Πρόγραμμα Μεταπτυχιακών Σπουδών στις Σύγχρονες Ηλεκτρονικές Τεχνολογίες

Τμήμα Φυσικής
Σχολη Θετικων Επιστημών
Πανεπιστημιο Ιωαννίνων

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ (Μ.Δ.Ε.)

Υλοποίηση γεννήτριας κυματομορφών βασισμένη σε συστοιχία πύλης προγραμματίσιμη από πεδίο

ΠΑΝΑΓΙΩΤΗΣ ΒΟΥΓΕΛΛΗΣ
Αριθμός Μητρώου: 744
Επιβλέπων Καθηγητής: Χριστοφιλάκης Βασίλειος

Ιωάννινα
2024

Στην οικογένειά μου

Περίληψη

Η παρούσα Μεταπτυχιακή Διπλωματική Εργασία, με τίτλο «Υλοποίηση γεννήτριας κυματομορφών βασισμένη σε συστοιχία πύλης προγραμματίσιμης από πεδίο», εκπονήθηκε υπό την επίβλεψη του Επίκουρου Καθηγητή κ. Β. Χριστοφιλάκη, στα πλαίσια του Προγράμματος Μεταπτυχιακών Σπουδών «Σύγχρονες Ηλεκτρονικές Τεχνολογίες», στο Τμήμα Φυσικής της Σχολής Θετικών Επιστημών του Πανεπιστημίου Ιωαννίνων.

Πρωταρχικός στόχος της είναι η σχεδίαση μιας γεννήτριας που θα παράγει προκαθορισμένες κυματομορφές χρησιμοποιώντας ψηφιακή λογική. Η σχεδίαση γίνεται βάσει σχηματικού διαγραμμάτος καθώς επίσης και με χρήση της VHDL στο λογισμικό Intel Quartus Prime Lite. Έπειτα, στόχος είναι η υπολοποίηση αυτού στην πλακέτα DE1 - SoC και πιο συγκεκριμένα στη Cyclone V FPGA. Αντικειμενικός σκοπός είναι η παραγωγή τέσσερων διαφορετικών κυματομορφών (τετραγωνική, ημιτονοειδής, τριγωνική και πριονωτή) σε δύο διαφορετικές συχνότητες ($1Hz$ και $200Hz$) στην έξοδο.

Το αντικείμενο της εργασίας συνοψίζεται στα κάτωθι, κατά σειρά πραγμάτωσης:

1. Στη βιβλιογραφική έρευνα και τη μελέτη αντίστοιχων δημοσιεύσεων κι άλλων έργων, με σκοπό την κατανόηση του τρόπου λειτουργίας και της χρησιμότητας των εν λόγω διατάξεων.

2. Στη μελέτη των εγχειριδίων των συνιστώμενων υλικών και λογισμικών, με σκοπό τη βέλτιστη αξιοποίησή τους.

3. Στη λογική σχεδίαση του κυκλώματος που θα παράγει τους αντίστοιχους παλμούς.

4. Στην προσομοίωση του κυκλώματος καθώς επίσης κι όλων των συναφών με αυτό τμημάτων κώδικα, ώστε να εξασφαλιστεί η εύρυθμη λειτουργία του και τελικώς στον προγραμματισμό επί της πλακέτας.

Έτσι, η παρούσα εργασία σκοπεύει να γεφυρώσει την ψηφιακή λογική και την σύνθεση αναλογικών σημάτων με ένα εύρος πιθανών εφαρμογών από την επεξεργασία σήματος έως και τις τηλεπικοινωνίες.

Abstract

The present Master's Thesis, entitled «Implementation of a waveform generator based on field-programmable gate array», was carried out under the supervision of Associate Professor Mr. V. Christofilakis, within the framework of the Postgraduate Studies Program «Modern Electronic Technologies» at the Department of Physics, School of Natural Sciences, University of Ioannina.

Its primary objective is to design a digital waveform generator that produces predefined waveforms using digital logic. The design is based on a schematic diagram as well as the use of VHDL in the Intel Quartus Prime Lite software. Subsequently, the aim is to implement this on the DE1 - SoC board and more specifically on the Cyclone V FPGA.

The main objective is to produce four different waveforms (square, sinusoidal, triangular, and sawtooth) at two different frequencies (1Hz and 200Hz) at the output.

The work involves the following stages:

1. Conducting a literature review and studying relevant publications and works to understand the operation and usefulness of these devices.
2. Studying the manuals of the recommended hardware and software aiming for their optimal utilization.
3. Logical design of the circuit that will generate the respective pulses.
4. Simulation of the circuit and all related code sections to ensure its smooth operation, followed by programming on the board.

Thus, this work aims to bridge digital logic and the synthesis of analog signals, with a wide range of potential applications from signal processing to telecommunications.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γεννήτριες κυματομορφών

Δεδομένου ότι η εργασία αυτή έχει ως σκοπό, όπως θα αναφερθεί επακριβώς και παρακάτω, την υλοποίηση μιας γεννήτριας κυματομορφών, η πρώτη αυτή ενότητα είναι αφιερωμένη σε αυτές.

Ως γεννήτρια κυματομορφών θεωρούμε κάθε ηλεκτρονική διάταξη ή συσκευή η οποία παράγει διαφόρων τύπων κυματομορφές, ρυθμιζόμενες συνήθως ως προς το πλάτος ή τη συχνότητα. Τυπικές κυματομορφές μπορούν να είναι η ημιτονοειδής, η συνημιτονοειδής, η τετραγωνική, η τριγωνική, η πριονωτή αλλά ακόμα και πιο σύνθετες. Οι διατάξεις αυτές τεχνικά παρέχουν στο χρήστη τον έλεγχο της συχνότητας, του πλάτους, της φάσης αλλά κι άλλων παραμέτρων [11][13][14][33][36].

Οι γεννήτριες κυματομορφών (ή γεννήτριες σήματος) ταξινομούνται σε δύο μεγάλες κατηγορίες:

1. Τις γεννήτριες αυθαίρετων κυματομορφών.
2. Τις γεννήτριες σημάτων RF.

Οι γεννήτριες αυθαίρετων κυματομορφών (AWG) αποτελούν μια κατηγορία γεννητριών σήματος, οι κυματομορφές των οποίων δημιουργούνται από μια σειρά σημείων - δεδομένων (data - point) [14]. Αυτές μπορούν να είναι διαφόρων μεγεθών και να δημιουργηθούν από τον σχεδιαστή. Σε σχέση με άλλες γεννήτριες, πλεονεκτούν λόγω της μεγαλύτερης ακρίβειας κι ανάλυσής τους. Περιλαμβάνονται σε ένα ευρύ φάσμα εφαρμογών στην εκπαίδευση ή σε πειράματα επιδείξεων ενώ σαφώς αποτέλουν βασικό πειραματικό εξοπλισμό σχεδον σε κάθε ερευνητικό εργαστήριο, αφού χρησιμοποιούνται στο design, το testing, το debugging ή και των καλιμπράρισμα άλλων διατάξεων ή εξοπλισμού αλλά και στην έρευνα σε ποικίλα επιστημονικά πεδία, συμπεριλαμβανομένης της ψηφιακής επεξεργασίας σήματος, της επιστημονικής οργανολογίας καθώς επίσης και των τηλεπικοινωνιών [1][11][14][33][34][36].

Οι γεννήτριες RF χρησιμοποιούνται για τον έλεγχο διατάξεων όπως κεραίες, ενισχυτές, φίλτρα και τηλεπικοινωνιακά συστήματα [39] και χωρίζονται σε τρεις κατηγορίες [39]:

- Τις αναλογικές, που προφανώς χρησιμοποιούν αναλογικές τεχνικές για τη δημιουργία συνεχών κυμάτων.
- Τις ψηφιακές, οι οποίες χρησιμοποιούν ψηφιακές τεχνικές όπως η άμεση ψηφιακή σύνθεση.
- Τις vector, που χρησιμοποιούν ένα συνδυασμό των δύο ανωτέρω και χρησιμοποιούνται ως επί τω πλείστον σε τηλεφωνικά δίκτυα, Wi-Fi ή Bluetooth.

Η ιστορική εξέλιξη των γεννητριών κυματομορφών στην Ηλεκτρονική αναδεικνύει την πρόοδο που επετεύχθη από στοιχειώδεις διατάξεις παραγωγής σημάτων σε εκλεπτυσμένες συσκευές, αναπόσπαστο πλέον τμήμα της σύγχρονης τεχνολογίας. Η πρώτη γεννήτρια κυματομορφών κατασκευάστηκε το 1928 από την General Radio. Η εν λόγω παρήγαγε, στην πρώτη της έκδοση, ένα σήμα σταθερής συχνότητας, ενώ υπήρχε η δυνατότητα αλλαγής μόνο του πλάτους του σήματος. Με την πάροδο του χρόνου και την τεχνολογική εξέλιξη, οι διατάξεις αυτές απέκτησαν τις επιπρόσθετες λειτουργίες της διαμόρφωσης πλάτους και συχνότητας, του συγχρονισμού πολλών σημάτων ή της παραγωγής περισσότερων τύπων κυματομορφών.

Το σημείο καμπής πάντως για τις γεννήτριες κυματομορφών ήταν η έλευση της ψηφιακής τεχνολογίας [14]. Με τη βοήθεια των μικροελεγκτών, των επεξεργαστών ψηφιακού σήματος (DSP, Digital Signal Processor) καθώς επίσης και προχωρημένων αλγορίθμων λογισμικού επετράπη η μεγαλύτερη ακρίβεια, η επαναπρογραμματισιμότητα, η δυνατότητα δημιουργίας αυθαίρετων κυματομορφών (arbitrary waveform) κλπ. [33]. Τις τελευταίες δεκαετίες, η τάση αυτή της συσχέτισης αναλογικών και ψηφιακών ηλεκτρονικών για την υλοποίηση γεννητριών κυματομορφών έχει ενταθεί [34]. Οι σημερινές συσκευές είναι σε θέση να εκμεταλλευόνται τα οφέλη και των δύο τεχνοτροπιών: τη σταθερότητα και το μεγάλο εύρος συχνοτήτων των αναλογικών συστημάτων με την ευελιξία και την προγραμματισιμότητα των ψηφιακών. Αρκετές από αυτές πλέον έχουν έγχρωμες οθόνες γραφικών και πολλούς ακροδέκτες.

Ανεξάρτητα του τρόπου κατασκευής τους (που περιγράφεται στις επόμενες παραγράφους), όλες οι γεννήτριες κυματομορφών έχουν ορισμένα κοινά χαρακτηριστικά [32][33]:

- Παρέχουν κάποιου είδους τρόπο επιλογής μεταξύ των κυματομορφών που θα παράξουν στην έξοδο.
- Παρέχουν κάποιο τρόπο επιλογής συχνότητας κυματομορφών, οι οποίες συνήθως κυμαίνονται από $10mHz$ έως μερικά MHz (για τις AWG) ενώ έως δεκάδες GHz (για τις γεννήτριες σημάτων RF), με ορισμένες να έχουν τη δυνατότητα έως κι $1THz$.
- Επιτρέπουν τη ρύθμιση του πλάτους της κυματομορφής εξόδου.
- Έχουν τουλάχιστον ένα κανάλι εξόδου.

Για την κατασκευή, λοιπόν, των συσκευών αυτών έχουν προταθεί στη διεθνή βιβλιογραφία ανά τα χρόνια διάφορες τεχνικές: από τη χρήση αναλογικών ηλεκτρονικών και μετατροπέων αναλογικού σε ψηφιακό και ψηφιακού σε αναλογικό σήματος (Analog to Digital Converter, ADC και Digital to Analog Converter, DAC), τη χρήση μικροελεγκτών και μικροεπεξεργαστών (Microcontroller και Microprocessor) έως και τις συστοιχίες επιτόπια προγραμματίσιμων πυλών (FPGA) [4][36][42]. Σε κάθε περίπτωση, μερικές από αυτές τις τεχνικές περιγράφονται παρακάτω .

Αρχικά, η άμεση ψηφιακή σύνθεση (DDS, από το Direct Digital Synthesis) είναι μία μέθοδος που χρησιμοποιείται για την παραγωγή ψηφιακών κυματομορφών με ακρίβεια συνθέτοντάς τες απευθείας, με χρήση ψηφιακής λογικής. Η DDS χρησιμοποιεί ένα σήμα διακριτού χρόνου, το οποίο το μετατρέπει σε αναλογικό με τη βοήθεια ενός DAC [12][33]. Για το σκοπό αυτό, απαιτεί έναν συσσωρευτή φάσης (Phase Accumulator), έναν ψηφιακό καταχωρητή Register, σκοπός του οποίου είναι ο προσδιορισμός της συχνότητας και της φάσης της κυματομορφής εξόδου. Εν συνεχεία, η έξοδός του συνδέεται συνήθως με έναν πίνακα αναζήτησης (LUT, Look Up Table), ούτως ώστε να παραχθούν οι πληροφορίες για το πλάτος της κυματομορφής. Η διεργασία αυτή ονομάζεται μετατροπή φάσης σε πλάτος (PAC, Phase to Amplitude Conversion). Τέλος, η ψηφιακή κυματομορφή που έχει δημιουργηθεί μετατρέπεται σε αναλογικό σήμα με τη βοήθεια και πάλι ενός γρήγορου DAC [12]. Το σήμα αυτό φιλτράρεται με σκοπό να περικοπούν αχρείαστες συχνότητες για να μπορεί η διάταξη να εγγυηθεί υψηλής ποιότητας

σήμα εξόδου. Η DDS είναι μια σχετικά γρήγορη μέθοδος εν συγκρίσει με άλλες, ικανή να παράγει κυματομορφές ακριβείας σε ένα εύρος συχνοτήτων.

Μία δεύτερη μέθοδος παραγωγής κυματομορφών είναι αυτή η οποία βασίζεται σε κάποιον απαριθμητή (Counter). Ο πυρήνας της μεθόδου αυτής είναι ένας απαριθμητής ο οποίος, αυξάνοντας τις τιμές του, συνεισφέρει στην παραγωγή του τελικού σήματος. Τα βασικότερα στοιχεία ενός τέτοιου κυκλώματος είναι ο απαριθμητής και κάποιοι LUT. Ο πρώτος είναι επιφορτισμένος με τη διαδικασία της καταμέτρησης συγκεκριμένων και προκαθορισμένων από το σχεδιαστή τιμών, επαναλαμβάνοντας τη διαδικασία αυτή συνεχώς. Για τους δεύτερους, αυτοί μπορεί να είναι κάποιες ψηφιακές μνήμες μόνο γι ανάγνωση (ROM, Read Only Memory), οι οποίες έχουν αποθηκευμένες διακριτές τιμές πλάτους των κυματομορφών, συνδέονται με τους αντίστοιχους απαριθμητές και σε κάθε κτύπο του αποδίδουν τις ψηφιακές τιμές σε κάποια αναλογική έξοδο. Τέλος, απαιτείται γι άλλη μια φορά και κάποιος DAC, σκοπός του οποίου είναι η μετατροπή των ψηφιακών σημάτων των LUT σε αναλογικά. Τα πλεονεκτήματα της μεθόδου αυτής είναι ως επί τω πλείστον δύο: πρόκειται για μια άμεση και σχετικά εύκολη στην εφαρμογή διαδικασία ενώ ταυτόχρονα η εξάρτηση του τελικού αποτελέσματος από «έτοιμους» πίνακες με τα δεδομένα των κυματομορφών προσφέρει ευελιξία. Η μέθοδος αυτή είναι αυτή που χρησιμοποιήθηκε στα πλαίσια της εργασίας.

Όπως ακριβώς ωστόσο και στην περίπτωση της μεθόδου με DDS, η ευκρίνεια των ψηφιακών διφύων (bit resolution) παίζει σημαίνων ρόλο, αφού όσο πιο περιορισμένη είναι, τόσο χειρότερη από άψοψη ποιότητας η τελική κυματομορφή. Από την άλλη, γρηγορότερες συχνότητες ρολογιού την αποδίδουν καλύτερα. Αυτές ωστόσο εξαρτώνται από τους εγγενείς περιορισμούς του υλικού που χρησιμοποιείται. Στην περίπτωση των γεννητριών βασισμένων σε απαριθμητές (Counter-Based Generators) πρέπει να ληφθούν υπόψιν κι οι απαιτήσεις μνήμης, αφού μεγαλύτεροι (και περιεκτικότεροι) LUT παράγουν ποιοτικότερα αποτελέσματα, καταλαμβάνοντας όμως περισσότερο χώρο και επιδρώντας στους πόρους του υλικού. Όσον αφορά τις ίδιες τις ROM, το πλάτος bit(bit width) τους είναι στενά συνδεδεμένο με το βαθμό ανάλυσης της κυματομορφής, με το χρήστη να επιδιώκει την όσο το δυνατόν λιγότερο κοκκιώδη υφή τους. Μεγαλύτερο πλάτος συνεπάγεται και μεγαλύτερη ανάλυση.

Τέλος, αξίζει να αναφερθεί πως αν κι οι μικροελεγκτές κι οι FPGA έχουν καθιερωθεί στον ψηφιακό κόσμο δεκαετίες τώρα, η χρησιμότητά τους σε καμία περίπτωση δεν έχει παρέλθει κι η συνεχής εξέλιξή τους τα καθιστά προϊόντα τεχνολογίας αιχμής. Επί του παρόντος και στα πλαίσια του θέματος της εργασίας αυτής, η εκμετάλλευση των δυνατοτήτων των διατάξεων αυτών για την παραγωγή κυματομορφών είναι μια πρακτική που συναντάται αρκετά στη βιβλιογραφία. Η διαδικασία παραγωγής κυματομορφών είναι παρόμοια αλλά όχι πανομοιότυπη για τις δύο προαναφερθείσες διατάξεις. Οι μικροελεγκτές δύνανται να αλληλεπιδράσουν με εξωτερικά συνδεδεμένους DAC για την παραγωγή κυματομορφών [4][35]. Οι μεν πρώτοι δημιουργούν τα ψηφιακά σήματα, οι δε δεύτεροι τα μετατρέπουν σε αναλογικά. Με χρήση χρονιστών (Timer) ή διαμόρφωσης πλάτους παλμών (PWM, Pulse Width Modulation) μπορούν να παραχθούν τετραγωνικοί λ.χ. παλμοί. Τα σήματα αυτά μπορούν να φιλτραριστούν καταλλήλως ούτως ώστε να αναπαριστούν αναλογικές τάσεις, επιτρέποντας έτσι τη σύνθεση κυματομορφών. Άλλες κυματομορφές μπορούν να παραχθούν από αλγορίθμους που εκτελούνται εντός της κεντρικής μονάδας επεξεργασίας (CPU, Central Processing Unit) ή απλώς με την παραγωγή ακολουθιών ψηφιακών τιμών. Η μνήμη της διάταξης μπορεί να χρησιμοποιηθεί για την αποθήκευση των δεδομένων που, με τη βοήθεια του χρονιστή, μπορούν να «ανακαλούνται» με τη σειρά. Σε κάθε περίπτωση, το αποτέλεσμα είναι το ίδιο με παραπάνω, η παραγωγή του σήματος στην έξοδο.

Κλείνοντας την πρώτη ενότητα αυτή της Εισαγωγής, αξίζει να γίνει ένα σύντομο σχόλιο για το μέλλον των συσκευών αυτών. Τα μεγάλα άλματα της βιομηχανίας ημιαγωγών, η συνεχής τεχνολογική εξέλιξη, οι όλο και πιο προχωρημένες μέθοδοι και τεχνικές στη σχεδίαση, σε συνάρτηση με την αμείωτη ζήτηση για γεννήτριες κυματομορφών όλο και μεγαλύτερης ευκρίνειας κι ακρίβειας, με περισσότερες

επιλογές για τον χρήστη είναι μόνο μερικοί από τους λόγους που το μέλλον τους μοιάζει λαμπρό. Άλλωστε, υπάρχει ήδη η τάση ενσωμάτωσής τους με άλλες συσκευές μέτρησης ή δοκιμών στα πλαίσια της όλο και συστηματικότερης αυτοματοποίησης της εποχής μας [1].

1.2 FPGA

Η παρούσα εργασία πραγματοποιήθηκε με βάση τις FPGA, οπότε ακολουθούν κάποιες πληροφορίες όσον αφορά αυτές, πριν συνεχιστεί η περιγραφή τρόπων χρήσης τους για την παραγωγή κυματομορφών. Τούτες κρίνονται αναγκαίες διότι η εργασία δεν αποτελεί μια εργαστηριακή αναφορά και ταυτόχρονα, δεν έχουν εμβαθύνει όλοι οι αναγνώστες το ίδιο στο θέμα.

Μία συστοιχία επιτόπια προγραμματίσιμων πυλών (FPGA, Field-Programmable Gate Array) είναι ένας τύπος ολοκληρωμένου κυκλώματος (IC, Integrated Circuit) που περιέχει ένα σύνολο διασυνδεδεμένων, λογικών τμημάτων (block) τα οποία είναι προγραμματίσιμα [4][16][23][41].

Βασίστηκαν εν πολλοίς στο σκεπτικό των προγραμματίσιμων μνημών μόνο γι ανάγνωση (PROM, Programmable Read Only Memory) και των προγραμματίσιμων λογικών διατάξεων PLD, Programmable Logic Device [40][41]. Κατά τις δεκαετίες του ΄70 και του ΄80, ήταν που άρχισαν να λαμβάνουν μορφή. Αυτές επέτρεπαν στους σχεδιαστές να υλοποιήσουν λογικές συναρτήσεις χωρίς την ανάγκη αντίστοιχης υλοποίησης ολοκληρωμένων κυκλωμάτων. Οι διατάξεις αυτές χρησιμοποιούσαν λογικές δομές δύο επιπέδων με σκοπό την υλοποίηση ψηφιακής λογικής. Το πρώτο επίπεδο, αυτό της AND ήταν σταθερό ενώ το δεύτερο, το της OR προγραμματίσιμο. Αν και βασίζονταν σε διακόπτες (ηλεκτρικές ασφάλειες, fuse), επομένως ήταν σχετικά απλές στη σχεδίαση και τη χρήση, σπανίως έφταναν στο τελικό προϊόν λόγω των εγγενών περιορισμών τους, ιδίως όταν η πολυπλοκότητα των ψηφιακών κυκλωμάτων αυξήθηκε.

Η πρώτη FPGA ήταν η EP300 της Altera το 1984, ενώ η πρώτη που έγινε εμπορικά διαθέσιμη ήταν η XC2024 της Xilinx, ένα χρόνο μετά [25][27]. Για την ακρίβεια, η συγκεκριμένη συσκευή ήταν η πρώτη που προσομοίαζε, κατά γενική ομολογία, αυτό που σήμερα αντιλαμβανόμαστε ως FPGA, όσον αφορά τη δομή και τα βασικά τους μέρη, ήτοι μια πληθώρα πυλών, συνδεδεμένων με προγραμματίσιμους διακόπτες [16]. Η δημιουργία της αποδίδεται στον Ross Freeman [25][27]. Στα μέσα της δεκαετίας του ΄90, ο Steve Casselman κατάφερε να αναπτύξει μια διάταξη η οποία περιείχε 600,000 επαναπρογραμματίσσιμες πύλες [25][27].

Η δεκαετία αυτή επομένως μπορεί να θεωρηθεί ως δεκαετία απότομης έκρηξης για τις FPGA [16][18][25][27]. Ενώ αρχικά η χρήση τους περιοριζόταν σε τηλεπικοινωνιακές εφαρμογές ή το διαδίκτυο, μέχρι τα τέλη της εισήχθησαν σε προϊόντα ευρείας κατανάλωσης και τις αντίστοιχες βιομηχανίες (προσωπικοί υπολογιστές, κινητήρες κλπ.) [23]. Σε αυτό συνεισέφερε άλλωστε το θεμελιώδες πλεονέκτημά τους της επαναπρογραμματισιμότητας καθώς επίσης κι αυτό της παράλληλης επεξεργασίας (parallel processing). Το τελευταίο αυτό άλλωστε αποτελεί και μια μείζονα διαφορά μεταξύ των γλωσσών προγραμματισμού (π.χ. της C) και των γλωσσών περιγραφής υλικού (HDL, Hardware Description Language), όπως οι VHDL και Verilog [23].

Η ενσωμάτωση επεξεργαστών, μνημών κι άλλων στοιχείων στις FPGA γέννησε την τεχνολογία των συστημάτων επί του ολοκληρωμένου (SoC FPGA, System on Chip Field Programmable Gate Arrays). Οι συσκευές αυτές συνδυάζουν τη δομή των FPGA με τους ARM ή τους RISK - V πυρήνες, επιτρέποντας στους προγραμματιστές να οικοδομήσουν περίπλοκα συστήματα με τόσο προγραμματίσιμη λογική όσο κι επεξεργαστική λειτουργικότητα επί του ίδιου chip [16][41].

Από τα μέσα της δεκαετίας του 2010 κι εντεύθεν, οι FPGA όχι μόνο έχουν ενταχθεί σχεδόν πλήρως στην ηλεκτροκίνηση αλλά πλέον παίζουν τεράστιο ρόλο στο IoT, το cloud computing, το crypto mining, την αεροδιαστημική κι αεροναυπηγική βιομηχανία ακόμα και τις διαδικτυακές μηχανές αναζήτησης

-όσον αφορά τους αντίστοιχους server, με αρκετές εταιρείες παγκοσμίως να έχουν καθιερώσει τη χρήση τους, όπως επίσης και τη χρηματοδότηση για έρευνα κι ανάπτυξη (R&D) [22][25][26][27].

Κάνοντας μια χρονολογική σύγκριση του καθαρού αριθμού των προγραμματίσσιμων πυλών που περιλαμβάνονται σε κάθε FPGA και μόνο κι αγνοώντας τον οικονομικό αντίκτυπο, μπορούμε να παρατηρήσουμε ότι την τελευταία τριαντακονταετία ο αριθμός αυτός έχει αυξηθεί κατά τρεις με τέσσερις τάξεις μεγέθους. Από 9,000 το 1987 σε αρκετές δεκάδες εκατομμύρια σήμερα [4][18][22].

Η γενικόλογη αναφορά ιστορικών στοιχείων δεν ωφελεί σε τίποτα, αν δε γίνει μια εκτενέστερη σχετικά παρουσίαση των ίδιων των διατάξεων αυτών. Αρχικά, πρέπει να αναφερθεί τι διακρίνει τις FPGA από τις PLD κι οι διαφορές αυτές είναι δύο [22][23]:

1. Δεν σχεδιάζονται με κάποιο σκοπό κατά νου. Αντιθέτως, το σκοπό αυτό τον δίνει ο εκάστοτε χρήστης τους με τον προγραμματισμό του.

2. Είναι σε θέση να υλοποιήσουν πολυεπίπεδη λογική, αντί για λογική δύο επιπέδων.

Έτσι, μία FPGA αποτελείται από αρκετά σημαντικά τμήματα κι αν και σε καμία περίπτωση δεν είναι όλες πανομοιότυπες μεταξύ τους, αυτά μπορούν να είναι γενικά [4][16][18][22][23][41]:

1. Τα προγραμματίσιμα λογικά τμήματα (CLB, Configurable Logic Blocks). Αυτά αποτελούν τη βάση ή τα θεμέλια της διάταξης και περιέχουν ένα σύνολο από λογικά στοιχεία όπως δισταθείς πολυδονητές (Flip Flop), πολυπλέκτες (Multiplexer), LUT κλπ. Τα στοιχεία αυτά μπορούν να τροποποιηθούν καταλλήλως ώστε να πραγματοποιηθούν διαφορετικές λογικές λειτουργίες. Οι σύγχρονες FPGA περιέχουν αρκετές χιλιάδες από αυτά.

2. Οι πόροι διεπαφής (IR, Interconnection Resources). Αυτά αποτελούν τα μονοπάτια με τα οποία διασυνδέονται τα CLB. Πρόκειται για τις διαδρομές τις οποίες ακολουθούν τα σήματα επί της πλακέτας.

3. Τα τμήματα εισόδου - εξόδου (IOB, Input - Output Blocks), τα οποία είναι υπεύθυνα για την επικοινωνία της FPGA με εξωτερικές συσκευές. Αφορούν τις διαδικασίες εισόδου κι εξόδου σημάτων και περιλαμβάνουν τις συνδέσεις για περιφερειακά, άλλες συσκευές, ακροδέκτες (pin) κλπ. Είναι προγραμματίσιμα, αν και -στις περισσότερες FPGA- αρκετά πιο αργά από τους πομποδέκτες (tranceivers) που μπορεί να έχει η διάταξη. Το ποια ακριβώς θα χρησιμοποιηθούν εξαρτάται από τον προγραμματιστή κι ασφαλώς επαφίεται σε τελική ανάλυση στη λειτουργικότητα του σχεδίου.

4. Τη μνήμη τυχαίας προσπέλασης τμημάτων (BRAM, Block Random Access Memory). Αυτά είναι τμήματα της συσκευής αφιερωμένα στην προσωρινή αποθήκευση δεδομένων. Σε ορισμένες συσκευές, τα block της είναι αφιερωμένα σε ειδικές λειτουργίες, όπως η επιδιόρθωση σφαλμάτων.

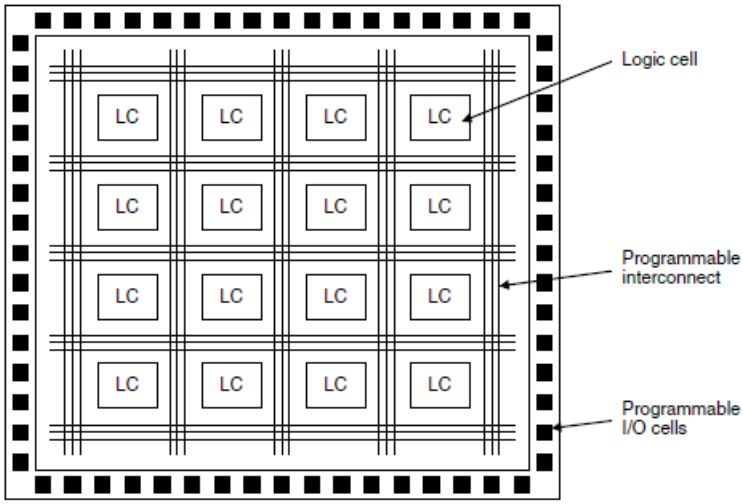
5. Τα τμήματα ψηφιακής επεξεργασίας σήματος (DSPB, Digital Signal Processing Blocks). Πρόκειται περί εξειδικευμένων τμημάτων της συσκευής τα οποία πραγματοποιούν διάφορες μαθηματικές λειτουργίες (όπως ο πολλαπλασιασμός) καθώς επίσης και λειτουργίες επεξεργασίας σήματος (όπως το φιλτράρισμα). Συνήθως περιλαμβάνουν μονάδες αριθμητικής λογικής. Επιτελούν τις διεργασίες τους αποδοτικότερα από τη χρήση πολλών, ξεχωριστών CLB.

6. Οι βρόχοι κλειδωμένης φάσης (PLL, Phase - Locked Loops) ή κι οι βρόχοι κλειδωμένης καθυστέρησης (DLL, Delay - Locked Loops), τα οποία αποτελούν αναπόσπαστα τμήματα της λειτουργίας χρονισμού της FPGA. Είναι υπεύθυνα για τη δημιουργία, τροποποίηση και παροχή σημάτων χρονισμού εντός της διάταξης.

7. Η μνήμη διαμόρφωσης Configuration Memory, η οποία είναι η εσωτερική μνήμη της συσκευής που χρησιμοποιείται για να «γνωρίζει» αυτή τη λειτουργικότητα των λογικών στοιχείων της.

Αυτά κι άλλα εξαρτήματα που μπορεί η κάθε πλακέτα να έχει ξεχωριστά από τις άλλες συνδυάζονται προκειμένου να δημιουργηθούν από τον εκάστοτε χρήστη ψηφιακά κυκλώματα για την πραγματοποίηση οποιουδήποτε σχεδίου.

Όπως ήδη ειπώθηκε, τα θεμελιωδέστερα στοιχεία μιας FPGA είναι τα CLB κι είναι ακριβώς αυτά τα οποία χρησιμοποιούν οι διατάξεις αυτές. Αυτό δίνει την ευχέρεια όχι μόνο της δρομολόγησης (routing) αλλά και το βέλτιστο προγραμματισμό των λογικών τμημάτων [22][41] (βλ. Σχήματα 1.1 κι 1.2). Σε γενικές γραμμές, τα CLB περιλαμβάνουν LUT και flip-flop, τα οποίοι μπορούν να τροποποιηθούν για την παραγωγή, με τον προγραμματισμό τους, λογικών συναρτήσεων (Σχήμα 1.3). Δεδομένου ότι μια τυπική FPGA της εποχής μας έχει χιλιάδες CLB γίνεται εύκολα αντιληπτό το πόσο ικανές είναι στην υλοποίηση περίπλοκων ψηφιακών συστημάτων [4]. Ο συνδυασμός της λογικής και της διασύνδεσης ονομάζεται «υφή» (fabric) των FPGA [41].



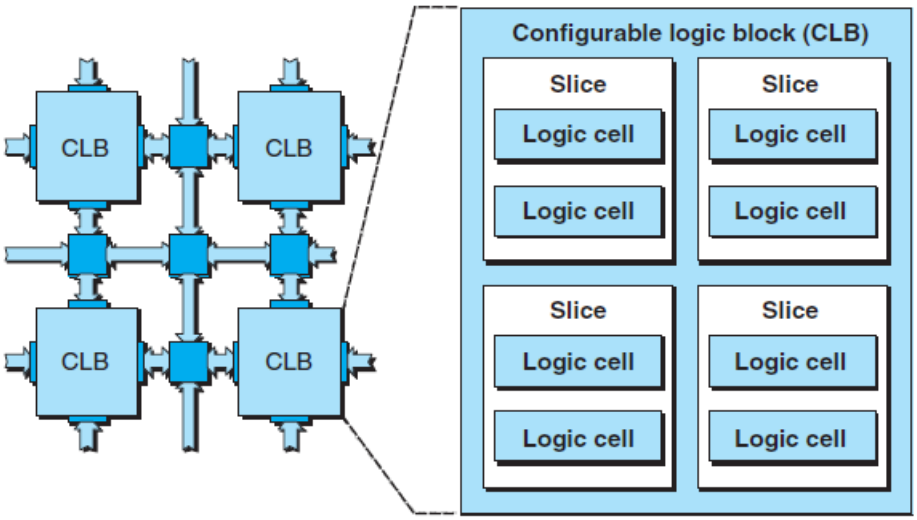
Σχήμα 1.1: Αρχιτεκτονική ενός FPGA [16].

Ο προγραμματισμός τους γίνεται σε διάφορα στάδια, τα οποία είναι τα εξής (Σχήμα 1.4):

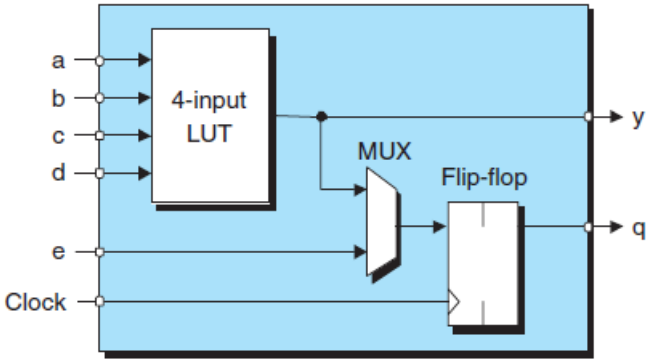
1. Ο βασικός σχεδιασμός. Πριν την οποιαδήποτε ερασιτεχνική ή επαγγελματική τους χρήση, για κάθε έργο (project) απαιτείται ένα προστάδιο σκέψης από τον χρήστη, το οποίο περιλαμβάνει τον τρόπο με τον οποίο θα αξιοποίησει τα πλεονεκτήματα της FPGA προς όφελός του.

2. Η λογική σχεδίαση (design), στάδιο κατά το οποίο γίνεται χρήση κάποιας γλώσσας περιγραφής υλικού (HDL, Hardware Description Language) όπως η VHDL ή η Verilog, είτε κατ΄ αποκλειστικότητα, είτε σε συνδυασμό με κάποιο σχηματικό (schematic).

3. Η προσομοίωση (simulation). Στο βήμα αυτό, ελέγχεται η λειτουργικότητα του σχεδίου, το κατά πόσον ανταποκρίνεται με τον τρόπο με τον οποίο προοριζόταν να ανταποκριθεί, με χρήση των εργαλείων προσομοίωσης που παρέχουν διάφορες εταιρείες για τις διαφορετικές πλακέτες.



Σχήμα 1.2: Αναπαράσταση ενός CLB που περιέχει τέσσερα slices [22].



Σχήμα 1.3: Εσωτερική αναπαράσταση ενός CLB μιας FPGA [22].

4. Η σύνθεση (synthesis), όπου ο κώδικας μετατρέπεται σε ένα netlist που αναπαριστά στοιχεία ψηφιακής λογικής, διασυνδέσεις κλπ.

5. Η υλοποίηση, κατά την οποία ό,τι έχει δημιουργηθεί από το προηγούμενο στάδιο της σύνθεσης, μέσω place and route αλγορίθμων «τοποθετείται» στα φυσικά στοιχεία της FPGA και δημιουργούνται οι μεταξύ τους συνδέσεις.

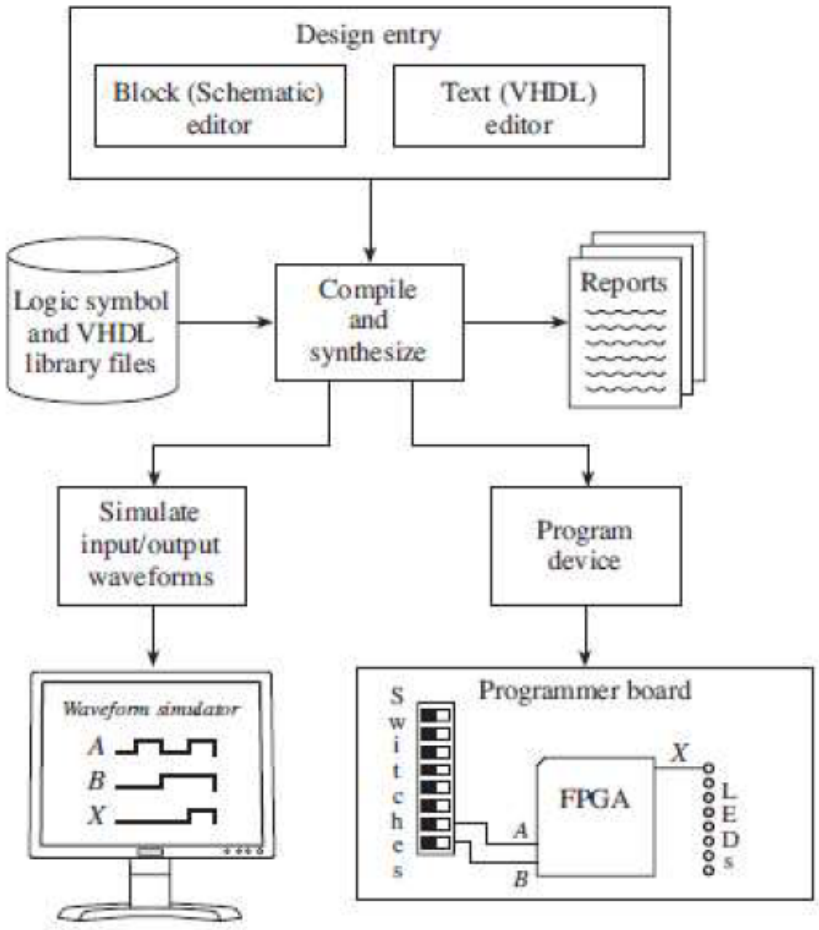
6. Η ανάλυση χρονισμού (timing analysis), όπου ο χρήστης, συνήθως με τη βοήθεια του προγράμματος που χρησιμοποιεί, βεβαιώνεται ότι το σχέδιό του τηρεί τις χρονικές απαιτήσεις και δεν δημιουργούνται προβλήματα χρονισμού λόγω των περιορισμών κάθε συσκευής.

7. Η διαμόρφωση κι ο προγραμματισμός (programming). Αν και στη βιβλιογραφία τείνουν ενίοτε

να θεωρούνται ξεχωριστά βήματα, εδώ, για χάρη συντομίας, ενοποιούνται. Κατά τη διαμόρφωση, δημιουργείται η ροή δεδομένων (bitstream) η οποία περιλαμβάνει σε δυαδική μορφή την πληροφορία που θα προγραμματίσει την πλακέτα. Κατά τον προγραμματισμό, τα δεδομένα αυτά μεταφέρονται στο υλικό (hardware) της FPGA.

8. Τελευταίο βήμα, γενικά θεωρείται η επιβεβαίωση της λειτουργίας (verification) από το χρήστη της συσκευής κατά τον τρόπο με τον οποίον σχεδίαζε να λειτουργήσει.

Εννοείται πως η διαδικασία αυτή ποικίλει ανάλογα με την εκάστοτε πλακέτα, τον κατασκευαστή, τα διαθέσιμα εργαλεία και τα αναπτυξιακά περιβάλλοντα (Xilinx Vivado, Intel Quartus Prime κλπ.) [4][23] .



Σχήμα 1.4: Γενικό FPGA Design Flow. [20]

Όπως γίνεται φανερό από τα ανωτέρω και σύμφωνα με όσα συζητήθηκαν στις προηγούμενες παραγράφους της Εισαγωγής, είναι δυνατή η σχεδίαση και προσομοίωση μιας γεννήτριας ψηφιακών κι

αναλογικών κυματομορφών με χρήση FPGA τεχνολογίας. Γενικά, υπάρχει μια πληθώρα επιλογών για τους σχεδιαστές όσον αφορά τη διάταξη που θα χρησιμοποιήσουν: επεξεργαστές, ολοκληρωμένα κυκλώματα εξειδικευμένης εφαρμογής (ASIC, Application Specific Integrated Circuit), ελεγκτές περιφερειακής διεπαφής (PIC, Peripheral Interface Controller), PLD, FPGA ή κάποιος συνδυασμός όλων αυτών. Το ποια ακριβώς τεχνοτροπία θα χρησιμοποιηθεί είναι ζήτημα που σε τελική ανάλυση επαφίεται στις απαιτήσεις του σχεδίου [4][16][22][23]. Καθώς οι FPGA προσφέρουν το ιδιαίτερο πλεονέκτημα της επαναπρογραμματισιμότητας και της γρήγορης τροποποίησης του υλικού, παίζουν σημαντικό ρόλο σε διεργασίες επεξεργασίας σε πραγματικό χρόνο, τους παράλληλους υπολογισμούς και τη γρήγορη διαχείριση δεδομένων, θεωρήθηκε ότι αποτελούν μια αρκετά καλή επιλογή για το ζητούμενο της εργασίας αυτής. Αξιοποιώντας το ενσωματωμένο της ρολόι μπορούν να παραχθούν οι επιθυμητές συχνότητες, ενώ στο στάδιο της λογικής σχεδίασης μπορούν να δημιουργηθούν οι μετρητές κι οι μνήμες όπου αποθηκεύονται τα δεδομένα των κυματομορφών.

Παρόλο που οι μικροελεγκτές είναι αρκετά οικονομικότεροι κι είναι ευκολότερος ο προγραμματισμός τους, οι FPGA παρουσιάζουν αρκετά συγκριτικά πλεονεκτήματα. Πρώτον, πρωτεύουν στην διαδικασία της παράλληλης επεξεργασίας. Εξαιτίας της αρχιτεκτονικής τους δηλαδή, μπορούν να εκτελούν αρκετές εργασίες ταυτόχρονα και για τον ίδιο λόγο είναι αποδοτικότερες [4][22]. Δεύτερον, τα σχέδια τα οποία βασίζονται σε FPGA μπορούν να εκτελούν αλγορίθμους στο υλικό, παρέχοντας γρηγορότερη κι αποτελεσματικότερη επεξεργασία σε σχέση με τους μικροελεγκτές που βασίζονται στο λογισμικό. Τρίτον, αξίζει να ειπωθεί πως η επαναπρογραμματισιμότητα είναι αρκετά δυσκολότερη στους μικροελεγκτές, ενώ στις FPGA, το σχέδιο μπορεί να τροποποιηθεί σε «χαμηλό επίπεδο», επιτρέποντας αλλαγές στη λειτουργικότητα μιας γεννήτριας κυματομορφών αισθητά γρηγορότερα και ριζικότερα. Οι FPGA χρειάζονται τα προγραμματίσιμα block και τις συνδέσεις προκειμένου να υλοποιηθούν οι κυματομορφές. Άλλωστε, ο προγραμματισμός τους είναι συνυφασμένος με τη λογική δομή τους. Έτσι, δεν απαιτείται να ανακαλούν εντολές, απλώς τις υλοποιούν. Μάλιστα, το χαρακτηριστικό τους αυτό, στη βιβλιογραφία, ονομάζεται «προσωπικότητα» [41]. Τέλος, μιας κι η διαδικασία του pin assignment, δηλαδή η αντιστοίχιση σημάτων εισόδου - εξόδου του σχεδίου με πραγματικές εισόδους - εξόδους της πλακέτας, γίνεται στο μεγαλύτερο δυνατό βαθμό από τον εκάστοτε χρήστη και δεν επαφίεται στο ολοκληρωμένο κύκλωμα, είναι δικό του ζήτημα η βελτιστοποίηση ή όχι του σχεδίου, όσον αφορά τα ζητήματα χρονικής καθυστέρησης και της ευχρησίας [16][22].

Η ιστορική πορεία των FPGA έχει σημαδευτεί από συνεχή καινοτομία. Το τεχνολογικό τους πεδίο εξελίσσεται συνεχώς, όσο η ενσωμάτωσή τους στις νέες τεχνολογίες και τους αυτοματισμούς κερδίζει έδαφος στη βιομηχανία ημιαγωγών. Τα τελευταία χρόνια λ.χ. κερδίζει όλο και μεγαλύτερη δημοτικότητα η υψηλού επιπέδου σύνθεση (HLS, High Level Synthesis), μια τεχνοτροπία που επιτρέπει στους σχεδιαστές να περιγράφουν τη λειτουργία του υλικού σε γλώσσες προγραμματισμού υψηλού επιπέδου όπως οι C, C++ κι OpenCL, επιτρέποντας έτσι την ευκολότερη συσχέτιση υλικού και λογισμικού [4].

Σε ένα κόσμο συνεχώς αυξανόμενων ενεργειακών απαιτήσεων και ταυτόχρονα μεγαλύτερου και γρηγορότερου αυτοματισμού, οι κατασκευαστές των FPGA μεταβαίνουν σε πιο προχωρημένες υλοποιήσεις ημιαγώγιμων διατάξεων με σκοπό τη βελτίωση της ταχύτητας, τη μείωση της ενεργειακής δαπάνης και την αύξηση των λογικών δυνατοτήτων των προϊόντων τους, ενώ, τέλος, με τις αυξανόμενες ανησυχίες γύρω από την ασφάλεια του υλικού, αρκετές εταιρείες εστιάζουν στη συμπερίληψη χαρακτηριστικών ασφαλείας στις FPGA, όπως τη δυνατότητα για ασφαλή εκκίνηση (boot), κρυπτογραφημένα bitstream καθώς επίσης και μηχανισμούς προστασίας διευθύνσεων διαδικτυακού πρωτοκόλλου (IP) [16][22].

1.3 Σκοπός της εργασίας

Βάσει των όσων αναφέρθηκαν στην Περίληψη και την Εισαγωγή, σκοπός της παρούσας εργασίας ήταν αρχικά η μελέτη της τρέχουσας βιβλιογραφίας γύρω από τις γεννήτριες κυματομορφών και τις FPGA. Έπειτα, η εξοικείωση με την αναπτυξιακή πλατφόρμα Cyclone V FPGA DE1 - SoC της Terasic, με στόχο τελικώς τη σχεδίαση κι η υλοποίηση μιας γεννήτριας επί αυτής που θα παρήγαγε τέσσερα διαφορετικά είδη κυματομορφών: τετραγωνική, ημιτονοειδή, τριγωνική και πριονωτή για δύο διαφορετικές συχνότητες, $1Hz$ και $200Hz$. Η γεννήτρια θα ήταν απόλυτα ελεγχόμενη από το χρήστη μέσω input (switches και buttons) της αναπτυξιακής πλατφόρμας.

Για την πραγματοποίησή της χρησιμοποιήθηκαν το ολοκληρωμένο αναπτυξιακό εργαλείο (IDE) Intel Quartus Prime Lite καθώς επίσης κι εργαλεία ανάπτυξης επιστημονικής γλώσσας προγραμματισμού Octave και MATLAB.

Το εγχείρημα αυτό εξελίχθηκε σε αρκετά στάδια, από την ψηφιακή σχεδίαση και τη σχεδίαση του σχηματικού διαγράμματος έως τη συγγραφή κώδικα σε VHDL και MATLAB, με σκοπό να προκύψει ένα όσο το δυνατόν καλύτερο αποτέλεσμα.

Η επιλογή των μεθόδων και των τεχνικών που ακολουθούν στις επόμενες σελίδες, αν και κοινότυπη, βασίστηκε στην επιθυμία υλοποίησης μιας διάταξης που θα προσφέρει εκτός από τα επιθυμητά αποτελέσματα, επαναπρογραμματισιμότητα και σχετική ευελιξία. Έτσι, η μεν τετραγωνική κυματομορφή παρήχθη με ένα απλό κύκλωμα διαιρέτη τάσης ενώ οι ημιτονοειδής, τριγωνική και πριονωτή με τη βοήθεια μνημών ROM. Έτσι, με κατάλληλες τροποποιήσεις στον κώδικα, θα μπορεί να μεταβληθεί όχι μόνο η συχνότητα των σημάτων εξόδου αλλά ακόμη κι η μορφή τους.

Στις ακόλουθες ενότητες είναι καταγεγραμμένες οι διαδικασίες σχεδίασης, υλοποίησης κι ελέγχου της γεννήτριας, συμπεριλμβανομένων των τμημάτων κώδικα που γράφτηκαν, του σχηματικού καθώς επίσης και των διαγραμμάτων χρονισμού των στοιχείων.

Κεφάλαιο 2

Μεθοδολογία

Το παρόν κεφάλαιο είναι αφιερωμένο στα υλικά και τις τεχνικές που χρησιμοποιήθηκαν κατά την εκπόνηση της εργασίας αυτής, καθώς επίσης και στις μεθόδους που αναπτύχθηκαν για την επίτευξη του αντικειμενικού σκοπού της.

Από τη φύση του στόχου της, απαιτήθηκε ο συνδυασμός υλικού και λογισμικού. Μία περιληπτική αναγραφή αυτών παρατίθεται κάτωθι, με σειρά προτεραιότητας:

1. Πλακέτα DE1 - SoC 5CSEMA5F31C6N, Cyclone V FPGA
2. Intel Quartus Prime Lite 22.1
3. Octave 8.3.0
4. MATLAB 8.5.0
5. Microsoft Office Excel 16
6. Questa Intel FPGA Starter 22.1
7. Texmaker 5.1.4

Στις επόμενες σελίδες ακολουθεί η αναλυτική περιγραφή του κάθε σταδίου της εργασίας με χρονολογική σειρά.

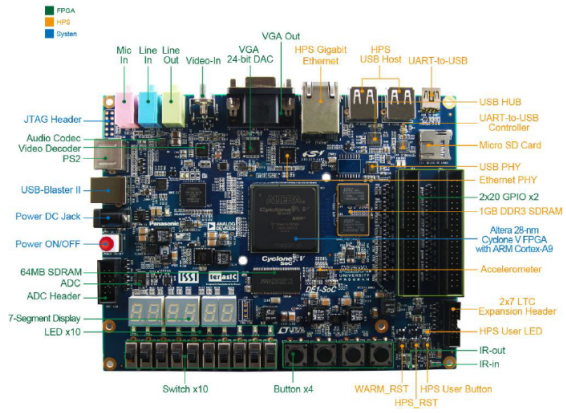
2.1 Αναπτυξιακό εργαλείο

Ισχυρό θεμέλιο για την πραγμάτωση της εργασίας αυτής αποτέλεσε η πλακέτα DE1 - SoC 5CSEMA5F31C6N Cyclone V, οπότε η πρώτη ενότητα είναι αφιερωμένη σε αυτή και τις δυνατότητές της.

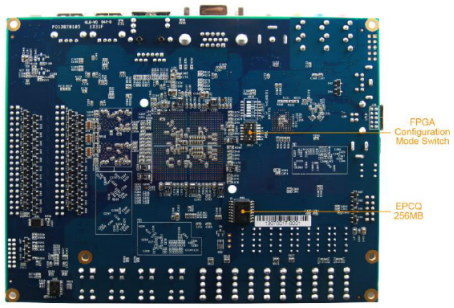
Το DE1 - SoC 5CSEMA5F31C6N αποτελεί μια αναπτυξιακή κι εκπαιδευτική πλατφόρμα που αναπτύχθηκε από την Terasic και βασίστηκε στην System On Chip (SoC) FPGA της Altera (νυν Intel). Οι Cyclone V FPGA κατασκευάζονται με βάση τη διαδικασία των $28nm$ [3]. Αν και το επίτευγμα της τεχνολογίας αυτής εντάσσεται χρονικά στις αρχές της προηγούμενης δεκαετίας (2011) και συνεπώς -με βάση την εξέλιξη έκτοτε- θεωρείται ξεπερασμένη, προσφέρει την απαραίτητη ισορροπία μεταξύ χαμηλής ενεργειακής κατανάλωσης κι υψηλού επιπέδου λογικών ικανοτήτων. Είναι άρρηκτα συνδεδεμένη με το

αναπτυξιακό περιβάλλον Intel Quartus και χρησιμοποιείται συχνά για εκπαιδευτικούς σκοπούς εντός της ακαδημαϊκής κοινότητας, ιδίως στη διδασκαλία της ψηφιακής λογικής σχεδίασης, του προγραμματισμού FPGA, της εκμάθησης των VHDL και Verilog καθώς επίσης και στα ενσωματωμένα συστήματα [2]. Ταυτόχρονα, η περιεκτικότητά του σε λειτουργίες και διεπαφές το καθιστούν ταυτόχρονα ένα ικανό και χρήσιμο εργαλείο σε διαδικασίες κατασκευής πρωτοτύπων (prototyping).

Η πλακέτα φαίνεται στις παρακάτω εικόνες (Σχήματα 2.1 και 2.2) όπου αναγράφονται και τα κυριότερα μέρη της, μια αναφορά για τα οποία γίνεται στις κάτωθι παραγράφους [2][3].



Σχήμα 2.1: Το εμπρόσθιο μέρος του αναπτυξιακού εργαλείου [3].



Σχήμα 2.2: Το οπίσθιο μέρος του αναπτυξιακού εργαλείου [3].

Η εν λόγω διάταξη προσφέρει στο χρήστη την ικανότητα να πραγματοποιήσει μια ποικιλία κυκλωμάτων που κυμαίνονται από πολύ απλά και τετριμμένα έως αρκετά σύνθετα.

Αρχικά, η FPGA είναι η Cyclone V της Altera. Περιέχει $110,000$ λογικά στοιχεία κι ενσωματωμένη μνήμη $5,570kbit$. Η RAM της είναι $516kB$. Η συσκευή έχει 516 εισόδους - εξόδους. Ο επεξεργαστής της είναι ο διπύρηνος ARM Cortex - A9 MPCore στα $800MHz$. Όσον αφορά την ίδια τη μνήμη της αλλά και τις εξωτερικές θέσεις μνήμης, περιέχει $1GB$ DDR3 SDRAM για 32-bit data

bus, SRAM $64kB$ και θέση για Micro - SD κάρτα. Για τη διασύνδεση της συσκευής με άλλες υπάρχουν δύο θύρες USB (επιτρέπουν και USB to UART), $10/100/1000Mbps$ Ethernet PHY, HDMI και VGA για εξόδους βίντεο ($24-bit$ CODEC), έξοδο ήχου κι είσοδο μικροφώνου. Παρέχεται επίσης η δυνατότητα σύνδεσης με ποντίκι ή πληκτρολόγιο Η/Υ καθώς επίσης και με Arduino ή ADC/DAC μέσω κατάλληλων header. Επιπρόσθετα στοιχεία επί της πλακέτας είναι αποκωδικοποιητής TV, πομπός και δέκτης IR, επιταχυνσιόμετρο, αισθητήρας θερμοκρασίας, ενσωματωμένο 7 - SEG Display, δύο $40pin$ header, LTC 2x7 header, κουμπιά, διακόπτες και LED. Για την τροφοδοσία της συσκευής περιέχεται σταθερή παροχή $5VDC$ και, βάσει σχηματικού, αρκετοί ρυθμιστές τάσης. Περιλαμβάνει επίσης έναν ταλαντωτή $50MHz$ που αποτελεί την πρωταρχική πηγή χρονισμού γι αρκετά σχέδια, επιτρέποντας όμως τη σύνδεση και με εξωτερικά ρολόγια εάν κριθεί απαραίτητο. Τέλος, για το προγραμματισμό και την αποσφαλμάτωση της διάταξης υπάρχουν ενσωματωμένοι JTAG header, ενώ αυτός της FPGA γίνεται μέσω του USB - Blaster II [3].

Ανάλογα με τους σκοπούς χρήσης της, υποστηρίζεται από διάφορα λογισμικά, όπως το SoC EDS για επεξεργαστές και το ARM DS-5 για την ανάπτυξη λογισμικού. Στα πλαίσια της παρούσας εργασίας προφανώς χρησιμοποιήθηκε το Intel Quartus Prime για τη σχεδίαση και τον προγραμματισμό της FPGA.

Έτσι, βάσει του αντικειμενικού σκοπού της, θεωρήθηκε πως οι πρώτοι επτά διακόπτες της πλακέτας, $SW[0], SW[1], SW[2], SW[3], SW[4], SW[5]$ κι $SW[6]$ χρησιμοποιήθηκαν για τον έλεγχο της συχνότητας και την παραγωγή των αντίστοιχων τετραγωνικών παλμών, την επαναφορά του ρολογιού, τον έλεγχο και την επαναφορά του μετρητή, καθώς επίσης και τον έλεγχο και την επαναφορά των στοιχείων που λειτούργησαν ως γεννήτορες των τριών άλλων κυματομορφών(βλ. ενότητα 2.2 Ψηφιακή Σχεδίαση). Εννοείται πως για τα ζητήματα χρονισμού των στοιχείων του σχεδίου, απαραίτητη ήταν η χρήση του $50MHz$ ταλαντωτή. Όσον δε αφορά τις εξόδους, αυτές αντιστοιχούνται κατά κανόνα στις GPIO (GPIO[27..0]), αν και κατά τα στάδια της δημιουργίας του project χρησιμοποιήθηκαν τα LED (LEDR[0], LEDR[1], LEDR[2] και LEDR[3]) και το 7 Segment Display (HEX[23..0]).

2.2 Ψηφιακή Σχεδίαση

Η τρέχουσα ενότητα είναι αφιερωμένη στο δεύτερο βήμα που αναφέρθηκε στην Εισαγωγή, τη λογική σχεδίαση.

Το λογισμικό που χρησιμοποιήθηκε για τη σχεδίαση του σχηματικού καθώς επίσης και τον προγραμματισμό της FPGA ήταν το Intel Quartus Prime Lite 22.1.

Το Quartus Prime Lite είναι μια σουίτα λογισμικού που αναπτύχθηκε από την Intel για τη σχεδίαση, την προσομοίωση, τη σύνθεση τον προγραμματισμό διατάξεων FPGA. Παρέχει ένα εύρος εργαλείων για την ανάπτυξη HDL, τόσο σε Verilog, System Verilog όσο και σε VHDL, υποστηρίζοντας ταυτόχρονα ποικίλες οικογένειες FPGA. Αξίζει να σημειωθεί πως το λογισμικό μπορεί, με τη βοήθεια ενσωματωμένων εργαλείων, να προτείνει βελτιώσεις του κώδικα -πέραν της όποιας ανάγνωσης σφαλμάτων-, ο οποίος με τη σειρά του μπορεί να «φορτωθεί» στη συσκευή.

Στην περίπτωση που επιλεγεί σχεδίαση βάση σχηματικού ή μεικτή (με κώδικα και σχηματικό, όπως ακριβώς στα πλαίσια της εργασίας ετούτης), περιλαμβάνεται το αντίστοιχο εργαλείο.

Θεωρείται αυτονόητο πως όπως τα αντίστοιχα προγράμματα των άλλων εταιρειών, έτσι και το Quartus προσφέρει τη δυνατότητα προσομοίωσης των σχεδίων πριν τη διαδικασία της σύνθεσης, επιτρέποντας στους σχεδιαστές να επιβεβαιώσουν ή όχι τη λειτουργικότητα των έργων τους και να αναγνωρίσουν σφάλματα της φάσης σχεδίασης. Στο ίδιο ακριβώς πνεύμα, επιτρέπει την ανάλυση χρόνου ούτως ώστε να εξακριβωθεί εάν το project πληροί χρονικούς περιορισμούς ή περιορισμούς απόδοσης.

Τέλος, δεδομένου ότι η Altera εξαγοράστηκε από την Intel, το Quartus Prime Lite (ανάλογα πάντοτε με την έκδοση) υποστηρίζει τις αντίστοιχες συσκευές FPGA.

2.2.1 Τετραγωνική κυματομορφή

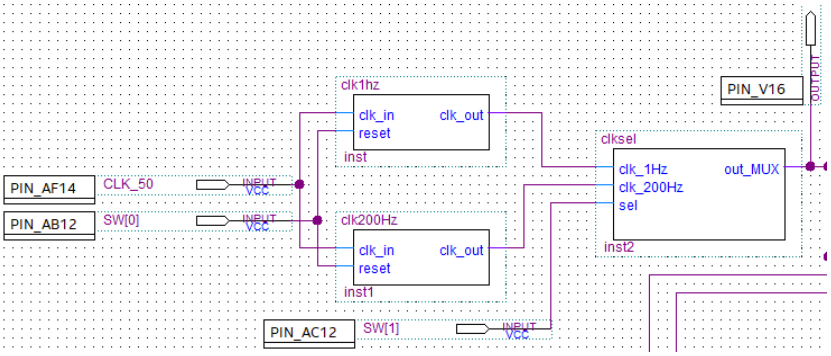
Ο χρονισμός αποτελεί κρίσιμο ζήτημα κάθε ψηφιακού σχεδίου, παρέχοντας τον απαιραίτητο συγχρονισμό κι εξασφαλίζοντας την χρονικά ορθή λειτουργία των στοιχείων του. Καταπώς έχει ήδη ειπωθεί, η DE1 - SoC έχει ενσωματωμένο ταλαντωτή $50MHz$, ο οποίος κι αποτέλεσε τη θεμελιώδη πηγή σημάτων χρόνου για το σχέδιο. Η αξιοποίησή του διευκόλυνε τον ακριβή χρονισμό, αφού αποτελεί ουσιαστικά το «βηματοδότη» του όλου project. Ήταν επομένως λογικό η σταθερή του συχνότητα να χρησιμοποιηθεί ως σημείο αναφοράς για πολλές λειτουργίες επί της FPGA που περιλαμβάνουν την παραγωγή τετραγωνικών παλμών, την ενορχήστρωση της μεταφοράς δεδομένων και το συγχρονισμό των λειτουργιών των υπόλοιπων στοιχείων.

Η πιο συνηθισμένη μέθοδος παραγωγής τετραγωνικών παλμών είναι η χρήση διαιρετών συχνότητας (Frequency Divider) κι είναι αυτή ακριβώς που χρησιμοποιήθηκε.

Στην παρακάτω εικόνα (Σχήμα 2.3) φαίνεται το τμήμα του σχηματικού που αναφέρεται σε αυτούς.

Πρόκειται ασφαλώς περί δύο διαιρετών, ένας προγραμματισμένος να αποδίδει σήματα κάθε $1sec$ (ήτοι συχνότητας $1Hz$) (το στοιχείο clk1hz) κι ένας ανά $5msec$ (ήτοι $200Hz$) (το στοιχείο clk200Hz). Όπως φαίνεται, οι είσοδοί τους είναι δύο: (α) ο ενσωματωμένος ταλαντωτής της πλατφόρμας (CLK_-50) που αποτελεί τον εκ των πραγμάτων το κεντρικό ρολόι -επί του οποίου έγιναν τροποποιήσεις με χρήση ψηφιακής λογικής- και (β) ένας διακόπτης (SW[0]) που συνδέεται με τις εισόδους reset των στοιχείων ώστε να ρυθμίζει τη λειτουργία τους.

Τα reset των διαιρετών συχνότητας αποκρίνονται ως εξής: σε λογικό ΄1΄ «μηδενίζουν» τον εκάστοτε μετρητή, οπότε δεν μετράει. Σε λογικό ΄0΄, η λειτουργία τους είναι η αναμενόμενη. Όπως γίνεται εύκολα κατανοητό, η δουλειά των δύο αυτών component ήταν να μειώνουν συστηματικά τη συχνότητα ρολογιού εισόδου έως ότου επιτευχθεί το επιθυμητό, σύμφωνα με τα ζητούμενα του σχεδίου, αποτέλεσμα. Η διαδικασία διαίρεσης αυτή ακριβώς ήταν που εξασφάλισε τη δημιουργία των τετραγωνικών παλμών.



Σχήμα 2.3: Το τμήμα του σχηματικού με το δικτύωμα των διαιρετών συχνότητας που λειτουργούν ως «πάροχοι» συνχότητας και ταυτόχρονα ως «γεννήτορες» τετραγωνικών παλμών. Διαφαίνεται επίσης κι ο πολυπλέκτης (επιλογέας συχνοτήτων) που επιτρέπει στο χρήστη την επιλογή ποιας κυματομορφής θέλει στην έξοδο και ταυτόχρονα τι συχνότητα θα διαμοιραστεί στα επόμενα στοιχεία του κυκλώματος.

Οι έξοδοί τους είναι συνδεδεμένες σε έναν πολυπλέκτη, το στοιχείο clksel. Πρόκειται περί ενός 2-σε-1 πολυπλέκτη που έχει ως στόχο την επιλογή της συχνότητας. Εάν το πλήκτρο SW[1] είναι σε

λογικό ′1′, τότε επιλέγει τη συχνότητα του $1Hz$. Αντίθετα, την $200Hz$. Αντίστοιχες διατάξεις στη βιβλιογραφία ονομάζονται κι επιλογείς συχνοτήτων. Η έξοδός του, τέλος, συνδέεται με τον παλμογράφο ή οποιαδήποτε άλλη διάταξη απαιτείται ώστε να φανεί καθαρά η τετραγωνική κυματομορφή που ο χρήστης επιλέγει. Ταυτόχρονα, διαμοιράζεται στα υπόλοιπα στοιχεία του σχηματικού προσφέροντας τη δυνατότητα επιλογής συχνότητας και γι αυτά (βλ. ενότητα 2.2.2).

Στα Σχήματα 2.4 και 2.5 φαίνονται οι αντίστοιχοι κώδικες σε VHDL.

```
1  --- Library Declaration ---
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  --- Entity Declaration ---
7
8  entity clk1hz is
9     port (clk_in: in STD_LOGIC;
10          reset: in STD_LOGIC;
11          clk_out: out STD_LOGIC
12          );
13 end clk1hz;
14
15 --- Architecture Declaration ---
16
17 architecture Behavioral of clk1hz is
18    signal temp: STD_LOGIC := '0';
19    signal count: integer := 0;
20 begin
21    frequency_divider: process (reset, clk_in)
22       begin
23          if (reset = '1') then
24             temp <= '0';
25             count <= 0;
26          elsif rising_edge (clk_in) then
27             count <= count + 1;
28             if (count = 24999999) then
29                temp <= NOT temp;
30                count <= 0;
31             end if;
32          end if;
33       clk_out <= temp;
34       end process;
35 end Behavioral;
36
```

Σχήμα 2.4: Ο κώδικας για τον διαιρέτη συχνότητας των $1Hz$.

Εμφανώς, η διαφορά μεταξύ των δύο τμημάτων κώδικα είναι στην κλιμακοποίηση (scaling), δηλαδή τη διαδικασία κατά την οποία «μετρούν» βήματα με τη βοήθεια δύο σημάτων ελέγχου, των temp και count. Το μεν πρώτο είναι του τύπου std_logic, υπακούει δηλαδή στην ψηφιακή λογική ′0′ ή ′1′ και συνεπώς είναι το κατ′ εξοχήν υπεύθυνο για τη δημιουργία του τετραγωνικού παλμού, ενώ το δεύτερο είναι ακέραιος και χρησιμοποιήθηκε για να μετράει τα ωρολογιακά «tick», επομένως σχετίζεται άμεσα με τη διαδικασία διαίρεσης της συχνότητας. Για μισή περίοδο στην έξοδο στέλνεται λογικό ′0′ ενώ στον άλλο μισό ′1′. Εφόσον ο ταλαντωτής της πλακέτας εκτελεί $50,000,000 \frac{cycles}{sec}$, στους μισούς κι ενώ το σήμα temp μετράει τον καθένα τους, το σήμα εξόδου εξαναγκάζεται σε μηδέν, με το αντίστροφο να συμβαίνει στους άλλους μισούς.

Όσον αφορά τον ακριβή υπολογισμό του scaling, αυτός δίνεται από τη μαθηματική σχέση:

$$\boxed{S = \frac{f_{in}}{f_{out}}} \tag{2.1}$$

Όπου f_{in} είναι η συχνότητα του ταλαντωτή κι f_{out} η επιθυμητή συχνότητα εξόδου. Δεδομένου ότι επιθυμούμε ένα κύκλο λειτουργίας 50%, το τελικό αποτέλεσμα (για το ποια πρέπει να είναι η μέγιστη

```
1  --- Library Declaration ---
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  --- Entity Declaration ---
7
8  entity clk200Hz is
9     port (clk_in: in std_logic;
10          reset: in std_logic;
11          clk_out: out std_logic
12          );
13 end clk200Hz;
14
15 --- Architecture Declaration ---
16
17 architecture Behavioral of clk200Hz is
18    signal temp: std_logic := '0';
19    signal count: integer := 0;
20 begin
21    frequency_divider: process (reset, clk_in)
22       begin
23          if (reset = '1') then
24             temp <= '0';
25             count <= 0;
26          elsif rising_edge (clk_in) then
27             count <= count + 1;
28                if (count = 124999) then
29                   temp <= NOT (temp);
30                   count <= 0;
31                end if;
32          end if;
33       clk_out <= temp;
34       end process;
35 end Behavioral;
36
```

Σχήμα 2.5: Ο κώδικας για τον διαιρέτη συχνότητας των $200Hz$.

τιμή της μεταβλητής count) θα δοθεί από τη σχέση (η αφαίρεση του 1 γίνεται για λόγους συμπερίληψης του 0 ως «κτύπου»):

$$\boxed{c_{max} = \frac{S}{2} - 1} \tag{2.2}$$

Τέλος, στο Σχήμα 2.6 φαίνεται ο κώδικας του πολυπλέκτη επιλογής συχνότητας για τις ROM και ταυτόχρονα αυτού που στέλνει στην έξοδο τον τετραγωνικό παλμό της επιλεγμένης συχνότητας. Η διαδικασία του είναι ευαίσθητη στις εναλλαγές των ρολογιών, κάτι που αποτελεί κατά γενική ομολογία μια καλή πρακτική προγραμματισμού.

Ο αναγνώστης μπορεί να δει τους κώδικες των στοιχείων που περιγράφηκαν παραπάνω σε ευκρινέστερη μορφή εάν ανατρέξει στα Παραρτήματα Β΄1, Β΄2 και Β΄3.

2.2.2 Ημιτονοειδής, τριγωνική και πριονωτή κυματομορφή

Η διαδικασία παραγωγής στην έξοδο των υπόλοιπων τριών κυματομορφών που αποτέλεσαν σκοπό αυτής της εργασίας ήταν αρκετά διαφορετική.

Ο λόγος είναι ότι η δημιουργία μιας τετραγωνικής κυματομορφής ένα μικρό κύκλωμα μπορεί να ελέγχει ή όχι την παροχή σήματος High ή Low στην έξοδο ανά χρονικά διαστήματα που ορίζονται από τον σχεδιαστή. Μια τέτοια διαδικασία είναι απολύτως πραγματοποιήσιμη από ένα ψηφιακό κύκλωμα, αφού αποτελεί τον πυρήνα της ίδιας της λογικής αυτής. Από την άλλη, ένα ημιτονοειδές σήμα λ.χ. είναι ένα αναλογικό σήμα, συνεχές κι όχι διακριτό. Κάτι αντίστοιχο ισχύει, με μικρές τροποποιήσεις,

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity clksel is
5      port (
6          clk_1Hz : in std_logic; --- Inputs from Clock Dividers ---
7          clk_200Hz : in std_logic;
8          sel : in std_logic;
9          out_MUX : out std_logic --- Output to ROMs ---
10     );
11 end entity clksel;
12
13 architecture behavioral of clksel is
14 begin
15     process(clk_1Hz, clk_200Hz)
16     begin
17         if sel = '0' then
18             out_MUX <= clk_200Hz; --- '0' sends 200 Hz to output ---
19         else
20             out_MUX <= clk_1Hz; --- '1' sends 1 Hz to output ---
21         end if;
22     end process;
23 end architecture behavioral;
```

Σχήμα 2.6: Ο κώδικας για τον πολυπλέκτη - επιλογέα συχνοτήτων.

για το τριγωνικό και το πριονωτό. Καθώς ένα απλό output pin δεν μπορεί να παράγει ένα τέτοιο σήμα χωρίς βοήθεια, επιλέχθηκε ο δρόμος των ROM.

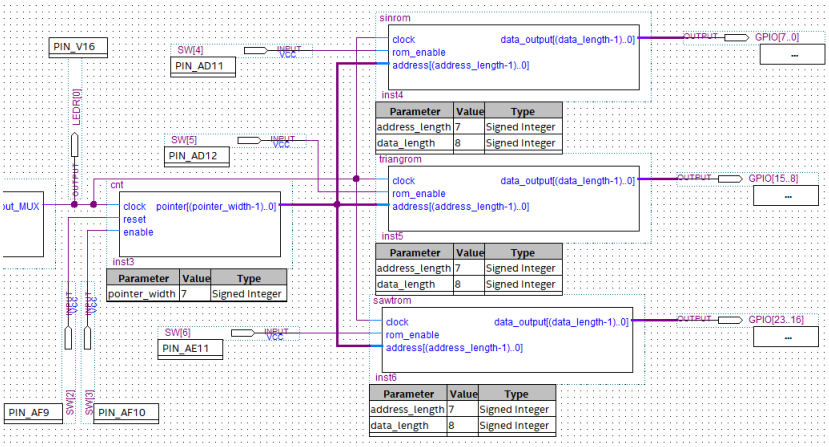
Η μνήμη μόνο γι ανάγνωση (Read - Only Memory, ROM) είναι ένας τύπος ηλεκτρονικής μνήμης που χρησιμοποιείται από τις ηλεκτρονικές διατάξεις και παρουσιάζει το χαρακτηριστικό της μη πτητικότητας (non volatile), δηλαδή να μην απεμπολεί τα δεδομένα που βρίσκονται αποθηκευμένα σε αυτήν, με τη διακοπή της τροφοδοσίας. Εντός μίας τέτοιας μνήμης μπορούν να αποθηκευτούν τιμές - δεδομένα συναρτήσεων, πραγματικά δηλαδή μαθηματικά σημεία (βλ. 2.3). Καθένα από αυτά αντιστοιχίζεται σε μια συγκεκριμένη ψηφιακή τιμή κι έχει τη δική του διεύθυνση (address) εντός της μνήμης. Όλες οι ψηφιακές τιμές τείνουν να προσομοιάσουν τη λειτουργία και τη χρονική εξέλιξη ενός αναλογικού σήματος. Ένας απαριθμητής που θα τροφοδοτείται από το ρολόι του σχεδίου (επομένως θα εξαρτάται από το ποια συχνότητα επιλέχθηκε) διαβάζει με τη σειρά τα δεδομένα αυτά, λειτουργώντας ως δείκτης (pointer) των θέσεων μνήμης. Τέλος, αυτά αποστέλλονται στην έξοδο. Καθώς οι μνήμες περιέχουν δεδομένα τα οποία μεταβάλλονται κατά τι, κάθε φορά, λ.χ. ένα bit, προσομοιάζεται με αυτό τον τρόπο η κλίση της κάθε καμπυλόγραμμης συνάρτησης.

Στην ακόλουθη εικόνα (Σχήμα 2.7) φαίνεται το τμήμα του σχηματικού που αναφέρεται στις μνήμες και τον απαριθμητή που αναφέρθηκε παραπάνω.

Αρχής γενομένης από τα αριστερά, φαίνεται ο πολυπλέκτης - επιλογέας συχνοτήτων για τον οποίον έγινε η συζήτηση στην ενότητα 2.2.1, η έξοδος του οποίου συνδέεται με την είσοδο χρονισμού του απαριθμητή - δείκτη (cnt). Η έξοδος του τετραγωνικού παλμού δε συνεισφέρει στη συγκεκριμένη διαπραγμάτευση κι ως εκ τούτου μπορεί να αγνοηθεί. Η δουλειά του είναι να μεταδίδει στην έξοδό του το σήμα αντίστοιχης συχνότητας που έχει επιλεγεί από τον διακόπτη. Το σήμα αυτό συγχρονίζει ταυτόχρονα τις μνήμες και τον απαριθμητή. Ο δε απαριθμητής επιλέχθηκε να γραφτεί σε κώδικα VHDL και να μη χρησιμοποιηθεί κάποιο έτοιμο, εμπορικό στοιχείο από τις βιβλιοθήκες (library) του Quartus καθώς η μοναδική του έξοδος έπρεπε να είναι ένα λογικό διάνυσμα που θα δείχνει τις θέσεις κάθε μνήμης. Η είσοδός του reset είναι συνδεδεμένη στον διακόπτη SW[2] και σε λογικό ′1′ τον μηδενίζει, ενώ σε λογικό ′0′ η λειτουργία της είναι η αναμενόμενη. Η είσοδος enable από την άλλη είναι συνδεδεμένη στο διακόπτη SW[3] κι όταν βρίσκεται σε 'H' επιτρέπει στον απαριθμητή να μετράει, ενώ τον σταματά όταν βρίσκεται σε 'L'. Η μοναδική του έξοδος είναι το διάνυσμα pointer που συνδέεται σύγχρονα με κάθε είσοδο address των ROM μέσω ενός διαύλου (bus).

Στην παρακάτω εικόνα (Σχήμα 2.8) φαίνεται ο κώδικας του απαριθμητή.

Στην ενότητα της οντότητας (entity) έχει οριστεί ένας ακέραιος ονόματι pointer_width με μήκος 7. Από το όνομά του γίνεται εμφανές ότι αυτός θα αναπαριστά το μήκος του διανύσματος εξόδου, το



Σχήμα 2.7: Το τμήμα του κυκλώματος λειτουργίας των ROM για τις υπόλοιπες τρεις κυματομορφές.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cnt is
    generic (
        pointer_width: integer := 7 -- Intentionally, for reprogrammability reasons.
    );
    port (
        clock: in std_logic;
        reset: in std_logic;
        enable: in std_logic;
        pointer: out std_logic_vector(pointer_width - 1 downto 0) -- See 1st comment.
    );
end cnt;

architecture behavioral of cnt is
    signal count: unsigned(pointer_width downto 0) := (others => '0');
begin
    process(clock, reset)
    begin
        if reset = '1' then
            count <= (others => '0');
        elsif rising_edge(clock) then
            if enable = '1' then
                if count = to_unsigned(2**pointer_width - 28, pointer_width) then -- If the counter reaches 100, it resets.
                    count <= (others => '0');
                else
                    count <= count + 1;
                end if;
            end if;
        end if;
    end process;
    pointer <= std_logic_vector(count(pointer_width - 1 downto 0)); -- Assigns the count to the pointer.
end architecture;
```

Σχήμα 2.8: Ο κώδικας το μετρητή - δείκτη διευθύνσεων μνήμης.

οποίο είναι 7 bit. Με την αντίστοιχη τροποποίηση του μήκους αυτού, τροποποπείται κι η λειτουργία του απαριθμητή, αφού ο αριθμός των bit καθορίζει τον αριθμό των «βημάτων» που θα εκτελέσει και συνεπώς τις θέσεις μνήμης που θα προσπελάσει. Εδώ επιλέχθηκε το μήκος αυτό καθότι με 7 bit «κατασκευάζεται» ο πλησιέστερος μεγαλύτερος αριθμός στο 100, που είναι το 128, βάσει της σχέσης:

$$\boxed{L = 2^N} \tag{2.3}$$

Όπου L είναι το μήκος του διανύσματος και N ο αριθμός των bit.

Η ενότητα της διαδικασίας είναι σχετικά απλή και τυπική ενός τέτοιου στοιχείου. Ευαίσθητη στην επαναφορά και το χρονισμό που παρέχεται από τον πολυπλέκτη, εάν το reset είναι σε λογικό ′1′, μηδενίζεται η μέτρηση, εάν όχι εκτελείται όπως αναμένεται. Σε κάθε παλμό του ρολογιού κι υπό την προϋπόθεση ότι το enable είναι επίσης σε λογικό ′1′, μετράει έως ότου φτάσει σε έναν αριθμό που

```
1   -- Library Declarations: prefer numeric over arithmetic.
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   -- Entity declaration.
8
9   entity sinrom is
10      generic(
11              address_length: natural := 7; -- For reprogrammability reasons.
12              data_length: natural := 8
13              );
14      port(
15              clock: in std_logic;
16              rom_enable: in std_logic;
17              address: in std_logic_vector((address_length - 1) downto 0); -- For
    reprogrammability reasons.
18              data_output: out std_logic_vector((data_length - 1) downto 0)
19              );
20  end sinrom;
21
22  -- Architecture declaration.
23
24  architecture behavioral of sinrom is
25      type rom_type is array (0 to (2**(address_length) - 28)) of
    std_logic_vector((data_length - 1) downto 0); -- For reprogrammability reasons. Always
    count parentheses.
26      constant mem: rom_type :=
27          (
28          "10000000", -- See text for full explanation of the transformation. Make each value
    positive, then normalize to 255 and convert to bits.
29          "10001000",
30          "10001111",
31          "10010111",
32          "10011111",
33          "10100111",
34          "10101110",
35          "10110110",
36          "10111101",
37          "11000100",
38          "11001010",
39          "11010001",
40          "11010111",
41          "11011100",
42          "11100010",
43          "11100111",
44          "11101011",
45          "11101111",
46          "11110011",
47          "11110110",
48          "11111001",
49          "11111011",
50          "11111101",
51          "11111110",
52          "11111111",
53          "11111111",
54          "11111111",
55          "11111110",
56          "11111101",
57          "11111011",
58          "11111001",
59          "11110110",
60          "11110011",
61          "11101111",
62          "11101011",
63          "11100111",
64          "11100010",
65          "11011100",
66          "11010111",
67          "11010001",
68          "11001010",
69          "11000100",
```

(α′)

```
70          "10111101",
71          "10110110",
72          "10101110",
73          "10100111",
74          "10011111",
75          "10010111",
76          "10001111",
77          "10001000",
78          "10000000",
79          "01110111",
80          "01110000",
81          "01101000",
82          "01100000",
83          "01011000",
84          "01010001",
85          "01001001",
86          "01000010",
87          "00111011",
88          "00110101",
89          "00101110",
90          "00101000",
91          "00100011",
92          "00011101",
93          "00011000",
94          "00010100",
95          "00010000",
96          "00001100",
97          "00001001",
98          "00000110",
99          "00000100",
100         "00000010",
101         "00000001",
102         "00000000",
103         "00000000",
104         "00000000",
105         "00000001",
106         "00000010",
107         "00000100",
108         "00000110",
109         "00001001",
110         "00001100",
111         "00010000",
112         "00010100",
113         "00011000",
114         "00011101",
115         "00100011",
116         "00101000",
117         "00101110",
118         "00110101",
119         "00111011",
120         "01000010",
121         "01001001",
122         "01010001",
123         "01011000",
124         "01100000",
125         "01101000",
126         "01110000",
127         "01110111",
128         "10000000"
129     );
130 begin
131 p_table: process(clock) -- Process to handle the data.
132 begin
133     if rising_edge(clock) then
134         if rom_enable = '1' then
135             data_output <= mem(to_integer(unsigned(address)));
136         else
137             data_output <= (others => '0'); -- Clear the output when ROM is disabled.
138         end if;
139     end if;
140 end process p_table;
141 end architecture;
```

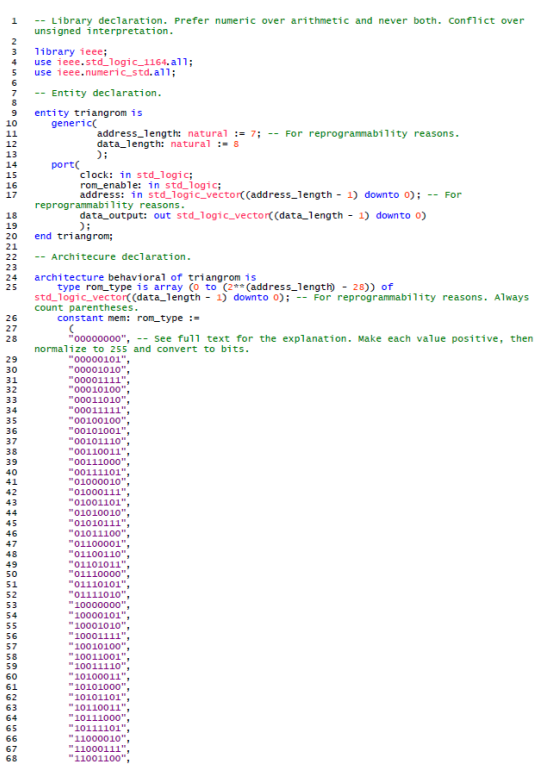
(β′)

Σχήμα 2.9: Ο κώδικας για τη ROM με τα δεδομένα της ημιτονοειδούς κυματομορφής.

ορίζεται από την παρένθεση της γραμμής 26. Πρόκειται δηλαδή επί της ουσίας για έναν απαριθμητή modulo. Επιλέχθηκε να ο τρόπος αυτός έκφρασης του μέγιστου ορίου του απαριθμητή κι όχι η απευθείας χρήση του αριθμού (100) για λόγους επαναπρογραμματισιμότητας κι ευελιξίας, κάνοντάς τον προσαρμόσιμο σε διάφορες εφαρμογές κι απαιτήσεις (βλ. Κεφάλαιο 4). Τέλος, το διάνυσμα pointer ενημερώνεται εντός του τμήματος της διαδικασίας. Λαμβάνει το αποτέλεσμα του σήματος count και το μετατρέπει σε λογικό διάνυσμα, παρέχοντας αποτελεσματικά την έξοδο βάσει αυτού.

Όσον αφορά τις μνήμες αυτές καθ′ εαυτές, που βρίσκονται στα δεξιά του Σχήματος 2.7, περιλαμβάνουν όλες τους τρεις εισόδους και μία έξοδο. Καθεμία έχει μία είσοδο επιλογής (enable), η οποία καθορίζεται από τους διακόπτες SW[4], SW[5] και SW[6] για τα στοιχεία με τα δεδομένα της ημιτονοειδούς, της τριγωνικής και της πριονωτής κυματομορφής αντίστοιχα. Σε λογικό ′1′, ο κάθε διακόπτης «ενεργοποιεί» τη μνήμη με την οποία είναι συνδεδεμένος, ενώ σε ′0′ την απενεργοποιεί. Οι άλλες δύο είσοδοί τους είναι το ρολόι, που καθορίζεται από τον πολυπλέκτη - επιλογέα συχνοτήτων και μια διανυσματική είσοδος address η οποία είναι συνδεδεμένη μέσω διαύλου με την έξοδο του απαριθμητή. Η μοναδική έξοδος των μνημών, data_output, είναι κι αυτή συνδεδεμένη με τις ψηφιακές εξόδους μέσω διαύλου κι οδηγείται στις GPIO της πλακέτας μέσω αντιστοίχισης ακροδεκτών (βλ. ενότητα 3.3).

Τα Σχήματα 2.9, 2.10 και 2.11 περιλαμβάνουν τον κώδικα που γράφτηκε για τις μνήμες. Λόγω της έκτασής του κώδικα έχουν χωριστεί σε δύο επιμέρους εικόνες το καθένα.



```
-- Library declaration. Prefer numeric over arithmetic and never both. Conflict over
unsigned interpretation.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Entity declaration.

entity triangrom is
   generic(
            address_length: natural := 7; -- For reprogrammability reasons.
            data_length: natural := 8
            );
    port(
         clock: in std_logic;
         rom_enable: in std_logic;
         address: in std_logic_vector((address_length - 1) downto 0); -- For
reprogrammability reasons.
         data_output: out std_logic_vector((data_length - 1) downto 0)
         );
end triangrom;

-- Architecure declaration.

architecture behavioral of triangrom is
    type rom_type is array (0 to (2**(address_length) - 28)) of
std_logic_vector((data_length - 1) downto 0); -- For reprogrammability reasons. Always
count parentheses.
    constant mem: rom_type :=
      (
       "00000000", -- See full text for the explanation. Make each value positive, then
normalize to 255 and convert to bits.
       "00001011",
       "00001010",
       "00001111",
       "00010100",
       "00011010",
       "00011111",
       "00100100",
       "00101001",
       "00101110",
       "00110011",
       "00111000",
       "00111101",
       "01000010",
       "01000111",
       "01001101",
       "01010010",
       "01010111",
       "01011100",
       "01100001",
       "01100110",
       "01101011",
       "01110000",
       "01110101",
       "01111010",
       "10000000",
       "10000101",
       "10001010",
       "10001111",
       "10010100",
       "10011001",
       "10011110",
       "10100011",
       "10101000",
       "10101101",
       "10110011",
       "10111000",
       "10111101",
       "11000010",
       "11000111",
       "11001100",
       "11010001",
       "11010110",
       "11011011",
       "11100000",
       "11100110",
       "11101011",
       "11110000",
       "11110101",
       "11111010",
       "11111111",
       "11111010",
       "11110101",
       "11110000",
       "11101011",
       "11100110",
       "11100000",
       "11011011",
       "11010110",
       "11010001",
       "11001100",
       "11000111",
       "11000010",
       "10111101",
       "10111000",
       "10110011",
       "10101101",
       "10101000",
       "10100011",
       "10011110",
       "10011001",
       "10010100",
       "10001111",
       "10001010",
       "10000101",
       "10000000",
       "01111010",
       "01110101",
       "01110000",
       "01101011",
       "01100110",
       "01100001",
       "01011100",
       "01010111",
       "01010010",
       "01001101",
       "01000111",
       "01000010",
       "00111101",
       "00111000",
       "00110011",
       "00101110",
       "00101001",
       "00100100",
       "00011111",
       "00011010",
       "00010100",
       "00001111",
       "00001010",
       "00000101",
       "00000000"
      );
begin
    p_table: process(clock) -- Process to handle the data.
begin
    if rising_edge(clock) then
        if rom_enable = '1' then
            data_output <= mem(to_integer(unsigned(address)));
        else
            data_output <= (others => '0'); -- Clear the output when ROM is disabled.
        end if;
    end if;
end process p_table;
end architecture;
```

(α′) (β′)

Σχήμα 2.10: Ο κώδικας για τη ROM με τα δεδομένα της τριγωνικής κυματομορφής.

```
-- Library Declaration. Avoid using both arithmetic and numeric. Prefer numeric.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Entity Declaration.

entity sawtrom is
   generic(address_length: natural := 7; -- For reprogrammability reasons.
           data_length: natural := 8
           );
   port(clock: in std_logic;
        rom_enable: in std_logic;
        address: in std_logic_vector((address_length - 1) downto 0); -- For reprogrammability reasons.
        data_output: out std_logic_vector((data_length - 1) downto 0)
        );
end entity;

-- Architecture Declaration.

architecture behavioral of sawtrom is
   type rom_type is array (0 to (2**(address_length) - 28)) of std_logic_vector((data_length - 1) downto 0); -- For reprogrammability reasons.
   constant mem: rom_type :=
              (
"00000000", -- See full text for an explanation on the values.
"00000011",
"00000101",
"00001000",
"00001010",
"00001101",
"00001111",
"00010010",
"00010100",
"00010111",
"00011010",
"00011100",
"00011111",
"00100001",
"00100100",
"00100110",
"00101001",
"00101011",
"00101110",
"00110000",
"00110011",
"00110110",
"00111000",
"00111011",
"00111101",
"01000000",
"01000010",
"01000101",
"01000111",
"01001010",
"01001101",
"01001111",
"01010010",
"01010100",
"01010111",
"01011001",
"01011100",
"01011110",
"01100001",
"01100011",
"01100110",
"01101001",
"01101011",
"01101110",
"01110000",
"01110011",
"01110101",
"01111000",
"01111010",
"01111101",
"10000000",
"10000010",
"10000101",
"10000111",
"10001010",
"10001100",
"10001111",
"10010001",
"10010100",
"10010110",
"10011001",
"10011100",
"10011110",
"10100001",
"10100011",
"10100110",
"10101000",
"10101011",
"10101101",
"10110000",
"10110011",
"10110101",
"10111000",
"10111010",
"10111101",
"10111111",
"11000010",
"11000100",
"11000111",
"11001001",
"11001100",
"11001111",
"11010001",
"11010100",
"11010110",
"11011001",
"11011011",
"11011110",
"11100000",
"11100011",
"11100110",
"11101000",
"11101011",
"11101101",
"11110000",
"11110010",
"11110101",
"11110111",
"11111010",
"11111100",
"00000000"
              );
begin
p_table: process(clock) -- Process to handle the data.
begin
     if rising_edge(clock) then
          if rom_enable = '1' then
               data_output <= mem(to_integer(unsigned(address)));
          else
               data_output <= (others => '0'); -- Clear the output when ROM is disabled.
          end if;
     end if;
end process p_table;
end architecture;
```

(α′) (β′)

Σχήμα 2.11: Ο κώδικας για τη ROM με τα δεδομένα της πριονωτής κυματομορφής.

Όσον αφορά τον κώδικα της sinrom, στοιχείου που περιέχει τα δεδομένα της ημιτονοειδούς μορφής, είναι εμφανές από τις εικόνες (α) και (β) του σχήματος 2.9, στην ενότητα της entity έχουν οριστεί γενικώς δύο μεταβλητές, μία για το μήκος της διεύθυνσης και μία για το μήκος των δεδομένων. Η τεχνοτροπία και το σημείο δήλωσής τους επιλέχθηκε ώστε να προσφέρει ευελιξία στον εκάστοτε χρήστη. Εάν δηλαδή η μνήμη θέλει να τροποποιηθεί ριζικά, προσφέροντας μεγαλύτερη ακρίβεια π.χ., θα πρέπει να αλλάξει το data_length. Εάν απαιτείται η τροποποίηση του αριθμού των δεδομένων που βρίσκονται στον πίνακά της, θα πρέπει να αλλάξει το address_length. Η μνήμη χρησιμοποιεί έναν πίνακα (array), τη σταθερά mem, ο οποίος περιέχει εκατό αποθηκευμένους προκαθορισμένους αριθμούς σε δυαδική μορφή, μήκους 8 bit. Οι τιμές αυτές αντιστοιχούν σε σημεία της ημιτονοειδούς συνάρτησης, δηλαδή από 0 έως $2\pi rad$, καλύπτουν δηλαδή μία πλήρη περίοδο κύματος κι η ακρίβεια της αναπαράστασης της συνάρτησης εξαρτάται από τον αριθμό των στοιχείων του πίνακα (βλ. Ενότητα 2.3 και Κεφάλαιο 4). Το μέγεθος του βήματος μεταξύ των στοιχείων καθορίζει την ανάλυση και την ομαλότητα της τελικής κυματομορφής στην έξοδο.

Τα επιλεγμένα μοτίβα των bit αναπαριστούν μια κανονικοποιημένη τιμή της συνάρητησης ημιτόνου. Για το πώς ακριβώς έγινε η κανονικοποίηση αυτή, ο αναγνώστης μπορεί να ανατρέξει στην ενότητα 2.4. Η επιλογή επομένως έγινε με βάση τις απαιτήσεις του συγκεκριμένου σχεδίου, επιτρέποντας όμως ταυτόχρονα την τροποποίησή του σε περίπτωση που αυτό κριθεί αναγκαίο. Για το πώς ελήφθησαν τα δεδομένα του, βλ. την ενότητα 2.3. Δεδομένου πάντως ότι ο αριθμός τους ήταν 100, το μήκος του πίνακα (που όφειλε να είναι σε bit) αντίστοιχο. Αφού για 7 bit ο μεγαλύτερος αριθμός που μπορούμε να λάβουμε είναι το 127 (΄1111111΄) και συνοπολογίζουμε το 0 ως αριθμό, αφαιρούμε 28 θέσεις, ούτως ώστε να μην προκύπτει πρόβλημα κατά τις μετρήσεις του απαριθμητή. Τέλος, εντός της αρχιτεκτονικής έχει οριστεί μια διαδικασία ευαίσθητη στην άνοδο του παλμού του ρολογιού. Υπό την προϋπόθεση ότι η μνήμη δεν είναι απενεργοποιημένη, σε κάθε άνοδο, η διεύθυνση εισόδου μετατρέπεται σε ακέραιο, ένα δείκτη της mem. Έπειτα, το μοτίβο των ψηφίων που είναι αποθηκευμένο εντός της αποστέλλεται στην έξοδο. Εάν δε το reset είναι σε λογικό ΄1΄, η έξοδος μηδενίζεται, σταματώντας τη λειτουργία της μνήμης.

Κάτι αντίστοιχο συμβαίνει και για τη ROM της τριγωνικής κυματομορφής, το στοιχείο triangrom. Ο κώδικας είναι παρόμοιος με μόνη ουσιαστική διαφοροποίηση τα περιεχόμενά του (εικόνες (α) και (β) του Σχήματος 2.10). Επί της ουσίας, ορίστηκε ο πίνακας rom_type ως ένας πίνακας διανυσμάτων και σε αυτόν αντιστοιχήθηκε η σταθερά mem. Το κάθε διάνυσμα έχει μήκος 8 που έχει οριστεί στο generic όπως επίσης κι η κάθε διεύθυνση, με μήκος 7 bit. Ο λόγος που έγινε αυτό είναι αυτός που περιγράφηκε παραπάνω. Η εντολή της γραμμής 23, (2*(address_length) - 28), ορίζει το μέγεθος του πίνακα (εδώ και πάλι 100 τιμών), επιτρέποντας ταυτόχρονα την επαναπρογραμματισιμότητά του χωρίς μεγάλες αλλαγές στο ευαίσθητο κομμάτι της αρχιτεκτονικής. Η σταθερά mem παίρνει τιμές που αναπαριστούν αυτή τη φορά μια τριγωνική κυματομορφή, τις οποίες τιμές τις περιέχει εντός του ορισθέντος εύρους διευθύνσεων. Και πάλι, κάθε στοιχείο του πίνακα αντιστοιχεί σε ένα σημείο της τριγωνομετρικής, μαθηματικής συνάρτησης. Τα σχόλια καταδεικνύουν το πώς εξελίσσονται τα μοτίβα ανόδου και καθόδου της (βήματα). Για τη διαδικασία της, εάν το reset είναι σε λογικό Ή΄, η μνήμη μηδενίζεται. Εάν το enable είναι σε λογικό 'L', δεν είναι ενεργοποιημένη. Εάν είναι ενεργοποιημένη -κι όντας ευαίσθητη στην άνοδο του ωρολογιακού παλμού-, επιλέγει μια έξοδο βάσει της διεύθυνσης. Τα μοτίβα των bit είναι τέτοια ώστε να προσομοιάζουν τη σταδιακή μείωση ή αύξηση που παρουσιάζει η συνάρτηση. Για την έξοδο, η εντολή της γραμμής 133, mem(to_integer(unsigned(address)))), μετατρέπει τη διεύθυνση εισόδου που είναι λογικό διάνυσμα σε έναν ακέραιο που λειτουργεί ως δείκτης του mem. Έπειτα, φέρνει το μοτίβο που είναι αποθηκευμένο στον πίνακα στην καταδειχθείσα τοποθεσία και τέλος, το στέλνει στην έξοδο data_output.

Τελευταία μνήμη είναι αυτή που περιέχει τις προκαθορισμένες τιμές της πριονωτής κυματομορφής.

Αυτή αναφέρεται στο component sawtrom του τμήματος του σχηματικού διαγράμματος (Σχήμα 2.7). Η γενική εικόνα του κώδικα είναι και πάλι παρόμοια από την άποψη ότι και στη συγκεκριμένη ROM ορίζεται και πάλι ένας τύπος, ο rom_type ως πίνακας 101 στοιχείων. Για την επιλογή των τιμών, ο αναγνώστης μπορεί να μεταβεί απευθείας στην ενότητα 2.3. Η επιλογή μεταξύ της χρήσης ακεραίων ή bit στη δημιουργία προκαθορισμένων πινάκων για την παραγωγή κυματομορφών σχετίζεται με ποικίλους παράγοντες. Καθώς το υλικό καταναλώνει περισσότερους πόρους για τους ακεραίους έναντι των bit και ταυτόχρονα αυξάνεται σχετικά κι η πολυπλοκότητα των εντολών που θα χειριστούν τη μετατροπή, επιλέχθηκε να αναπαραστηθούν τα δεδομένα σε δυαδική μορφή. Η δομή του κώδικα περιλαμβάνει ένα προκαθορισθέν μήκος διεύθυνσης και δεδομένων στο τμήμα generic, ώστε να προσφέρεται η δυνατότητα του εύκολου προγραμματισμού ή της τροποποίησης κατά το δοκούν. Εντός της διαδικασίας, κατά την άνοδο του παλμού κι εφόσον η μνήμη είναι ενεργή (με τα reset κι enable στα αντίστοιχα λογικά ′0′ κι ′1′), επιλέγεται μια τιμή εξόδου βάσει της διεύθυνσης στην είσοδο. Η εντολή to_integer(unsigned(address)) μετατρέπει τη διεύθυνση εισόδου -ένα std_logic_vector- σε ακέραιο, προκειμένου να καταδειχθεί η θέση των δεδομένων εντός του πίνακα mem. Η mem(to_integer(unsigned(address))) φέρνει (fetch) την αποθηκευμένη στον πίνακα ακέραια τιμή, στην τοποθεσία που ορίζεται από την address κι αντιστοιχίζει την τιμή αυτή στην έξοδο.

Στα Παραρτήματα Β′4, Β′5, Β′6 και Β′7 παρέχονται οι αντίστοιχοι κώδικες σε ευκρινέστερη μορφή.

2.3 Δειγματοληψία

Η παροχή στην έξοδο του συστήματος ενός τετραγωνικού παλμού είναι μια σχετικά απλή διαδικασία που πραγματοποιήθηκε με τη δημιουργία δύο διαιρετών συχνότητας για τις δύο ζητούμενες συχνότητες των $1Hz$ και $200Hz$ σε συνδυασμό με τον πολυπλέκτη - επιλογέα συχνοτήτων (βλ. Ενότητα 2.2.1). Η λειτουργία αυτή είναι τετριμμένη. Παροχή στην έξοδο τάσης $+V$ συνεπάγεται λογικό ′1′. Παροχή 0, συνεπάγεται λογικό ′0′.

Για τις εναπομείνασες τρεις κυματομορφές όμως, ήτοι ημιτονοειδή, τριγωνική και πριονωτή, η διαδικασία ήταν πολύ διαφορετική καθώς απαιτείται το σύστημα να γνωρίζει επακριβώς τι δεδομένα θα στείλει στην έξοδο και με τι σειρά. Αυτό έγινε με τον προγραμματισμό των αντίστοιχων, τριών μνημών ROM (βλ. Ενότητα 2.2.2) σε συνδυασμό με τον απαριθμητή - δείκτη διευθύνσεων. Παραδείγματος χάριν, ένα ημίτονο είναι μια συνεχής συνάρτηση με πεδίο ορισμού το $(-\infty, +\infty)$ και πεδίο τιμών το $[-1, +1]$. Είναι πρακτικά αδύνατον από άποψη πόρων κι ενέργειας να επιλεγούν όλες αυτές οι τιμές για να αναπαρασταθούν. Ωστόσο, όσο περισσότερες τιμές έχει το σύστημα γι αυτή τη συνάρητηση, τόσο καλύτερη θα είναι η προσομοίωσή της, στα πλαίσια πάντοτε των εργοστασιακών του δυνατοτήτων και των γενικότερων τεχνολογικών περιορισμών. Κοντολογίς, λίγες τιμές κι η αναπαράσταση του ημιτόνου δεν είναι ποιοτικά ακριβής. Πολλές τιμές και το σύστημα γίνεται εκτός από ενεργοβόρο και χρονοβόρο. Όμοια και για τις άλλες δύο κυματομορφές. Επομένως, κρίνεται αναγκαία η εξεύρεση μιας «χρυσής τομής» μεταξύ των δύο, ενός σημείου ισορροπίας μεταξύ ποιότητας και ποσότητας. Πάντως, σε τελική ανάλυση, αυτό ακριβώς το σημείο εξαρτάται από το ποια πλατφόρμα προγραμματίζεται, με τι σκοπό, τι ακρίβεια επιθυμεί ο σχεδιαστής, τι έχει σκοπό να πετύχει και τι δύναται ή όχι να θυσιάσει από άποψη χρόνου κι ενέργειας.

Για την άντληση δεδομένων για τις κυματομορφές, επί της ουσίας σημείων των γραφικών τους αναπαραστάσεων, κρίθηκε αναγκαία η χρήση των προγραμμάτων Octave 8.3.0 και MATLAB 8.5.0, με το δεύτερο να λειτουργεί κυρίως επικουρικά εκεί που οι δυνατότητες του πρώτου εξαντλήθηκαν.

Η MATLAB είναι μια υψηλού επιπέδου γλώσσα προγραμματισμού κι επιπλέον ένα διαδραστικό περιβάλλον που έχει αρχικά σχεδιαστεί για αριθμητικούς υπολογισμούς, ανάλυση δεδομένων, ανάπτυξη αλγορίθμων κι οπτικοποίηση αποτελεσμάτων. Παρέχει ένα μεγάλο αριθμό μαθηματικών συναρτήσεων

και λειτουργιών για διάφορα πεδία όπως η επεξεργασία σήματος, εικόνας, τα συστήματα ελέγχου ή κι η μηχανική μάθηση. Οι χρήστες μπορούν να πραγματοποιήσουν αριθμητικούς υπολογισμούς, να δημιουργήσουν γραφικές παραστάσεις και να αποθηκεύσουν τιμές - δεδομένα των ανωτέρω. Σε συνδυασμό με τη ευκολία του κατά τη διάρκεια αποσφαλμάτωσης, αποτελεί ένα χρήσιμο εργαλείο της επιστημονικής κοινότητας. Το όνομά της αποτελεί συντομογραφία του MATrix LABoratory. Αναπτύχθηκε αρχικά από τον Cleve Moler στα τέλη της δεκαετίας του ΄70, στο Πανεπιστήμιο του Νέου Μεξικού, αν κι η εμπορική της έκδοση κυκλοφόρησε για πρώτη φορά το 1984 από την MathWorks.

Από την άλλη, το Octave είναι το αντίστοιχο ανοικτού κώδικα πρόγραμμα. Μοιράζεται πολλές από τις δυνατότητες του MATLAB, αν κι όχι όλες. Εν συγκρίσει βέβαια με το τελευταίο είναι αρκετά γρηγορότερο και καθώς κάθε έκδοσή του είναι προσβάσιμη στο μέσο χρήστη, πάντοτε ενημερωμένο. Δημιουργήθηκε από τον John W. Eaton στο Πανεπιστήμιο του Ουισκόνσιν, στα 1990. Το Octave περιέχει ένα φιλικό περιβάλλον στο οποίο μπορούν να δημιουργηθούν αρχεία κώδικα, τα script, που μπορούν να αυτοματοποιήσουν διαδικασίες.

Σε αυτά, έγινε η δημιουργία αρχείων επέκτασης $.m$ τα οποία περιείχαν τον κώδικα στην ομώνυμη γλώσσα για τις συναρτήσεις. Στην παρακάτω εικόνα (Σχήμα 2.12) φαίνεται ο κώδικας που συνεγράφη.

Να σημειωθεί ότι η τετραγωνική κυματομορφή περιλαμβάνεται μόνο για λόγους πληρότητας του κειμένου, αφού δημιουργήθηκε με τελείως διαφορετικό τρόπο (βλ. Ενότητα 2.2.1). Τα αποτέλεσματα τιμών κατά την εκτέλεση του κώδικα γι αυτήν ήταν δεδομένα κι ως εκ τούτου μπορεί να αγνοηθεί με ασφάλεια.

Κατά την εκτέλεσή του, το πρόγραμμα ζητάει από το χρήστη την εισαγωγή δύο τιμών: μία για το πλάτος και μία για τη συχνότητα. Τη διαδικασία αυτή την επαναλαμβάνει τέσσερις φορές. Όσο αυτό λαμβάνει χώρα, σε δύο διαφορετικά παράθυρα τυπώνονται οι γραφικές παραστάσεις (πριν κάθε εισαγωγή τιμών) και στο παράθυρο εντολών αποδίδονται 101 τιμές - δείγματα που λαμβάνει από τις συναρτήσεις (αυτό αποδίδεται με τη «δειγματοληψία» από το 0 έως το 1 με βήμα $\frac{1}{100}$.

Στις εικόνες που ακολουθούν (Σχήματα 2.13 έως και 2.16) φαίνονται οι κάτωθι γραφικές παραστάσεις για την τετραγωνική, την ημιτονοειδή, την τριγωνική και την πριονωτή κυματομορφή αντίστοιχα.

Για κάθε μία συνάρτηση ελήφθησαν 101 διαφορετικές τιμές από το παράθυρο εντολών της εφαρμογής. Οι τιμές αυτές καταγράφηκαν σε πίνακα με χρήση του Microsoft Excel.

Η διαδικασία αυτή υπήρξε κρίσιμη διότι με αυτό τον τρόπο, τα δεδομένα εισήχθησαν στις αντίστοιχες ROM του σχηματικού διαγράμματος της γεννήτριας, η οποία μπορεί σε κάθε χτύπο του ρολογιού (για τις δύο ζητούμενες συχνότητες) να τα διαβάζει και να τα στέλνει στην έξοδο ως σήματα.

Ο κώδικας που χρησιμοποιήθηκε βρίσκεται σε ευκρινέστερη μορφή στο Παράρτημα Β΄8.

2.4 Κωδικοποίηση

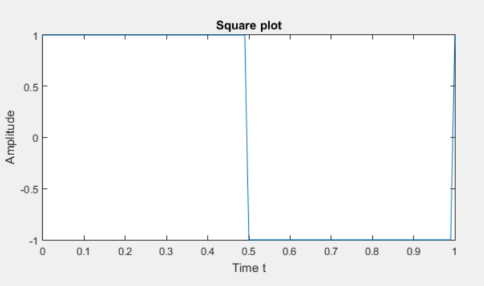
Με τον όρο κωδικοποίηση (encoding) καλείται η διαδικασία συσχέτισης ενός συνόλου χαρακτήρων (εν γένει αλφαριθμητικών) με ένα άλλο σύνολο, σαφώς ορισμένο.

Στα πλαίσια αυτή της εργασίας, η συσχέτιση αυτή είναι μεταξύ θετικών, πραγματικών αριθμών με τις δυαδικές τους αναπαραστάσεις. Δεδομένου ότι επελέγη η τεχνοτροπία προσθήκης των τιμών των κυματομορφών στις μνήμες ROM, ανέκυψε το ζήτημα της μορφής τους. Με άλλα λόγια, το πώς ακριβώς οι μνήμες θα χειριστούν τιμές που είναι είτε αρνητικές, είτε περιέχουν δεκαδικά ψηφία, είτε και τα δύο μαζί. Για το σκοπό αυτό, κρίθηκε αναγκαία η μετατροπή των σημείων των κυματοσυναρτήσεων που προέκυψαν από τη διαδικασία της δειγματοληψίας (βλ. ενότητα 2.3) σε δυαδική μορφή με τη βοήθεια ενός απλού αλγορίθμου, στο Microsoft Office Excel.

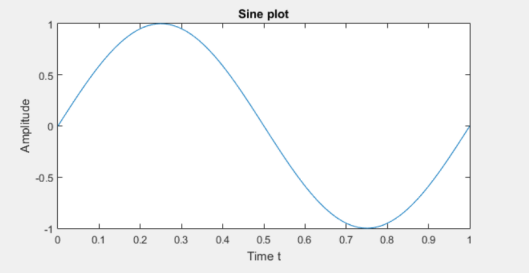
```
clc
close all
clear
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y1 = A*sin(2*pi*F*t) %sinusoidal%
subplot (2, 2, 1)
plot (t,y1)
title ('Sine plot')
xlabel ('Time t')
ylabel ('Amplitude')
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y2 = A*square(2*pi*F*t) %square%
subplot (2, 2, 2)
plot (t,y2)
title ('Square plot')
xlabel ('Time t')
ylabel ('Amplitude')
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y3 = A*sawtooth(2*pi*F*t) %sawtooth%
subplot (2, 2, 3)
plot (t,y3)
title ('Sawtooth plot')
xlabel ('Time t')
ylabel ('Amplitude')
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y4 = A*sawtooth(2*pi*F*t, 0.5) %triangular%
subplot (2, 2, 4)
plot (t,y4)
title ('Triangular plot')
xlabel ('Time t')
ylabel ('Amplitude')
```

Σχήμα 2.12: Ο κώδικας σε MATLAB των κυματομορφών.

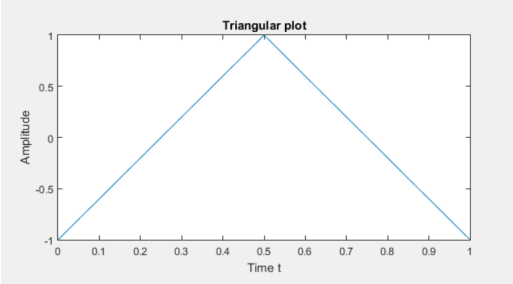
Το συγκεκριμένο λογισμικό σχεδιάστηκε το 1985 από τον Bill Gates κι έφερε την επανάσταση στην ανάλυση και διαχείριση δεδομένων. Αναπτύχθηκε από την Microsoft και κυκλοφόρησε για πρώτη φορά το 1987, ενώ η σουίτα του πλέον αναγνωρίζεται από όλα τα λειτουργικά συστήματα (Windows, Linux, iOS, Android, macOS).



Σχήμα 2.13: Η γραφική παράσταση της τετραγωνικής κυματομορφής από το MATLAB.



Σχήμα 2.14: Η γραφική παράσταση της ημιτονοειδούς κυματομορφής από το MATLAB.



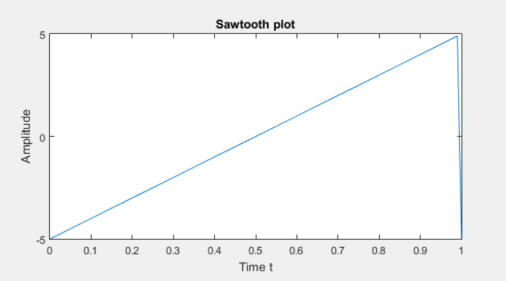
Σχήμα 2.15: Η γραφική παράσταση της τριγωνικής κυματομορφής από το MATLAB.

Δύο από τα βασικότερα χαρακτηριστικά του που υπήρξαν κρίσιμα για την πραγματοποίηση της παρούσας εργασίας είναι οι βιβλιοθήκες συναρτήσεών του κι η δυνατότητα οπτικοποίησης των δεδομένων μέσω γραφικών παραστάσεων.

Τα δεδομένα που ελήφθησαν από το MATLAB τοποθετήθηκαν σε μία στήλη, σε υπολογιστό φύλλο. Εφόσον για όλες τις κυματομορφές ο ελάχιστος αριθμός ήταν το -1 κι ο μέγιστος το $+1$ κι έπρεπε όλα τα δεδομένα να εκπεφραστούν ως συστοιχίες bit μήκους 8, ακολουθήθηκε η εξής διαδικασία:

1. Σε κάθε τιμή (v) από το MATLAB προστέθηκε ο αριθμός 1. Με αυτό τον τρόπο, όλα τα δεδομένα έγιναν θετικοί αριθμοί, με ελάχιστη τιμή πλέον το 0 και μέγιστη το 2 (v').

2. Η νέα αυτή τιμή πολλαπλασιάστηκε με το 127.5. Εφόσον απαιτήθηκε αναπαράσταση 8bit, ο



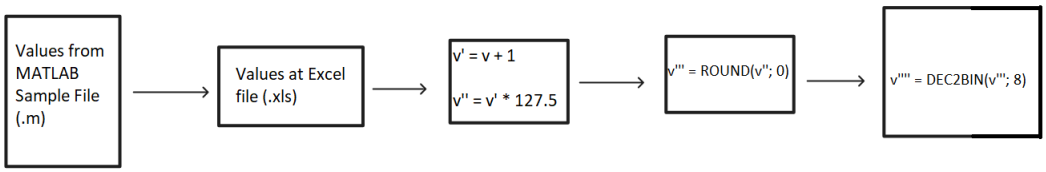
Σχήμα 2.16: Η γραφική παράσταση της πριονωτής κυματομορφής από το MATLAB.

μεγαλύτερος δυνατός αριθμός είναι το 255, το μισό του οποίου είναι το 127.5. Έτσι, λ.χ. το 0 παρέμεινε 0, ενώ το 2 έγινε ακριβώς 255. Η διαδικασία αυτή κανονικοποίησε όλα τα σημεία των κυματομορφών στο 255, σε νέες τιμές (v'').

3. Καθώς η απόλυτη πλειοψηφία των τιμών περιείχε δεκαδικά ψηφία, έγινε στρογγυλοποίηση της τιμής στον πλησιέστερο ακέραιο χωρίς ψηφία μετά την υποδιαστολή, με χρήση της συνάρτησης (function) $ROUND(v''; 0)$. v''' είναι η εκάστοτε στρογγυλοποιημένη τιμή.

4. Τέλος, ακολούθησε η μετατροπή των δεδομένων σε δυαδική μορφή με χρήση της συνάρτησης $DEC2BIN(v'''; 8)$. Ο αριθμός 8 αναπαριστά το μήκος bit. Αυτό έδωσε τις τελικές τιμές σε δυαδική μορφή (v'''').

Μια σχηματική αναπαράσταση του αλγορίθμου φαίνεται στην παρακάτω εικόνα (Σχήμα 2.17).



Σχήμα 2.17: Ο αλγόριθμος μετατροπής των δεδομένων των κυματομορφών που ελήφθησαν από το MATLAB σε δυαδική μορφή, με την οποία τοποθετήθηκαν εντός των ROM.

Τα Παραρτήματα Γ′1, Γ′2 και Γ′3 περιέχουν τους πίνακες από τα υπολογιστικά φύλλα σε ευκρινέστερη μορφή, για τον αναγνώστη που προτίθεται να διασταυρώσει τα όσα συζητήθηκαν στην ενότητα 2.2.2.

2.5 Συγγραφή

Για τη συγγραφή της εργασίας αυτής χρησιμοποιήθηκε ο κειμενογράφος Texmaker 5.1.4 που αποτελεί πρόγραμμα ανοικτού κώδικα βασισμένο σε LaTeX.

Η ειδική αυτή μνεία κρίνεται αναγκαία όχι τόσο για το απαραίτητο της χρήσης του καθ΄ όλη τη διάρκεια της εκπόνησης αλλά κυρίως λόγω της κομψότητάς του, η οποία τον έχει καθιερώσει στην επιστημονική κι όχι μόνο κοινότητα.

Η LaTeX είναι μια γλώσσα συγγραφής εγγράφων που αναπτύχθηκε από τον Leslie Lamport γύρω στα 1980 και βασίστηκε στο σύστημα συγγραφής του Donald Knuth (TeX). Κατά τη συγγραφή, ο χρήστης χρησιμοποιεί απλό κείμενο ενώ η τροποποίηση γίνεται είτε αυτόματα είτε με χρήση εντολών. Οι πιο περίπλοκες από αυτές μπορούν είτε να βρεθούν μέσω των εκπαιδευτικών παραθύρων του προγράμματος, είτε συμβουλευόμενοι το εγχειρίδιο του εκάστοτε κειμενογράφου, είτε εν τέλει με τη βοήθεια της διαδικτυακής της κοινότητας. Η στοιχειοθεσία γίνεται αυτόματα, συμπεριλαμβανομένης της αρίθμησης, της σελιδοποίησης, της τομεοποίησης και της συμπερίληψης γραφικών. Ένα από τα γνωστότερα χαρακτηριστικά της LaTex είναι η αξιοσημείωτη ικανότητά της να χειρίζεται επίσης επιστημονικό συμβολισμό, μαθηματικές εκφράσεις και βιβλιογραφία, παρέχοντας ταυτόχρονα και την απαραίτητη υποστήριξη.

Αν η LaTex είναι η γλώσσα, ο Texmaker είναι ο πάπυρος επί του οποίου ξεδιπλώνεται η χρησιμότητά της και μετατρέπεται σε κάτι ουσιώδες. Πρόκειται για ένα κειμενογράφο που δημιουργήθηκε από τον Pascal Brachet και κυκλοφόρησε το 2003, με αρκετές λειτουργίες κι ένα φιλικό προς το χρήστη περιβάλλον. Η σύνταξη των προτάσεων ακολουθεί χρωματικό κώδικα. Το πρόγραμμα περιέχει και τον αντίστοιχο μεταγλωτιστή. Σε περίπτωση εσφαλμένης εισαγωγής κειμένου ή κώδικα, το παράθυρο διαλόγου αναγράφει τα πιθανά σφάλματα και προτείνει τρόπους επίλυσής τους. Το τελικό αποτέλεσμα μπορεί να αποδοθεί σε μια πληθώρα επεκτάσεων με τη χρησιμότερη να είναι η .pdf. Ένα μεγάλο πλεονέκτημα της χρήσης της LaTeX είναι ότι το αρχικό αρχείο στο οποίο γίνεται η συγγραφή είναι εύκολα τροποποιήσιμο και κατ΄ επέκταση και το ίδιο το τελικό κείμενο όπως αυτό είναι να παρουσιαστεί. Ταυτόχρονα, αντιμετωπίζεται ένα βασικό πρόβλημα των υπόλοιπων κειμενογράφων που είναι η κακής ποιότητας μεταγλώττισή τους ή ακόμα κι η μη συμβατότητά τους σε διαφορετικά λειτουργικά συστήματα (Windows, Linux, macOS κλπ.).

Σε κάθε περίπτωση, ο συνδυασμός LaTex και Texmaker αποτέλεσαν ένα ισχυρό εργαλείο για τη συγγραφή της εργασίας αφού όχι μόνο επέτρεψαν τη δημιουργία περιεχομένου αλλά ταυτόχρονα εγγυήθηκαν επαγγελματικού επιπέδου μορφοποίηση.

Κεφάλαιο 3

Αποτελέσματα

3.1 Προσομοίωση

Η προσομοίωση ενός ψηφιακού σχεδίου σε FPGA ενέχει πληθώρα πλεονεκτημάτων που συνεισφέρουν στην αποτελεσματικότητα και την ακρίβεια της ίδιας της σχεδιαστικής διαδικασίας. Επιτρέπει την επιδιόρθωση λαθών εάν αυτά υφίστανται και σε κάθε περίπτωση δίνει την ευκαιρία στο σχεδιαστή να δει το πώς περίπου θα λειτουργεί το σχέδιό του στην πραγματικότητα [16][40][42].

Η προσομοίωση μειώνει την ανάγκη για εκτεταμένη φυσική πρωτοτυποποίηση, μειώνοντας το κόστος που σχετίζεται με τις επαναλαμβανόμενες εκδόσεις υλικού και μειώνοντας τον χρόνο μέχρι την κυκλοφορία των προϊόντων στην αγορά. Ο επαναπρογραμματισμός ελαττωματικών σχεδίων ή η συνεχής βελτίωσή τους είναι ασφαλώς μια γρηγορότερη, οικονομικότερη κι ευκολότερη διαδικασία. Κυρίως όμως επιτρέπει τον πειραματισμό μέσω της μεταβολής διάφορων παραμέτρων ή ρυθμίσεων του αρχικού σχεδίου και την βαθύτερη κατανόηση του τρόπου λειτουργίας του υπό κανονικές ή άλλες συνθήκες, χωρίς να υπάρχει η ανάγκη τροποποίησης ολόκληρου του υλικού ή δημιουργίας του από την αρχή. Οι σχεδιαστές δύνανται να κάνουν τα έργα τους αποδοτικότερα από άποψη χρήσης πόρων. Επομένως, η προσομοίωση αποτελεί τον ακρογωνιαίο λίθο στη διαδικασία επαλήθευσης (verification) [16][40][42]. Τέλος, αν κι αποτελεί μια ξεχωριστή διαδικασία από τη μελέτη του χρονισμού, συνήθως οι δύο τους πάνε μαζί. Άλλωστε, ένα επιτυχημένο σχέδιο είναι όχι μόνο αυτό το οποίο γενικά κι αόριστα λειτουργεί αλλά ταυτόχρονα βρίσκεται εντός κάποιων χρονικών πλαισίων, ορισμένων από τις απαιτήσεις του σχεδίου, του χρήστη και βεβαίως του ίδιου του υλικού που χρησιμοποιεί [16][40][42].

Στα πλαίσια αυτής της εργασίας, χρησιμοποιήθηκαν δύο διακριτές αλλά παραπλήσιες μέθοδοι για τη διαδικασία της προσομοίωσης.

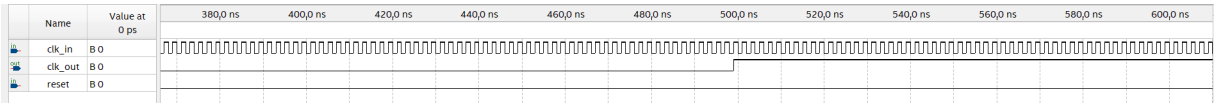
- Τα αρχεία επαλήθευσης (verification files) του Quartus κι ο Simulation Waveform Editor.
- Η προσομοίωση στο Questa FPGA.

Βεβαίως, εκεί που η ακριβής προσομοίωση δεν ήταν εφικτή ή η αποδοτικότερη λύση, προτιμήθηκε ο διαχωρισμός του κυκλώματος σε μικρότερα και το ενδελεχές testing επί της πλακέτας για το καθένα. Ένα τέτοιο παράδειγμα αποτελεί ο διαιρέτης συχνότητας του $1Hz$. Συνδέοντας ένα LED στην έξοδο και την είσοδο του αντίστοιχου κυκλώματος με το ταλαντωτή των $50MHz$, πράγματι, το αποτέλεσμα ήταν το αναμενόμενο.

Στις παρακάτω υποενότητες γίνεται μια αναφορά στις προσομοιώσεις των τμημάτων του σχηματικού της γεννήτριας.

3.1.1 Διαιρέτες συχνότητας - Τετραγωνική κυματομορφή

Όπως αναφέρθηκε προηγουμένως, η προσομοίωση του τετραγωνικού παλμού στις συχνότητες που καθορίστηκαν για την παρούσα εργασία δεν ήταν αποδοτική. Ο λόγος γι αυτό έχει να κάνει με το scaling που αναφέρθηκε στην ενότητα 2.2.1. Για λόγους ευκρίνειας του αποτελέσματος επομένως, επιλέχθηκε να γίνει η προσομοίωση με αρκετές τάξεις μεγέθους μικρότερη κλίμακα, αφού ούτε το $1Hz$, ούτε τα $200Hz$ θα ήταν επαρκώς εμφανή, εάν θέλουμε να σεβαστούμε τα όρια λειτουργίας τόσο του λογισμικού, όσο και του υλικού. Η συχνότητα που επιλέχθηκε είναι το $100kHz$, ώστε η μετάβαση να είναι εμφανής σε περίοδο $1\mu sec$. Με τις απαραίτητες διορθώσεις στον κώδικα του διαιρέτη τάσης (αμφότεροι έτσι κι αλλιώς είχαν την ίδια βάση, αυτό που άλλαζε ήταν οι κύκλοι που μετρούσαν τα εσωτερικά σήματα πριν αλλάξουν), προκύπτει η ακόλουθη εικόνα (Σχήμα 3.1):



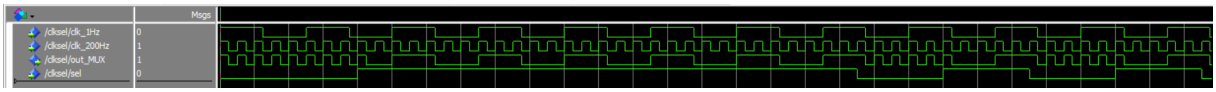
Σχήμα 3.1: Το διάγραμμα χρονισμού των διαιρετών τάσης και ταυτόχρονα γεννήτορων των τετραγωνικών παλμών.

Δεδομένου ότι ο κώδικας τροποποιήθηκε ώστε η έξοδος να επιτραπεί στα $500nsec$, η εύρυθμη λειτουργία του είναι εμφανής.

3.1.2 Πολυπλέκτης - Επιλογέας συχνοτήτων

Όσον αφορά τον επιλογέα συχνοτήτων, έναν πολυπλέκτη 2-σε-1, η προσομοίωσή του είναι αρκετά εύκολη. Το διάγραμμα χρονισμού του φαίνεται παρακάτω (Σχήμα 3.2). Στα ρολόγια που ανατέθηκαν στα σήματα εισόδου clk_1Hz και clk_200Hz έγιναν κατάλληλες τροποποιήσεις στις περιόδους τους ώστε να είναι ευκρινή. Επομένως, δεν ανταποκρίνονται ούτε στις πραγματικές συχνότητες που έχουν κατά τη λειτουργία της διάταξης, ούτε και κι ο μεταξύ τους λόγος συχνοτήτων είναι αυτός που έχει οριστεί στα πλαίσια της εργασίας.

Όταν το select βρίσκεται σε λογικό ′0′, στην έξοδο περνάει το σήμα των $200Hz$, ενώ το αντίθετο συμβαίνει στην περίπτωση του λογικού ′1′. Μια ακόμη αξιοσημείωτη λεπτομέρεια είναι ότι ο πολυπλέκτης, όντας μη ευαίσθητος στο select (βλ. 2.2.1) εφαρμόζει τις αλλαγές που του επιβάλλονται στην άνοδο του επόμενου παλμού.

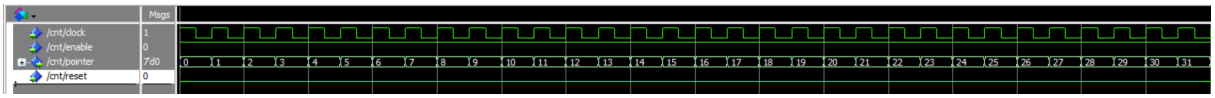


Σχήμα 3.2: Το διάγραμμα χρονισμού του πολυπλέκτη - επιλογέα συχνοτήτων.

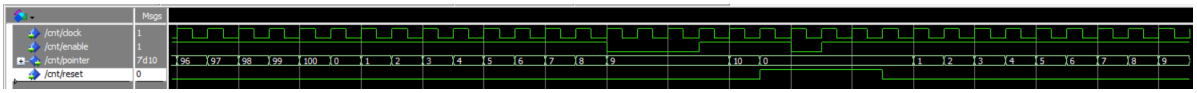
3.1.3 Απαριθμητής - Δείκτης διευθύνσεων μνήμης

Το έργο κι η σημασία του συγκεκριμένου στοιχείου περιγράφηκε με αρκετή λεπτομέρεια στην ενότητα 2.2.2. Συνοπτικά, πρόκειται για component ευαίσθητο στο διακόπτη επαναφοράς (reset) καθώς επίσης και στη συχνότητα ρολογιού εισόδου, για την ακρίβεια στην άνοδο του παλμού.

Η προσομοίωση της συμπεριφοράς του φαίνεται στο διάγραμμα χρονισμού του (Σχήματα 3.3 και 3.4):



Σχήμα 3.3: Το διάγραμμα χρονισμού του απαριθμητή - δείκτη διευθύνσεων μνήμης. Με την άνοδο του παλμού, εκκινεί η μέτρηση.



Σχήμα 3.4: Το διάγραμμα χροσνιμού του απαριθμητή - δείκτη διευθύνσεων μνήμης. Αντίδραση του στοιχείου στους διακόπτες επαναφοράς κι ενεργοποίησης

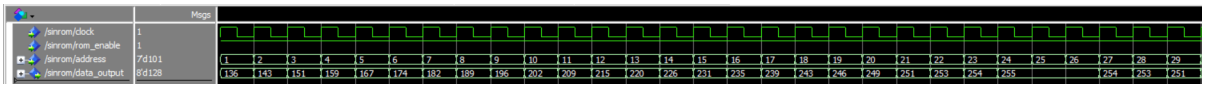
Με την άνοδο του παλμού του ρολογιού, ο απαριθμητής ξεκινά την καταμέτρηση. Πρέπει να σημειωθεί ότι οι ακέραιοι αριθμοί που φαίνονται στο διάγραμμα χρονισμού αποτελούν έναν συγκεκριμένο τρόπο έκφρασης της τιμής του διανύσματος pointer στα πλαίσια της προσομοίωσης κι όχι την τιμή του αυτή καθ΄ εαυτή, η οποία εκφράζεται με μια συστοιχία 7 δυαδικών ψηφίων. Μόλις φτάσει στο 100, ο απαριθμητής ξεκινά την καταμέτρηση από την αρχή. Στο Σχήμα 3.4 μάλιστα φαίνεται το πώς αντιδρά στις εναλλαγές των διακοπτών enable (που τον ενεργοποιεί) και reset (που τον επαναφέρει). Μόλις το enable τεθεί σε λογικό ΄0΄, ο απαριθμητής διατηρεί την τιμή του διανύσματος σταθερή. Εάν ο διακόπτης επανέλθει σε λογικό ΄1΄, συνεχίζει τη μέτρηση από εκεί από όπου είχε σταματήσει. Σε περίπτωση που το reset τεθεί σε λογικό Ή΄, ο απαριθμητής επαναφέρεται στην αρχική του κατάσταση, εκκινεί δηλ. τη μέτρητη ξανά από το 0.

3.1.4 Τριγωνική, ημιτονοειδής, πριονωτή κυματομορφή

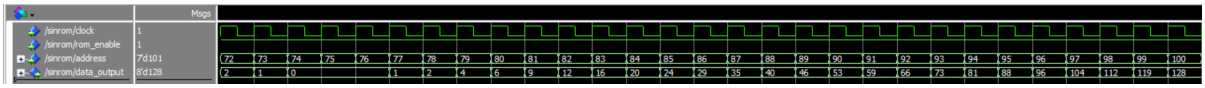
Η έξοδος του απαριθμητή είναι αυτή που με τη σειρά της τροφοδοτεί τις εισόδους address των τριών ROM, οι οποίες τροφοδοτούνται επίσης από τον πολυπλέκτη και τους αντίστοιχους διακόπτες ενεργοποίησής τους. Αυτό εξασφαλίζει αφ΄ ενός μεν ότι η συχνότητα του ρολογιού θα είναι σταθερή συνεχώς κι αφ΄ ετέρου δε ότι -για όσο δεν έχει προκύψει κάποια αλλαγή στους διακόπτες του- ο απαριθμητής θα συνεχίσει να μετράει έως ότου φτάσει στο 100 κι ύστερα θα επαναλμβάνει τη διαδικασία. Αποτέλεσμα αυτής της κατάστασης είναι η ανυπαρξία ασυνέχειας κατά τη λειτουργία των μνημών. Αφού διαγράψουν μία φορά τη συνάρτηση την οποία αναπαριστούν, εκκινούν τη διαδικασία από την αρχή.

Εντός των μνημών ROM, οι διαδικασίες είναι ευαίσθητες στο ρολόι. Έτσι, σύμφωνα με τον κώδικά τους, σε κάθε άνοδο του παλμού του ρολογιού και για μία περίοδό του ακριβώς εξέρχονται τα δεδομένα που βρίσκονται αποθηκευμένα εντός της εκάστοτε μνήμης από την έξοδό της αντίστοιχα (data_output). Οι δύο παρακάτω εικόνες (Σχήματα 3.5 και 3.6) αναπαριστούν τα διαγράμματα χρονισμού της ROM που περιείχε αποθηκευμένες τις τιμές των σημείων της ημιτονοειδούς κυματομορφής, τα οποία προέκυψαν από την προσομοίωσή της:

Ένα πλεονέκτημα της προσομοίωσης στον Questa FPGA εν συγκρίσει με αυτή των κυματομορφών στον Simulation Waveform Editor είναι ότι το πρώτο επιτρέπει και την παρουσίαση των δεδομένων σε αναλογική μορφή. Αυτό επέτρεψε τον έλεγχο του κατά πόσο τα δεδομένα εντός των μνημών

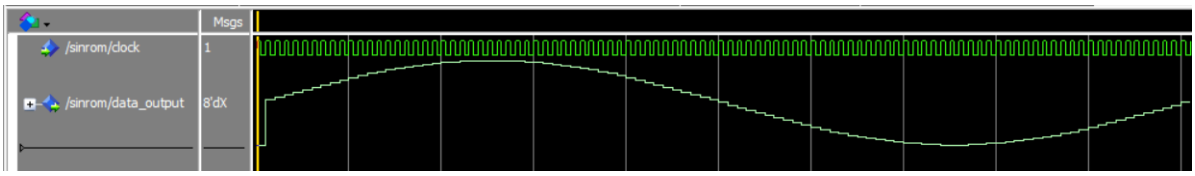


Σχήμα 3.5: Το διάγραμμα χρονισμού της ROM που εμπεριέχει τα δεδομένα της ημιτονειδούς κυματομορφής. Οι θέσεις μνήμης αυξάνονται διαδοχικά και τα δεδομένα αποστέλλονται στις έξοδο όπως ακριβώς βρίσκονται στον πίνακα.



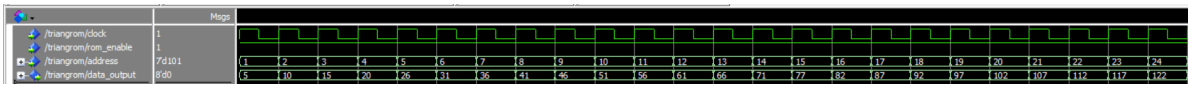
Σχήμα 3.6: Το τελικό τμήμα του διαγράμματος χρονισμού της ROM που εμπεριέχει τα δεδομένα της ημιτονοειδούς κυματομορφής.

ανταποκρίνονται πράγματι στις κυματομορφές ή όχι. Επιλέγοντας λοιπόν την αναλογική προσομοίωση των δεδεμένων, προκύπτει η εικόνα του Σχήματος 3.7, η οποία όντως αναπαριστά την περίοδο ενός ημιτόνου.

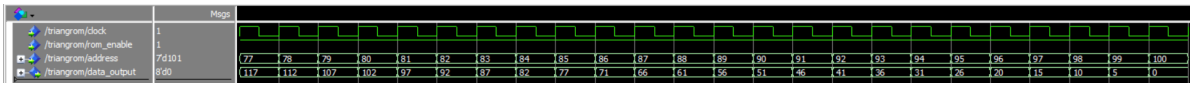


Σχήμα 3.7: Η αναλογική προσομοίωση της μνήμης με τα δεδομένα της ημιτονοειδούς κυματομορφής.

Αντίστοιχο αποτέλεσμα ελήφθη κι από την προσομοίωση της μνήμης με τα δεδομένα της τριγωνικής κυματομορφής. Στα Σχήματα 3.8 και 3.9 φαίνεται το διάγραμμα χρονισμού της. Και σε αυτή την περίπτωση, τα δεδομένα βγαίνουν με τη σειρά που έχουν προγραμματιστεί στη ROM σε κάθε ανερχόμενο παλμό.



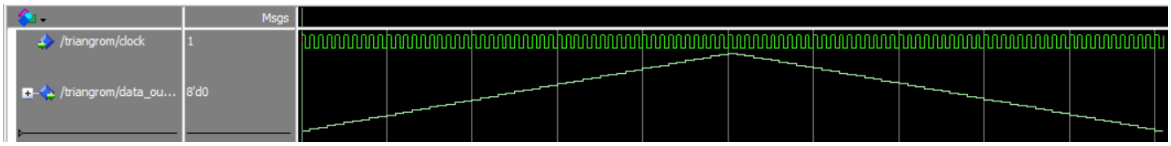
Σχήμα 3.8: Το διάγραμμα χρονισμού της ROM με τα δεδομένα της τριγωνικής κυματομορφής. Αποστέλλονται στην έξοδο σε κάθε αλλαγή της διεύθυνσης μνήμης, με τη σειρά που βρίσκονται αποθηκευμένα στον πίνακα.



Σχήμα 3.9: Το τελικό τμήμα του διαγράμματος χρονισμού της ROM με τα δεδομένα της τριγωνικής κυματομορφής.

Το δε Σχήμα 3.10 καταδεικνύει ότι τα δεδομένα εντός της μνήμης πράγματι αναπαριστούν διακριτές

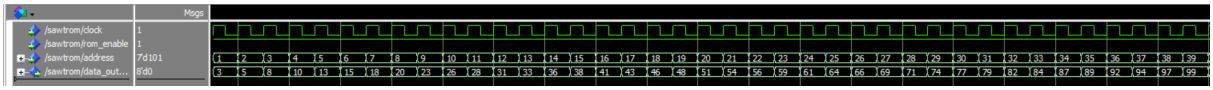
τιμές μιας τριγωνικής κυματομορφής.



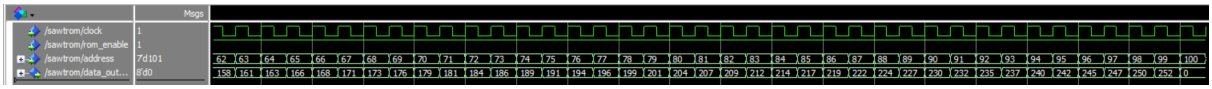
Σχήμα 3.10: Η αναλογική προσομοίωση της μνήμης με τα δεδομένα της τριγωνικής κυματομορφής.

Όπως ακριβώς και στην ημιτονοειδή κυματομορφή, έτσι κι εδώ, οι αναπαραστάσεις δεν είναι τέλειες ευθείες ή καμπύλες. Ο λόγος που συμβαίνει αυτό είναι ότι χωρίς τη βοήθεια ενός DAC δε μπορεί να αναπαρασταθεί ομαλά ένα αναλογικό σήμα. Αντ' αυτού, προκύπτουν αυτές οι μικρές ασυνέχειες που οφείλονται στη διακριτή αύξηση και μείωση των bit. Αποτελούν ωστόσο μια αρκετά καλή μέθοδο προσομοίωσης των κυματομορφών.

Τέλος, όσον αφορά την προσομοίωσης της μνήμης ROM με τα δεδομένα της πριονωτής κυματομορφής, μια αντίστοιχη διαδικασία παρείχε τα αποτελέσματα που φαίνονται στα Σχήματα 3.11 και 3.12.

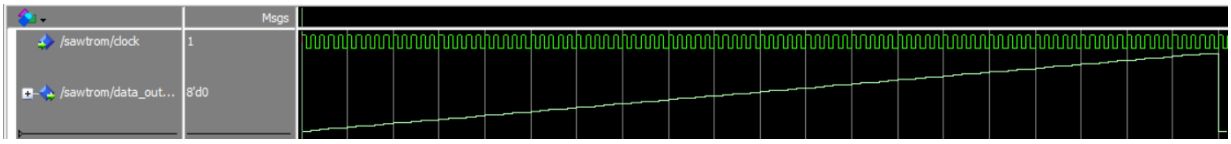


Σχήμα 3.11: Το διάγραμμα χρονισμού της ROM με τα δεδομένα της πριονωτής κυματομορφής. Η παροχή τους στην έξοδο γίνεται με τη σειρά με την οποία βρίσκονται τοποθετημένα στον αντίστοιχο πίνακα.



Σχήμα 3.12: Το τελικό τμήμα του διαγράμματος χρονισμού της ROM με τα δεδομένα της πριονωτής κυματομορφής.

Η δε αναλογική της αναπαράσταση φαίνεται στο Σχήμα 3.13.



Σχήμα 3.13: Η αναλογική προσομοίωση της μνήμης με τα δεδομένα της πριονωτής κυματομορφής.

Σε επίπεδο προσομοίωσης επομένως και το τμήμα του σχηματικού διαγράμματος που αναφέρεται στις μνήμες λειτουργεί ως όφειλε. Όπως ακριβώς και με τον απαριθμητή, έτσι κι εδώ επιλέχθηκε να εκφραστούν οι τιμές των μνημών στην έξοδο σε δεκαδική μορφή χωρίς πρόσημο (unsigned), για χάρη της καλύτερης ευκρίνειας της προσομοίωσης. Για έναν ενδελεχή έλεγχο της σειράς των δεδομένων εντός των πινάκων mem, βλ. Ενότητα 2.2.2 ή τα Παραρτήματα Β΄5, Β΄6 και Β΄7 αντίστοιχα, σε συνδυασμό με τους αντίστοιχούς τους πίνακες κωδικοποίησης (Παραρτήματα Γ΄1, Γ΄2 και Γ΄3).

3.2 Ανάλυση χρονισμού

Η ανάλυση χρονισμού αποτελεί ένα ακόμη ουσιώδες τμήμα ενός σχεδίου. Σε μικρότερα ή μεσαίου μεγέθους έργα, όπως η συγκεκριμένη εργασία, θεωρείται γενικά δύσκολο να προκύψουν σοβαρά ζητήματα με το χρονισμό [19]. Όσο όμως αυξάνεται η πολυπλοκότητα του σχεδίου, τόσο αυξάνεται κι η σημασία της.

Σε γενικές γραμμές, η ανάλυση χρονισμού χρησιμοποιείται για την επιβεβαίωση του ότι τα ψηφιακά σήματα εντός της FPGA πληρούν κάποιες συγκεκριμένες χρονικές απαιτήσεις. Ελέγχει δηλαδή εάν η συσκευή λειτουργεί εντός των ορισθέντων χρονικών ορίων, ελαχιστοποιώντας τις καθυστερήσεις και μεγιστοποιώντας την απόδοση [19]. Αυτό σημαίνει ότι όχι μόνο το σχέδιο παράγει τις απαιτούμενες εξόδους αλλά το κάνει και στην επιθυμητή ταχύτητα.

Οι FPGA έχουν αυστηρούς χρονικούς περιορισμούς (timing constraints), απαιτείται δηλαδή τα σήματα να καταφθάνουν στον προορισμό τους εντός ενός χρονικού παραθύρου. Η παραβίασή τους ενδέχεται να οδηγήσει σε λανθασμένη λήψη ή αποστολή, απώλεια των δεδομένων ή ζητήματα μεταστατθερότητας (metastability), κατάσταση κατά την οποία η τιμή των δεδομένων είναι άγνωστη [19].

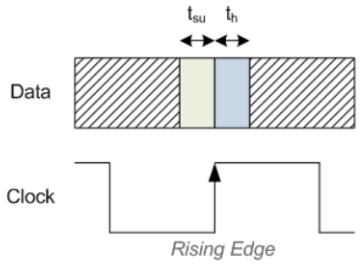
Οι βασικότερες παράμετροι των χρονικών περιορισμών περιλαμβάνουν (αλλά δεν εξαντλούνται κατ' αποκλειστικότητα σε) το χρόνο διάταξης (setup time), το χρόνο κράτησης (hold time) και τη μέγιστη συχνότητα ρολογιού (maximum clock frequency) [19].

Η ταχύτητα διάδοσης της πληροφορίας σε ένα σύστημα έχει μια πεπερασμένη τιμή. Η αύξηση της περιπλοκότητας του συστήματος (λ.χ. λόγω πολλών μονοπατιών) προκαλεί καθυστερήσεις (slack) στους χρόνους αυτούς. Με τη σειρά τους, αυτές απαιτείται να βρίσκονται εντός κάποιων ορίων, τα οποία σε τελική ανάλυση ορίζονται τόσο από το σχεδιαστή και τις ανάγκες του σχεδίου όσο κι από τις ίδιες τις δυνατότητες της FPGA.

Δεδομένου ότι οι διατάξεις αυτές αποτελούνται από εσωτερικά flip - flop, τα σήματα εντός τους διαδίδονται κατά μήκος των μονοπατιών που ενώνουν αυτά τα στοιχεία. Ο χρόνος αυτός ονομάζεται καθυστέρηση διάδοσης (propagation delay), t_p. Λόγω της λογικής των σχεδίων, τα σήματα μεταβάλλονται. Με την άφιξή τους όμως από την έξοδο του πρώτου στην είσοδο του δεύτερου flip - flop πρέπει να είναι σταθερά για κάποιο χρονικό διάστημα ώστε να προσπελαθούν σωστά [19].

Ο setup time επομένως, t_{su}, είναι ο χρόνος τον οποίον απαιτείται τα δεδομένα να είναι σταθερά πριν τον ωρολογιακό παλμό ώστε να ληφθούν και να διαδοθούν σωστά (χωρίς απώλειες).

Ο holt time, t_h, είναι ο χρόνος που απαιτείται η είσοδος ενός flip-flop να είναι σταθερή μετά την αλλαγή του ρολογιού. Μία αναπαράσταση των χρόνων αυτών φαίνεται στην παρακάτω εικόνα (Σχήμα 3.14).



Σχήμα 3.14: Setup και hold time Πηγή: www.nandland.com.

Ο χρόνος τον οποίο τα δεδομένα μπορούν να καθυστερήσουν κατά την άφιξή τους στο flip -

Πίνακας 3.1: Ο πίνακας με τις χρονικές καθυστερήσεις.

α/α	**Slack**	**Time (nsec)**
1	Setup	15.738
2	Hold	0.376
3	Worst - Case	15.738

flop χωρίς να παραβιαστεί ο setup time ονομάζεται καθυστέρηση διάταξης setup slack. Από την άλλη, ο χρόνος στον οποίο τα δεδομένα πρέπει να είναι σταθερά πριν τον επόμενο ωρολογιακό παλμό προκειμένου να μην παραβιαστεί ο hold time ονομάζεται καθυστέρηση κράτησης. Οι καθυστερήσεις αυτές ορίζονται από τις ακόλουθες μαθηματικές σχέσεις [19]:

$$\boxed{\begin{aligned} S_{su} &= t_{req} - t_{arr} = \\ &= t_{clk} - t_{su} - t_{clkQ} - t_{del} \end{aligned}} \tag{3.1}$$

$$\boxed{\begin{aligned} S_h &= t_{arr} - t_{req} = \\ &= t_{clkQ} + t_{del} - t_h \end{aligned}} \tag{3.2}$$

t_{clkQ} είναι η χρονική διάρκεια μεταξύ διαδοχικών ανερχόμενων (ή κατερχόμενων αλλά εδώ επιλέχθηκαν οι πρώτοι) παλμών (ή ο χρόνος για να μεταβούν τα δεδομένα στον κτύπο του παλμού, στην έξοδο ενός flip - flop) και t_{del} αυτός που απαιτείται για τη μετάβαση των δεδομένων μέσω συνδέσεων ή πυλών από τη μία έξοδο στην επόμενη είσοδο.

Το Quartus παρέχει το εργαλείο Timing Analyzer, υπεύθυνο για τον ορισμό των χρονικών περιορισμών μέσω της δημιουργίας των *.sdc* αρχείων, την ανάλυση χρονισμού και τις αναφορές που σχετίζονται με αυτή [3]. Για το συγκεκριμένο project, οι καθυστερήσεις φαίνονται στον Πίνακα 3.1.

Θετικές setup και hold slack υποδεικνύουν ότι το σχέδιο πληροί τους χρονικούς περιορισμούς και μάλιστα έχει περιθώρια περεταίρω βελτίωσης ή δυνατότητες επιπλέον προσθηκών. Η worst - case slack αναφέρεται στην τιμή καθυστέρησης η οποία είναι πιο κρίσιμη για το σχέδιο. Τυπώνεται από τον Timing Analyzer αφού καταγράψει και συγκρίνει όλες τις καθυστερήσεις. Για την εργασία αυτή, ταυτίστηκε με την setup slack.

Δεδομένου ότι το ρολόι που χρησιμοποιήθηκε ήταν αυτό του ταλαντωτή της πλακέτας στη συχνότητα των $f = 50MHz$, η περίοδός του υπολογίστηκε σε:

$$\begin{aligned} T &= \frac{1}{f} \Leftrightarrow \\ \Leftrightarrow T &= \frac{1}{50MHz} \Leftrightarrow \\ \Leftrightarrow T &= 20nsec \end{aligned} \tag{3.3}$$

Επομένως, το πλάτος παλμού -δεδομένου ότι ο κύκλος λειτουργίας του ήταν 50%- ήταν $10nsec$. Η ανάλυση χρονισμού δίνει και το ελάχιστο πλάτος παλμού (minimum pulse width), η οποία είναι η ελάχιστη χρονική διάρκεια που απαιτείται το οποιοδήποτε σήμα να μείνει σε κατάσταση λογικού ΄1΄ ή ΄0΄ ώστε να αναγνωριστεί σωστά από τα στοιχεία του κυκλώματος. Η τιμή που προέκυψε είναι:

$$\boxed{w_{min} = 9.405nsec} \tag{3.4}$$

Η γνώση της τιμής της ποσότητας αυτής είναι σημαντική για την αποφυγή καταστάσεων μεταστα-θερότητας και διαφθοράς των δεδομένων. Υψηλότερες τιμές καταδεικνύουν καλύτερη σταθερότητα του συστήματος.

Τέλος, οι αναφορές της χρονικής ανάλυσης παρέχουν και τη μέγιστη επιτρεπτή συχνότητα, f_{max}, η οποία είναι η μέγιστη συχνότητα στην οποία το σχέδιο μπορεί να είναι αξιόπιστα λειτουργικό, χωρίς να παραβιάζει τους χρονικούς περιορισμούς. Με τη σειρά της, αυτή υπολογίζεται από τη σχέση [19]:

$$\boxed{f_{max} = \frac{1}{T_{clkQ} + T_{logic} + T_{routing} + T_{setup} - T_{skew}}} \tag{3.5}$$

Όπου:

$t_{routing}$ είναι ο ελάχιστος χρόνος που απαιτείται ώστε τα δεδομένα να φτάσουν, πριν την άνοδο του επόμενου παλμού, στην είσοδο D, δηλαδή ο χρόνος που απαιτείται για να διαδοθεί ένα σήμα μεταξύ των διάφορων διασυνδέσεων των λογικών στοιχείων,

T_{logic} είναι η καθυστέρηση διάδοσης μεταξύ των flip - flop, η χρονική καθυστέρηση της συνδυαστικής λογικής

και τέλος T_{skew} είναι η καθυστέρηση διάδοσης μεταξύ του αρχικού και του τελικού flip - flop του σχεδίου [19].

Για το σχέδιο της εργασίας αυτής, ο Timing Analyzer υπολόγισε ότι:

$$\boxed{f_{max} = 234.63MHz} \tag{3.6}$$

Επομένως, η γεννήτρια δύναται να λειτουργήσει ως αυτή τη συχνότητα, η οποία όμως είναι πολύ μεγαλύτερη από αυτές που απαιτούνται, αφήνοντας έτσι ανοικτό το περιθώριο για περεταίρω βελτιώσεις, τροποποιήσεις ή προσθήκες στο project (βλ. Κεφάλαιο 4).

3.3 Προγραμματισμός

Ο προγραμματισμός μιας FPGA αναφέρεται στη ρύθμιση του υλικού της με τρόπο τέτοιο ώστε να πραγματοποιεί το κάθε φορά επίδικο ενός σχεδίου. Σε αυτό το σημείο ακριβώς είναι που παίζει το βασικότερο ρόλο η επαναπρογραμματισιμότητά τους, για την οποία έγινε αναφορά σε προηγούμενες ενότητες (βλ. 1.2). Σε κάθε περίπτωση, πριν ένα σχέδιο φτάσει στο σημείο να προγραμματιστεί επί της διάταξης, προηγείται το στάδιο της προσομοίωσης και της επαλήθευσης της ορθής λειτουργίας του μέσω του verification (βλ. 3.1) [23].

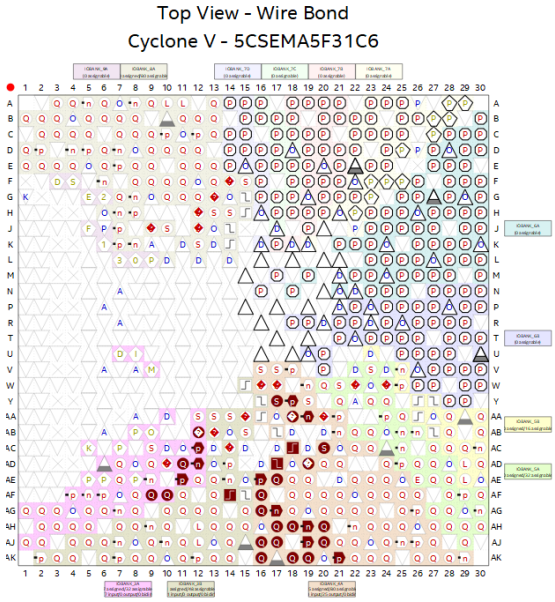
Εκτός από το ότι οι FPGA μπορούν να προγραμματιστούν συνεχώς και με διαφορετικούς κάθε φορά στόχους, η διαδικασία αυτή μπορεί να γίνει και με διαφορετικούς τρόπους, όπως π.χ. η JTAG (Joined Test Action Group) ή διεπαφές επάνω στην πλατφόρμα [23]. Για το συγκεκριμένο project επιλέχθηκε ο πρώτος τρόπος. Για την αλληλεπίδραση της συσκευής με τον προγραμματιστή απαιτείται η αντιστοίχιση των ψηφιακών εισόδων κι εξόδων με πραγματικές, όπως είναι οι διακόπτες, τα κουμπιά, τα LED ή οι GPIO. Αυτό αποτελεί αντικείμενο της αντιστοίχισης ακροδεκτών (pin planning) [23].

Το αντίστοιχο εργαλείο που παρέχει το Quartus είναι ο Pin Planner. Καθώς το εγχειρίδιο χρήσης της πλακέτας περιλαμβάνει τα ακριβή ονόματα των σημάτων, η διαδικασία αυτή επαφίεται στο χρήστη κι εξαρτάται όχι μόνο από τις προτιμήσεις ή τη βολικότητα που παρέχει ο εκάστοτε ακροδέκτης αλλά κι από παράγοντες που σχετίζονται με το χρόνο [3]. Επιλέγοντας ακροδέκτες κι αντιστοιχώντας τους σε λειτουργίες ο Pin Planner εγγυάται την ορθή συνδεσιμότητα κι επιτρέπει την οπτική επιβεβαίωση των συνδέσεων μεταξύ σχεδίου και φυσικής διάταξης.

Πίνακας 3.2: Πίνακας αντιστοίχισης διακοπτών - λειτουργιών.

Διακόπτης	**'1'**	**'0'**
SW[0]	Clocks - Reset	Clocks - No Reset
SW[1]	1 Hz Frequency Select	200 Hz Frequency Select
SW[2]	Counter - Reset	Counter - No Reset
SW[3]	Counter - Enable	Counter - No Enable
SW[4]	Sin ROM - Enable	Sin ROM - No Enable
SW[5]	Triangular ROM - Enable	Triangular ROM - No Enable
SW[6]	Sawtooth ROM - Enable	Sawtooth ROM - No Enable

Στα πλαίσια της παρούσας εργασίας, εγινε η αντιστοίχιση ακροδεκτών της παρακάτω εικόνας (Σχήμα 3.15).



Σχήμα 3.15: Η αντιστοίχιση ακροδεκτών για την τρέχουσα εργασία.

Πέραν του προφανούς σκοπού της, η εικόνα αποτελεί και μια ένδειξη των δυνατοτήτων τόσο των FPGA γενικά όσο και του DE1 - SoC ιδιαίτερα.

Ο Πίνακας 3.2 συνοψίζει τις εισόδους των διακοπτών με τις αντίστοιχες λειτουργίες τους.

Ο προγραμματισμός της FPGA έγινε μέσω JTAG, η οποία είναι μια πρότυπη μέθοδος ελέγχου κι επαλήθευσης ηλεκτρονικών διατάξεων, ιδιαίτερα ολοκληρωμένων κυκλωμάτων, μικροελεγκτών, επεξεργαστών κι FPGA. Επιτρέπει στους προγραμματιστές να δοκιμάσουν τα κυκλώματα κατά τη διαδικασία της παραγωγής, με σκοπό να επιβεβαιωθεί η ορθή λειτουργία τους. Είναι αρκετά αποτελεσματική στην εξεύρεση ελαττωμάτων στους κώδικες ή το υλικό. Ταυτόχρονα, χρησιμοποείται ευρέως και γι απο-

σφαλμάτωση σε πραγματικό χρόνο (real time debugging) υλικού, λογισμικού ή υλικολογισμικού κατά τη φάση της ανάπτυξής τους. Τέλος, σαν μέθοδος χρησιμοποιείται για τον προγραμματισμό και τη ρύθμιση configuration των μνημών ή των FPGA, όπως ακριβώς έγινε και στα πλαίσια της εργασίας αυτής [23].

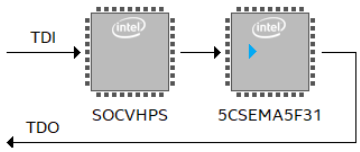
Τα βασικά στοιχεία της μεθόδου JTAG είναι τα κάτωθι [23]:

- Το TAP (Test Access Port) που λειργουγεί εν είδει διεπαφής μεταξύ των ακροδεκτών JTAG και του εσωτερικού κυκλώματος της διάταξης.
- Το TCK (Test Clock) που παρέχει το χρονισμό για την αποστολή των δεδομένων.
- Την TDI (Test Data In) είσοδο.
- Την TDO (Test Data Out) έξοδο.
- Το TMS (Test Mode Select) που ελέγχει τις μεταβάσεις κατάστασης του TAP.

Οι διατάξεις με πολλαπλά στοιχεία που επιτρέπουν τη JTAG συνδέδονται με τη μορφή αλυσίδας (chain). Έτσι, η αρχιτεκτονική της μεθόδου χρησιμοποιεί σειριακούς καταχωρητές ολίσθησης, διάταξης που επιτρέπει τη σειριακή έξοδο των δεδομένων συχρονισμένη με ένα ρολόι [23].

Για τον προγραμματισμό της συγκεκριμένης FPGA, κατά τη διαδικασία της σύνθεσης κι αφού πραγματοποιήθηκε η αντιστοίχιση ακροδεκτών, δημιουργήθηκε το αρχείο επέκτασης *.sof*, αυτό που περιέχει όλη την πληροφορία του σχεδίου σε δυαδική μορφή. Η πλατφόρμα συνδέθηκε σε Η/Υ μέσω USB - Blaster II καλωδίου [3].

Υπό το χρονισμό του TCK, το bitstream των δεδομένων του αρχείου αυτού ολισθαίνει σειριακά μέσω του TDI στην FPGA. Με τη σειρά της, η συσκευή ρυθμίζει το εσωτερικό της κύκλωμα ώστε να ανταποκρίνεται στα δεδομένα αυτά. Τέλος, τα δεδομένα εξέρχονται επίσης σιεριακά στην έξοδο TDO με σκοπό την επαλήθευση του ορθού προγραμματισμού (Σχήμα 3.16).



Σχήμα 3.16: Προγραμματισμός της DE1 - SoC Cyclone V FPGA μέσω JTAG.

Κεφάλαιο 4

Συμπεράσματα

4.1 Χρησιμότητα των FPGA

Το τελευταίο αυτό κεφάλαιο αναφέρεται στα συμπεράσματα της πονήματος αυτού και ταυτόχρονα φιλοδοξεί να θέσει τα θεωρητικά σημεία εκκίνησης βελτίωσής του.

Αντικειμενικός σκοπός της παρούσας εργασίας ήταν -όπως τέθηκε ξεκάθαρα στην ενότητα 1.3- η σχεδίαση κι η υλοποίηση μιας γεννήτριας κυματομορφών επί της πλακέτας DE1 - SoC 5CSEMA5F31C6N, σχεδιασμένη γύρω από την Cyclone V FPGA. Η γεννήτρια θα ήταν σε θέση να παράγει τέσσερις διαφορετικές κυματομορφές (τετραγωνική, ημιτονοειδή, τριγωνική και πριονωτή) σε δύο διαφορετικές συχνότητες ($1Hz$ και $200Hz$). Η επιλογή της κυματομορφής και της συχνότητας εξόδου θα ήταν απόλυτα ελέγξιμες από το χρήστη. Μέσω πολλαπλών διαδοχικών σταδίων που περιλαμβάνουν τη συγγραφική κώδικα σε VHDL και MATLAB, τη σχεδίαση σχηματικού διαγράμματος, την προσομοίωση, την ανάλυση χρονισμού και την υλοποίηση, ο σκοπός αυτό επετεύχθη.

Το πρώτο και βασικότερο συμπέρασμα είναι αυτό που θεμελιώθηκε στην Ενότητα 1.2 για τις FPGA: η επαναπρογραμματισιμότητά τους. Εκεί, είχε αναφερθεί πως, για την υλοποίηση ενός σχεδίου, αποτελούν συμφέρουσες επιλογές καθώς επιτρέπουν γρήγορες τροποποιήσεις, βελτιώσεις ή προσθήκες και σε καμία περίπτωση δεν επιβαρύνουν το σχεδιαστή με το επαναλαμβανόμενο κόστος αγοράς νέου υλικού [4][22][41].

Αυτό κατέστη από πολύ νωρίς σαφές στο συγκεκριμένο έργο. Πριν την τελική μορφή του που αποδίδεται μέσω του κειμένου και των εικόνων αυτής της εργασίας (βλ. Παραρτήματα), υπήρξαν αρκετά σχέδια που τροποποιήθηκαν αρκετές φορές και σε μεγάλο επίπεδο. Από ένα χρονικό σημείο και μετά (σταθμός του οποίου υπήρξε η εξοικείωση με το λογισμικό που χρησιμοποιήθηκε), η διαδικασία αυτή γινόταν αυτόματα και σε σύντομο χρονικό διάστημα αφού κάθε αποτυχημένη απόπειρα άφηνε παρακαταθήκη λειτουργικό υλικό (αρχεία κώδικα, χρονισμού, σχηματικού, προσομοίωσης κλπ.). Αν και δεν δοκιμάστηκε, θεωρείται σχεδόν βέβαιο πως αυτό δε θα ήταν το ίδιο εύκολο εάν το σχέδιο γινόταν χωρίς τη χρήση FPGA αλλά μόνο με μεικτή κι αναλογική και ψηφιακή σχεδίαση -παρά το όποιο επιστημονικό ενδιαφέρον που παρουσιάζει και την τεράστια θεωρητική και πρακτική σημασία της τεχνοτροπίας αυτής.

Ένα δεύτερο βασικό συμπέρασμα είναι αυτό που σχετίζεται άμεσα με τις υπόλοιπες (πλήν επαναπρογραμματισμότητας) δυνατότητες των FPGA [4][22][41]. Ένα σχέδιο μπορεί να κατατμηθεί σε μικρότερα, συνήθως πιο γρήγορα επιτεύξιμα κι ευκολότερα στο χειρισμό, τα οποία στο τέλος να συντεθούν σε μια ενιαία, κυρίαρχη δομή. Αυτό αποτελεί ασύγκριτης σημασίας πλεονέκτημα αφού επέτρε-

ψε την έγκαιρη ενημέρωση για σφάλματα, παρείχε συνεχώς τις απαραίτητες προειδοποιήσεις κι ήταν πολύ ευκολότερα προσομοιώσιμη. Η ίδια διαδικασία με χρήση μικροεπεξεργαστών θα ήταν αρκετά δυσκολότερη ενώ με μεικτή σχεδίαση θα ήταν μεταξύ αυτών κι αρκετά πιο χρονοβόρα.

4.2 Τροποποιήσεις στο σχέδιο

Όπως κάθε ανθρώπινο έργο, έτσι και το τρέχον, είναι ευαίσθητο σε σφάλματα και παραλείψεις.

Οι σουίτες λογισμικού που συνοδεύουν τις αναπτυξιακές FPGA είναι ικανές στις πλείστες των περιπτώσεων να αναγνωρίζουν αυτά τα σφάλματα εάν είναι αρκετά σημαντικά ή να παρέχουν τις απαραίτητες (κριτικές ή μη) προειδοποιήσεις αν κάποιο τμήμα του σχεδίου δεν εξελίσσεται σωστά. Σε κάθε περίπτωση όμως αποτελούν εξίσου ανθρώπινα κατασκευάσματα και μάλιστα λογισμικό Η/Υ, συνεπώς δεν είναι σε θέση να κρίνουν πάντοτε αξιόπιστα την ποιότητα ή την ορθότητα αυτού που ο σχεδιαστής έχει κατά νου ή προσπαθεί να υλοποιήσει.

Κατά τη διάρκεια της αναπτυξιακής διαδικασίας ανέκυψαν διάφορα ζητήματα πολλά από τα οποία διέφυγαν της προσοχής του Quartus αλλά είχαν ως αποτέλεσμα τον εσφαλμένο προγραμματισμό που οδήγησε σε κακή λειτουργία του ίδιου του project. Η πληθώρα αυτών των λαθών οφειλόταν στον κώδικα του γράφοντα κι όχι κατ΄ ανάγκη στο λογισμικό ή το υλικό και περιστρέφονταν γύρω από δύο ζητήματα: τη φύση των δεδομένων εντός των ROM και την παραβίαση των χρονικών περιορισμών.

Όσον αφορά τις τρεις μνήμες, αυτές περιέχουν πίνακες με τα δεδομένα που ελήφθησαν από το MATLAB αντιστοιχισμένα σε ακολουθίες bit. Αν και δεν αποτελεί πρόβλημα σε επίπεδο λειτουργίας της ίδιας της γεννήτριας, ο κώδικας θα μπορούσε να βελτιωθεί με αρκετούς τρόπους, όπως λ.χ. η συμπερίληψη των δεδομένων σε μορφή ακεραίων τόσο για χάρη της ευμορφίας όσο και για την ευκολία της συντήρησης (βλ. ενότητα 4.3).

Το δεύτερο ζήτημα έχει να κάνει τόσο με τη χρήση των στοιχείων IP (Intellectual Property) όσο και με τους χρονικούς περιορισμούς του έργου. Αρκετές εταιρείες ανάπτυξης και παραγωγής FPGA παρέχουν έτοιμες βιβλιοθήκες με προσχεδιασμένα στοιχεία τα οποία μπορούν να ενσωματωθούν σε ένα ευρύτερο σχέδιο. Το αποτέλεσμα είναι η αισθητή μείωση του χρόνου ανάπτυξης του σχεδίου (εφόσον δεν απαιτείται να δημιουργηθεί «από το μηδέν») κι η παραγωγή ποιοτικότερων έργων, εφόσον τα στοιχεία των βιβλιοθηκών αυτών έχουν αναπτυχθεί από έγκυρους κατασκευαστές (π.χ. AMD, Intel, Lattice, Microchip κλπ.) κι έχουν δοκιμαστεί πολλαπλώς [16][19][23][31][41]. Από την άλλη, όσον αφορά του χρονικούς περιορισμούς, ένα αρκετά συχνό λάθος των σχεδιαστών, ιδίως των μη έμπειρων, είναι η μη ενασχόλησή τους με αυτούς. Τούτο προκύπτει ως αποτέλεσμα της εξοικείωσής τους με μικρά έργα, όπου τα ζητήματα αυτά σπανίως να αποτελούν τροχοπέδη της όλης υλοποίησης, όσο όμως αυξάνονται σε πολυπλοκότητα, τόσο αυξάνεται κι η σημασία τους.

Με διάθεση εμβάθυνσης και περεταίρω ενασχόλησης με τη γλώσσα VHDL, οι διαιρέτες συχνότητας συνεγράφησαν σε αυτή. Μια αποτελεσματικότερη (κι ίσως ευκολότερη) μέθοδος όμως θα μπορούσε να είναι, όπως αναφέρθηκε και πιο πριν, η χρήση ενός PLL από τη βιβλιοθήκη του Quartus, όπου αντί γι αρχεία κώδικα, θα γίνονταν μικρές τροποποιήσεις στη διαίρεση του αρχικού ρολογιού του σχεδίου ενώ ταυτόχρονα θα ανανγωριζόταν αυτόματα από τον Timing Analyzer. Παρόλο που επελέγη ο δρόμος της συγγραφής και του *.sdc* αρχείου, πριν αυτό συμβεί, κατά τις δοκιμές της συμπεριφοράς της προγραμματισμένης γεννήτριας ανέκυπταν ζητήματα.

Λίγο - πολύ, το ίδιο μπορεί αν ειπωθεί και για τον πολυπλέκτη - επιλογέα συχνοτήτων. Ο Timing Analyzer ακολουθεί μια εσωτερική διαδικασία κρίσης των σημάτων ως ρολόγια ή όχι, η οποία εξαρτάται από ποικίλους παράγοντες όπως τα ονόματά τους, η συχνότητα εναλλαγών μεταξύ ′0′ κι ′1′, η συσχέτισή τους με πολυπλέκτες κλπ. Για τις διατάξεις αυτές που βρίσκονται ενσωματωμένες στις βιβλιοθήκες του Quartus, η διαδικασία είναι αισθητά ευκολότερη από ό,τι για τα components που έχουν δημιουργηθεί

από αρχεία *.vhd*. Μάλιστα, προκειμένου, λ.χ., ο διακόπτης SW[1] να μην αναγνωριστεί ως ρολόι, πρέπει χειροκίνητα να οριστεί false path στο αρχείο περιορισμών του σχεδίου ή μία συχνότητα πολύ μεγάλη (σε *sec*), ώστε ο Timing Analyzer να το αγνοήσει.

4.3 Μελλοντικές προσθήκες

Στην ενότητα 3.2 γίνεται μια αναφορά στο χρονισμό του σχεδίου. Εκεί, προέκυψε το συμπέρασμα ότι οι ληφθέντες χρόνοι καθυστέρησης καθώς επίσης κι η μέγιστη συχνότητα του ρολογιού έχουν τιμές τέτοιες που να επιτρέπουν την επέκταση του σχεδίου περεταίρω. Έχοντας κατά νου αυτό στην τελευταία αυτή ενότητα ακολουθούν κάποιες προτάσεις με πιθανές προσθήκες στο σχέδιο.

4.3.1 Επιπλέον συχνότητες

Η ευκολότερη ίσως προσθήκη είναι αυτή των επιπλέον συχνοτήτων στο σχέδιο. Αυτό μπορεί να πραγματοποιηθεί είτε με τη δημιουργία περισσότερων διαιρετών τάσης, είτε με τη χρήση PLL.

Η τροποποίηση αυτή είναι αρκετά άμεση και θα μπορούσε να εμπλουτίσει το σχέδιο, ωστόσο πρέπει να γίνει με σεβασμό στα όριά του και με τον απαραίτητο έλεγχο χρονισμού, αφού η αύξηση της πολυπλοκότητάς του (ακόμα και με αυτόν τον χρονισμό περισσότερων στοιχείων από το ίδιο ρολόι) επιβαρύνει κατά τι το σύστημα.

4.3.2 Επέκταση μνήμης

Μία ακόμη πιθανή τροποποίηση του σχεδίου μπορεί να είναι αυτή του μεγέθους των υπάρχουσων ROM.

Αυτές συνεγράφησαν σε VHDL και μάλιστα στα αντίστοιχα αρχεία τους έχει οριστεί το μήκος της διεύθυνσης κάθε μίας να είναι ίσο με 7 bit. Αυτό σημαίνει ότι έχουν 128 θέσεις μνήμης. Σε περίπτωση που απαιτηθεί από μελλοντικά σχέδια, τροποποίηση του φυσικού αυτού αριθμού αυξάνει ή μειώνει αντίστοιχα τον αριθμό των δεδομένων που μπορούν να αποθηκευτούν εντός τους, εν είδει LUT. Βεβαίως, αντίστοιχη τροποποίηση πρέπει να υπάρξει και στο μήκος των δεδομένων, αφού μεγαλύτεροι ή μικρότεροι αριθμοί απαιτούν διαφορετικό αριθμό δυαδικών ψηφίων ώστε να περιγραφούν. Στη συγκεκριμένη περίπτωση, τα 8 bit που είναι το μήκος των δεδομένων δύνανται να αναπαραστήσουν έως και τον αριθμό 255 ενώ για τον αμέσως επόμενο απαιτούνται 9 bit κ.ο.κ..

Ο εμπλουτισμός των πινάκων με περισσότερα στοιχεία απαιτεί ασφαλώς με τη σειρά του εκτενέστερη δειγματοληψία στο MATLAB. Ταυτόχρονα, ενώ, όπως έχει ήδη ειπωθεί, όσες περισσότερες τιμές έχει η μνήμη για την κκυματομορφή, τόσο ποιοτικότερα θα την αναπαριστά, αυτό θα συνεπάγεται το αντίστοιχο κόστος σε χρόνους σύνθεσης και κατανάλωσης πόρων του υλικού.

4.3.3 Επιπλέον κυματομορφές

Οι τέσσερις κυματομορφές που παρήχθησαν για τις ανάγκες της εργασίας αυτής δεν είναι οι μοναδικές. Υπάρχει μια πληθώρα κυματομορφών που συνηθίζουν να αναπαρίστανται και δεν περιλαμβάνονται εδώ, όπως τα παλμοκύματα (pulse wave), οι ράμπες (ramp wave), τα κύματα θορύβου, οι γκαουσιανοί παλμοί, οι ψευδοτυχαίες κυματομορφές (pseudo - random wave), οι συναρτήσεις βήματος step function κλπ.

Το MATLAB περιέχει στις βιβλιοθήκες του αρκετές από αυτές τις συναρτήσεις κι επομένως, με το κατάλληλο script μπορούν να αναπαρασταθούν και να ληφθούν δείγματα. Αυτά, με τη σειρά τους, μπορούν να τοποθετηθούν στις μνήμες με τρόπο παρόμοιο με αυτόν που χρησιμοποιήθηκε εδώ.

Βεβαίως, πρέπει να ομολογηθεί ότι ο τρόπος αυτός δεν είναι αποδοτικός για όλες τις προαναφερθείσες κυματομορφές. Λ.χ. μια τυχαία κυματομορφή είναι ένα σήμα το οποίο δεν ακολουθεί ένα συγκεκριμένο, προκαθορισμένο μοτίβο ή συνάρτηση. Εν αντιθέσει με τις τυπικές κυματομορφές, όπως αυτές που εντάχθηκαν στο πλαίσιο της εργασίας αυτής, οι τυχαίες είναι πιο ευέλικτες και μπορούν να πάρουν οποιοδήποτε σχήμα ή μορφή.

Τυπικά, μία ROM σαν αυτές που δημιουργήθηκαν στα πλαίσια της εργασίας αυτής περιέχει προκαθορισμένες τιμές. Αυτό έρχεται σε αντίθεση με τον ορισμό των τυχαίων κυματομορφών που αναφέρθηκε προηγουμένως, κάνοντάς τες ακατάλληλες για τη δημιουργία τους.

Από την άλλη, ένα ramp wave είναι παρόμοιο με μια πριονωτή συνάρτηση και μάλιστα οι δύο όροι χρησιμοποιούνται στη βιβλιογραφία εναλλάξ. Όταν γίνεται διάκριση μεταξύ τους, αυτή αφορά στο ότι το πριονωτό κύμα αυξάνεται γραμμικά κι ύστερα επανέρχεται απότομα στην αρχική του κατάσταση, ενώ η ράμπα μπορεί να μειώνεται επίσης γραμμικά αλλά με κλίση διαφορετική από αυτή με την οποία ανερχόταν (έτσι, διαφέρει κι από την τριγωνική συνάρτηση). Ένας τέτοιος παλμός είναι απολύτος πραγματοποιήσιμος με τη βοήθεια μιας ROM, χωρίς να απαιτούνται κατ΄ ανάγκη άλλες τροποποιήσεις στο σχηματικό ή προσθήκη διαφορετικών αλγορίθμων.

Κάτι αντίστοιχο ισχύει και για το παλμικό κύμα. Ένα τέτοιο κύμα προσομοιάζει τον τετραγωνικό παλμό, με τη διαφορά ότι ο κύκλος λειτουργίας του δεν είναι 50%. Η υλοποίηση ενός τέτοιου σήματος δεν απαιτεί καν τη χρήση μνήμης, αφού αρκεί απλώς η τροποποίηση λειτουργίας των διαιρετών τάσης σύμφωνα με τις απαιτήσεις του σχεδιαστή.

Τέλος, θα μπορούσε να υλοποιηθεί και μια ψευδο - τυχαία κυματορφή. Πρόκειται περί σήματος που μοιάζει τυχαίο όπως πηγάζει από κάποια αιτιοκρατική διαδικασία, μιμείται δηλαδή την τυχαιότητα. Συνήθως αυτές οι κυματομορφές παράγονται με τη χρήση αλγορίθμων που ονομάζονται γεννήτορες ψευδοτυχαίων αριθμών (PRNG, pseudo - random number generators). Αυτοί παράγουν αριθμητικές ακολουθίες φαινομενικά τυχαίες αλλά απολύτως αναπαράξιμες αφού καθορίζονται εν πολλοίς από τις αρχικές τους συνθήκες. Για την πραγματοποίηση ενός τέτοιου στοιχείου, εκτός από τη ROM θα απαιτούνταν κι ο μηχανισμός παραγωγής των ψευδοτυχαίων αριθμών. Ένας τέτοιος μηχανισμός μπορεί να γραφτεί σε γλώσσα προγραμματισμού.

4.3.4 Χρήση DAC

Η βασικότερη ίσως προσθήκη που θα μπορούσε να γίνει στο σχέδιο είναι αυτή ενός DAC.

Στην υποενότητα 3.1.4 παρουσιάστηκαν τρία διαφορετικά σχήματα (Σχήματα 3.7, 3.10 και 3.13) τα οποία δείχνουν την αναπαράσταση των τριών κυματομορφών σε αναλογική μορφή στον προσομοιωτή Questa FPGA.

Οι αναπαραστάσεις είναι αρκετά ευκρινείς και πράγματι αντιπροσωπεύουν πιστά το σκοπό για τον οποίο δημιουργήθηκαν, ωστόσο μια λεπτομερέστερη παρατήρηση ανακαλύπτει το θεμελιώδες τους «ελάττωμα»: πρόκειται περί ψηφιακών σημάτων, τα οποία έχουν χειραγωγηθεί με τρόπο τέτοιο ώστε να προσομοιάζουν αναλογικά.

Προκειμένου να αποφευχθεί η μορφή αυτή, σε βήματα, κρίνεται αναγκαία η χρήση ενός DAC, ο οποίος θα λαμβάνει το ψηφιακό σήμα από τις εξόδους της FPGA (που αναπαριστά διακριτές τιμές) και θα το μετατρέπει σε αναλογικό, προσεγγίζοντας αυτή την τιμή με κάποια αναλογική τάση ή ρεύμα. Με αυτό τον τρόπο, η έξοδος του DAC θα παράγει ένα πιστότερο, συνεχές σήμα.

Όσον αφορά τις εισόδους και τις εξόδους του, η πλατφόρμα περιέχει αρκετούς ακροδέκτες που μπορούν να χρησιμοποιηθούν για το σκοπό αυτό ενώ ταυτόχρονα υποστηρίζει κι αρκετά πρωτόκολλα επικοινωνίας μεταξύ αυτής και του DAC, όπως SPI (Serial Peripheral Interface), I2C (Inter - Inegrated Curcuit), UART (Universal Asynchronous Receiver - Transmitter), USB (Universal Serial Bus) και

βεβαίως τα GPIO, για τα οποία ήδη έχει γίνει λόγος στις προηγούμενες ενότητες.

Τα πρωτόκολλα αυτά προσφέρουν ευελιξία στη διεπαφή της FPGA με εξωτερικές συσκευές, κάνοντάς την τελικώς κατάλληλη για το σκοπό της ίδιας της εργασίας.

4.3.5 Έλεγχος μέσω UI

Η απευθείας χρήση της πλακέτας για τη λειτουργίας της γεννήτριας κυματομορφών είναι μια βολική λύση σε επίπεδο σχεδίασης.

Μια ενδιαφέρουσα ωστόσο προσθήκη θα μπορούσε να είναι ο προγραμματισμός ενός GUI, ελεγχόμενο από το χρήστη μέσω Η/Υ.

Η επικοινωνία μεταξύ της FPGA και του υπολογιστή θα μπορούσε να καταστεί δυνατή είτε μέσω UART, είτε μέσω USB. Το αναπτυξιακό εργαλείο προσφέρει αμφότερες τις επιλογές. Το UART για παράδειγμα θα μπορούσε να είναι το πρωτόκολλο επικοινωνίας μεταξύ υπολογιστή και συσκευής ώστε ο χειρισμός της να γίνεται απομακρυσμένα, από τον ίδιο τον υπολογιστή.

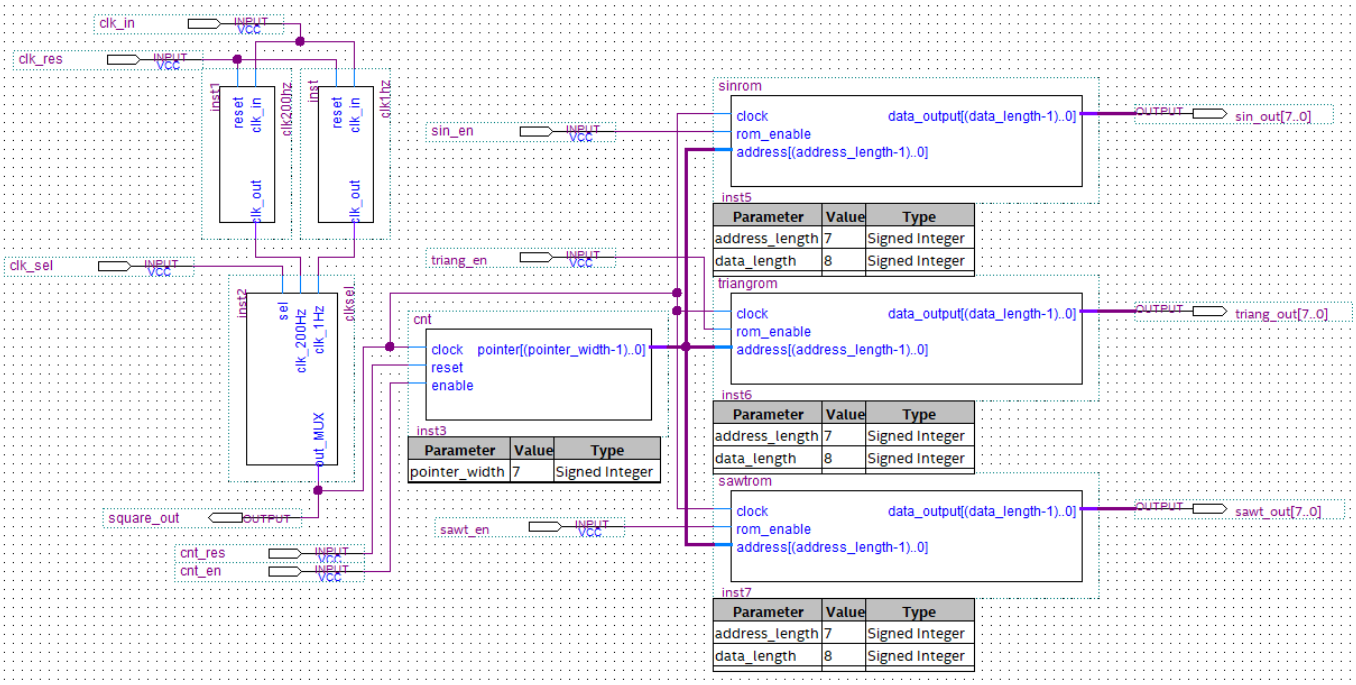
Η διαδικασία αυτή βεβαίως θα απαιτούσε την υλοποίηση ενός συστήματος εντός του σχεδίου που θα μπορεί να διαβάζει τα σήματα αυτά και να τα ερμηνεύει ως τις αντίστοιχες εντολές επιλογής.

Όσον αφορά το γραφικό περιβάλλον, αυτό δύναται να γραφτεί σε μια πληθώρα γλωσσών, όπως οι Python, Visual Basic ή LabView. Η επιλογή επαφίεται στις προτιμήσεις και τις γνώσεις του σχεδιαστή. Αξίζει βέβαια να αναφερθεί ότι λ.χ. η Visual Basic κι η Python παρέχουν τα εργαλεία για τη δημιουργία ενός GUI, με τη πρώτη μάλιστα να πρωτεύει σε αυτό, παρά την παλαιότητά της.

Από την άλλη, η Labview είναι μια γλώσσα που σχεδιάστηκε, σε τελική ανάλυση, για επιστημονικές εφαρμογές, η οποία παρέχει όχι μόνο εκτεταμένες βιβλιοθήκες κι οδηγούς για την επικοινωνία με το υλικό αλλά και τη δυνατότητα οπτικοποίησης των δεδομένων.

Παράρτημα Α΄

Σχηματικό διάγραμμα



Παράρτημα Β΄

Κώδικες

Β΄.1 Διαιρέτης συχνότητας 1 Hz

```
library ieee;
use ieee.std_logic_1164.all;
entity clk1hz is
        port (clk_in: in std_logic;
              reset: in std_logic;
              clk_out: out std_logic);
end clk1hz;
architecture behavioral of clk1hz is
        signal temp: std_logic := '0';
        signal count: integer := 0;
begin
        frequency_divider: process (reset, clk_in)
                begin
                        if (reset = '1') then
                                temp <= '0';
                                count <= 0;
                        elsif rising_edge (clk_in) then
                                count <= count + 1;
                                if (count = 24999999) then
                                        temp <= not temp;
                                        count <= 0;
                                end if;
                        end if;
                clk_out <= temp;
                end process;
end behavioral;
```

Β΄.2 Διαιρέτης συχνότητας 200 Hz

```
library ieee;
use ieee.std_logic_1164.all;
entity clk200Hz is
        port (clk_in: in std_logic;
              reset: in std_logic;
              clk_out: out std_logic);
end clk200Hz;
architecture behavioral of clk200Hz is
        signal temp: std_logic := '0';
        signal count: integer := 0;
begin
        frequency_divider: process (reset, clk_in)
                begin
                        if (reset = '1') then
                                temp <= '0';
                                count <= 0;
                        elsif rising_edge (clk_in) then
                                count <= count + 1;
                                        if (count = 124999) then
                                                temp <= not (temp);
                                                count <= 0;
                                        end if;
                        end if;
                clk_out <= temp;
                end process;
end behavioral;
```

Β΄.3 Πολυπλέκτης - επιλογέας συχνοτήτων

```
library ieee;
use ieee.std_logic_1164.all;
entity clksel is
    port(clk_1Hz : in std_logic;
         clk_200Hz : in std_logic;
         sel : in std_logic;
         out_MUX : out std_logic
         );
end entity clksel;
architecture behavioral of clksel is
begin
    process(clk_1Hz, clk_200Hz)
    begin
        if sel = '0' then
            out_MUX <= clk_200Hz;
        else
            out_MUX <= clk_1Hz;
        end if;
    end process;
end architecture behavioral;
```

Β΄.4 Απαριθμητής - δείκτης διευθύνσεων

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity cnt is
    generic(pointer_width: integer := 7);
    port(clock: in std_logic;
         reset: in std_logic;
         enable: in std_logic;
         pointer: out std_logic_vector(pointer_width - 1 downto 0)
         );
end cnt;
architecture behavioral of cnt is
    signal count: unsigned(pointer_width downto 0) := (others => '0');
begin
    process(clock, reset)
    begin
        if reset = '1' then
            count <= (others => '0');
        elsif rising_edge(clock) then
            if enable = '1' then
                if count = to_unsigned(2**pointer_width - 28, pointer_width) then
                    count <= (others => '0');
                else
                    count <= count + 1;
                end if;
            end if;
        end if;
    end process;
    pointer <= std_logic_vector(count(pointer_width - 1 downto 0));
end architecture;
```

Β΄.5 Μνήμη ημιτονοειδούς κυματομορφής

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity sinrom is
        generic(address_length: natural := 7;
                data_length: natural := 8);
        port(clock: in std_logic;
             rom_enable: in std_logic;
             address: in std_logic_vector((address_length - 1) downto 0);
             data_output: out std_logic_vector((data_length - 1) downto 0));
end sinrom;
architecture behavioral of sinrom is
    type rom_type is array (0 to (2**(address_length) - 28)) of
    std_logic_vector((data_length - 1) downto 0);
    constant mem: rom_type :=
("10000000", "10001000", "10001111", "10010111",
"10011111", "10100111", "10101110", "10110110",
"10111101", "11000100", "11001010", "11010001",
"11010111", "11011100", "11100010", "11100111",
"11101011", "11101111", "11110011", "11110110",
"11111001", "11111011", "11111101", "11111110",
"11111111", "11111111", "11111111", "11111110",
"11111101", "11111011", "11111001", "11110110",
"11110011", "11101111", "11101011", "11100111",
"11100010", "11011100", "11010111", "11010001",
"11001010", "11000100", "10111101", "10110110",
"10101110", "10100111", "10011111", "10010111",
"10001111", "10001000", "10000000", "01110111",
"01110000", "01101000", "01100000", "01011000",
"01010001", "01001001", "01000010", "00111011",
"00110101", "00101110", "00101000", "00100011",
"00011101", "00011000", "00010100", "00010000",
"00001100", "00001001", "00000110", "00000100",
"00000010", "00000001", "00000000", "00000000",
"00000000", "00000001", "00000010", "00000100",
"00000110", "00001001", "00001100", "00010000",
"00010100", "00011000", "00011101", "00100011",
"00101000", "00101110", "00110101", "00111011",
"01000010", "01001001", "01010001", "01011000",
"01100000", "01101000", "01110000", "01110111",
"10000000");
```

```
begin
p_table: process(clock)
begin
    if rising_edge(clock) then
        if rom_enable = '1' then
            data_output <= mem(to_integer(unsigned(address)));
        else
            data_output <= (others => '0');
        end if;
    end if;
end process p_table;
end architecture;
```

Β΄.6 Μνήμη τριγωνικής κυματομορφής

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity triangrom is
        generic(address_length: natural := 7;
                data_length: natural := 8);
        port(clock: in std_logic;
             rom_enable: in std_logic;
             address: in std_logic_vector((address_length - 1) downto 0);
             data_output: out std_logic_vector((data_length - 1) downto 0));
end triangrom;
architecture behavioral of triangrom is
    type rom_type is array (0 to (2**(address_length) - 28)) of
    std_logic_vector((data_length - 1) downto 0);
    constant mem: rom_type :=
("00000000", "00000101", "00001010", "00001111", "00010100",
"00011010", "00011111", "00100100", "00101001", "00101110",
"00110011", "00111000", "00111101", "01000010", "01000111",
"01001101", "01010010", "01010111", "01011100", "01100001",
"01100110", "01101011", "01110000", "01110101", "01111010",
"10000000", "10000101", "10001010", "10001111", "10010100",
"10011001", "10011110", "10100011", "10101000", "10101101",
"10110011", "10111000", "10111101", "11000010", "11000111",
"11001100", "11010001", "11010110", "11011011", "11100000",
"11100110", "11101011", "11110000", "11110101", "11111010",
"11111111", "11111010", "11110101", "11110000", "11101011",
"11100110", "11100000", "11011011", "11010110", "11010001",
"11001100", "11000111", "11000010", "10111101", "10111000",
"10110011", "10101101", "10101000", "10100011", "10011110",
"10011001", "10010100", "10001111", "10001010", "10000101",
"10000000", "01111010", "01110101", "01110000", "01101011",
"01100110", "01100001", "01011100", "01010111", "01010010",
"01001101", "01000111", "01000010", "00111101", "00111000",
"00110011", "00101110", "00101001", "00100100", "00011111",
"00011010", "00010100", "00001111", "00001010", "00000101",
"00000000");
```

```
begin
p_table: process(clock)
begin
    if rising_edge(clock) then
        if rom_enable = '1' then
            data_output <= mem(to_integer(unsigned(address)));
        else
            data_output <= (others => '0');
        end if;
    end if;
end process p_table;
end architecture;
```

Β΄.7 Μνήμη πριονωτής κυματομορφής

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity sawtrom is
        generic(address_length: natural := 7;
                data_length: natural := 8);
        port(clock: in std_logic;
             rom_enable: in std_logic;
             address: in std_logic_vector((address_length - 1) downto 0);
             data_output: out std_logic_vector((data_length - 1) downto 0));
end entity;
architecture behavioral of sawtrom is
        type rom_type is array (0 to (2**(address_length) - 28)) of
        std_logic_vector((data_length - 1) downto 0);
        constant mem: rom_type :=
("00000000", "00000011", "00000101", "00001000", "00001010",
"00001101", "00001111", "00010010", "00010100", "00010111",
"00011010", "00011100", "00011111", "00100001", "00100100",
"00100110", "00101001", "00101011", "00101110", "00110000",
"00110011", "00110110", "00111000", "00111011", "00111101",
"01000000", "01000010", "01000101", "01000111", "01001010",
"01001101", "01001111", "01010010", "01010100", "01010111",
"01011001", "01011100", "01011110", "01100001", "01100011",
"01100110", "01101001", "01101011", "01101110", "01110000",
"01110011", "01110101", "01111000", "01111010", "01111101",
"10000000", "10000010", "10000101", "10000111", "10001010",
"10001100", "10001111", "10010001", "10010100", "10010110",
"10011001", "10011100", "10011110", "10100001", "10100011",
"10100110", "10101000", "10101011", "10101101", "10110000",
"10110011", "10110101", "10111000", "10111010", "10111101",
"10111111", "11000010", "11000100", "11000111", "11001001",
"11001100", "11001111", "11010001", "11010100", "11010110",
"11011001", "11011011", "11011110", "11100000", "11100011",
"11100110", "11101000", "11101011", "11101101", "11110000",
"11110010", "11110101", "11110111", "11111010", "11111100",
"00000000");
```

```
begin
p_table: process(clock)
begin
    if rising_edge(clock) then
        if rom_enable = '1' then
            data_output <= mem(to_integer(unsigned(address)));
        else
            data_output <= (others => '0');
        end if;
    end if;
end process p_table;
```

Β΄.8 Δειγματοληψία κυματομορφών

```
clc
close all
clear
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y1 = A*sin(2*pi*F*t) %sinusoidal%
subplot (2, 2, 1)
plot (t,y1)
title ('Sine plot')
xlabel ('Time t')
ylabel ('Amplitude')
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y2 = A*square(2*pi*F*t) %square%
subplot (2, 2, 2)
plot (t,y2)
title ('Square plot')
xlabel ('Time t')
ylabel ('Amplitude')
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y3 = A*sawtooth(2*pi*F*t) %sawtooth%
subplot (2, 2, 3)
plot (t,y3)
title ('Sawtooth plot')
xlabel ('Time t')
ylabel ('Amplitude')
A = input ('Enter Amplitude') %1 V%
F = input ('Enter Frequency') %1 Hz%
t = 0:0.01:1 %100 samples%
y4 = A*sawtooth(2*pi*F*t, 0.5) %triangular%
subplot (2, 2, 4)
plot (t,y4)
title ('Triangular plot')
xlabel ('Time t')
ylabel ('Amplitude')
```

Παράρτημα Γ΄

Πίνακες κωδικοποίησης

Γ΄.1 Ημιτονοειδής κυματομορφή

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
0	1	127,5	128	10000000
0,06279052	1,06279052	135,5057913	136	10001000
0,125333234	1,125333234	143,4799873	143	10001111
0,187381315	1,187381315	151,3911177	151	10010111
0,248689887	1,248689887	159,2079606	159	10011111
0,309016994	1,309016994	166,8996667	167	10100111
0,368124553	1,368124553	174,4358805	174	10101110
0,425779292	1,425779292	181,7868597	182	10110110
0,481753674	1,481753674	188,9235934	189	10111101
0,535826795	1,535826795	195,8179164	196	11000100
0,587785252	1,587785252	202,4426196	202	11001010
0,63742399	1,63742399	208,7715587	209	11010001
0,684547106	1,684547106	214,779756	215	11010111
0,728968627	1,728968627	220,4434999	220	11011100
0,770513243	1,770513243	225,7404385	226	11100010
0,809016994	1,809016994	230,6496667	231	11100111
0,844327926	1,844327926	235,1518106	235	11101011
0,87630668	1,87630668	239,2291017	239	11101111
0,904827052	1,904827052	242,8654491	243	11110011
0,929776486	1,929776486	246,046502	246	11110110
0,951056516	1,951056516	248,7597058	249	11111001
0,968583161	1,968583161	250,994353	251	11111011
0,982287251	1,982287251	252,7416245	253	11111101
0,992114701	1,992114701	253,9946244	254	11111110
0,998026728	1,998026728	254,7484078	255	11111111

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
1	2	255	255	11111111
0,998026728	1,998026728	254,7484078	255	11111111
0,992114701	1,992114701	253,9946244	254	11111110
0,982287251	1,982287251	252,7416245	253	11111101
0,968583161	1,968583161	250,994353	251	11111011
0,951056516	1,951056516	248,7597058	249	11111001
0,929776486	1,929776486	246,046502	246	11110110
0,904827052	1,904827052	242,8654491	243	11110011
0,87630668	1,87630668	239,2291017	239	11101111
0,844327926	1,844327926	235,1518106	235	11101011
0,809016994	1,809016994	230,6496667	231	11100111
0,770513243	1,770513243	225,7404385	226	11100010
0,728968627	1,728968627	220,4434999	220	11011100
0,684547106	1,684547106	214,779756	215	11010111
0,63742399	1,63742399	208,7715587	209	11010001
0,587785252	1,587785252	202,4426196	202	11001010
0,535826795	1,535826795	195,8179164	196	11000100
0,481753674	1,481753674	188,9235934	189	10111101
0,425779292	1,425779292	181,7868597	182	10110110
0,368124553	1,368124553	174,4358805	174	10101110
0,309016994	1,309016994	166,8996667	167	10100111
0,248689887	1,248689887	159,2079606	159	10011111
0,187381315	1,187381315	151,3911177	151	10010111
0,125333234	1,125333234	143,4799873	143	10001111
0,06279052	1,06279052	135,5057913	136	10001000

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
0	1	127,5	128	10000000
-0,06279052	0,93720948	119,4942087	119	01110111
-0,125333234	0,874666766	111,5200127	112	01110000
-0,187381315	0,812618685	103,6088823	104	01101000
-0,248689887	0,751310113	95,79203941	96	01100000
-0,309016994	0,690983006	88,10033327	88	01011000
-0,368124553	0,631875447	80,56411949	81	01010001
-0,425779292	0,574220708	73,21314027	73	01001001
-0,481753674	0,518246326	66,07640657	66	01000010
-0,535826795	0,464173205	59,18208364	59	00111011
-0,587785252	0,412214748	52,55738037	53	00110101
-0,63742399	0,36257601	46,22844128	46	00101110
-0,684547106	0,315452894	40,22024399	40	00101000
-0,728968627	0,271031373	34,55650006	35	00100011
-0,770513243	0,229486757	29,25956152	29	00011101
-0,809016994	0,190983006	24,35033327	24	00011000
-0,844327926	0,155672074	19,84818944	20	00010100
-0,87630668	0,12369332	15,7708983	16	00010000
-0,904827052	0,095172948	12,13455087	12	00001100
-0,929776486	0,070223514	8,953498035	9	00001001
-0,951056516	0,048943484	6,24029421	6	00000110
-0,968583161	0,031416839	4,005646973	4	00000100
-0,982287251	0,017712749	2,258375498	2	00000010
-0,992114701	0,007885299	1,005375623	1	00000001
-0,998026728	0,001973272	0,25159218	0	00000000

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
-1	0	0	0	00000000
-0,998026728	0,001973272	0,25159218	0	00000000
-0,992114701	0,007885299	1,005375623	1	00000001
-0,982287251	0,017712749	2,258375498	2	00000010
-0,968583161	0,031416839	4,005646973	4	00000100
-0,951056516	0,048943484	6,24029421	6	00000110
-0,929776486	0,070223514	8,953498035	9	00001001
-0,904827052	0,095172948	12,13455087	12	00001100
-0,87630668	0,12369332	15,7708983	16	00010000
-0,844327926	0,155672074	19,84818944	20	00010100
-0,809016994	0,190983006	24,35033327	24	00011000
-0,770513243	0,229486757	29,25956152	29	00011101
-0,728968627	0,271031373	34,55650006	35	00100011
-0,684547106	0,315452894	40,22024399	40	00101000
-0,63742399	0,36257601	46,22844128	46	00101110
-0,587785252	0,412214748	52,55738037	53	00110101
-0,535826795	0,464173205	59,18208364	59	00111011
-0,481753674	0,518246326	66,07640657	66	01000010
-0,425779292	0,574220708	73,21314027	73	01001001
-0,368124553	0,631875447	80,56411949	81	01010001
-0,309016994	0,690983006	88,10033327	88	01011000
-0,248689887	0,751310113	95,79203941	96	01100000
-0,187381315	0,812618685	103,6088823	104	01101000
-0,125333234	0,874666766	111,5200127	112	01110000
-0,06279052	0,93720948	119,4942087	119	01110111
0	1	127,5	128	10000000

Γ΄.2 Τριγωνική κυματομορφή

Τιμή	Θετικοποίηση	Κανονικοποίηση	Στρογγυλοποίηση	Κωδικοποίηση
-1	0	0	0	00000000
-0,96	0,04	5,1	5	00000101
-0,92	0,08	10,2	10	00001010
-0,88	0,12	15,3	15	00001111
-0,84	0,16	20,4	20	00010100
-0,8	0,2	25,5	26	00011010
-0,76	0,24	30,6	31	00011111
-0,72	0,28	35,7	36	00100100
-0,68	0,32	40,8	41	00101001
-0,64	0,36	45,9	46	00101110
-0,6	0,4	51	51	00110011
-0,56	0,44	56,1	56	00111000
-0,52	0,48	61,2	61	00111101
-0,48	0,52	66,3	66	01000010
-0,44	0,56	71,4	71	01000111
-0,4	0,6	76,5	77	01001101
-0,36	0,64	81,6	82	01010010
-0,32	0,68	86,7	87	01010111
-0,28	0,72	91,8	92	01011100
-0,24	0,76	96,9	97	01100001
-0,2	0,8	102	102	01100110
-0,16	0,84	107,1	107	01101011
-0,12	0,88	112,2	112	01110000
-0,08	0,92	117,3	117	01110101
-0,04	0,96	122,4	122	01111010
0	1	127,5	128	10000000

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
0,04	1,04	132,6	133	10000101
0,08	1,08	137,7	138	10001010
0,12	1,12	142,8	143	10001111
0,16	1,16	147,9	148	10010100
0,2	1,2	153	153	10011001
0,24	1,24	158,1	158	10011110
0,28	1,28	163,2	163	10100011
0,32	1,32	168,3	168	10101000
0,36	1,36	173,4	173	10101101
0,4	1,4	178,5	179	10110011
0,44	1,44	183,6	184	10111000
0,48	1,48	188,7	189	10111101
0,52	1,52	193,8	194	11000010
0,56	1,56	198,9	199	11000111
0,6	1,6	204	204	11001100
0,64	1,64	209,1	209	11010001
0,68	1,68	214,2	214	11010110
0,72	1,72	219,3	219	11011011
0,76	1,76	224,4	224	11100000
0,8	1,8	229,5	230	11100110
0,84	1,84	234,6	235	11101011
0,88	1,88	239,7	240	11110000
0,92	1,92	244,8	245	11110101
0,96	1,96	249,9	250	11111010
1	2	255	255	11111111

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
0,96	1,96	249,9	250	11111010
0,92	1,92	244,8	245	11110101
0,88	1,88	239,7	240	11110000
0,84	1,84	234,6	235	11101011
0,8	1,8	229,5	230	11100110
0,76	1,76	224,4	224	11100000
0,72	1,72	219,3	219	11011011
0,68	1,68	214,2	214	11010110
0,64	1,64	209,1	209	11010001
0,6	1,6	204	204	11001100
0,56	1,56	198,9	199	11000111
0,52	1,52	193,8	194	11000010
0,48	1,48	188,7	189	10111101
0,44	1,44	183,6	184	10111000
0,4	1,4	178,5	179	10110011
0,36	1,36	173,4	173	10101101
0,32	1,32	168,3	168	10101000
0,28	1,28	163,2	163	10100011
0,24	1,24	158,1	158	10011110
0,2	1,2	153	153	10011001
0,16	1,16	147,9	148	10010100
0,12	1,12	142,8	143	10001111
0,08	1,08	137,7	138	10001010
0,04	1,04	132,6	133	10000101
0	1	127,5	128	10000000

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
-0,04	0,96	122,4	122	01111010
-0,08	0,92	117,3	117	01110101
-0,12	0,88	112,2	112	01110000
-0,16	0,84	107,1	107	01101011
-0,2	0,8	102	102	01100110
-0,24	0,76	96,9	97	01100001
-0,28	0,72	91,8	92	01011100
-0,32	0,68	86,7	87	01010111
-0,36	0,64	81,6	82	01010010
-0,4	0,6	76,5	77	01001101
-0,44	0,56	71,4	71	01000111
-0,48	0,52	66,3	66	01000010
-0,52	0,48	61,2	61	00111101
-0,56	0,44	56,1	56	00111000
-0,6	0,4	51	51	00110011
-0,64	0,36	45,9	46	00101110
-0,68	0,32	40,8	41	00101001
-0,72	0,28	35,7	36	00100100
-0,76	0,24	30,6	31	00011111
-0,8	0,2	25,5	26	00011010
-0,84	0,16	20,4	20	00010100
-0,88	0,12	15,3	15	00001111
-0,92	0,08	10,2	10	00001010
-0,96	0,04	5,1	5	00000101
-1	0	0	0	00000000

Γ΄.3 Πριονωτή κυματομορφή

Τιμή	Θετικοποίηση	Κανονικοποίηση	Στρογγυλοποίηση	Κωδικοποίηση
-1	0	0	0	00000000
-0,98	0,02	2,55	3	00000011
-0,96	0,04	5,1	5	00000101
-0,94	0,06	7,65	8	00001000
-0,92	0,08	10,2	10	00001010
-0,9	0,1	12,75	13	00001101
-0,88	0,12	15,3	15	00001111
-0,86	0,14	17,85	18	00010010
-0,84	0,16	20,4	20	00010100
-0,82	0,18	22,95	23	00010111
-0,8	0,2	25,5	26	00011010
-0,78	0,22	28,05	28	00011100
-0,76	0,24	30,6	31	00011111
-0,74	0,26	33,15	33	00100001
-0,72	0,28	35,7	36	00100100
-0,7	0,3	38,25	38	00100110
-0,68	0,32	40,8	41	00101001
-0,66	0,34	43,35	43	00101011
-0,64	0,36	45,9	46	00101110
-0,62	0,38	48,45	48	00110000
-0,6	0,4	51	51	00110011
-0,58	0,42	53,55	54	00110110
-0,56	0,44	56,1	56	00111000
-0,54	0,46	58,65	59	00111011
-0,52	0,48	61,2	61	00111101
-0,5	0,5	63,75	64	01000000
-0,48	0,52	66,3	66	01000010
-0,46	0,54	68,85	69	01000101
-0,44	0,56	71,4	71	01000111
-0,42	0,58	73,95	74	01001010
-0,4	0,6	76,5	77	01001101
-0,38	0,62	79,05	79	01001111
-0,36	0,64	81,6	82	01010010
-0,34	0,66	84,15	84	01010100
-0,32	0,68	86,7	87	01010111
-0,3	0,7	89,25	89	01011001

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
-0,28	0,72	91,8	92	01011100
-0,26	0,74	94,35	94	01011110
-0,24	0,76	96,9	97	01100001
-0,22	0,78	99,45	99	01100011
-0,2	0,8	102	102	01100110
-0,18	0,82	104,55	105	01101001
-0,16	0,84	107,1	107	01101011
-0,14	0,86	109,65	110	01101110
-0,12	0,88	112,2	112	01110000
-0,1	0,9	114,75	115	01110011
-0,08	0,92	117,3	117	01110101
-0,06	0,94	119,85	120	01111000
-0,04	0,96	122,4	122	01111010
-0,02	0,98	124,95	125	01111101
0	1	127,5	128	10000000
0,02	1,02	130,05	130	10000010
0,04	1,04	132,6	133	10000101
0,06	1,06	135,15	135	10000111
0,08	1,08	137,7	138	10001010
0,1	1,1	140,25	140	10001100
0,12	1,12	142,8	143	10001111
0,14	1,14	145,35	145	10010001
0,16	1,16	147,9	148	10010100
0,18	1,18	150,45	150	10010110
0,2	1,2	153	153	10011001
0,22	1,22	155,55	156	10011100
0,24	1,24	158,1	158	10011110
0,26	1,26	160,65	161	10100001
0,28	1,28	163,2	163	10100011
0,3	1,3	165,75	166	10100110
0,32	1,32	168,3	168	10101000
0,34	1,34	170,85	171	10101011
0,36	1,36	173,4	173	10101101

Τιμή	**Θετικοποίηση**	**Κανονικοποίηση**	**Στρογγυλοποίηση**	**Κωδικοποίηση**
0,38	1,38	175,95	176	10110000
0,4	1,4	178,5	179	10110011
0,42	1,42	181,05	181	10110101
0,44	1,44	183,6	184	10111000
0,46	1,46	186,15	186	10111010
0,48	1,48	188,7	189	10111101
0,5	1,5	191,25	191	10111111
0,52	1,52	193,8	194	11000010
0,54	1,54	196,35	196	11000100
0,56	1,56	198,9	199	11000111
0,58	1,58	201,45	201	11001001
0,6	1,6	204	204	11001100
0,62	1,62	206,55	207	11001111
0,64	1,64	209,1	209	11010001
0,66	1,66	211,65	212	11010100
0,68	1,68	214,2	214	11010110
0,7	1,7	216,75	217	11011001
0,72	1,72	219,3	219	11011011
0,74	1,74	221,85	222	11011110
0,76	1,76	224,4	224	11100000
0,78	1,78	226,95	227	11100011
0,8	1,8	229,5	230	11100110
0,82	1,82	232,05	232	11101000
0,84	1,84	234,6	235	11101011
0,86	1,86	237,15	237	11101101
0,88	1,88	239,7	240	11110000
0,9	1,9	242,25	242	11110010
0,92	1,92	244,8	245	11110101
0,94	1,94	247,35	247	11110111
0,96	1,96	249,9	250	11111010
0,98	1,98	252,45	252	11111100
-1	0	0	0	00000000

Παράρτημα Δ΄

Σφάλματα

Δ΄.1 Προσομοιωτές

Το λογισμικό Quartus υπόκειται σε συχνές ενημερώσεις ασφαλείας καθώς επίσης κι ενημερώσεις που έχουν να κάνουν με την ίδια του τη σημασία ως προϊόν. Δηλαδή, καθώς τίθενται σε κυκλοφορία όλο και νεότερα μοντέλα FPGA, αναδύονται και νεότερες εκδόσεις του λογισμικού.

Η Intel έχει αποφασίσει να αποσύρει ανά τακτά χρονικά διαστήματα τις παλαιότερες εκδόσεις του λογισμικού αυτού καθ΄ εαυτού όπως επίσης και τα παρ΄ αυτώ: προσομοιωτές κι αρχεία υποστήριξης FPGA, όπως ο Simulation Waveform Editor.

Το αποτέλεσμα είναι ότι σε νεότερες εκδόσεις των Microsoft Windows (10 ή 11), η εκτέλεση του προγράμματος να είναι από εξαιρετικά περίπλοκη έως αδύνατη λόγω σφαλμάτων. Σε κάθε περίπτωση, αυτό δεν πρέπει να μεταβληθεί είναι το όνομα του παραγόμενου αρχείου ή η θέση αποθήκευσής του (Waveform.vwf).

Για τους χρήστες, λοιπόν, του Intel Quartus Prime Lite οι οποίοι αντιμετωπίζουν προβλήματα με την προσομοίωση, είτε μέσω Modelsim FPGA Starter Edition, είτε μέσω Quartus FPGA Starter Edition, είτε μέσω Simulation Waveform Editor συνίσταται η εξής διόρθωση στο script του προσομοιωτή. Αντ΄ αυτού:

```
onerror {exit -code 1}
vlib work
vlog -work work                $file_name$     .vo
vlog -work work   $file_name$.vwf.vt
vsim -novopt -c -t 1ps              $board_inforamtion$
vcd file -direction                 $file_name$   msim.vcd
vcd add -internal                   $file_name$   _vlg_vec_tst/*
vcd add -internal                   $file_name$   _vlg_vec_tst/i1/*
proc simTimestamp {} {
   echo "Simulation time: $::now ps"
   if { [string equal running [runStatus]] } {
      after 2500 simTimestamp
   }
}
after 2500 simTimestamp
run -all
quit -f
```

Αυτή η διόρθωση:

```
onerror {exit -code 1}
vlib work
vlog -work work                $file_name$ .vo
vlog -work work  $file_name$ .vwf.vt
vsim    voptargs="+acc"  -c -t 1ps      $board  information$
vcd file -direction                    $file_name$  .msim.vcd
vcd add -internal                      $file_name$  _vlg_vec_tst/*
vcd add -internal                      $file_name$  _vlg_vec_tst/i1/*
proc simTimestamp {} {
   echo "Simulation time: $::now ps"
   if { [string equal running [runStatus]] } {
      after 2500 simTimestamp
   }
}
after 2500 simTimestamp
run -all
quit -f
```

Μια παρόμοια κατηγορία σφάλματος είναι αυτή η οποία δεν επιτρέπει την εκκίνηση των προομοιωτών Modelsim ή Questa FPGA Starter Edition.

Υπό την αυστηρή προϋπόθεση ότι δεν τίθεται θέμα με τις άδειες χρήσης του λογισμικού (η λήψη κι εγκατάσταση των οποίων θεωρείται εκ των ων ουκ άνευ), ενδέχεται κατά την εκτέλεση των προσομοιωτών μέσω NativeLink να εμφανιστεί μήνυμα σφάλματος στην οθόνη.

Για την επίλυσή του άπαξ και διά παντός, συνίστανται δύο μικρές διορθώσεις:

- Στο μενού $Tools \rightarrow Options \rightarrow General \rightarrow EDATool$, αφού καθοριστεί η ακριβής διαδρομή στην οποία βρίσκονται εγκατεστημένα τα εκτελέσιμα αρχεία των προσομοιωτών, προσθέτουμε μία / στο τέλος της διεύθυνσης.

  Δηλαδή, π.χ., αντί για:

  $C : /intelFPGA\_lite/22.1std/questa\_fse/win64$

  Διορθώνουμε ως εξής:

  $C : /intelFPGA\_lite/22.1std/questa\_fse/win64/.$

- Στο φάκελο εγκατάστασης του Quartus, βρίσκουμε κι ανοίγουμε με κειμενογράφο το αρχείο $qnativelinkflow.tcl$, στη θέση:

  $C : /intelFPGA\_lite/22.1std/quartus/common/tcl/internal/nativelink$

  Στη γραμμή 122 βρίσκεται η εντολή:

  $set\_questa\_installation"\$questa\_fse\_directory$

  Από αυτήν, αφαιρούμε τα εισαγωγικά. Δηλαδή:

  $set\_questa\_installation\$questa\_fse\_directory$

Ευχαριστίες

Η παρούσα εργασία δε θα μπορούσε να πραγματοποιηθεί δίχως τη στήριξη ενός συνόλου ανθρώπων από την αρχή ως το τέλος. Έτσι, ετούτες οι τελευταίες αλλά διόλου ασήμαντες γραμμές είναι αφιερωμένες σε αυτούς, ως ελάχιστη δυνατή μνεία.

Ιδιαίτερα και πρώτα από όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Επίκουρο Καθηγητή κ. Χριστοφιλάκη Βασίλειο για την εμπιστοσύνη που μου έδειξε από την αρχή αυτού του ταξιδιού, την πολύτιμη καθοδήγησή του, τις παραγωγικές υποδείξεις του, την καίρια συμβολή του στη διατήρηση ενός εξαιρετικού κλίματος συνεργασίας μεταξύ μας και κυρίως την αμέριστη υπομονή που έδειξε, όλον αυτό τον καιρό.

Θα ήθελα επίσης να ευχαριστήσω και την οικογένειά μου, συγκεκριμένα τους γονείς μου Ευστράτιο και Μαρία, καθώς επίσης και την αδερφή μου Αγάπη για την επικοδομοιητική κριτική τους και τη συνεχή στήριξή τους σε δύσκολους καιρούς, θυμίζοντάς μου ότι εγώ επέλεξα το ταξίδι αυτό, σε μένα χρωστάω την πραγματοποίησή του.

Τέλος, θα ήθελα να ευχαριστήσω όλους αυτούς τους φίλους και γνωστούς, παλιούς και τρέχοντες συναδέλφους, τμήμα της γενιάς μας, που κατανοώντας τις δυσκολίες και τις ανάγκες της εποχής, μέσα από τα λίγο ή πολύ κοινά βιώματά μας, μου συμπαραστάθηκαν.

Ευρετήριο

Βιβλιογραφία

[1] *«32-Channel Waveform Generator Implemented Using Actel's Axcelerator FPGA»*, Actel Corporation, 2004.

[2] *«DE1 - SoC My First FPGA»*, Altera University Program.

[3] *«DE1 - SoC User Manual»*, Altera University Program.

[4] *«Introduction to FPGA Design with Vivado High - Level Synthesis»*, Xilinx, 2019.

[5] *«My First FPGA Design Tutorial»*, Altera, 2008.

[6] *«The VHDL Golden Reference Guide»*, Doulos, UK, 1995.

[7] Al Bustam H., Shahzama M. *«A VHDL Based DAC Implementation on FPGA»*, ResearchGate, 2013.

[8] Alpert T., Werz M., Lang F., Ferenci D., Masini M., Grözing M., Berroth M. *«Arbitrary Waveform Generator Based on FPGA and High-Speed DAC with Real-Time Interface»*, PRIME, 2012.

[9] Ashenden P. J. *«The Designer's Guide to VHDL», 3rd Edition, Morgan Kaufman, USA, 2008.*

[10] Clemente J. A. *«Introduction to VHDL Programming»*, ResearchGate, 2014.

[11] *«Differences Between Function Generators, Arbitrary Function Generators, and Arbitrary Waveform Generators»*, NI Knowledge Base, National Instruments Corporation.

[12] Ding S., An A., Gou X. *«Digital Waveform Generator Based on FPGA»*, Research Journal of Applied Sciences, Engineering and Technology, 2012.

[13] Donnellan S., Hill I. R., Bowden W., Hobson R. *«A scalable arbitrary waveform generator for atomic physics experiments based on field-programmable gate array technology»*, Review of Scientific Instruments, 2019.

[14] *«Engineer's Toolkit: Understanding Arbitrary Waveform Generators vs Function Generators»*, Keysight Tech Guides, Keysight Technologies Inc.

[15] Goman A. *«Waveform Generator Implemented in FPGA with an Embedded Processor»*, Linköping University, 2003.

[16] Grout I. *«Digital Systems Design with FPGAs»*, Newnes, Elsevier, Oxford, 2008.

[17] Hansen J. S. *«GNU Octave: Begginer's Guide»*, Packt Publishing, Birmingham, 2011.

[18] Haskell R. E., Hanna D. M. *«Introduction to Digital Design using Digilent FPGA Boards - Block Diagram/VHDL Examples»*, LBE Books LLC, Rochester Hills, MI, 2009.

[19] Kilts S. *«Advanced FPGA Design: Architecture, Implementation and Optimization»*, John Wiley & Sons Inc., New Jersey, 2007.

[20] Kleitz W. *«Digital Electronics: A Practical Approach with VHDL» (9th Edition)*, State University of New York, Pearson, New Jersey, 2012.

[21] Λάμπρος Σ., *«Σχεδίαση κι Υλοποίηση Πομπού Βασικής Ζώνης σε Altera Flex»*, Πανεπιστήμιο Ιωαννίνων, 2008.

[22] Maxfield C. *«FPGAs: Instant Access - World Class Designs»*, Newnes, USA, 2008.

[23] Maxfield C. *«The Design Warrior's Guide to FPGAs: Devices, Tools and Flows»*, Newnes, Elsevier, USA, 2004.

[24] Mealy B., Tappero F. *«Free Range VHDL»*, 2012.

[25] Mencer O., Allison D., Blatt E., Flynn M. J., Harris J., Hewitt C., Jacobson Q., Lavasani M., Moazani M., Murray H., Nikravesh M., Nowatzyk A., Shand M., Shirazi S. *«The History, Status and Future of FPGAs»*, ACM Queue, 2020.

[26] Moore A., Wilson R. *«FPGAs for dummies» (2nd Edition)*, John Wiley & Sons Inc., New Jersey, 2017.

[27] Nguyen C., *«Introduction to Field Programmable Gate Arrays (FPGAs)»*, University of Scranton, 2021.

[28] Μπαλντούμας Γ., *«Ψηφιακή Διαμόρφωση Σήματος με χρήση DDS»*, Πανεπιστήμιο Ιωαννίνων, 2006.

[29] Μωυσής Λ., Τσικαλοπούλου Μ., Διαμαντίδης Δ., Λύκου Ρ., Βουγιούκα Γ., Τσαουσίδου Μ., Τσολάκης Χ., Τσάπαρης Θ., Παπαθεοδώρου Δ., *«Εισαγωγή στη LaTeX για φοιτητές»*, 2014.

[30] Ξενοφώντος Χ., *«Οδηγός MATLAB γι αρχάριους»*, Πανεπιστήμιο Κύπρου, Τμήμα Μαθηματικών και Στατιστικής.

[31] Parab J. S., Gad R. S., Naik G. M. *«Hands-on Experience with Altera FPGA Development Boards»*, Springer Private Ltd., New Delhi, India, 2018.

[32] Pengra D. *«The Oscilloscope and the Function Generator: Some introductory exercises for students in the advanced labs»*, arXiv.org, 2007.

[33] Peterson D. *«Function Generator and Arbitrary Waveform Generator Guidebook»*, B&K Precision.

[34] Qi J., Sun Q., Wu X., Wang C., Chen L. *«Design and Analysis of a Low Cost Wave Generator Based on Direct Digital Synthesis»*, Journal of Electrical and Computer Engineering, 2015.

[35] Rajewski J. *«Learning FPGAs: Digital Design for Beginners with Mojo and Lucid HDL» (1st Edition)*, O' Reilly Media Inc., USA, 2017.

[36] Rueda L. E. G., Silva E., Centeno A., Roa E. *«All-Digital FPGA-based DAC with None or Few External Components»*, arXiv.org, 2020.

[37] Sizemore J., Mueller J. P. *«MATLAB for dummies»*, John Wiley and Sons, New Jersey, 2015.

[38] Tirmare A. H., Mohite S. R., Mali P. S., Suryavanshi V. A. *«FPGA based function generator»*, International Research Journal of Engineering and Technology (IRJET), 2015.

[39] *«What Is an Rf Signal Generator? Different Types, Applications & More!»*, Keysight Tech Guides, Keysight Technologies Inc.

[40] Wilson P. R. *«Design Recipes for FPGAs»*, Elsevier, Oxford, 2007.

[41] Wolf W. *«FPGA - Based System Design»*, PTR Prentice Hall, New Jersey, 2004.

[42] Zwolinski M. *«Digital System Design with VHDL» (2nd Edition)*, Pearson Education Ltd., Harlow, England, 2004.