η ROM θα καθυστερούσε πολύ τη λειτουργία του, καθώς οι ταχύτητες που δουλεύει είναι αρκετές τάξεις μεγέθους μικρότερες του κυρίως ρολογιού του DDS. Έτσι, για λόγους ταχύτητας, επιλέγεται η δημιουργία μία λογικής συνάρτησης η οποία αποτελείται από λογικές πύλες που για κάθε είσοδο (φάση), δίνουν την αντίστοιχη έξοδο (πλάτος).

Εφόσον η ανάλυση φάσης έχει επιλεχθεί να είναι $N = 14\ bit$, το Look-Up Table (LUT) του PAC θα πρέπει να περιέχει $2^{14} = 16.384$ θέσεις για τα πλάτη του ημιτονοειδούς σήματος. Επειδή όμως το ημίτονο είναι μία συμμετρική συνάρτηση με άρτια και περιττή συμμετρία, δίνεται η δυνατότητα να αποθηκεύσουμε μόνο το $\frac{1}{4}$ των τιμών μίας περιόδου. Έτσι, καταγράφουμε τις τιμές μόνο του πρώτου τεταρτημόριου και με βάση αυτές παράγουμε τις ζητούμενες τιμές για τα υπόλοιπα τεταρτημόρια. Το LUT λοιπόν, χρειάζεται μόνο $\frac{16.384}{4} = 4.096$ θέσεις μνήμης.

Για τη δημιουργία των ζητούμενων τιμών ενός ημιτονοειδούς σήματος, συγγράφηκε ένα script σε γλώσσα MATLAB. Το script αυτό, που είναι διαθέσιμο στο παράρτημα MATLAB Script – Μνήμη PAC, χωρίζεται σε τρία βασικά μέρη:

i. Δημιουργία διακριτών τιμών 1ου τεταρτημόριου ημιτόνου
ii. Μετατροπή τους σε δυαδικούς αριθμούς ακρίβειας $P = 12\ bit$
iii. Επαλήθευση με ανακατασκευή του αρχικού ημιτόνου

3.5.2.1 Δημιουργία διακριτών τιμών 1ου τεταρτημόριου

Όπως είναι γνωστό, το πρώτο τεταρτημόριο του ημιτόνου ορίζεται στο διάστημα φάσης $\left[0, \frac{\pi}{2}\right)\ rad$. Επομένως, πρέπει να χωρίσουμε το διάστημα αυτό σε $2^{N-2} = 2^{12} = 4.096$ διαφορετικά σημεία. Με την παραπάνω σκέψη έχουμε την $n-$οστή τιμή του πλάτους:

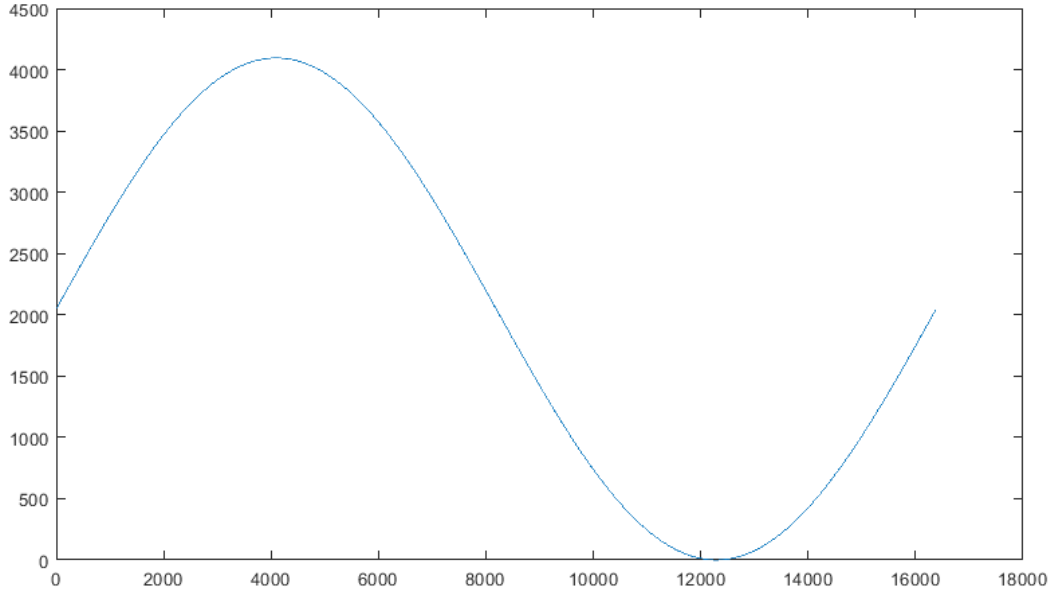
$$A_n = \sin\left(\frac{\pi}{2} \cdot \frac{a_n}{2^{N-2}}\right) = \sin\left(\frac{\pi}{2} \cdot \frac{a_0 + n \cdot \omega}{2^{N-2}}\right) = \sin\left(\frac{\pi}{2} \cdot \frac{n}{2^{N-2}}\right) = \sin\left(\frac{n\pi}{2^{N-1}}\right) \qquad (6)\,,$$

όπου a_n αριθμητική πρόοδος με αρχική τιμή $a_0 = 0$ και βήμα $\omega = 1$ και $n \in [0, 2^{N-2})$.

Όπως προκύπτει από την εξίσωση (6), η ελάχιστη τιμή του πλάτους είναι ίση με $A_{min} = A_0 = \sin(0) = 0$ και η μέγιστη τιμή είναι η $A_{max} = A_{2^{N-2}-1} = \sin\left(\frac{\pi}{2} \cdot \frac{2^{N-2}-1}{2^{N-2}}\right) = \sin\left(\frac{\pi}{2} - 2^{-(N-2)}\right) \cong \sin\left(\frac{\pi}{2}\right) = 1.$

Σε συντακτικό MATLAB, το παραπάνω μπορεί να εκφραστεί χρησιμοποιώντας τον τελεστή επανάληψης $a_0 : \omega : n_{max}$ για τιμές όπως ορίστηκαν στην προηγούμενη εξίσωση. Για μεγαλύτερη ακρίβεια, αντί να γίνεται διαίρεση με την προσεγγιστική σταθερά $pi = \pi$ του MATLAB, χρησιμοποιείται η συνάρτηση $sinpi$, η οποία είναι ίδια με την sin εάν πολλαπλασιαστεί η φάση της τελευταίας με π, το οποίο και γίνεται εσωτερικά της συνάρτησης. Οι τιμές που προκύπτουν είναι πιο ακριβείς και αποθηκεύονται στο διάνυσμα pac.

Στη συνέχεια γίνεται η μεταφορά των τιμών του 1ου τεταρτημόριου από το σύνολο τιμών του ημιτόνου $[-1, 1]$, στο επιθυμητό $[0, 2^P - 1] = [0,\ 4095]$, ώστε να συμφωνεί με τις διευθύνσεις μνήμης του PAC. Τέλος, οι τιμές στρογγυλοποιούνται στον κοντινότερο ακέραιό τους με τη συνάρτηση $round$. Το βήμα αυτό συμβαίνει αρχικά για λόγους ευκολίας της διαχείρισης των τιμών στη δυαδική τους αναπαράσταση χωρίς να χάνουμε σημαντικά σε ακρίβεια, και ακόμη για λόγους συμβατότητας με τον D/A Converter στην έξοδο του DDS, ο οποίος συνήθως απαιτεί ακέραιους δυαδικούς αριθμούς.



Εικόνα 3-14 Περίοδος ημιτόνου 2^{14} διακριτών τιμών εύρους [0, 4095]
Η γραφική παράσταση έχει ανακατασκευαστεί με τιμές μόνο από το 1ο τεταρτημόριο του ημιτόνου

Για την επαλήθευση των υπολογισμών μας, δημιουργούμε ολόκληρο το ημίτονο χρησιμοποιώντας τις τιμές μόνο από το 1ο τεταρτημόριο. Αξιοποιώντας τις συμμετρίες, το 2ο τεταρτημόριο δημιουργείται διαβάζοντας τις τιμές του 1ου με αντίστροφη σειρά, το 3ο προκύπτει ως η διαφορά των τιμών του 1ου από τη μέγιστη τιμή $2^P - 1$ και τέλος το 4ο είναι η διαφορά των τιμών του 2ου από το $2^P - 1$. Τα παραπάνω περιγράφονται σε συντακτικό MATLAB στον πίνακα που ακολουθεί και αντίστοιχη λογική θα χρησιμοποιηθεί για την υλοποίησή τους με γλώσσα Verilog σε επόμενη ενότητα.

Τεταρτημόριο	Συνάρτηση Περιγραφής (MATLAB)
1ο	pac
2ο	$pac(end:-1:1)$
3ο	$2^P - 1 - pac$
4ο	$2^P - 1 - pac(end:-1:1)$

Πίνακας 3-9 Σχέσεις τεταρτημόριων ημιτόνου σε γλώσσα MATLAB

3.5.2.2 Μετατροπή σε δυαδικούς αριθμούς ακρίβειας P και αποθήκευση τιμών

Στη συνέχεια οι αριθμοί μετατρέπονται σε δυαδικούς ακρίβειας $P = 12\ bit$ μέσω της συνάρτησης $de2bi$ του MATLAB. Το MSB επιλέγεται να είναι στην αρχή της δυαδικής ακολουθίας, καθώς σε όλες τις μονάδες στη Verilog ακολουθείται το συγκεκριμένο πρότυπο (little endian).

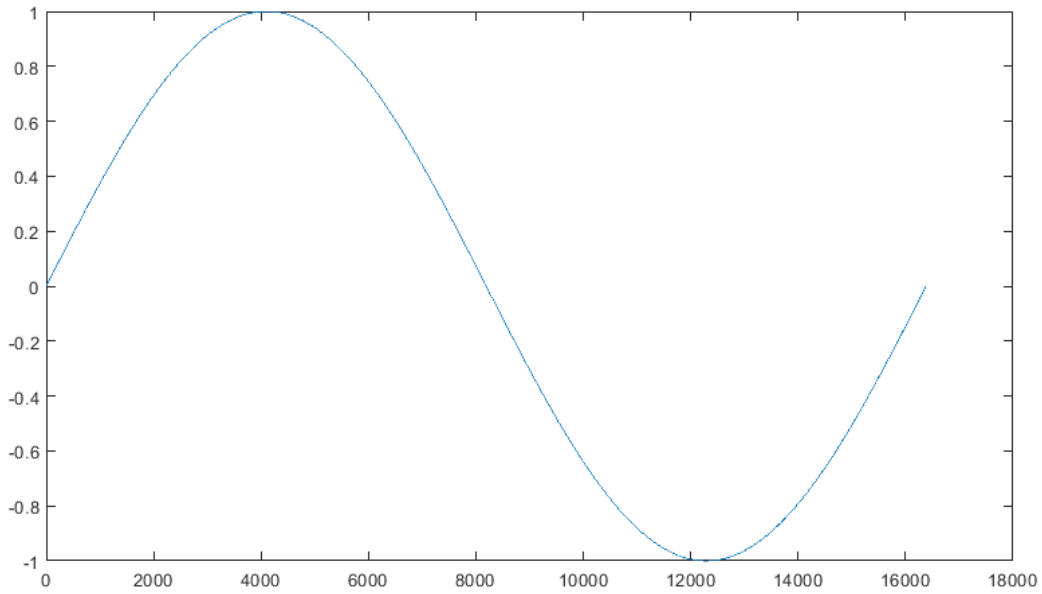
Ακολουθεί η αποθήκευση των τιμών σε διαδοχικές σειρές στο αρχείο κειμένου sine_4.txt. Η σειρά εμφάνισής τους είναι κατά αύξουσα φάση και επομένως η διεύθυνση της μνήμης αυξάνεται όσο και η φάση και αντίστροφα. Με τον τρόπο αυτό, απλοποιείται η λειτουργία του PAC, καθώς υπάρχει 1-1 αντιστοιχία μεταξύ διεύθυνσης LUT και φάσης και επομένως ο υπολογισμός του πλάτους γίνεται αρκετά πιο εύκολα.

Για τη διευκόλυνση της εισαγωγής των τιμών στον κώδικα Verilog, δημιουργείται ξεχωριστό αρχείο με κατάλληλη μορφοποίηση, η οποία είναι σύμφωνη με το συντακτικό της Verilog. Έτσι, αντί απλά να γράψουμε στο αρχείο εξόδου τις τιμές, χρησιμοποιούμε τη μορφή «$\{a\}: data =' b\{τιμή\};$», όπου $\{a\}$ είναι η διεύθυνση μνήμης, $data$ ο καταχωρητής δεδομένων του PAC (βλ. Λειτουργία) και $\{τιμή\}$ η δυαδική αναπαράσταση $12\ bit$ της τρέχουσας τιμής.

3.5.2.3 Ανακατασκευή αρχικού ημιτόνου και επαλήθευση

Ολοκληρώνοντας την εξαγωγή των τιμών για το LUT του PAC, ακολουθούμε την αντίστροφη πορεία, ώστε να επαληθευτεί η ορθότητα των τιμών. Έτσι, μετατρέπουμε τους δυαδικούς αριθμούς στο δεκαδικό σύστημα και έπειτα τους ανάγουμε στο σύνολο τιμών $[-1, 1]$, ώστε να προκύψει και πάλι το αρχικό ημίτονο.

Η Εικόνα 3-15 παρουσιάζει το ανακατασκευασμένο ημίτονο, που πλέον έχει πλάτος ± 1. Στη μορφή αυτή αναμένεται να είναι και η έξοδος ενός D/A Converter εάν προστεθεί στην έξοδο του DDS. Βέβαια, οι μέγιστες και ελάχιστες τιμές πλάτους ενδέχεται να είναι διαφορετικές ανάλογα με τις εκάστοτε απαιτήσεις (π.χ. $\pm 5\ V$). Παρατηρούμε πως η ακρίβεια του σήματος χρησιμοποιώντας 2^{14} τιμές $12\ bit$ είναι αρκετά υψηλή. Μπορούμε τέλος να συμπεράνουμε πως το DDS θα έχει πολύ καλή απόδοση και σε υψηλότερες συχνότητες, όπου θα αξιοποιούνται λιγότερες από τις διαθέσιμες τιμές της μνήμης του PAC.



Εικόνα 3-15 Ανακατασκευή αρχικού ημιτόνου 2^{14} διακριτών τιμών πλάτους ± 1

3.5.3 Λειτουργία

Για να λειτουργήσει σύμφωνα με το επιθυμητό, το phase to amplitude converter θα χρειαστεί μερικές επιπλέον μεταβλητές εκτός των εισόδων και εξόδων του. Οι τρεις από αυτές είναι registers και η τελευταία είναι τύπου wire.

Αρχικά λοιπόν, ορίζουμε τον καταχωρητή eos (enable output synchronously), ο οποίος χρησιμοποιείται για τη σύγχρονη ενεργοποίηση / απενεργοποίηση της εξόδου. Δεδομένου ότι το eo μπορεί να τεθεί οποιαδήποτε στιγμή σε υψηλό δυναμικό, η έξοδος του PAC δε θα εμφανιστεί συγχρονισμένα με το ρολόι. Έτσι, αν η έξοδος ελέγχεται από το eos, το οποίο λαμβάνει την εκάστοτε τιμή του eo, αλλά πάντα σε θετικό παλμό του ρολογιού, τότε το πρόβλημα συγχρονισμού της εξόδου επιλύεται.

Ο λόγος που η ενεργοποίηση ή απενεργοποίηση της εξόδου αποφεύγεται να είναι ασύγχρονη, οφείλεται στο γεγονός ότι το PAC πρέπει να είναι πλήρως συγχρονισμένο με το υπόλοιπο σύστημα και όλα τα πλάτη να εμφανίζονται ομοιόμορφα στην έξοδο. Για παράδειγμα μπορεί το eo να γίνει 1 πριν προλάβει η φάση να ανανεωθεί (π.χ. κατά την αρχικοποίησή της) οδηγώντας έτσι στην εμφάνιση λάθος τιμής μέχρι τον επόμενο θετικό παλμό που η φάση θα λάβει τη σωστή τιμή. Βέβαια, κάτι τέτοιο θα επηρέαζε ανεπαίσθητα την έξοδο, αλλά καλό είναι να μην εμφανίζονται ποτέ τιμές garbage.

Επιπλέον, δημιουργούμε τα register address και data. Το πρώτο αποτελεί τη διεύθυνση του LUT, η οποία για να υπολογιστεί χρησιμοποιούνται τα $N-2$ πιο σημαντικά bits της φάσης. Η απευθείας χρήση της φάσης μήκους $N = 14\ bit$ δεν είναι δυνατή, καθώς υπενθυμίζεται πως το LUT περιέχει $2^{N-2} = 2^{12} = 4096$ θέσεις μνήμης και γι' αυτό χρειάζεται πρώτα μία επεξεργασία. Το data από την άλλη, χρησιμοποιείται για τη μεταφορά των δεδομένων της μνήμης και

παίρνει την τιμή πλάτους που αντιστοιχεί στην τρέχουσα διεύθυνση. Όσο η έξοδος του PAC είναι ενεργοποιημένη (eos = 1), η έξοδος amplitude συνδέεται με το data.

Ορίζουμε ακόμη το wire quadrant το οποίο υποδεικνύει κάθε στιγμή σε ποιο τεταρτημόριο βρισκόμαστε. Αποτελείται ουσιαστικά από τα 2 πιο σημαντικά bits της εισόδου phase. Για να γίνει κατανοητός ο υπολογισμός του τεταρτημόριου από τη φάση, μπορούμε να θεωρήσουμε ότι $quadrant = phase\ div\ 2^{N-2}$, δηλαδή ότι το quadrant είναι το ακέραιο πηλίκο της φάσης με το πλήθος των τιμών ανά τεταρτημόριο. Επειδή όμως το εύρος τιμών του είναι το $[0, 2^2) = [0, 2^2 - 1] = [0, 3]$, η αρίθμηση των τεταρτημόριων στο εξής θα ξεκινάει από το 0 και όχι από το 1 (π.χ. το 1ο τεταρτημόριο αντιστοιχεί σε quadrant = 00).

Στη συνέχεια ακολουθεί η κυρίως λειτουργία του PAC, όπου ορίζονται τα δεδομένα της μνήμης και γίνονται όλοι οι απαραίτητοι μετασχηματισμοί ανάλογα με το τεταρτημόριο που βρίσκεται η φάση. Σύμφωνα με τον πίνακα της σελίδας 58, παρατηρούμε πως στο 2ο και 4ο τεταρτημόριο οι τιμές διαβάζονται με αντίστροφη σειρά. Έτσι, εάν το LSB του quadrant είναι 1, δηλαδή αν το quadrant είναι μονός αριθμός (αντιστοιχεί στα τεταρτημόρια 2, 4), τότε η διεύθυνση που πρόκειται να διαβαστεί προκύπτει ως η διαφορά της τρέχουσας φάσης από τη μέγιστη τιμή της φάσης, ή αλλιώς $address = 2^{N-2} - 1 - phase_{N-2}$. Η πράξη αυτή σε δυαδική λογική αντιστοιχεί με το συμπλήρωμα ως προς 1 της φάσης.

Για την απόδοση τιμής στο data, χρησιμοποιείται η δομή επιλογής case της Verilog. Έτσι, ανάλογα με τη διεύθυνση που μόλις υπολογίστηκε, το data λαμβάνει την τιμή πλάτους που της αντιστοιχεί. Το κομμάτι αυτό αποτελεί τη μνήμη του PAC, το οποίο όταν υλοποιηθεί σε hardware, θα αποτελείται από μία συστοιχία λογικών πυλών που θα δίνουν τη λογική συνάρτηση του LUT. Το σκεπτικό είναι αντίστοιχο με μίας ROM με μόνη διαφορά ότι η ROM δεν υλοποιεί τη λογική συνάρτηση με λογικές πύλες.

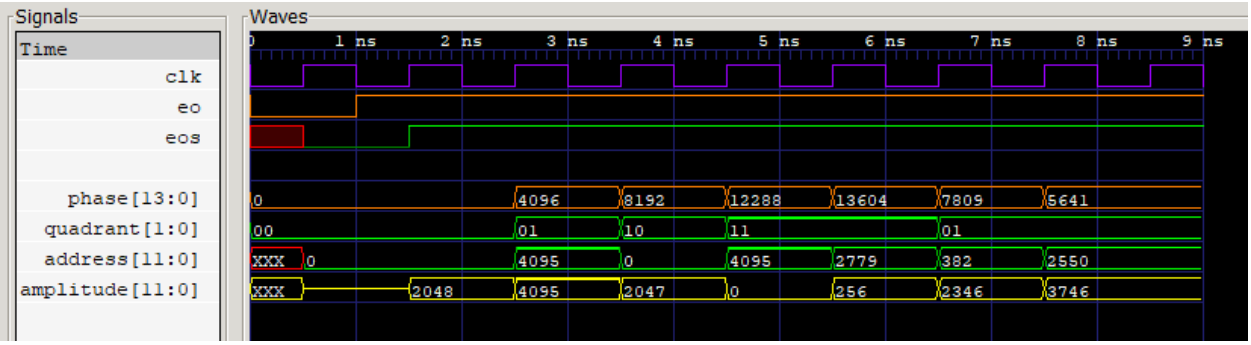
Εάν τέλος βρισκόμαστε σε κάποιο από τα δύο τελευταία τεταρτημόρια και επομένως το MSB του quadrant είναι 1, δηλαδή quadrant > 1 (αντιστοιχεί στα τεταρτημόρια 3, 4), τότε η ζητούμενη τιμή προκύπτει ως η διαφορά του data από τη μέγιστη τιμή του πλάτους $2^P - 1 = 2^{12} - 1$, ή $data = 2^P - 1 - data$. Ομοίως με τον υπολογισμό της διεύθυνσης, και αυτή η πράξη ισοδυναμεί με το συμπλήρωμα του data ως προς 1 (βλ. Πίνακας 3-9).

3.5.4 Προσομοίωση

Το testbench για την επαλήθευση της λειτουργίας του PAC ακολουθεί το πρότυπο όλων των προηγούμενων μονάδων. Έτσι, ορίζεται η χρονική κλίμακα, τα προσωρινά registers και wires για τον έλεγχο των εισόδων και παρακολούθηση των εξόδων αντίστοιχα κτλ. Αφού λοιπόν οριστθούν όλα τα απαραίτητα στοιχεία για τη λειτουργία του testbench και τα registers λάβουν τις αρχικές τιμές τους, ορίζουμε την περίοδο του ρολογιού στο 1 ns αλλάζοντας την κατάστασή του κάθε 0.5 ns.

Η ανάλυση που θα πραγματοποιηθεί αφορά αρχικά τη λειτουργία ενεργοποίησης εξόδου και στη συνέχεια την απόκριση του PAC για διάφορες τιμές φάσης από κάθε τεταρτημόριο.

Αρχικά δοκιμάζονται οι πρώτες τιμές κάθε τεταρτημόριου (0, 4096, 8192, 12288) και στη συνέχεια ακολουθούν 3 τυχαίες τιμές.



Εικόνα 3-16 Προσομοίωση λειτουργιών PAC

Ξεκινώντας τη λειτουργία του PAC, παρατηρούμε πως μέχρι τον πρώτο θετικό παλμό οι τιμές των εσωτερικών του καταχωρητών είναι άγνωστες (κατάσταση X), όπως συμβαίνει σε κάθε κύκλωμα μέχρι την αρχικοποίησή του. Αντί όμως για reset, εδώ η αρχικοποίηση συμβαίνει απευθείας από την είσοδο phase. Έτσι, με τον πρώτο θετικό παλμό όλα τα registers παίρνουν τις τιμές που τους αναλογούν. Παρατηρούμε ακόμη, ότι παρά την ασύγχρονη μετάβαση του eo σε υψηλό δυναμικό, η έξοδος δεν ενεργοποιείται μέχρι τον επόμενο θετικό παλμό, όπου το eos γίνεται 1 και έτσι το amplitude συνδέεται με το data.

Με την ενεργοποίηση της εξόδου, εμφανίζεται το πλάτος που αντιστοιχεί στη μηδενική φάση. Για κάθε φάση που εισάγεται βλέπουμε τον σωστό υπολογισμό του τεταρτημόριου στον καταχωρητή quadrant. Εύκολα μπορεί να παρατηρήσει κανείς ότι και η έξοδος ανταποκρίνεται σε αυτή που έχει υπολογιστεί για το ημίτονο, αφού στην αρχή κάθε τεταρτημόριου τα πλάτη που προκύπτουν είναι τα 2048, 4095, 2047, 0, τα οποία σε κλίμακα [-1, 1] αντιστοιχούν στα 0, 1, 0, -1, αντίστοιχα.

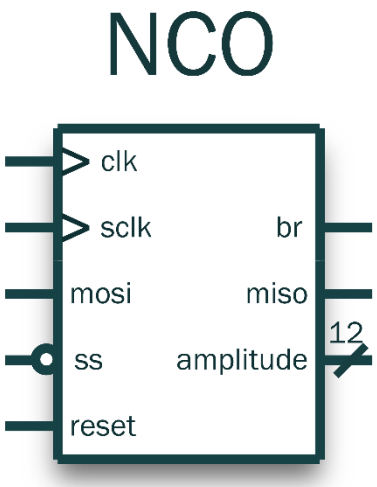
Παρατηρούμε πως για όλες τις περιπτώσεις που ελέγχονται στην ανάλυση του PAC, οι τιμές των σημάτων συμφωνούν με τις αναμενόμενες όπως υπολογίστηκαν στο MATLAB και επομένως επαληθεύεται η ορθή λειτουργία του PAC, αφού οι τιμές έχουν αποθηκευτεί σωστά και ο μετασχηματισμός τους για το κάθε τεταρτημόριο είναι σωστός. Τα παραπάνω γίνονται εμφανή στον παρακάτω πίνακα.

Phase	Quadrant $= phase\ div\ 2^{N-2}$	Address [5]	Amplitude
0	$0 = 00_b \rightarrow 1^o$	$phase_{N-2} = 0$	$data = 2.048$
4.096	$1 = 01_b \rightarrow 2^o$	$2^{N-2} - 1 - phase_{N-2} = 4.095$	$data = 4.095$
8.192	$2 = 10_b \rightarrow 3^o$	$phase_{N-2} = 0$	$2^P - 1 - data = 2.047$
12.288	$3 = 11_b \rightarrow 4^o$	$2^{N-2} - 1 - phase_{N-2} = 4.095$	$2^P - 1 - data = 0$
13.604	$3 = 11_b \rightarrow 4^o$	$2^{N-2} - 1 - phase_{N-2} = 2.779$	$2^P - 1 - data = 256$
7.809	$1 = 01_b \rightarrow 2^o$	$2^{N-2} - 1 - phase_{N-2} = 382$	$data = 2.346$
5.641	$1 = 01_b \rightarrow 2^o$	$2^{N-2} - 1 - phase_{N-2} = 2.550$	$data = 3.746$

Πίνακας 3-10 Αναμενόμενες τιμές σημάτων, σύμφωνα με το MATLAB script

3.6 Αριθμητικώς Ελεγχόμενος Ταλαντωτής (Numerically Controlled Oscillator, NCO)

3.6.1 Δομή



Εικόνα 3-17 Μονάδα NCO

I/O Pin	I/O Name	I/O Type	I/O Length	I/O Logic	Triggered On
clk	Clock	input	1	positive	positive edge
sclk	Serial Clock	input	1	positive	positive edge
mosi	Master Out Slave In	input	1	positive	-
ss	Slave Select	input	1	negative	-
reset	Reset	input	1	positive	positive edge
br	Buffer Ready	output	1	positive	-
miso	Master In Slave Out	output	1	positive	-
amplitude	Amplitude	output	12	positive	-

Πίνακας 3-11 Είσοδοι / Έξοδοι PAC

[5] Η χρήση κάτω δείκτη σε ονομασία ενός καταχωρητή reg_x, παραπέμπει στην περικοπή των $L - x$ σημαντικότερων ψηφίων, όπου L το μήκος του καταχωρητή, και επομένως $reg_x = reg[x - 1 : 0]$.

Η μονάδα NCO φέρει σχεδόν τις ίδιες εισόδους και εξόδους με την OPC, με μόνη διαφορά την έξοδο amplitude αντί της phase που έχει το OPC. Η phase συνδέεται εσωτερικά του NCO μεταξύ των OPC και PAC.

Το NCO είναι η τελική μονάδα που πλαισιώνει ολόκληρη τη λειτουργία του DDS και επομένως περιέχει αποκλειστικά όσες εισόδους χρειάζονται για τη σειριακή επικοινωνία και τον προγραμματισμό του DDS, την αρχικοποίησή του (reset) και τέλος την έξοδο του πλάτους για την εξαγωγή της παραγόμενης κυματομορφής.

3.6.2 Λειτουργία

Η λειτουργία του NCO δεν εισάγει καμία επιπρόσθετη λογική στο συνολικό σύστημα. Ουσιαστικά αποτελεί την ομαδοποίηση όλων των προηγούμενων μονάδων και την συντονισμένη λειτουργία τους κάτω από μία κοινή μονάδα.

Το μόνο σημείο που ελέγχει το NCO, είναι η είσοδος eo του PAC. Έτσι, η έξοδος απενεργοποιείται (κατάσταση υψηλής αντίστασης Z) όταν βρισκόμαστε σε reset ή το mode του PA δεν είναι 0 και επομένως ισχύει outr = 0. Με άλλα λόγια, η κυματομορφή εμφανίζεται στην έξοδο μόνο όταν το DDS έχει σε κανονική λειτουργία. Τέλος, πάρα την ασύγχρονη φύση του reset, η έξοδος αλλάζει κατάσταση πάντα σε συγχρονισμό με το ρολόι clk, όπως αναλύεται στη λειτουργία του PAC.

3.6.3 Προσομοίωση

Έχοντας επαληθεύσει τις λειτουργίες των OPC και PAC ξεχωριστά, μένει μόνο να ελέγξουμε την ορθή λειτουργία όταν «συνεργάζονται» μεταξύ τους. Τα μόνα σημεία που δεν έχουμε ελέγξει λοιπόν, είναι η κανονική λειτουργία και η λειτουργία αδράνειας του DDS.

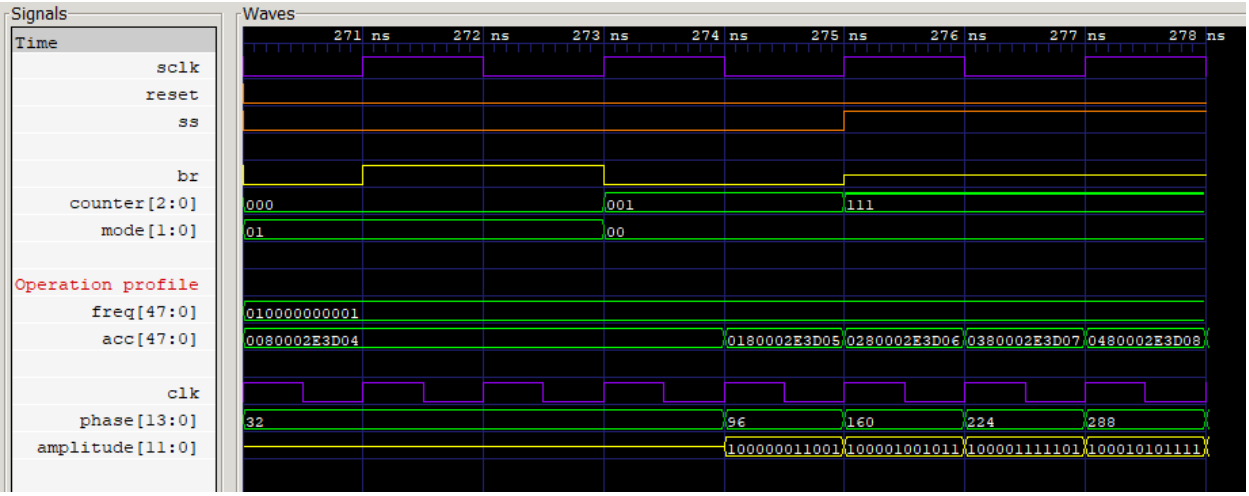
Δημιουργούμε ένα testbench για το NCO, το οποίο μοιάζει σε μεγάλο βαθμό με εκείνο του OPC. Στην αρχή προγραμματίζουμε το operation profile του DDS στις τιμές $FCW = 010000000001$ και $phase\ offset = 0080002E3D04$, στην δεκαεξαδική τους αναπαράσταση. Έπειτα, δίνουμε εντολή για έναρξη της κανονικής λειτουργίας του DDS, την οποία μετά από λίγο διακόπτουμε προσωρινά, ώστε να επαληθεύσουμε και τη λειτουργία αδράνειας.

Στην Εικόνα 3-18 προσομοιώνεται η κανονική λειτουργία του NCO και επομένως του DDS. Αρχικά, ολοκληρώνεται το instruction byte και το mode του PA σωστά λαμβάνει την τιμή 00, ώστε να ξεκινήσει η κανονική λειτουργία με το operation profile που προγραμματίσαμε στην αρχή του testbech.

Παρά την αλλαγή του mode τη χρονική στιγμή 273 ns, η κυματομορφή ξεκινά να παράγεται από τον αμέσως επόμενο θετικό παλμό του clk. Το γεγονός αυτό οφείλεται στην πύλη buffer που είχε συνδεθεί στην είσοδο mode του PA. Έτσι, η μικρή καθυστέρηση που εισάγεται από την πύλη, κάνει το mode του PA να αλλάζει ελαφρώς μετά τον παλμό. Ως αποτέλεσμα η φάση παραμένει ίση με το phase offset την πρώτη περίοδο της κανονικής λειτουργίας.

Εάν δεν υπήρχε η εν λόγω καθυστέρηση, το phase θα άλλαζε σε 96 κατευθείαν με την αλλαγή του mode σε 00. Στο μεταξύ η έξοδος και πάλι θα ενεργοποιούταν στα 274 ns, λόγω της

καθυστέρησης μέχρι να ανανεωθεί η τιμή του outr. Επομένως, η πρώτη τιμή amplitude που θα εμφανιζόταν θα ήταν αυτή που αντιστοιχεί σε φάση 96, χάνοντας έτσι το πλάτος της φάσης 32.



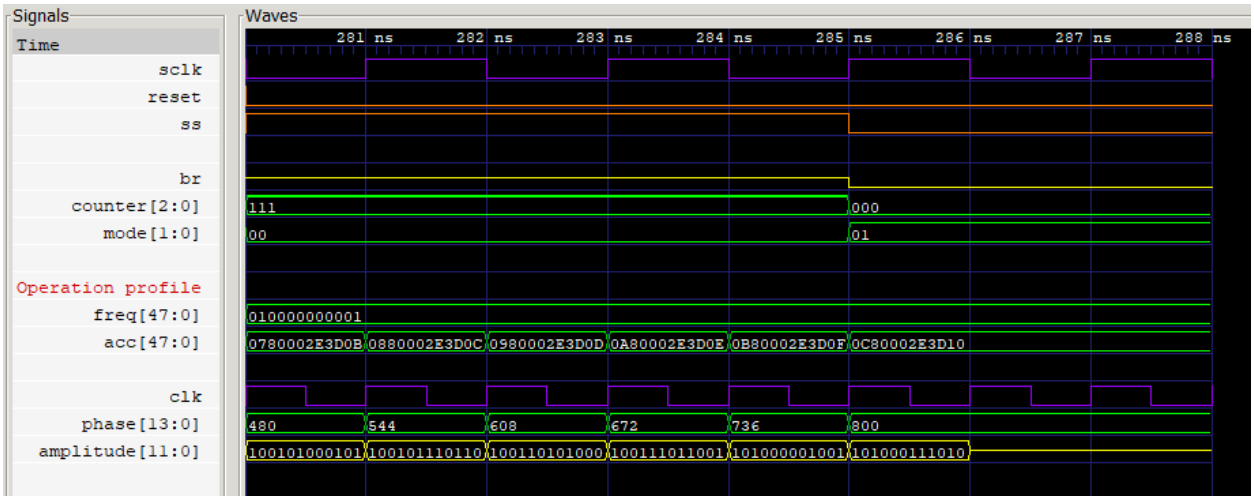
Εικόνα 3-18 Προσομοίωση NCO: κανονική λειτουργία

Επιβεβαιώνεται επίσης, η σωστή αύξηση του acc σε κάθε περίοδο και επομένως και της φάσης. Ακόμη, παρατηρούμε πως το πλάτος αλλάζει την αμέσως επόμενη περίοδο από αυτή που αλλάζει η φάση, λόγω της καθυστέρησης από την ανανέωση του acc έως την ανάγνωση του LUT του PAC. Το γεγονός αυτό δεν επηρεάζει σε κάτι, καθώς διατηρείται σε όλη τη διάρκεια της κανονικής λειτουργίας.

Επιπλέον, επειδή το ss είναι απαραίτητο να παραμένει σε χαμηλό επίπεδο καθ' όλη τη διάρκεια προγραμματισμού, δεν μπορεί να αλλάξει τη χρονική στιγμή 273 ns, διότι σε αυτή την περίπτωση δε θα άλλαζε η τιμή του mode. Έτσι, είναι απαραίτητο το ss να μεταβεί στην κατάσταση 1 μετά την ολοκλήρωση του instruction byte είτε σύγχρονα, είτε όχι με το σειριακό ρολόι. Στο testbench αυτό συμβαίνει μία περίοδο μετά το instruction. Παρατηρούμε ότι ταυτόχρονα με την αλλαγή του ss, αρχικοποιείται και το counter του OPC, ώστε να είναι έτοιμο για επόμενη εντολή αν το ss ξαναγίνει 0. Σημειώνεται πως αν αφήσουμε το ss στην κατάσταση 0, δε θα επηρεάσει την κανονική λειτουργία του NCO έως ότου ο counter μηδενιστεί. Αν λοιπόν, περάσουν $7 \cdot 8 = 56$ περίοδοι του sclk και το ss είναι ακόμη σε χαμηλό δυναμικό, τότε επειδή το counter θα γίνει 0, το OPC θα μεταβεί αυτομάτως σε λειτουργία αδράνειας και θα ξεκινήσει την ανάγνωση του instruction byte.

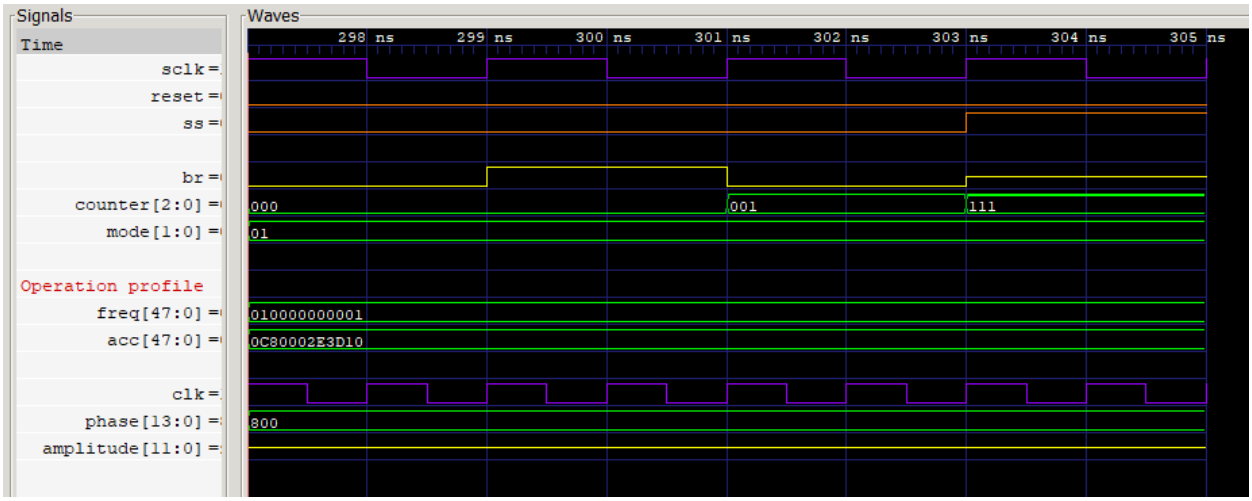
Ολοκληρώνοντας την προσομοίωση, ελέγχουμε και τη λειτουργία αδράνειας. Τη χρονική στιγμή 285 ns θέτουμε ss = 0 και το DDS μεταβαίνει ξανά σε κατάσταση προγραμματισμού. Το instruction byte που δίνουμε αυτή τη φορά είναι το $0001XXXX$ για να υποδείξουμε το halt mode. Όπως φαίνεται στην Εικόνα 3-19Εικόνα 3-19, το τελευταίο πλάτος εμφανίζεται μία περίοδο του clk μετά την αλλαγή του ss σε 0 και αντιστοιχεί στη φάση 736. Έτσι, η κυματομορφή άρχισε μία περίοδο μετά την έναρξη της κανονικής λειτουργίας και τελειώνει επίσης μία περίοδο μετά τη μετάβαση σε λειτουργία αδράνειας. Αντίστοιχα, και η φάση αυξάνεται, ώστε όταν

το DDS βρεθεί ξανά σε κανονική λειτουργία, η έξοδος να ξεκινήσει από το αμέσως επόμενο πλάτος από αυτό που είχε σταματήσει.



Εικόνα 3-19 Προσομοίωση NCO: προγραμματισμός κατά την κανονική λειτουργία

Τέλος, στην Εικόνα 3-20 παρουσιάζεται το πέρας του instruction byte και η παραμονή στη λειτουργία αδράνειας μετά τον προγραμματισμό του DDS.

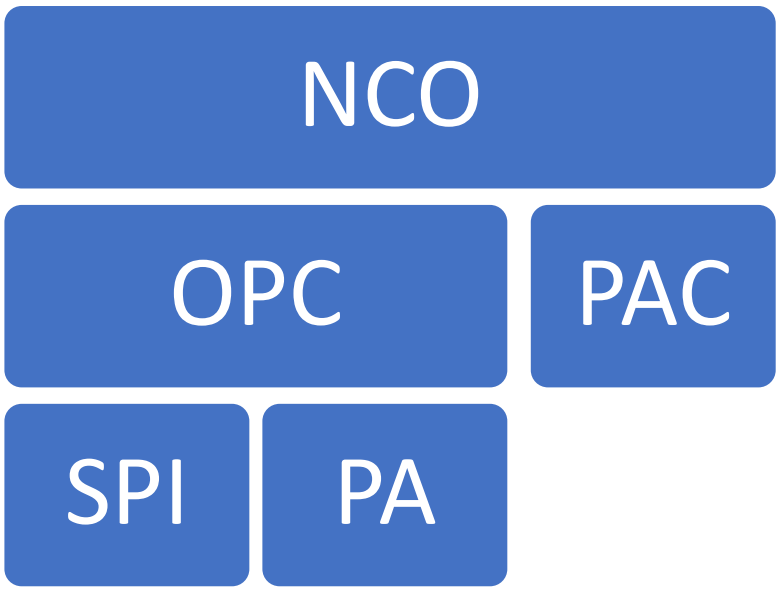


Εικόνα 3-20 Προσομοίωση NCO: λειτουργία αδράνειας

3.7 Ιεραρχία σχεδίασης

Η σχεδίαση του DDS έχει πλέον ολοκληρωθεί. Οι πέντε μονάδες που το συνιστούν, κάνουν τη συνολική κατανόηση του συστήματος αρκετά ευκολότερη, ενώ απλοποιείται ιδιαιτέρως και η διαχείριση της σχεδίασης. Με άλλα λόγια, αν χρειαστεί να γίνει μία αλλαγή, όπως για παράδειγμα η προσθήκη ενός νέου χαρακτηριστικού, τότε αρκεί η επεξεργασία της μονάδας που αφορά η αλλαγή. Επίσης, ο κώδικας είναι αρκετά ευανάγνωστος και μπορεί να κατανοηθεί με άνεση από οποιονδήποτε χρειαστεί να τον διαβάσει.

Στην Εικόνα 3-21 παρουσιάζεται το δέντρο ιεραρχίας των μονάδων:



Εικόνα 3-21 Ιεραρχία μονάδων DDS

Στο σημείο αυτό, το κύκλωμα έχει σχεδιαστεί με όλα τα επιθυμητά χαρακτηριστικά, τα οποία έχουν επαληθευτεί μέσα από τις προσομοιώσεις που έλαβαν χώρα. Έτσι, είναι έτοιμο για τη σύνθεση και στη συνέχεια τη σχεδίαση σε φυσικό επίπεδο.

4 Σύνθεση κυκλώματος

Το Cadence Genus, με το οποίο γίνεται η σύνθεση του κυκλώματος, υποστηρίζει τη γλώσσα προγραμματισμού TCL. Επιτρέπεται λοιπόν στον σχεδιαστή να δημιουργήσει ένα script εντολών TCL, περιγράφοντας όλη τη διαδικασία που πρόκειται να ακολουθηθεί. Έτσι, αντί να εισάγουμε τις εντολές μία – μία στο terminal του προγράμματος, γράφουμε ένα script το οποίο μπορούμε να τρέξουμε όσες φορές θέλουμε με άνεση και ευκολία.

Για να τρέξουμε το script, χρησιμοποιούμε την εντολή source συνοδευόμενη από το path όπου είναι αποθηκευμένο το αρχείο TCL.

Στο Παράρτημα TCL Script – Genus υπάρχει διαθέσιμο το TCL script που υλοποιήθηκε και χρησιμοποιείται καθ' όλη τη διαδικασία που ακολουθεί.

4.1 Προετοιμασία

Προτού ξεκινήσει η σύνθεση του κυκλώματος, χρειάζεται να προηγηθούν κάποια βήματα. Το πρώτο από αυτά είναι ο ορισμός των βιβλιοθηκών της τεχνολογίας που πρόκειται να χρησιμοποιηθεί. Για τη συγκεκριμένη εργασία επιλέχθηκε η τεχνολογία 65 nm της TSMC. Δίνουμε λοιπόν το κατάλληλο μονοπάτι στη μεταβλητή lib_search_path και έπειτα επιλέγεται το αρχείο της βιβλιοθήκης με την εντολή library. Στο σημείο αυτό, δηλώνεται με την εντολή lef_library η βιβλιοθήκη lef, η οποία αφορά τη φυσική σχεδίαση και είναι απαραίτητο να δηλωθεί από την αρχή (πριν το elaboration).

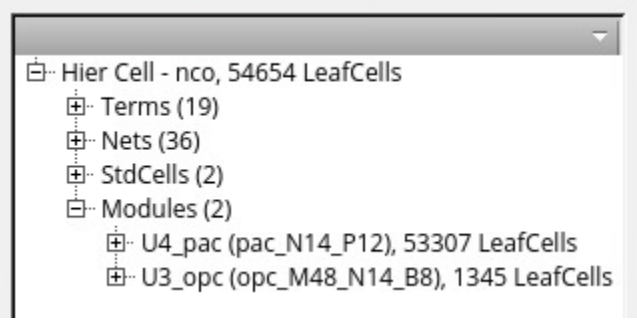
Στη συνέχεια ορίζουμε το auto_ungroup στην τιμή both. Με τη ρύθμιση αυτή επιτρέπουμε στο Genus να «σπάσει» τις μονάδες εάν είναι απαραίτητο και να τις ενοποιήσει. Η λέξη both αναφέρεται τόσο στο χρόνο όσο και στην ενέργεια, εννοώντας ότι για τη βελτιστοποίηση της σύνθεσης λαμβάνονται υπόψη και τα δύο μεγέθη. Με τον τρόπο αυτό, η σύνθεση γίνεται πιο ευέλικτη και συνήθως τα αποτελέσματα που προκύπτουν είναι καλύτερα από όταν δε γίνεται ungroup. Δεδομένου ακόμη πως το συνολικό κύκλωμα είναι σχετικά μικρό και ότι ο διαχωρισμός σε επιμέρους μονάδες έγινε για να εξυπηρετήσει τη σχεδίαση, η ενοποίησή τους σε μία κοινή δε θα επηρεάσει αρνητικά τη σύνθεση ή τη φυσική σχεδίαση.

Αφού μεταφερθούμε στον φάκελο όπου θέλουμε να δουλέψουμε, ξεκινά η ανάγνωση του κώδικα. Έτσι, ρυθμίζουμε το init_hdl_search_path στην τοποθεσία των αρχείων κώδικα και με το hdl_language τη γλώσσα που χρησιμοποιείται με την έκδοσή της. Με την εντολή read_hdl ξεκινά η ανάγνωση του αρχείου HDL που επιθυμούμε. Σημειώνεται πως εάν η σχεδίαση αποτελείται από πολλαπλές μονάδες, τότε αρκεί να υποδείξουμε το αρχείο της ανώτερης στην ιεραρχία μονάδας, διότι σε αυτή γίνονται αναφορές και στις υπόλοιπες (π.χ. με το include της Verilog). Επομένως, καλούμε την read_hdl μόνο για το NCO, το οποίο περιέχει όλες τις υπόλοιπες μονάδες στο εσωτερικό του.

Κατά την ανάγνωση των αρχείων κώδικα, το Genus επισημαίνει τυχόν συντακτικά λάθη που μπορεί να υπάρχουν, ενώ μας προειδοποιεί για διάφορα άλλα πιθανά λογικά σφάλματα. Εάν λοιπόν η ανάγνωση του κώδικα είναι επιτυχής, τότε το κύκλωμα είναι έτοιμο για

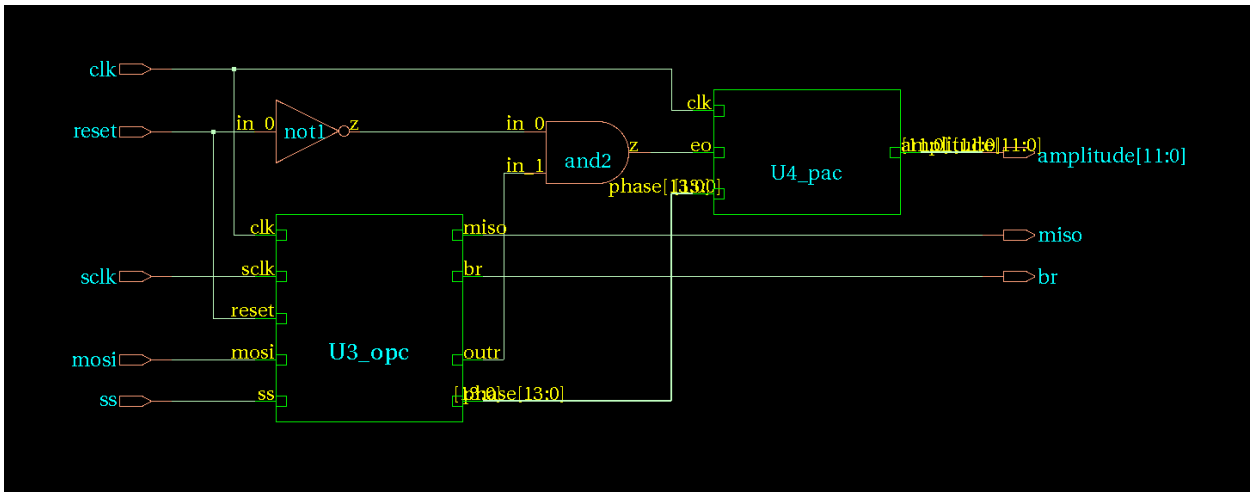
elaboration. Όπως αναφέρεται στην ενότητα 1.3.2, το elaboration είναι απαραίτητο να προηγείται της σύνθεσης, καθώς εκτελεί διάφορες προεργασίες και βελτιώσεις στον κώδικα.

Εκτελούμε λοιπόν την εντολή elaborate και λαμβάνουμε τα πρώτα αποτελέσματα. Στην Εικόνα 4-1 φαίνονται τα αποτελέσματα του elaboration. Όπως παρατηρούμε, το PAC αποτελεί με διαφορά το μεγαλύτερο μέρος του DDS. Τα στοιχεία του βέβαια αναμένουμε να μειωθούν αρκετά, έπειτα από τις βελτιστοποιήσεις κατά τη σύνθεση. Στο πρώτο αυτό στάδιο έχει γίνει απλά μία «μετάφραση» της Verilog σε στοιχειώδη ψηφιακά στοιχεία και γι' αυτό το κύκλωμα φαίνεται τεράστιο.



Εικόνα 4-1 Αποτελέσματα elaboration

Σημειώνεται τέλος πως εφόσον δε δηλώθηκαν σε κανένα σημείο οι τιμές των παραμέτρων του NCO (M, N, P, B), αυτομάτως το Genus τους δίνει τις default τιμές τους, όπως φαίνεται παραπάνω.

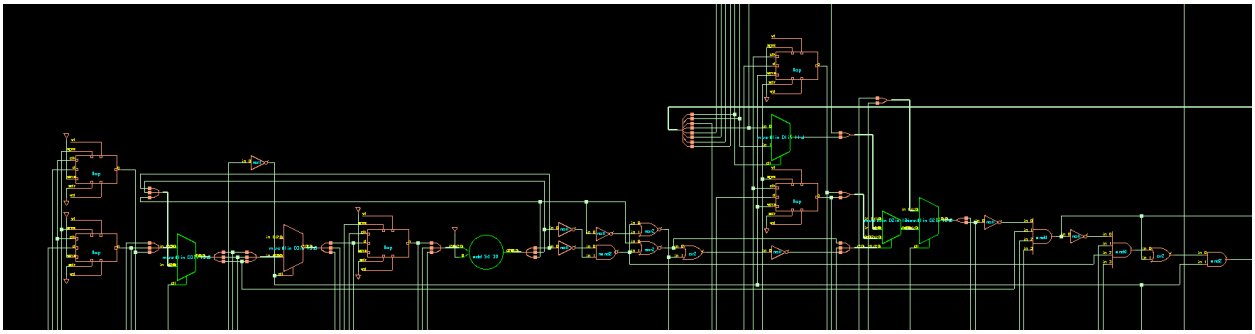


Εικόνα 4-2 Σχηματικό NCO μετά το elaboration

Στην Εικόνα 4-2 παρουσιάζεται το σχηματικό του NCO. Όλες οι διασυνδέσεις έχουν γίνει όπως σχεδιάστηκε και ακόμη η είσοδος eo του PAC συνδέεται με μία πύλη AND που έχει ως εισόδους το NOT του reset και το outr του OPC (eo = !reset && outr).

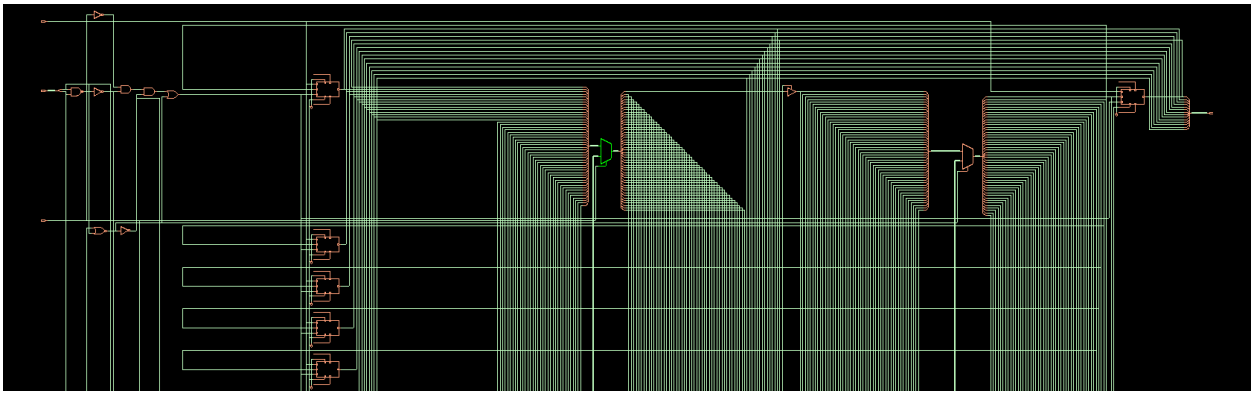
Το OPC αποτελείται συνολικά από 1.345 στοιχεία και έτσι το σχηματικό του είναι αρκετά μεγάλο για να παρουσιαστεί ολόκληρο. Παρουσιάζεται λοιπόν ένα μέρος αυτού στην Εικόνα

4-3, όπου διακρίνονται οι καταχωρητές counter (3 flip-flop στα αριστερά) και mode (2 flip-flop στο κέντρο). Στη μέση επίσης, φαίνεται ο αθροιστής (κυκλικό σύμβολο) ο οποίος συνδέεται με την κατάλληλη λογική, ώστε να αυξάνεται το counter κατά μία μονάδα όποτε απαιτείται. Οι υπόλοιπες πύλες υλοποιούν τη λογική του OPC για να εκτελεί όλες τις λειτουργίες, όπως έχουν σχεδιαστεί.



Εικόνα 4-3 Μέρος σχηματικού OPC μετά το elaboration

Ακολουθεί ένα μέρος του σχηματικού του PA (βλ. Εικόνα 4-4), καθώς και αυτή η μονάδα είναι αρκετά μεγάλη για να παρουσιαστεί ολόκληρη. Στα αριστερά διακρίνονται οι είσοδοι clk, mode και fph μαζί με τη λογική που τα συνοδεύει. Λίγο πιο δίπλα φαίνονται τα 5 MSB του phase register, τα οποία αποτελούν και κομμάτι της εξόδου phase. Στο κέντρο του σχηματικού, οι δύο πολυπλέκτες (mux) ελέγχουν τις συνδέσεις των acc και word ανάλογα με τη ροή των δεδομένων από ή προς το PA.

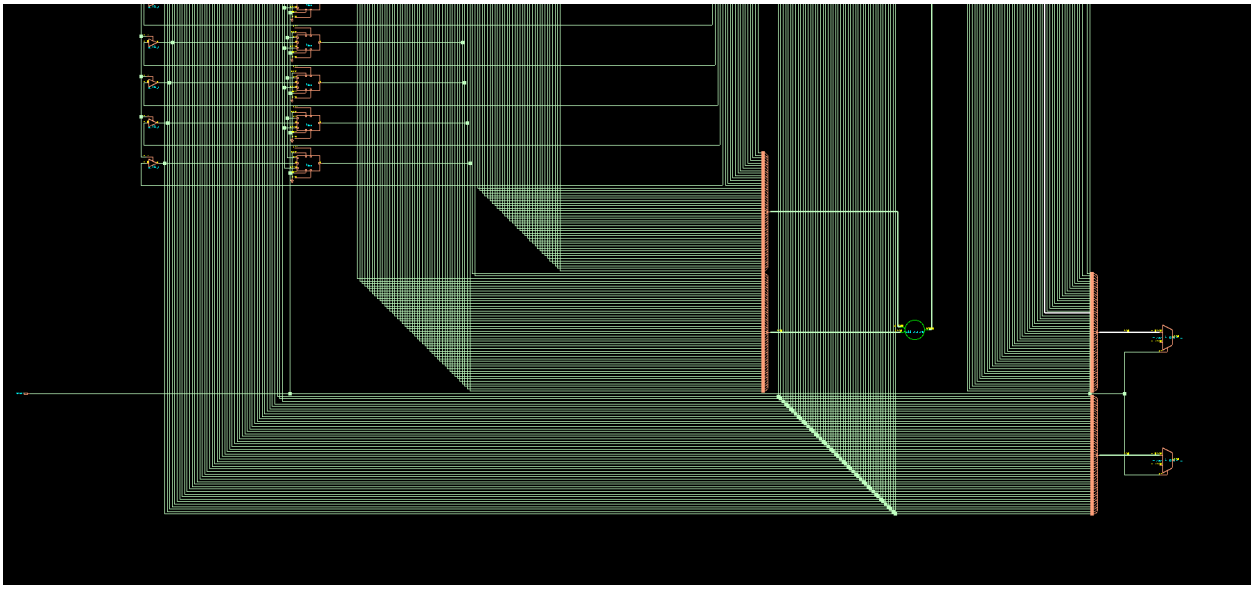


Εικόνα 4-4 Άνω μέρος σχηματικού PA μετά το elaboration

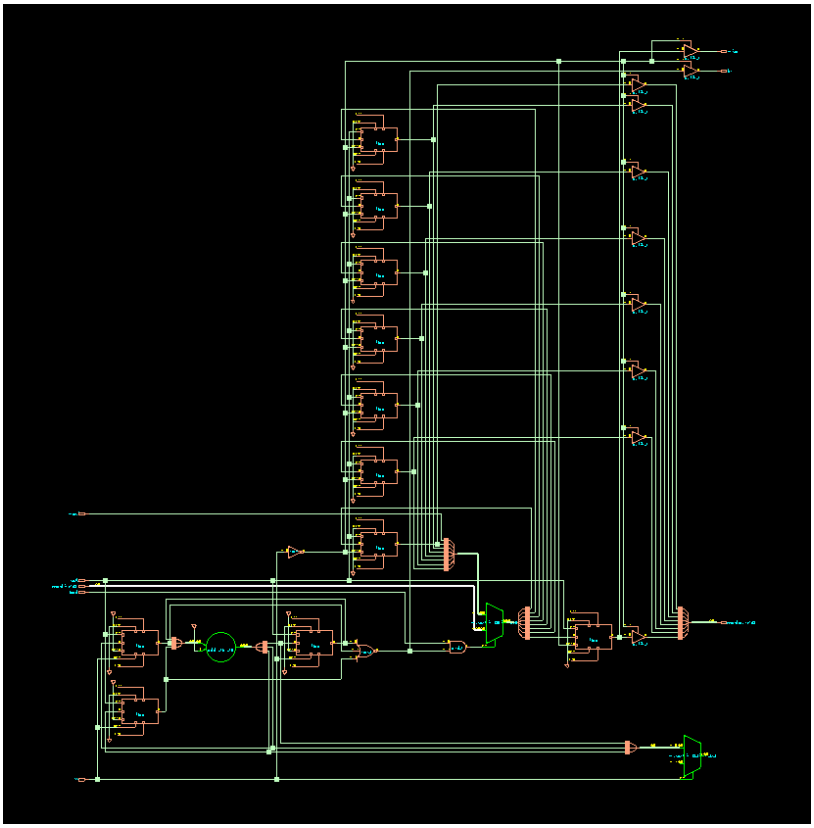
Στο κάτω μέρος του σχηματικού του PA στην Εικόνα 4-5 φαίνονται ακόμη δύο πολυπλέκτες (στα δεξιά), οι οποίοι έχουν αντίστοιχη χρησιμότητα με αυτούς που περιγράφονται στην προηγούμενη εικόνα, αλλά αφορούν το frequency register. Επιπλέον, διακρίνεται και ο αθροιστής της μονάδας στον οποίο συνδέονται τα acc και freq και η έξοδός του επιστρέφει το αποτέλεσμα στον acc. Τέλος, πάνω αριστερά βλέπουμε τα 5 LSB του frequency register.

Στην Εικόνα 4-6 παρουσιάζεται το σχηματικό του SPI. Η μονάδα αυτή είναι η μικρότερη όλων, αποτελούμενη μόλις από 44 στοιχεία κατά το elaboration. Στο επάνω τμήμα διακρίνεται

το buffer register μαζί με τα tristate buffers που ελέγχουν αν συνδέεται με το wordout ή αν γράφεται σε αυτό η τιμή του wordin. Τα 3 flip-flop στα αριστερά συνιστούν το counter του SPI, τα οποία συνδέονται κατάλληλα και με έναν αθροιστή για την αύξηση του counter.



Εικόνα 4-5 Κάτω μέρος σχηματικού PA μετά το elaboration



Εικόνα 4-6 Σχηματικό SPI μετά το elaboration

Σημειώνεται τέλος, πως το σχηματικό του PAC δεν παρουσιάζεται σε κάποια εικόνα, καθώς το μέγεθός του στο elaboration είναι απαγορευτικό. Ωστόσο, το κύριο μέρος του σχηματικού του είναι η υλοποίηση του LUT. Επειδή το τελευταίο περιέχει 4.096 τιμές ακρίβειας 12 bit η κάθε μία, το μέγεθος του PAC ήταν αναμενόμενο να είναι πολύ μεγάλο πριν λάβουν χώρα ο διάφορες βελτιστοποιήσεις.

4.2 Σύνθεση

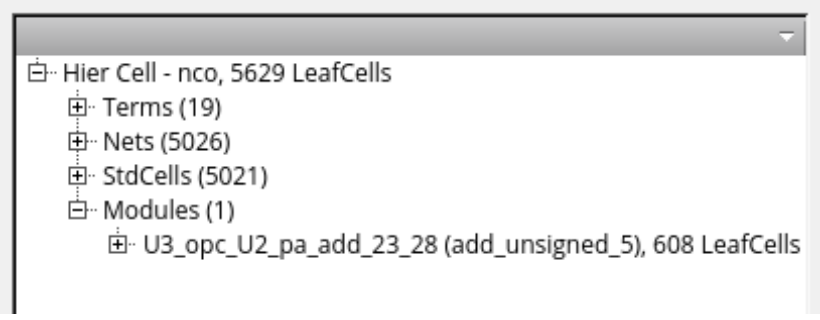
Για να γίνει η βέλτιστη σύνθεση, είναι απαραίτητο να έχει προηγηθεί η δήλωση όλων των περιορισμών του κυκλώματος. Έτσι, στο σημείο αυτό δημιουργούμε τα ρολόγια του συστήματος μέσω της εντολής define_clock. Πιο συγκεκριμένα, ορίζουμε την ελάχιστη περίοδο με τις παραμέτρους -period και -divide_period (για μεγαλύτερη ακρίβεια), καθώς και το αντίστοιχο pin για το κάθε ρολόι. Ορίζουμε ακόμη τις εξωτερικές καθυστερήσεις στα pins εισόδων / εξόδων, οι οποίες αφορούν τον χρόνο από τον θετικό παλμό ενός ρολογιού έως την αλλαγή της κατάστασης του αντίστοιχου pin. Οι τελευταίες είναι ενδεικτικές, αλλά είναι απαραίτητες για τη χρονική ανάλυση του κυκλώματος.

Στον παρακάτω πίνακα παρουσιάζονται οι εντολές σύνθεσης του Genus. Η syn_gen πραγματοποιεί απλά βελτιστοποιήσεις σε επίπεδο καταχωρητών RTL, η syn_map αντικαθιστά τις γενικές πύλες του προηγούμενου βήματος με αυτές της βιβλιοθήκης που χρησιμοποιείται, δοκιμάζοντας διάφορους συνδυασμούς για την εύρεση του βέλτιστου (incremental optimizations) Τέλος, η syn_opt συνοδευόμενη από την παράμετρο -physical, πραγματοποιεί βελτιστοποιήσεις αφού πρώτα τοποθετήσει τα στοιχεία στο φυσικό σχέδιο (placement). Παρατηρούμε πως οι τρεις εντολές είναι ανεξάρτητες μεταξύ τους. Για παράδειγμα αν τρέξουμε πρώτα την syn_map, θα εκτελέσει και τις ενέργειες της syn_gen, ενώ αν καλέσουμε τη syn_gen μετά τη syn_map, θα αναιρέσει τις ενέργειες της δεύτερης. Στη συνέχεια εκτελούνται διαδοχικά οι εντολές σύνθεσης για ευκολότερη παρακολούθηση των αποτελεσμάτων.

Specified Command	Current Design State		
	RTL	**Generic**	**Mapped**
syn_gen	▪ RTL Optimization		▪ Unmapping
syn_map	▪ RTL Optimization ▪ Mapping ▪ Incremental optimizations	▪ Mapping ▪ Incremental optimizations	▪ Unmapping ▪ Mapping ▪ Incremental optimizations
syn_opt -physical	▪ RTL Optimization ▪ Mapping ▪ Placement ▪ Post-placement incremental optimizations	▪ Mapping ▪ Placement ▪ Post-placement incremental optimizations	▪ Placement ▪ Post-placement incremental optimizations

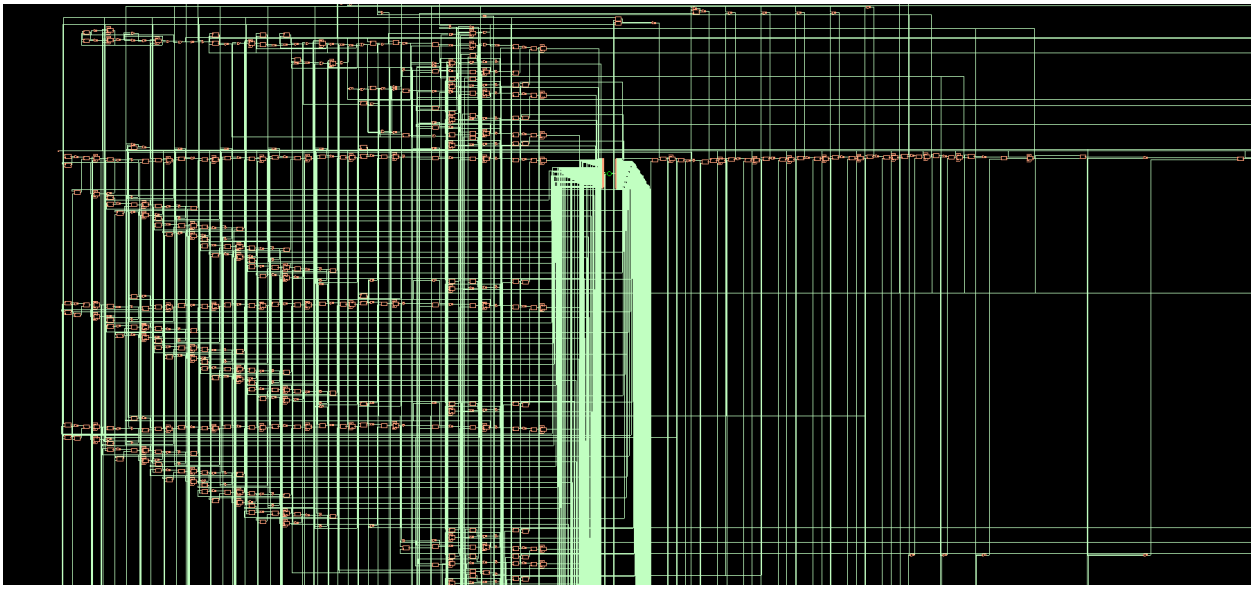
Πίνακας 4-1 Ενέργειες που εκτελούνται από τις εντολές σύνθεσης

Με το πέρας της γενικής σύνθεσης στην Εικόνα 4-7, παρατηρούμε δύο μεγάλες διαφορές σε σχέση με το elaboration. Αρχικά τα συνολικά στοιχεία του κυκλώματος έχουν μειωθεί από 54.654 σε μόλις 5.629, δηλαδή κατά 89,7%. Η μείωση αυτή αφορά κυρίως τα στοιχεία του PAC, λόγω των απλοποιήσεων της λογικής συνάρτησης του LUT από το syn_gen. Η άλλη αξιοσημείωτη αλλαγή αφορά το ungrouping των μονάδων. Έτσι, πλέον υπάρχουν μόνο δύο μονάδες στην ιεραρχία: η NCO και η OPC. Οι υπόλοιπες συγχωνεύθηκαν η κάθε μία με την αμέσως ανώτερη ιεραρχικά μονάδα. Για παράδειγμα, στο OPC πλέον έχουν ενσωματωθεί τα PA και SPI σχηματίζοντας μία νέα ενιαία μονάδα με ίδια λειτουργικότητα, αλλά καλύτερη συμπεριφορά ως προς τον χρόνο και την κατανάλωση ενέργειας.

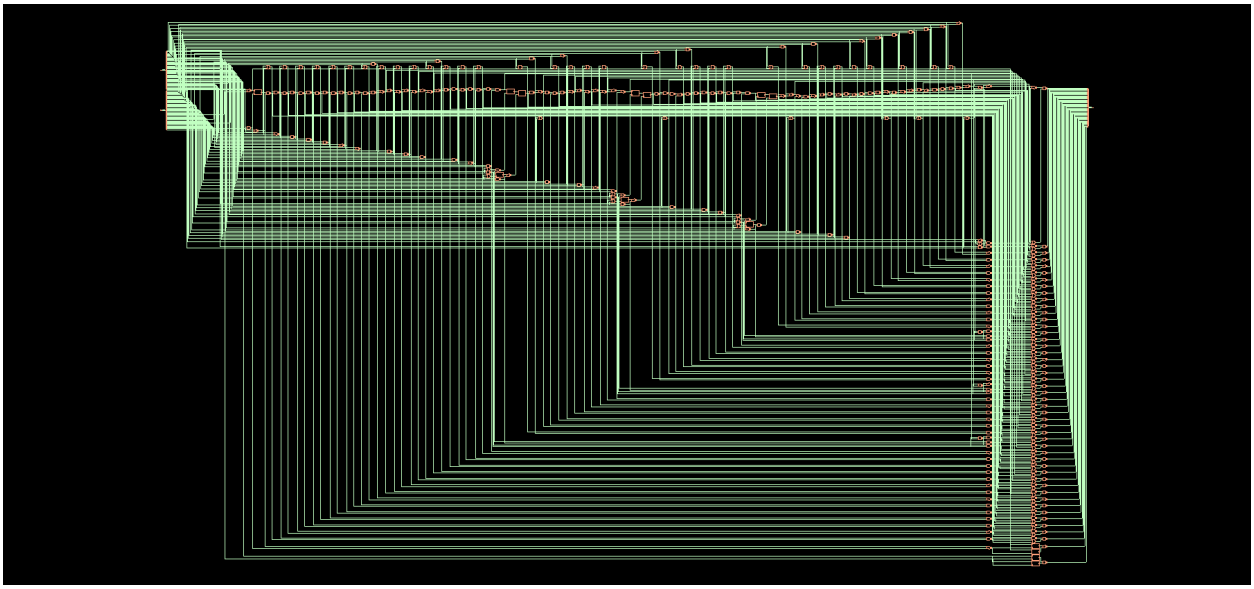


Εικόνα 4-7 Αποτελέσματα syn_gen

Παρακάτω ακολουθούν τα σχηματικά των δύο μονάδων όπως έχουν διαμορφωθεί μετά τη γενική σύνθεση:

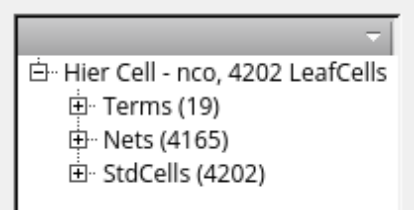


Εικόνα 4-8 Μέρος σχηματικού NCO μετά τη syn_gen



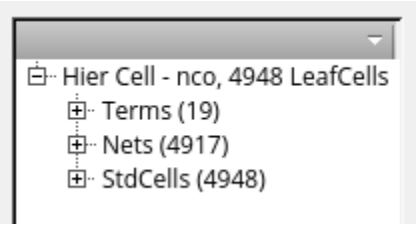
Εικόνα 4-9 Σχηματικό ungrouped OPC μετά τη syn_gen

Συνεχίζοντας με τη syn_map, τα συνολικά στοιχεία μειώνονται κατά 25,3%. Τα στοιχεία αυτά πλέον είναι πραγματικά και περιέχονται όλα στη βιβλιοθήκη της τεχνολογίας που χρησιμοποιούμε. Επιπλέον, έχει πραγματοποιηθεί περεταίρω ungrouping, ενοποιώντας όλες τις μονάδες με την ανώτερη NCO. Η ungrouped μονάδα NCO φαίνεται στην Εικόνα 4-12 της σελίδας 75. Εποπτικά μπορούμε να διαχωρίσουμε το σχηματικό αυτό σε δύο τμήματα: i) το PAC στα δεξιά, και ii) τις υπόλοιπες μονάδες στο πάνω μέρος.



Εικόνα 4-10 Αποτελέσματα syn_map

Ολοκληρώνοντας τη σύνθεση, εκτελούμε τη syn_opt, τα αποτελέσματα της οποίας παρουσιάζονται στην Εικόνα 4-11. Αυτή τη φορά τα συνολικά στοιχεία αυξάνονται μετά τις βελτιστοποιήσεις, ώστε οι περιορισμοί να ικανοποιούνται όσο το δυνατόν καλύτερα λαμβάνοντας υπόψη και τη σχεδίαση σε φυσικό επίπεδο.



Εικόνα 4-11 Αποτελέσματα syn_opt

4.3 Αποτελέσματα & αναφορές

Μετά τη σύνθεση του κυκλώματος, έχουμε μία πρώτη εκτίμηση διάφορων μετρικών που αφορούν το σύστημα. Με την εντολή report του Genus λαμβάνουμε αναφορές που αφορούν τη χρονική ανάλυση, την επιφάνεια ολοκλήρωσης, τα ρολόγια, την κατανάλωση ενέργειας κ.α. Ακόμη καταγράφονται λεπτομερώς όλα τα στοιχεία που χρησιμοποιούνται, καθώς και οι μεταξύ τους συνδέσεις. Οι σημαντικότερες από τις αναφορές αυτές, παρουσιάζονται στους πίνακες που ακολουθούν.

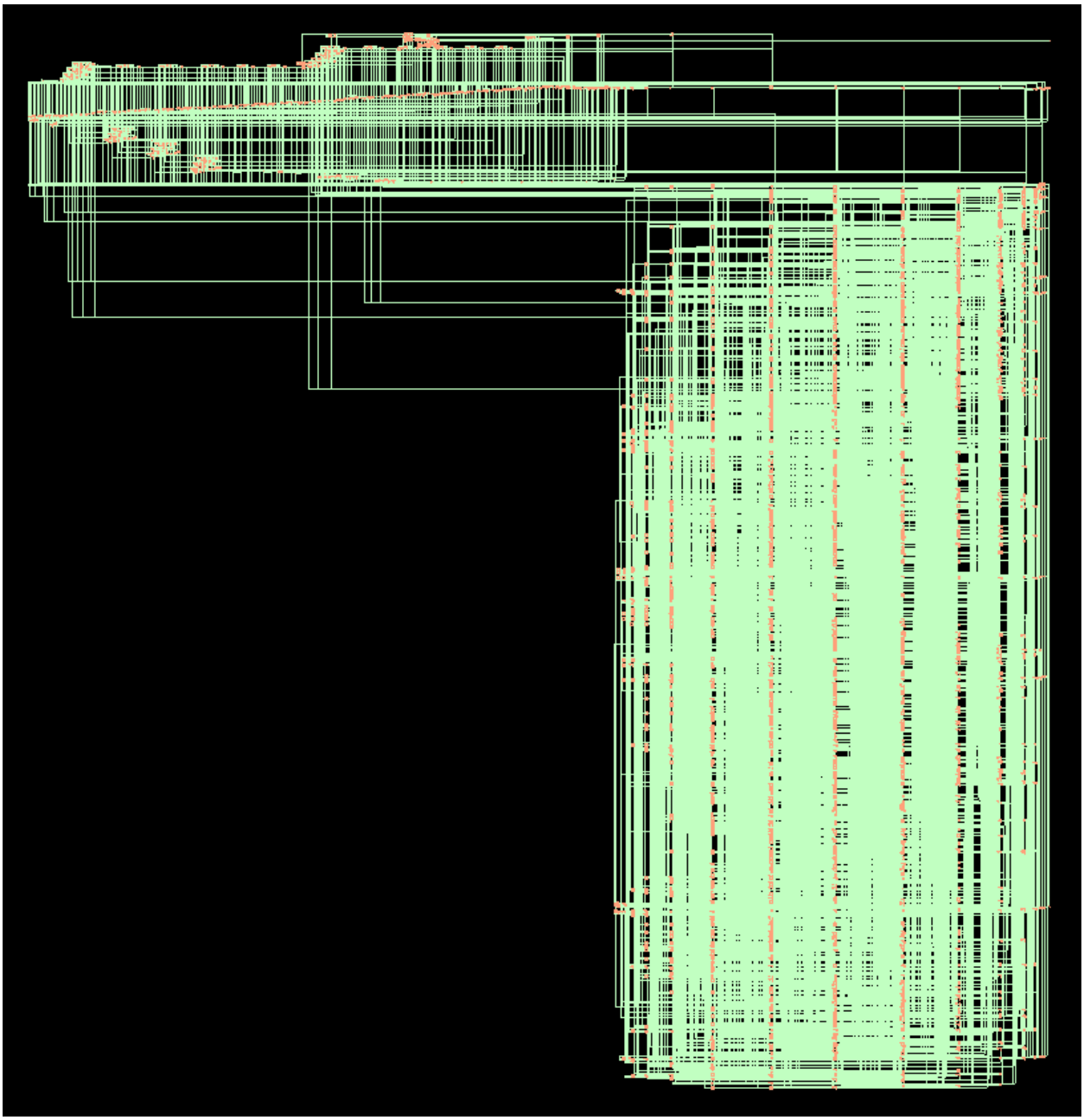
Area Report [6]	
Cell Area	18.570,60
Net Area	9.721,37
Total Area	28.291,97

Πίνακας 4-2 Αναφορά επιφάνειας ολοκλήρωσης

Power Report (nW)	
Leakage Power	581.937,89
Dynamic Power	30.205.026,14
Total Power	30.786.964,03

Πίνακας 4-3 Αναφορά κατανάλωσης ενέργειας

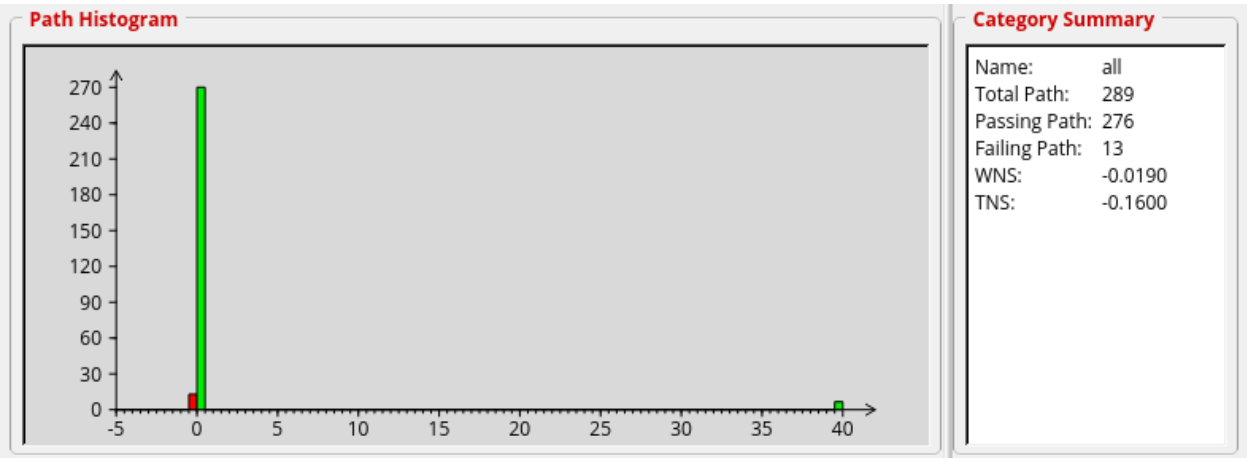
[6] Η μονάδα μέτρησης επιφάνειας δεν είναι διαθέσιμη από το Genus, καθώς εξαρτάται από τη βιβλιοθήκη της τεχνολογίας. Συνήθως χρησιμοποιούνται τα τετραγωνικά μικρόμετρα μm^2.



Εικόνα 4-12 Σχηματικό ungrouped NCO μετά τη syn_map
Μετά τη syn_opt, το σχηματικό δεν παρουσιάζει μεγάλες διαφορές.

Στην Εικόνα 4-13 παρουσιάζεται η χρονική ανάλυση του κυκλώματος μετά τη βελτιστοποιημένη σύνθεση, η οποία έχει λάβει υπόψη και το placement των στοιχείων. Παρατηρούμε πως υπάρχουν 13 μονοπάτια που δεν ικανοποιούν τους περιορισμούς της σχεδίασης και τα οποία θα προσπαθήσουμε να διορθώσουμε με την προσθήκη ενός δέντρου ρολογιού (CTS) κατά τη φυσική σχεδίαση και τις διάφορες βελτιστοποιήσεις που θα ακολουθήσουν.

Τα περισσότερα μονοπάτια βρίσκονται σε μία μικρή περιοχή δεξιά του μηδενός, ενώ υπάρχουν μερικά ακόμη στην άκρη του ιστογράμματος με περιθώριο (slack) κοντά στα 40 ps. Τα πρώτα αφορούν το κυρίως ρολόι, ενώ τα υπόλοιπα το σειριακό.



Εικόνα 4-13 Ιστόγραμμα περιθωρίων καθυστερήσεων (slack) μονοπατιών (ps)

Στο σημείο αυτό ολοκληρώνεται η διαδικασία της σύνθεσης. Για να προχωρήσουμε στη σχεδίαση σε φυσικό επίπεδο, πρέπει πρώτα να εξάγουμε κάποια αρχεία από το Genus. Αυτό επιτυγχάνεται με τις εντολές write και το όνομα του αρχείου προς αποθήκευση. Η write_encounter εξάγει όλα τα απαιτούμενα αρχεία και δημιουργεί scripts για την αυτοματοποιημένη εισαγωγή της σχεδίασης στο Innovus. Βέβαια, επειδή χρησιμοποιήσαμε την παράμετρο physical στη syn_opt, το Genus έχει ήδη εξάγει αρχεία για το Innovus, τα οποία αφορούν τη φυσική σχεδίαση.

Σημειώνεται τέλος πως κατά τη σύνθεση του κυκλώματος εξάγονται κι άλλα αρχεία που αφορούν τη διαδικασία της σύνθεσης, όπως για παράδειγμα κώδικας Verilog με την mapped εκδοχή του κυκλώματος. Επομένως, δε χρειάζεται να κάνουμε save την εργασία μας στο Genus, αφού μπορούμε απλά να το επαναφέρουμε στην ίδια κατάσταση με τα αρχεία που δημιουργεί.

5 Σχεδίαση κυκλώματος σε φυσικό επίπεδο

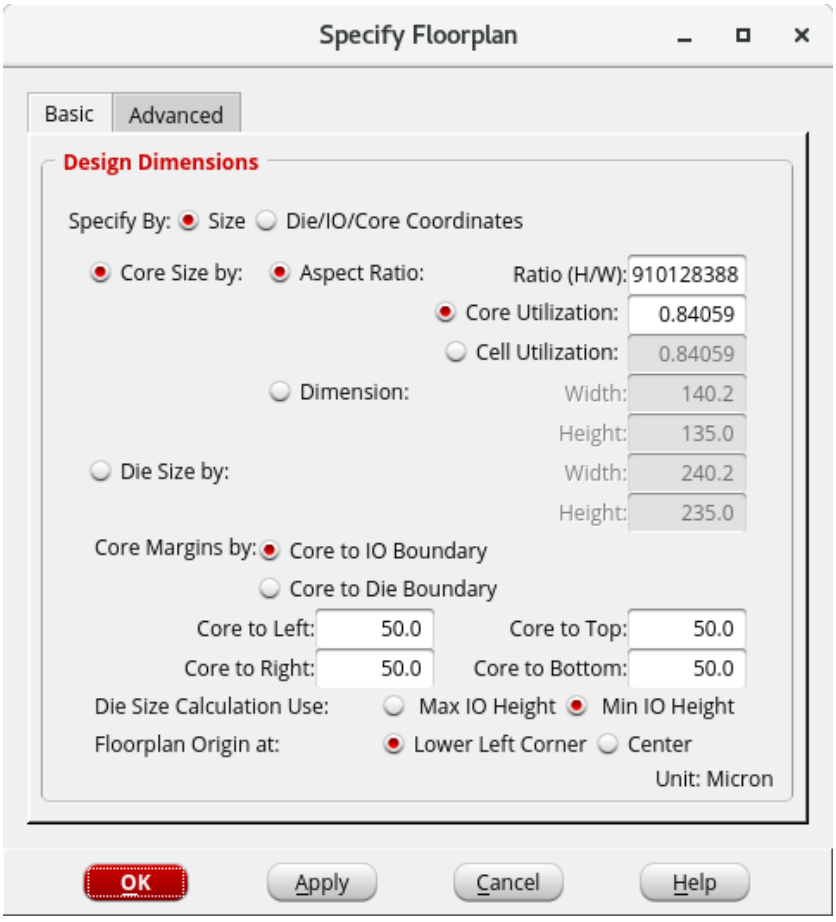
5.1 Έναρξη σχεδίασης – PrePlace

Για τη σχεδίαση σε φυσικό επίπεδο, το Innovus αποτελεί την άμεση συνέχεια του Genus. Έτσι, εκκινούμε το Innovus και φορτώνουμε το TCL script που δημιούργησε αυτομάτως το Genus κατά τη σύνθεση. Σε αυτό περιλαμβάνονται όλες οι απαραίτητες βιβλιοθήκες, καθώς και οι πληροφορίες του κυκλώματος μετά τη σύνθεση. Το μόνο που χρειάζεται να συμπληρώσουμε πριν ξεκινήσει η σχεδίαση, είναι τα καλώδια τροφοδοσίας VDD και VSS. Το πρώτο δηλώνεται στη μεταβλητή init_power_nets και το δεύτερο στην init_ground_nets.

5.1.1 Σχεδίαση διάταξης μονάδων (Floorplanning)

Δεδομένου ότι έχει πραγματοποιηθεί ungrouping των επιμέρους μονάδων του DDS, δε χρειάζεται να υποδείξουμε το σημείο όπου επιθυμούμε να τοποθετηθεί η κάθε μία. Ωστόσο, χρειάζεται να επεκτείνουμε το χώρο του floorplan, ώστε να προσθέσουμε την τροφοδοσία και τα pins του ολοκληρωμένου κυκλώματος.

Για να το πετύχουμε αυτό, επιλέγουμε από το μενού του Innovus το Floorplan → Specify Floorplan… και στο παράθυρο που ανοίγει ορίζουμε τις αποστάσεις από το core στο 50, όπως φαίνεται στην Εικόνα 5-1.

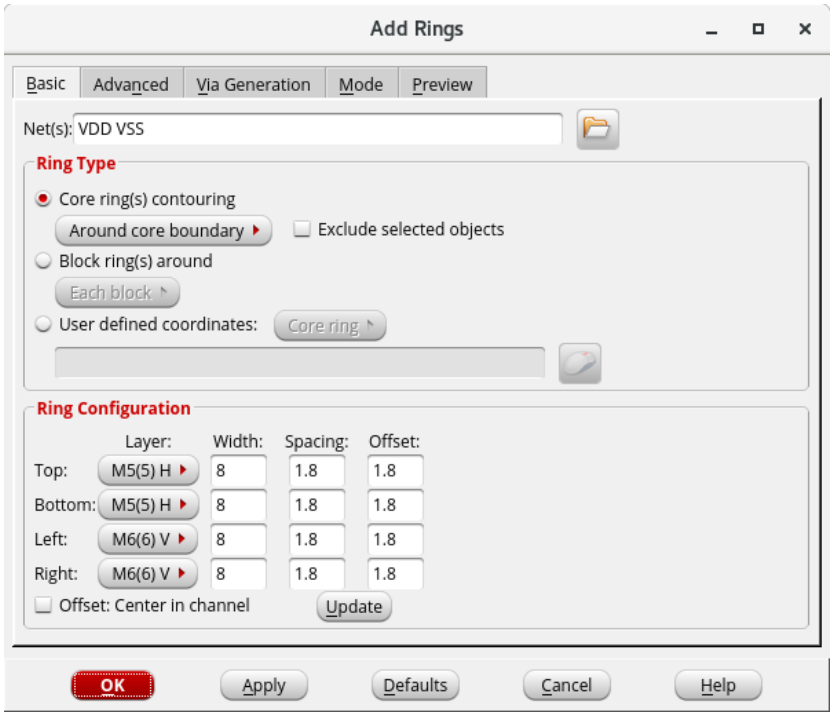


Εικόνα 5-1 Specify Floorplan

5.1.2 Σχεδίαση ενέργειας / τροφοδοσίας (Power planning)

Κατά το power planning, προσπαθούμε να σχεδιάσουμε όσο το δυνατόν πιο αποτελεσματικά την τροφοδοσία του κυκλώματος, ώστε όλα τα στοιχεία να έχουν άμεση και εύκολη πρόσβαση σε αυτή. Το βήμα αυτό είναι αρκετά σημαντικό για την επίτευξη του βέλτιστου placement και routing των στοιχείων.

Αρχικά, δημιουργούμε ένα «δαχτυλίδι» τροφοδοσίας γύρω από τον πυρήνα, επιλέγοντας το Power →Power Planning → Add Ring… όπως φαίνεται στην Εικόνα 5-2. Επιλέγουμε τα καλώδια VDD και VSS που δημιουργήσαμε προηγουμένως. Επίσης, επιλέγουμε το δαχτυλίδι να είναι γύρω από τον πυρήνα και τέλος ορίζουμε τα οριζόντια καλώδια να είναι στο μέταλλο 5, ενώ τα κατακόρυφα στο 6. Όλοι οι αγωγοί ορίζονται να έχουν πλάτος 8.

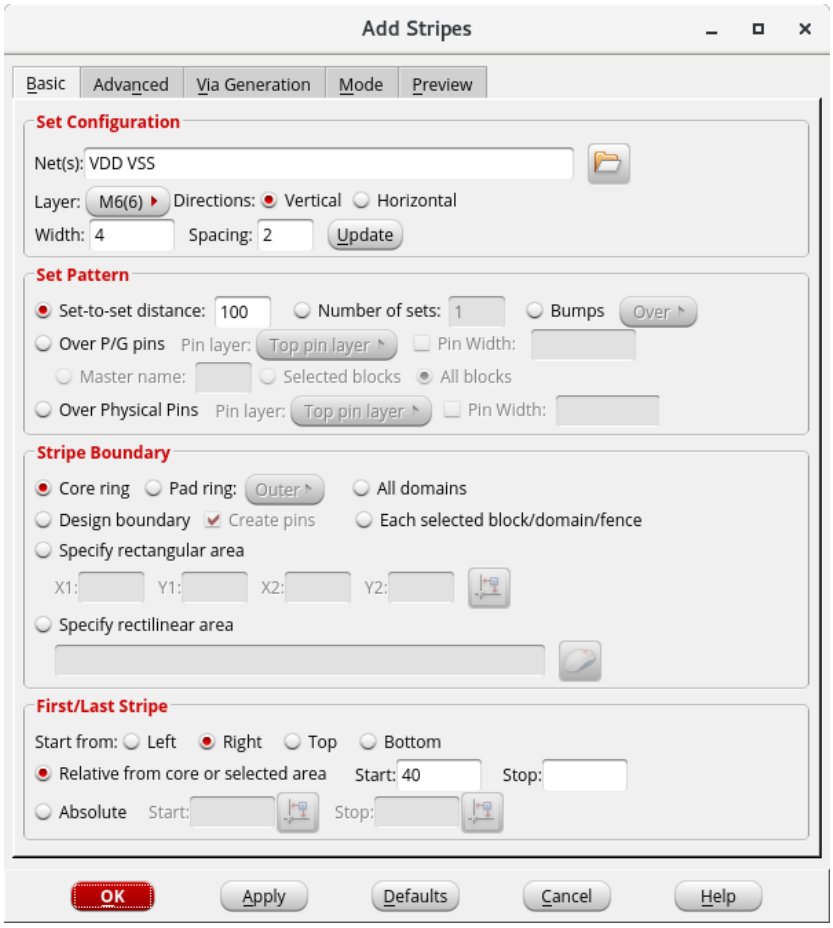


Εικόνα 5-2 Προσθήκη power ring

Συνεχίζοντας με το σχεδιασμό της τροφοδοσίας, επιλέγουμε το Power → Power Planning → Add Stripes… για την εισαγωγή επιπλέον ραβδώσεων κατά μήκος ή/και κατά πλάτος του πυρήνα, ώστε να επεκτείνουμε την τροφοδοσία και στο εσωτερικό του πυρήνα. Θα δημιουργήσουμε λοιπόν, δύο κατακόρυφα stripes. Το κάθε ένα περιέχει δύο αγωγούς: έναν για το VSS και έναν για το VDD. Και πάλι επιλέγουμε το μέταλλο 6 για την τοποθέτησή τους, αλλά αυτή τη φορά ορίζουμε μικρότερο πλάτος στους αγωγούς και ίσο με 4. Τέλος, για την ομοιόμορφη τοποθέτησή τους, υπολογίζουμε τη μεταξύ τους απόσταση (αρχή της μίας με αρχή της επόμενης) σύμφωνα με τον απλό τύπο:

$$w_c = N_s \cdot (2 \cdot w_n + s_n) + (N_s + 1) \cdot w_e \Leftrightarrow \boxed{w_e = \frac{w_c - N_s \cdot (2 \cdot w_n + s_n)}{N_s + 1}} \qquad (6)$$

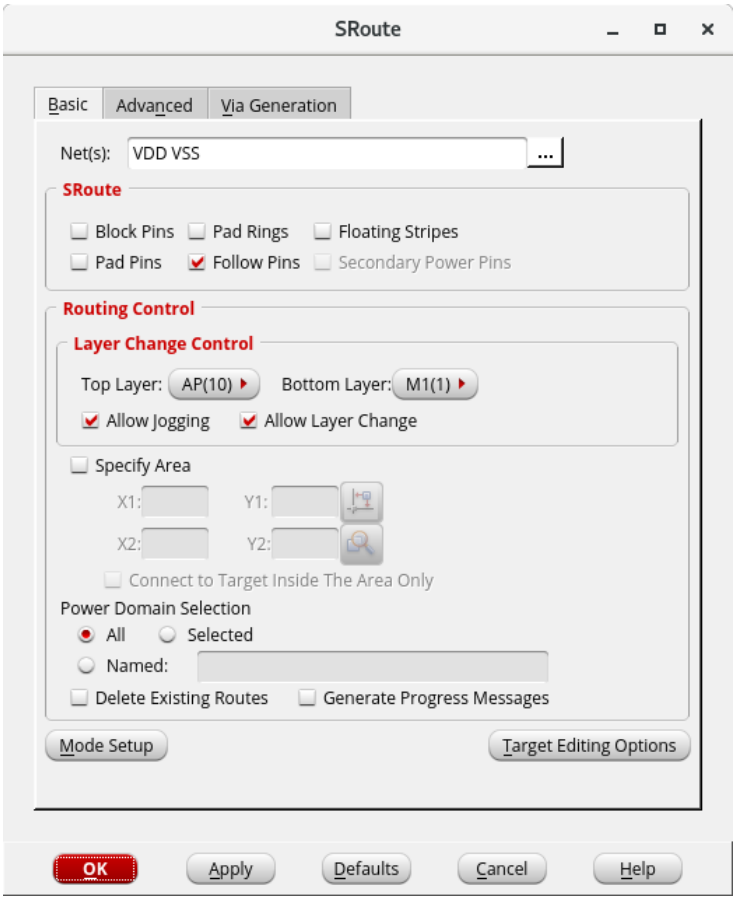
όπου w_c το πλάτος του πυρήνα, w_e το πλάτος των διαστημάτων μεταξύ των stripes, N_s το πλήθος των stripes, w_n και s_n το πλάτος του κάθε αγωγού του stripe και η μεταξύ τους απόσταση, αντίστοιχα.



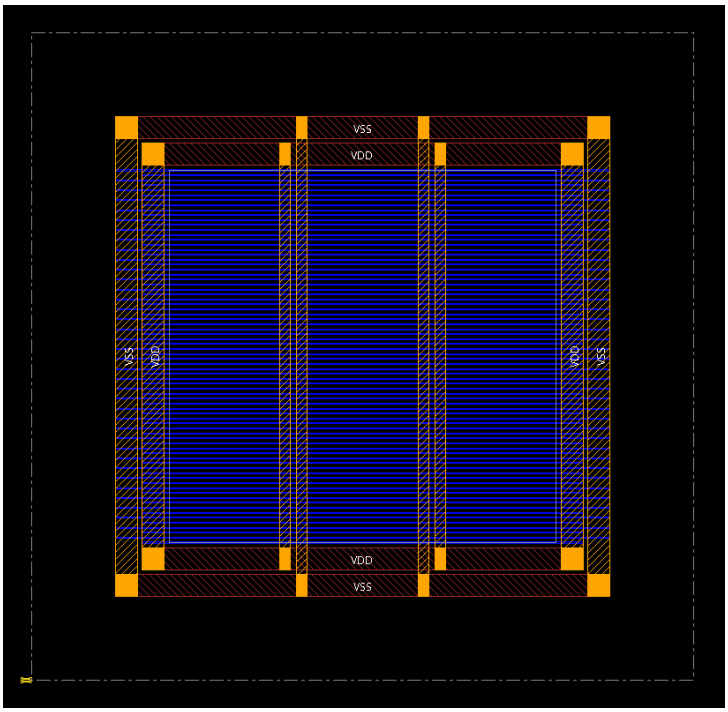
Εικόνα 5-3 Προσθήκη power stripes

Κλείνοντας την προετοιμασία της τροφοδοσίας, θα προσθέσουμε οριζόντιες διασυνδέσεις μεταξύ των αγωγών τροφοδοσίας, αλλά αυτή τη φορά λεπτότερους και στο μέταλλο 1. Έτσι, όλα τα στοιχεία θα τροφοδοτηθούν αμεσότερα και ευκολότερα. Επιλέγουμε λοιπόν το Route →Special Route… και από εκεί το Follow Pins, όπως παρουσιάζεται στην Εικόνα 5-4.

Τα αποτελέσματα της διαδικασίας του power planning παρουσιάζονται στην Εικόνα 5-5. Προτού συνεχίσουμε με τη σχεδίαση, εκτελούμε την εντολή check_floorplan για να επαληθεύσουμε την ορθότητα των προηγούμενων διαδικασιών.



Εικόνα 5-4 Special routing

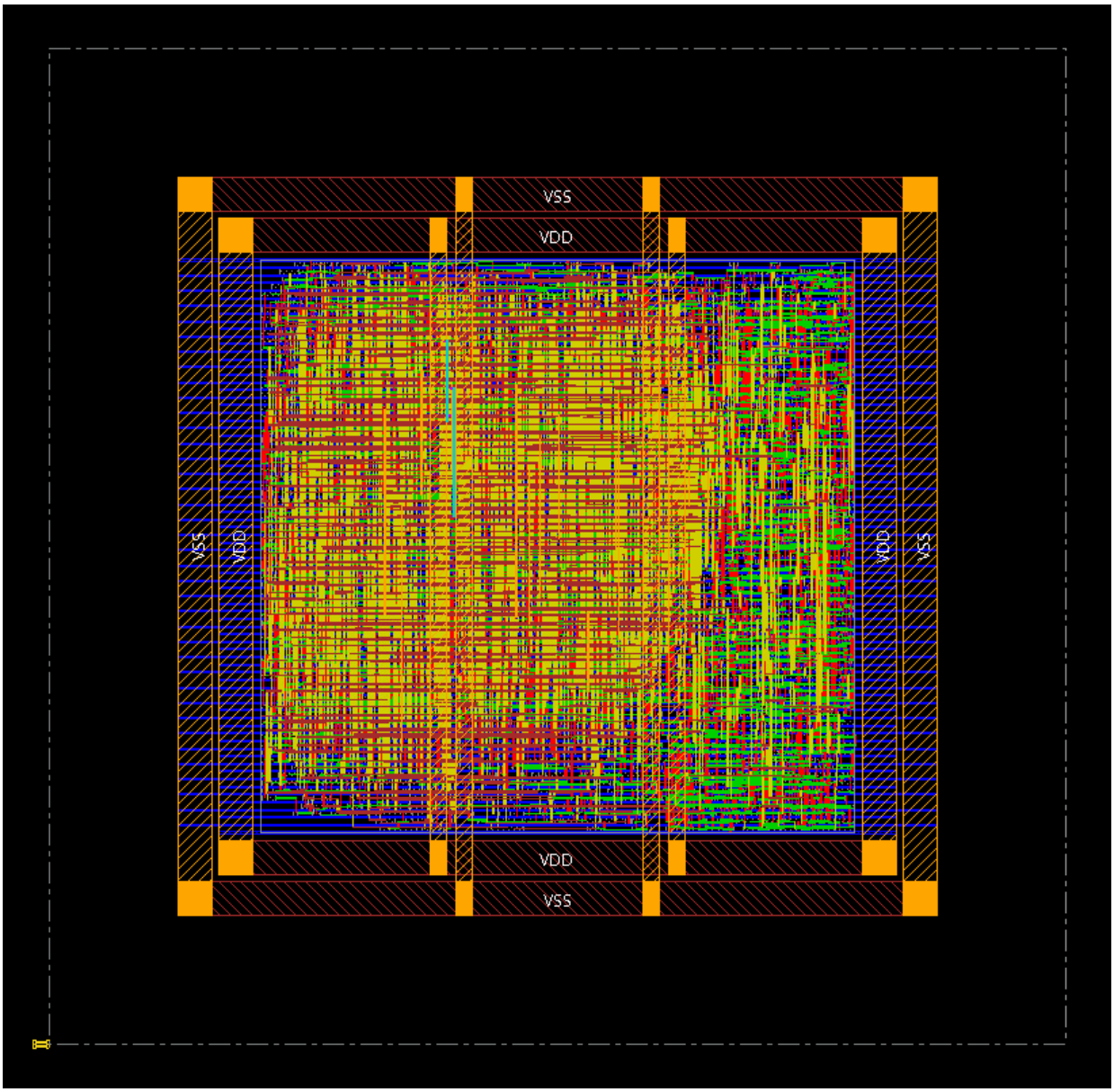


Εικόνα 5-5 Power planning

5.2 Τοποθέτηση (Placement) – PreCTS

Πλέον είναι όλα έτοιμα για να γίνει η τοποθέτηση των στοιχείων στον πυρήνα του ολοκληρωμένου κυκλώματος. Έτσι, εκτελούμε την εντολή place_opt_design για να ξεκινήσει το βέλτιστο placement ή επιλέγουμε το μενού Place → Place Standard Cell… Το Innovus εκτελεί την ίδια διαδικασία με το placement που έκανε το Genus κατά τη syn_opt, επομένως αναμένουμε αντίστοιχα αποτελέσματα.

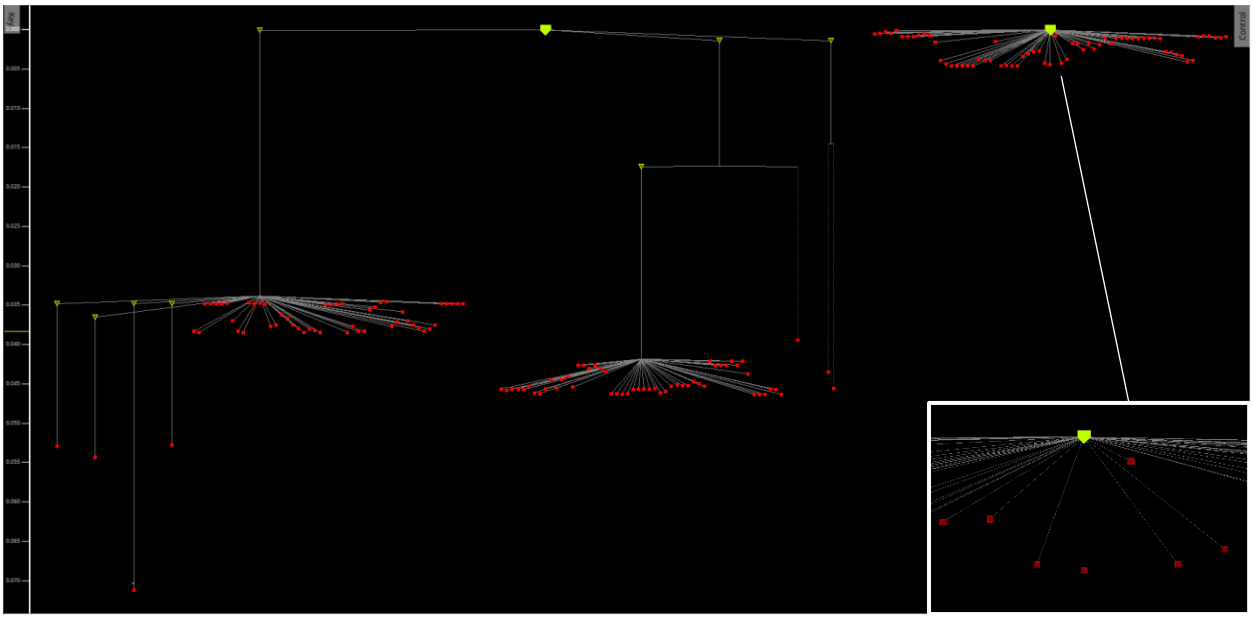
Πράγματι τρέχοντας τη χρονική ανάλυση με την εντολή time_design -pre_cts, οι καθυστερήσεις των μονοπατιών είναι ίδιες. Με την παράμετρο -hold στην προηγούμενη εντολή μπορούμε να δούμε την αντίστοιχη ανάλυση για την κατάσταση hold των στοιχείων του κυκλώματος.



Εικόνα 5-6 Physical view μετά το placement

5.3 Σύνθεση Δέντρου Ρολογιού (Clock Tree Synthesis) – PostCTS

Εκτελώντας την εντολή create_clock_tree_spec δημιουργούνται τα δέντρα για τα δύο ρολόγια του συστήματος, τα οποία μπορούμε να οπτικοποιήσουμε από το μενού Clock →CCOpt Clock Tree Debugger…



Εικόνα 5-7 Clock trees

Στη συνέχεια εκτελούμε την εντολή ccopt_design, η οποία πραγματοποιεί βελτιστοποίηση της τοποθέτησης των στοιχείων με βάση τα ρολόγια του συστήματος (Clock Concurrent Optimization, CCOpt). Ξανατρέχουμε χρονική ανάλυση, αυτή τη φορά με την παράμετρο -post_cts, και παρατηρούμε πως οι καθυστερήσεις των μονοπατιών βελτιώνονται αρκετά. Εάν όμως υπάρχουν ακόμη μονοπάτια με αρνητικά slacks, μπορούμε να ζητήσουμε επιπλέον βελτιστοποίηση με την εντολή opt_design -post_cts -setup -hold.

5.4 Διασύνδεση (Routing) – PostRoute

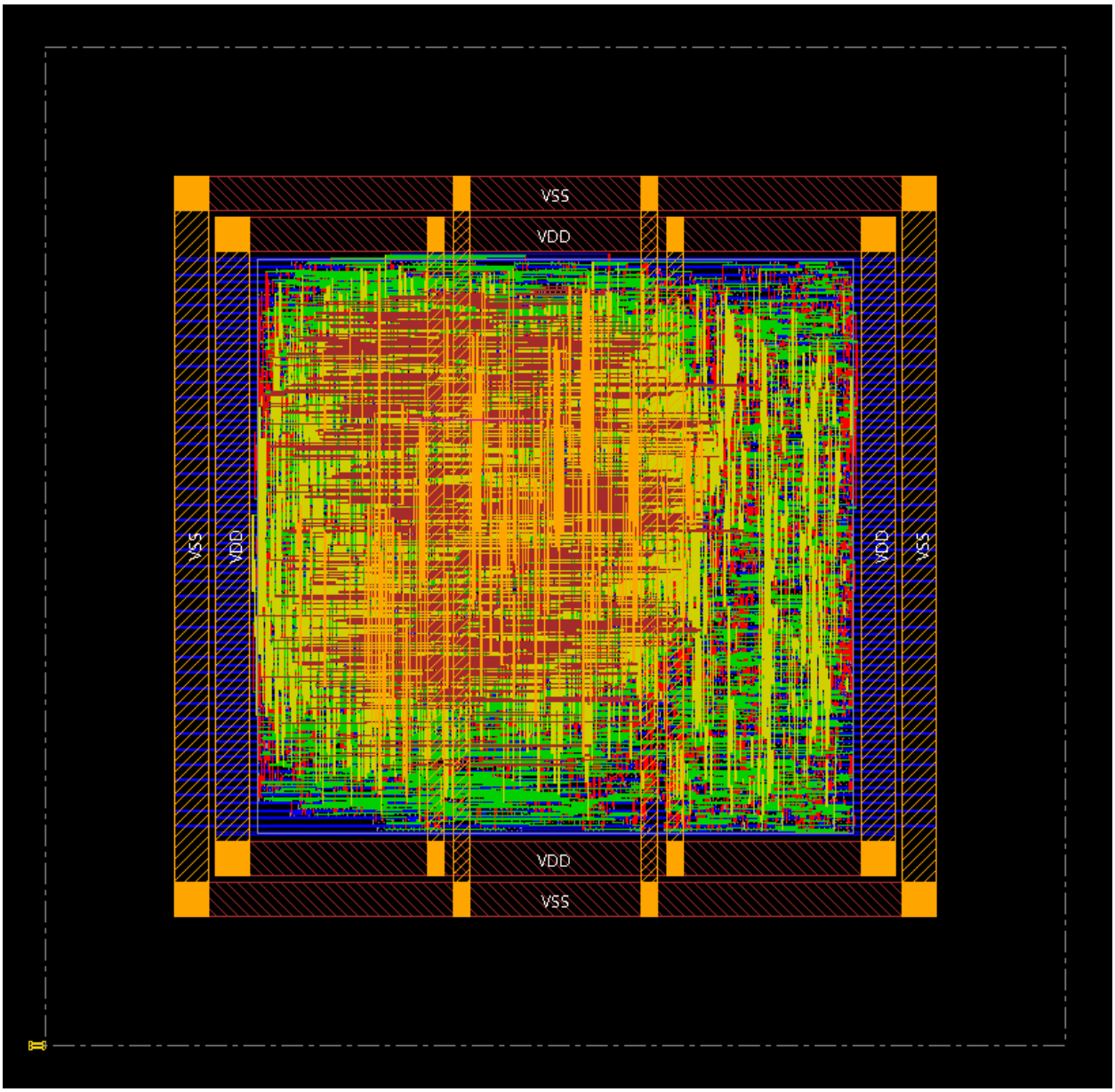
Προτού ξεκινήσει η διασύνδεση των στοιχείων, κάνουμε μερικές παραμετροποιήσεις για καλύτερα αποτελέσματα. Δεδομένου λοιπόν, πως οι αγωγοί των ρολογιών (clk και sclk) λειτουργούν σε πολύ μεγάλη συχνότητα, είναι καλό να τους δώσουμε περισσότερο χώρο σε σχέση με τα υπόλοιπα καλώδια. Με τον τρόπο αυτό, μειώνεται η πιθανότητα για εισαγωγή παρασίτων στο υπόλοιπο κύκλωμα. Έτσι, μέσω του μενού Route → NanoRoute → Specify Attribute… ή με την εντολή set_route_attributes -net clk -preferred_extra_space_tracks 2, ορίζουμε την επιπλέον απόσταση για τους εν λόγω αγωγούς. Ακόμη, θέτουμε στην τιμή medium τη μεταβλητή route_design_detail_use_multi_cut_via_effort.

Επιλέγουμε Route → NanoRoute →Route… για να ξεκινήσει το αναλυτικό routing των στοιχείων του κυκλώματος. Το παράθυρο που ανοίγει διακρίνεται στην Εικόνα 5-8.

Εικόνα 5-8 Routing

Αφού ολοκληρωθεί το routing, πρέπει και πάλι να τρέξουμε χρονική ανάλυση για το κύκλωμα για να σιγουρευτούμε ότι τηρούνται οι περιορισμοί που έχουμε θέσει. Η ανάλυση στο στάδιο αυτό πρέπει να είναι τύπου ocv και έτσι θέτουμε τη μεταβλητή timing_analysis_type σε ocv. Αυτή τη φορά η παράμετρος στην εντολή time_design είναι η -post_route. Εάν υπάρχουν αρνητικά slacks, τρέχουμε την opt_design με παράμετρο επίσης -post_route. Σε περίπτωση που εξακολουθούν να υπάρχουν αρνητικά slacks ακόμα και μετά την τελευταία βελτιστοποίηση, τότε πρέπει να ορίσουμε πιο χαλαρούς περιορισμούς ή να κάνουμε αλλαγές στη σχεδίασή μας, διότι οι αρχικοί περιορισμοί δεν μπορούν να ικανοποιηθούν.

Στην Εικόνα 5-9 παρουσιάζεται η σχεδίαση όπως έχει προκύψει μετά το routing και την post_route βελτιστοποίηση.

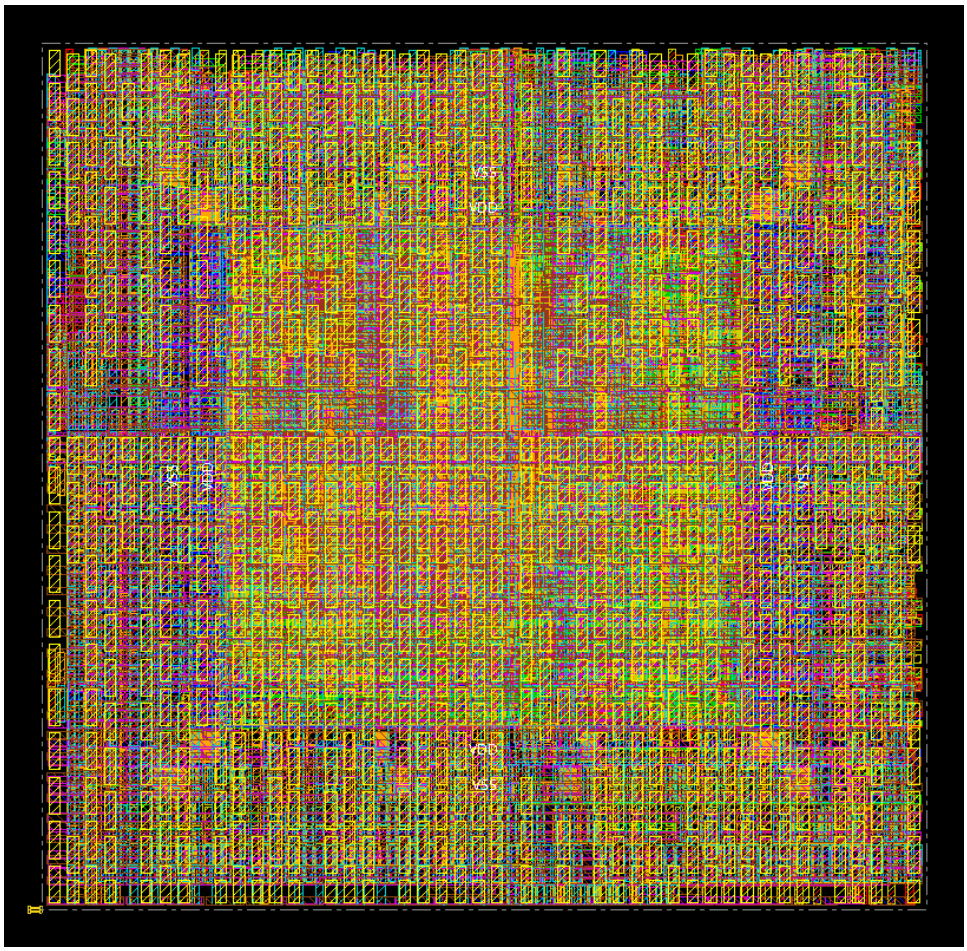


Εικόνα 5-9 Physical view μετά το routing

5.5 Οριστικοποίηση σχεδίασης – SignOff

5.5.1 Συμπλήρωση μετάλλου (Metal fill)

Κάθε στρώση (layer) του chip πρέπει να περιέχει ένα συγκεκριμένο ποσοστό μετάλλου. Επειδή όμως, συνήθως αυτό δεν ικανοποιείται μόνο από τα στοιχεία και τους αγωγούς (π.χ. σε μερικές στρώσεις υπάρχουν λίγα στοιχεία), χρειάζεται να προσθέσουμε επιπλέον στοιχεία που ονομάζονται fillers. Από το μενού Place → Filler → Add Filler… γίνεται η επιλογή των fillers που θέλουμε να χρησιμοποιηθούν. Για να γίνει η συμπλήρωση μετάλλου, επιλέγουμε το Route → Metal Fill --> Add… Στο παράθυρο που ανοίγει έχουμε τη δυνατότητα να ρυθμίσουμε διάφορες παραμέτρους, όπως για παράδειγμα σε ποια layers επιθυμούμε να γίνει το fill. Τα αποτελέσματα της διαδικασίας παρουσιάζονται στην παρακάτω εικόνα.



Εικόνα 5-10 Metal fill
Τα ορθογώνια στοιχεία που διακρίνονται αποτελούν τα fillers, τα οποία τοποθετούνται σε όλη την επιφάνεια εκτός και εντός του πυρήνα.

5.5.2 Επαλήθευση (Verification)

Ολοκληρώνοντας, χρειάζεται να επιβεβαιώσουμε την ορθότητα της σχεδίασης. Επιλέγουμε λοιπόν από το μενού Check, τα παρακάτω verifications:

i. Geometry
ii. Metal Density
iii. Connectivity

Εάν προκύψουν σφάλματα (violations) σε κάποιο έλεγχο, είναι διαθέσιμα στο Tools → Violation Browser… με τις κατάλληλες πληροφορίες, ώστε να προβούμε στη διόρθωσή τους. Αν όλα είναι εντάξει, τότε η σχεδίαση έχει φτάσει στο τέλος της. Προτού το ολοκληρωμένο κύκλωμα είναι έτοιμο για παραγωγή, συνήθως λαμβάνουν χώρα επιπλέον έλεγχοι διαφόρων τύπων, ώστε να μειωθούν οι πιθανότητες να προκύψουν προβλήματα κατά τη λειτουργία του chip σε πραγματικές συνθήκες. Ένας από τους ελέγχους αυτούς είναι το IR drop κατά το οποίο χαρτογραφείται η παραγωγή θερμότητας ανά περιοχή του πυρήνα.

6 Επίλογος

Με το πέρας της παρούσας διπλωματικής εργασίας, σχεδιάστηκε και προσομοιώθηκε επιτυχώς ο Άμεσος Ψηφιακός Συνθέτης όπως περιεγράφηκε αρχικά με όλες τις προδιαγραφές που ορίστηκαν. Όλα τα επιμέρους τμήματα του DDS λειτουργούν σύμφωνα με το επιθυμητό, ώστε να επιτευχθεί η λειτουργικότητα του συνολικού συστήματος.

Η σχεδίαση έγινε σε όλα τα στάδια σύμφωνα με τις πιο πρόσφατες τεχνικές και τα νεότερα λογισμικά σχεδίασης, ώστε να ανταποκρίνεται στις σύγχρονες σχεδιαστικές και κατασκευαστικές ανάγκες.

Επιπρόσθετα και αφού προηγηθεί ανάλυση του κυκλώματος σε καταστάσεις και συνθήκες πραγματικής λειτουργίας (για παράδειγμα ανάλυση θερμοκρασίας), το ολοκληρωμένο κύκλωμα του DDS θα είναι έτοιμο για παραγωγή σε ASIC chips και στη συνέχεια για κανονική χρήση σε οποιαδήποτε επιθυμητή εφαρμογή.

6.1 Μελλοντικές προσθήκες

Για την πληρότητα του DDS, προτείνεται η ενσωμάτωση ενός RF D/A Converter αμέσως μετά τη μονάδα του NCO, ώστε το παραγόμενο σήμα να μετατρέπεται σε αναλογικό εντός του ολοκληρωμένου κυκλώματος.

Ακόμη, το phase accumulator μπορεί να εμπλουτιστεί με περισσότερα ζεύγη phase και frequency registers, ώστε το DDS να μπορεί να προγραμματίζεται ταυτόχρονα για περισσότερα operation profiles. Με τον τρόπο αυτό, η εναλλαγή μεταξύ των διαφόρων προφίλ λειτουργίας θα μπορούσε να συμβαίνει πολύ γρήγορα και χωρίς την ανάγκη για επαναπρογραμματισμό.

Εκτός από την παραγωγή ημιτονοειδών σημάτων, το DDS θα μπορούσε να τροποποιηθεί κατάλληλα για την υποστήριξη διαφορετικών τύπων σημάτων στην έξοδό του. Με τον κατάλληλο προγραμματισμό θα μπορούσε να εξάγει τετραγωνικούς, τριγωνικούς, πριονωτούς κ.α. παλμούς. Εάν δε τροποποιηθεί και η μνήμη του PAC, ώστε να υποστηρίζει δυνατότητες εγγραφής, τότε το DDS θα είναι σε θέση να αναπαράγει οποιοδήποτε περιοδικό σήμα. Βέβαια, όλες οι παραπάνω τροποποιήσεις έρχονται με κόστος την ταχύτητα, καθώς μία RAM ή μία μνήμη Flash για παράδειγμα είναι αρκετά πιο αργές.

Τέλος, θα μπορούσε να ενσωματωθεί και ένα PLL στο DDS, ώστε το ρολόι να μπορεί να παράγεται εσωτερικά για ακόμα μικρότερες καθυστερήσεις και επομένως καλύτερες επιδόσεις.

Παράρτημα Α – Αρχεία Verilog

Α1. Κώδικας SPI

```
module spi(sclk, mosi, miso, ss, wordin, wordout, load, br);
parameter B = 8;

input sclk, mosi, ss, load;
input [B-1:0] wordin;
output miso, br;
output [B-1:0] wordout;

reg [B-1:0] buffer;
reg [$clog2(B)-1:0] counter;

assign br      = !ss ? !counter : 'bz;
assign wordout = !ss ? buffer : 'bz;
assign miso    = !ss ? buffer[0] : 'bz;

always @(posedge sclk, posedge ss) begin
    if (ss) begin
        counter <= 'b0;
    end
    else begin
        if (load && !counter)
            buffer <= wordin;
        else
            buffer <= {mosi, buffer[B-1:1]};

        counter <= counter + 1;
    end
end
endmodule
```

A2. Κώδικας PA

```
module phase_accumulator(clk, reset, mode, fph, word, phase);
    parameter M = 48;
    parameter N = 14;

    input clk, reset, fph;
    input [1:0] mode;
    inout [M-1:0] word;
    output [N-1:0] phase;

    reg [M-1:0] freq, acc;

    assign phase = acc[M-1:M-N];
    assign word  = mode == 2'b10 ? (fph ? freq : acc) : 'bz;

    always @ (posedge clk, posedge reset)
    begin
        if (reset) begin
            freq  <= 'b0;
            acc   <= 'b0;
        end
        else begin
            if (!mode)
                acc <= acc + freq;
            else if (mode == 2'b11) begin
                if (fph)
                    freq <= word;
                else
                    acc  <= word;
            end
        end
    end
endmodule
```

A3. Κώδικας OPC

```
`include "serial_peripheral_interface/spi.v"
`include "phase_accumulator/phase_accumulator.v"

module opc(clk, sclk, reset, mosi, miso, ss, br, outr, phase);
    parameter M = 48;
    parameter N = 14;
    parameter B = 8;

    input clk, sclk, reset, mosi, ss;
    output miso, br, outr;
    output [N-1:0] phase;

    reg fph;
    reg [1:0] mode;
    reg [M-1:0] buffer;
    reg [$clog2(M/B+2)-1:0] counter;

    wire [B-1:0] word_read;
    wire [M-1:0] pa_word;

    assign pa_word = mode == 2'b10 || counter != 3'b111 ? 'bz: buffer;
    assign outr    = !mode;

    spi #(.B(B)) U1_spi (
        .sclk       (sclk),
        .mosi       (mosi),
        .miso       (miso),
        .ss         (ss),
        .wordin     (buffer[B-1:0]),
        .wordout    (word_read),
        .load       (mode == 2'b10),
        .br         (br)
    );
```

```
phase_accumulator #(.M(M), .N(N)) U2_pa (
    .clk        (clk),
    .reset      (reset),
    .mode       (~(~mode)),
    .fph        (fph),
    .word       (pa_word),
    .phase      (phase)
);

always @(posedge sclk, posedge reset) begin
    if (reset) begin
        fph     = 1'b0;
        mode    = 2'b01;
        buffer  =  'b0;
        counter = 3'b111;
    end
    else begin
        if (!ss) begin
            if (br) begin
                counter = counter + 1;
                if (counter == 3'b000) begin
                    mode = 2'b01;
                end
                else if (counter == 3'b001) begin
                 mode[1] = word_read[7];
                 mode[0] = word_read[7]?word_read[6]:word_read[4];
                 fph = word_read[5];
                end
                else begin
                    if (mode[1]) begin
                        buffer = buffer >> B;
                        if (mode[0]) begin
                            buffer[M-1:M-B] = word_read;
                        end
                    end
                end
```

```
                end

                if (counter == 3'b001 && mode == 2'b10) begin
                    buffer = pa_word;
                end
            end
            else begin
                counter = 3'b111;
            end
        end
    end
endmodule
```

A4. Κώδικας PAC

```
module pac(clk, eo, phase, amplitude);
parameter N = 14;
parameter P = 12;

input clk, eo;
input [N-1:0] phase;
output [P-1:0] amplitude;

reg eos;
reg [P-1:0] data;
wire [1:0] quadrant;

assign amplitude = eos ? data : 'bz;
assign quadrant  = phase[N-1:N-2];

always @(posedge clk) begin
    eos = eo;
    address = quadrant[0] ? ~phase[N-3:0] : phase[N-3:0];
    case (address)
         0 : data = quadrant[1] ? 12'b011111111111 : 12'b100000000000;
         1 : data = quadrant[1] ? 12'b011111111111 : 12'b100000000000;
         2 : data = quadrant[1] ? 12'b011111111110 : 12'b100000000001;
                                  ⋮
       382 : data = quadrant[1] ? 12'b011011010101 : 12'b100100101010;
                                  ⋮
      2550 : data = quadrant[1] ? 12'b000101011101 : 12'b111010100010;
                                  ⋮
      2779 : data = quadrant[1] ? 12'b000100000000 : 12'b111011111111;
                                  ⋮
      4093 : data = quadrant[1] ? 12'b000000000000 : 12'b111111111111;
      4094 : data = quadrant[1] ? 12'b000000000000 : 12'b111111111111;
      4095 : data = quadrant[1] ? 12'b000000000000 : 12'b111111111111;
     endcase
end
endmodule
```

A5. Κώδικας NCO

```
`include "operation_profile_configurator/opc.v"
`include "phase_to_amplitude_converter/pac.v"

module nco(clk, sclk, reset, mosi, miso, ss, br, amplitude);
    parameter M = 48;
    parameter N = 14;
    parameter P = 12;
    parameter B = 8;

    input clk, sclk, reset, mosi, ss;
    output miso, br;
    output [P-1:0] amplitude;

    wire outr;
    wire [N-1:0] phase;

    opc #(.M(M), .N(N), .B(B)) U3_opc (
        .clk        (clk),
        .sclk       (sclk),
        .reset      (reset),
        .mosi       (mosi),
        .miso       (miso),
        .ss         (ss),
        .br         (br),
        .outr       (outr),
        .phase      (phase)
    );

    pac #(.N(N), .P(P)) U4_pac (
        .clk        (clk),
        .eo         (!reset && outr),
        .phase      (phase),
        .amplitude  (amplitude)
    );
endmodule
```

Παράρτημα Β – Αρχεία Testbenches

B1. Testbench SPI

```
`include "serial_peripheral_interface/spi.v"
`timescale 1ns/100ps

module spi_tb;
    reg sclk, mosi, ss, load;
    reg [7:0] temp;
    wire miso, br;
    wire [7:0] word;

    spi #(.B(8)) U1_spi_tb (
        .sclk       (sclk),
        .mosi       (mosi),
        .miso       (miso),
        .ss         (ss),
        .wordin     (temp),
        .wordout    (word),
        .load       (load),
        .br         (br)
    );

    initial begin
        $dumpfile ("serial_peripheral_interface/output/spi.vcd");

        sclk = 0;
        mosi = 0;
        ss   = 1;
        temp = 0;
        load = 0;

        $dumpvars;
        #36 $finish;
    end
```

```
    always begin
        #1 sclk = ~sclk;
    end

    initial begin
        #0.5 ss   = 0;

        #0.5 mosi = 1;
        #2   mosi = 0;
        #2   mosi = 0;
        #2   mosi = 1;
        #2   mosi = 1;
        #2   mosi = 1;
        #2   mosi = 0;
        #2   mosi = 1;

        #2   load = 1;
             temp = 'b10010101;
             mosi = 0;

        #2   load = 0;
             temp = 'b0;

        #14   load = 1;
             temp = 'b10000000;

        #2   load = 0;
             temp = 'b0;
    end
endmodule
```

B2. Testbench PA

```
`include "phase_accumulator/phase_accumulator.v"
`timescale 1ns/100ps

module phase_accumulator_tb;
    reg clk, reset, fph;
    reg [1:0] mode;
    reg [4:0] temp;

    wire [3:0] phase;
    wire [4:0] word;

    assign word = mode == 2'b10 ? 'bz : temp;

    phase_accumulator #(.M(5), .N(4)) U2_pa_tb (
        .clk    (clk),
        .reset  (reset),
        .mode   (mode),
        .fph    (fph),
        .word   (word),
        .phase  (phase)
    );

    initial begin
        $dumpfile ("phase_accumulator/output/phase_accumulator.vcd");

        clk   = 0;
        reset = 0;
        temp  = 5;
        mode  = 'b0;
        fph   = 0;

        $dumpvars;
        #9 $finish;
    end
```

```
    always begin
        #0.5 clk = ~clk;
    end

    initial begin
        #0.6 reset = 1;
        #0.4 reset = 0;

        #0.5 mode  = 2'b11;
             fph   = 1;

        #1   fph   = 0;
             temp  = 2;

        #1   mode  = 2'b10;
             fph   = 1;
             temp  = 'b0;

        #1   mode  = 'b0;
             temp  = 1;

        #2   mode  = 2'b01;

        #1   mode  = 'b0;
    end
endmodule
```

B3. Testbench OPC

```
`include "operation_profile_configurator/opc.v"
`timescale 1ns/100ps

module opc_tb;
    reg clk, sclk, reset, mosi, ss;
    wire miso, br, outr;
    wire [13:0] phase;

    opc #(.M(48), .N(14), .B(8)) U3_opc_tb (
        .clk        (clk),
        .sclk       (sclk),
        .reset      (reset),
        .mosi       (mosi),
        .miso       (miso),
        .ss         (ss),
        .br         (br),
        .outr       (outr),
        .phase      (phase)
    );

    initial begin
        $dumpfile ("operation_profile_configurator/output/opc.vcd");

        clk     = 0;
        sclk    = 0;
        reset   = 0;
        mosi    = 0;
        ss      = 1;

        $dumpvars;
        $finish;
    end

    always begin
        #0.5 clk = ~clk;
    end
```

```
always begin
    #1 sclk = ~sclk;
end

initial begin
    #0.2 reset  = 1;
    #0.3 reset  = 0;
         ss     = 0;

    //Instruction Byte 1
    //Write to freq
    #0.5 mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 1; #2 mosi = 1; #2 mosi = 1;
    //Frequency
    #2   mosi = 1; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 1; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    //Sync Byte
    #2 mosi  = 0;
    #14
```

```
        //Instruction Byte 2
        //Write to acc
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 1; #2 mosi = 1;
        //Phase Offset
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 1; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 1; #2 mosi = 0; #2 mosi = 1; #2 mosi = 1;
        #2   mosi = 1; #2 mosi = 1; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 0; #2 mosi = 1; #2 mosi = 1; #2 mosi = 1;
        #2   mosi = 0; #2 mosi = 1; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 1;

        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        //Sync Byte
        #2 mosi  = 0;
        #14
        //Instruction Byte 3
        //Read from freq
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 1; #2 mosi = 0; #2 mosi = 1;
        //Sync Byte
        //Reads garbage
        #16
        //Data read
        #96 ;
    end
endmodule
```

B4. Testbench PAC

```
`include "phase_to_amplitude_converter/pac.v"
`timescale 1ns/100ps

module pac_tb;
    reg clk, eo;
    reg [13:0] phase;
    wire [11:0] amplitude;

    pac #(.N(14), .P(12)) U4_pac_tb (
        .clk        (clk),
        .eo         (eo),
        .phase      (phase),
        .amplitude  (amplitude)
    );

    initial begin
        $dumpfile ("phase_to_amplitude_converter/output/pac.vcd");
        clk = 0; eo = 0; phase = 0;
        $dumpvars;
        #9 $finish;
    end

    always begin
        #0.5 clk = ~clk;
    end

    initial begin
        #1   eo    = 1;
        #1.5 phase = 4096;
        #1   phase = 8192;
        #1   phase = 12288;
        repeat (3) begin
            #1 phase = $urandom % (1<<14);
        end
    end
endmodule
```

B5. Testbench NCO

```
`include "numerically_controlled_oscillator/nco.v"
`timescale 1ns/100ps

module nco_tb;
    reg clk, sclk, reset, mosi, ss;
    wire miso, br;
    wire [11:0] amplitude;

    nco #(.M(48), .N(14), .P(12), .B(8)) U5_nco_tb (
        .clk        (clk),
        .sclk       (sclk),
        .reset      (reset),
        .mosi       (mosi),
        .miso       (miso),
        .ss         (ss),
        .br         (br),
        .amplitude  (amplitude)
    );

    initial begin
      $dumpfile ("numerically_controlled_oscillator/output/nco.vcd");
      clk = 1; sclk = 0; reset= 0; ss = 1; mosi = 0;
      fork
          #270 $dumpvars;
          #305 $finish;
      join
    end

    always begin
        #0.5 clk   = !clk;
    end

    always begin
        #1 sclk   = !sclk;
    end
```

```
initial begin
    #0.2 reset = 1;
    #0.3 reset = 0;
         ss    = 0;
    //Instruction Byte 1
    //Write to freq
    #0.5 mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 1; #2 mosi = 1; #2 mosi = 1;
    //Frequency
    #2   mosi = 1; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

    #2   mosi = 1; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    //Sync Byte
    #2 mosi  = 0;
    #14
    //Instruction Byte 2
    //Write to acc
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 1; #2 mosi = 1;
    //Phase Offset
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 1; #2 mosi = 0;
    #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
```

```
        #2   mosi = 1; #2 mosi = 0; #2 mosi = 1; #2 mosi = 1;
        #2   mosi = 1; #2 mosi = 1; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 0; #2 mosi = 1; #2 mosi = 1; #2 mosi = 1;
        #2   mosi = 0; #2 mosi = 1; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 1;

        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        //Sync Byte
        #2 mosi  = 0;
        #14
        //Instruction Byte 3
        //Normal operation
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 1;
        #2   mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

        #4 ss = 1;
        #10 ss = 0;
        //Instruction Byte 4
        //Halt operation
             mosi = 0; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;
        #2   mosi = 1; #2 mosi = 0; #2 mosi = 0; #2 mosi = 0;

        #4 ss = 1;
    end
endmodule
```

ΠΑΡΑΡΤΗΜΑ Γ – Αρχεία Script

Γ1. TCL Script – Genus

```
#Initialization
set_db lib_search_path /mnt/apps/prebuilt/eda/designkits/TSMC_65nm/N65RF
      /TSMCHOME/digital/Front_End/timing_power_noise/NLDM/tcbn65gplus_200a
set_db library tcbn65gplusbc.lib
set_db lef_library /mnt/apps/prebuilt/eda/designkits/TSMC_65nm/N65RF/TSMCHOME
      /digital/Back_End/lef/tcbn65gplus_200a/lef/tcbn65gplus_9lmT2.lef
set_db auto_ungroup both
cd ~/dds/work/genus

#Elaboration
set_db init_hdl_search_path ~/dds/Code
set_db hdl_language v2001
read_hdl numerically_controlled_oscillator/nco.v
ela

#Timing Constraints
define_clock -period 1000000 -divide_period 2800 -name main_clk {clk}
define_clock -period 1000000 -divide_period   25 -name serial_clk {sclk}
external_delay -output 10 -clock main_clk [all_outputs]
external_delay -input 2 -clock main_clk [all_inputs]
set_driving_cell -cell DFD4 -pin Q [all_inputs -no_clocks]
set_db [all_outputs] .external_pin_cap 1

#Synthesis
syn_gen
syn_map
syn_opt -physical

#Reports
cd ~/dds/Reports/Synthesis
report timing > timing.txt
report design > design.txt
report area > area.txt
report summary > summary.txt
report gates > gates.txt
report clocks > clock.txt
report clocks -ideal > clock_ideal.txt
report clocks -generated > clock_generated.txt
report nets > nets.txt
report power > power.txt

#Exports for Physical Design
cd ~/dds/Exports/Synthesis
write -mapped > nco_map.v
write_script > nco_constr.g
write_sdf > nco.sdf
write_sdc > nco.sdc
cd ~/dds/Exports/Synthesis/Innovus
write_encounter design -basename "nco" -lef "/mnt/apps/prebuilt/eda/
      designkits/TSMC_65nm/N65RF/TSMCHOME/digital/Back_End/lef/tcbn65gplus_20
      0a/lef/tcbn65gplus_9lmT2.lef"

#Open GUI
gui_show
```

Γ2. MATLAB Script – Μνήμη PAC

```
%%%%% NCO LUT Calculation %%%%%
%% Initialization
clear

M = 48;
N = 14;
P = 12;

%% Create discrete values
pac = sinpi( 0.5 * (0 : 2^-(N-2) : 1 - 2^-(N-2)) );
pac = (pac + 1) * (2^P-1) / 2;
pac = round(pac);

figure
plot(0:2^N-1, [pac, pac(end:-1:1), 2^P-1 - pac, 2^P-1 - pac(end:-1:1)]);

%% Convert values to binary
pac = de2bi(pac, 'left-msb');

file = fopen('sine_4_test.txt', 'w');

format_bin = '%d';
for i = 1:P-1
    format_bin = strcat(format_bin, '%d');
end

% Export for Verilog
% format = strcat("%d : data = quadrant[1] ? %d'b", format_bin);
% format = strcat(format, " : %d'b");
% format = strcat(format, format_bin);
% format = strcat(format, ";\n");

% Export of simple values
format = strcat(format_bin, "\n");

for i = 1:2^(N-2)
%   Print Verilog cases
%   fprintf(file, format, i-1, P, ~pac(i, :), P, pac(i, :));

%   Print simple values
    fprintf(file, format, pac(i, :));
end
fclose(file);

%% Recalculate original values
dac = bi2de(pac, 'left-msb') * 2 / (2^P-1) - 1;

figure
plot(0:2^N-1, [dac; dac(end:-1:1); - dac; - dac(end:-1:1)]);
```

Βιβλιογραφία

[B1] N. H. E. Weste, D. M. Harris, «Σχεδίαση Ολοκληρωμένων Κυκλωμάτων CMOS VLSI» 4η έκδοση, Εκδόσεις Παπασωτηρίου, 2011

[B2] Σ. Ι. Σουραβλάς, Μάνος Ρουμελιώτης, «Ψηφιακά Συστήματα, Μοντελοποίηση & Προσομοίωση με τη γλώσσα VHDL», Εκδόσεις Τζιόλα, 2012

[B3] Ν. Ι. Μάργαρης, «Μη γραμμική θεωρία του αναλογικού PLL», Εκδόσεις Τζιόλα, Αύγουστος 2000

[B4] E. Murphy, C. Slattery, "All About Direct Digital Synthesis", Analog Dialogue 38-08, August 2004

[B5] Analog Devices, "AD9914 Data Sheet: 3.5 GSPS Direct Digital Synthesizer with 12-Bit DAC", 2012

[B6] Λ. Κατσέλας, «Σχεδίαση Ψηφιακού Κυκλώματος Συστήματος Συγκομιδής Ενέργειας», διπλωματική εργασία τμήματος ηλεκτρολόγων μηχανικών και μηχανικών υπολογιστών, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2014

[B7] Cadence, "Genus User Guide", 2019

[B8] Cadence, "Innovus User Guide", 2019

[B9] T. Finateu, F. Badets, Y. Deval, JB Begueret, D. Belot, "A 65nm CMOS 2.4 GHz Phase Shifter based Direct Digital Synthesizer", IEEE 11th International Conference on Solid-State and Integrated Circuit Technology, October 2012

[B10] H. Omran, K. Sharaf, M. Ibrahim, "An All-Digital Direct Digital Synthesizer Fully Implemented on FPGA", 4th International Design and Test Workshop (IDT), November 2009

[B11] R. O. R. Cardoso, J. A. J. Ribeiro, M. Silveira, "Direct Digital Synthesizer Using FPGA", Global Congress on Engineering and Technology Education, March 2005

Διαδικτυακές πηγές

[Δ1] en.wikipedia.org/wiki/Moore%27s_law

[Δ2] el.wikipedia.org/wiki/VHDL

[Δ3] en.wikipedia.org/wiki/Verilog

[Δ4] en.wikipedia.org/wiki/NCSim

[Δ5] www.edaboard.com/showthread.php?341462-moved-RTL-Compiler-elaboration-command

[Δ6] en.wikipedia.org/wiki/AND-OR-Invert

[Δ7] en.wikipedia.org/wiki/Numerically_controlled_oscillator

[Δ8] en.wikipedia.org/wiki/Serial_Peripheral_Interface

[Δ9] only-vlsi.blogspot.com/2008/04/setup-and-hold-time.html?m=1