ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

3 ΙΟΥΝΙΟΥ 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ Μηχανικών Πληροφορικής ΤΕ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση ψηφιακού συστήματος με γλώσσα περιγραφής υλικού

Κοτσολάρι Βασίλ

Επιβλέπων: Φώτης Βαρτζιώτης

τίτλος, βαθμίδα

Τόπος έκδοσης, Μήνας, Έτος ολοκλήρωσης

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

SCHOOL OF ENGINEERING INFORMATION AND TELECOMMUNICATIONS

DEPARTMENT OF IT ENGINEERING TE

Thesis

Design of digital system using a Hardware Description Language

Kocollari Vasil

Supervisor: Fotis Bartziotis

Title, Rank

Place of issue, Month, Year of completion

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Φώτιος Βαρτζιώτης

2. Μέλος επιτροπής

   Γρηγόριος Δουμένης

3. Μέλος επιτροπής

   Ελευθέριος Στεργίου

© Κοτσολάρι, Βασίλ, 2025.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Κοτσολάρι, Βασίλ

Υπογραφή

.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω την οικογένεια μου που μου συμπαραστάθηκε καθ'όλη την διάρκεια της ακαδημαϊκής μου καριέρας, τον Φώτιο Βαρτζιώτη για την υπομονή, την καθοδήγηση και την βοήθεια του για την συγγραφή αυτής της εργασίας. Και τέλος θα ήθελα να δώσω τις ευχαριστίες μου στον Κύριο Ιωάννη Παπαγεωργίου, που μου συμπαραστάθηκε στις ποιο κακές στιγμές της ακαδημαϊκής μου καριέρας.

ΠΕΡΙΛΗΨΗ

Τα νευρωνικά δίκτυα στη σημερινή εποχή παίζουν έναν πολύ σημαντικό ρόλο στην καθημερινότητά μας, είτε στο να λύσουμε απλά ζητήματα, είτε στο να απαντήσουμε σε πιο περίπλοκες ερωτήσεις που έχει ο άνθρωπος από τα αρχαία χρόνια μέχρι και σήμερα. Από τα πρώτα ερωτήματα που τέθηκαν ήταν: Πώς δουλεύει ο ανθρώπινος εγκέφαλος; και Πώς θα μπορούσαμε να φτιάξουμε έναν τεχνητό εγκέφαλο;

Παρόλο που βρίσκονται ακόμα σε ένα αρχικό στάδιο εξέλιξης, τα νευρωνικά δίκτυα είναι ήδη εφαρμοσμένα σε έναν μεγάλο αριθμό περιπτώσεων, είτε ως τεχνητή νοημοσύνη με "δικές της πρωτοβουλίες και σκέψεις", είτε ως μηχανή αναζήτησης, είτε ακόμα και ως μηχανισμός επίλυσης περίπλοκων προβλημάτων (όπως προβλήματα κβαντικής φυσικής).

Σε αυτή την εργασία θα εξηγήσουμε αναλυτικά τον όρο Νευρωνικά Δίκτυα, πώς αναπτύχθηκαν, πώς προέκυψε η ιδέα για την ανάγκη δημιουργίας τους, σε τι χρησιμεύουν στην καθημερινότητά μας, τα διαφορετικά είδη που υπάρχουν σήμερα, πώς μπορούμε να εκπαιδεύσουμε τέτοια δίκτυα, τα δύο βασικά εργαλεία της υλοποίησης: το **Quartus Prime Lite Edition** και το **Modelsim Altera**, καθώς και θα υλοποιήσουμε το δικό μας νευρωνικό δίκτυο.

Η κατανόηση των νευρωνικών δικτύων είναι σημαντική, καθώς πρόκειται για συστήματα που αναπαράγουν βασικές λειτουργίες του ανθρώπινου εγκεφάλου, μαθαίνοντας και προσαρμόζονατς μέσα από την εμπειρία. Μέσα από την εκπαίδευση και τις επαναληπτικές διεργασίες, είναι σε θέση να επιλύουν προβλήματα, να εντοπίζουν μοτίβα σε δεδομένα και να προσφέρουν προβλέψεις με αξιοσημείωτη ακρίβεια.

Τα εργαλεία υλοποίησης που θα παρουσιαστούν, χρησιμοποιούνται ευρέως στην ανάπτυξη και προσομοίωση ψηφιακών κυκλωμάτων, και μέσω αυτών μπορούμε να σχεδιάσουμε και να ελέγξουμε την αρχιτεκτονική ενός νευρωνικού δικτύου σε επίπεδο υλικού. Με αυτόν τον τρόπο, η εργασία συνδέει την θεωρία με την πρακτική εφαρμογή, προσφέροντας ένα ολοκληρωμένο πλαίσιο κατανόησης και ανάπτυξης.

Λέξεις-κλειδιά: Νευρωνικά Δίκτυα, εργαλεία υλοποίσης Quartus Prime Lite Edition και Modelsim Altera, ανάπτυξη, εκπαίδευση, διάφορα είδη δικτύων

ABSTRACT

Neural networks today play a very important role in our daily lives, whether in solving simple problems or in answering more complex questions that humans have had from ancient times to the present day. Some of the first questions that were asked were: How does the human brain work? and How could we build an artificial brain?

Although they are still in an early stage of development, neural networks are already applied in a large number of cases, either as artificial intelligence with "its own initiatives and thoughts", or as a search engine, or even as a mechanism for solving complex problems (such as quantum physics problems).

In this paper, we will explain in detail the term Neural Networks, how they were developed, how the idea for the need to create them arose, what they are used for in our everyday lives, the different types that exist today, how we can train such networks, the two main tools of the implementation: Quartus Prime Lite Edition and Modelsim Altera, as well as we will implement our own neural network.

Understanding neural networks is important, as they are systems that replicate basic functions of the human brain, learning and adapting through experience. Through training and iterative processes, they are able to solve problems, identify patterns in data, and offer predictions with remarkable accuracy.

The implementation tools that will be presented are widely used in the development and simulation of digital circuits, and through them we can design and control the architecture of a neural network at the hardware level. In this way, the work connects theory with practical application, offering a comprehensive framework for understanding and development.

Keywords: Neural Networks, Quartus Prime Lite Edition implementation tools and Altera Modelsim, development, training, various types of networks

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1.Εισαγωγή

Νευρωνικό δίκτυο (Neural Network) ονομάζουμε ένα υπολογιστικό σύστημα με μεγάλο βαθμό παραλληλίας και διασυνδέσεων που ξεχωριστά ονομάζονται νευρώνες. Αυτά τα χαρακτηριστικά θέτουν μεγάλη πρόκληση για τους κατασκευαστές στην υλοποίηση τους, λόγω της μεγάλης ποσότητας υλικού που απαιτείται για την κατασκευή ενός απλού τέτοιου είδους δικτύου.

1.1 Στόχος Πτυχιακής

Ο κύριος σκοπός αυτής της πτυχιακής εργασίας είναι η επεξήγηση των νευρωνικών δικτύων, των τεχνολογιών που εφαρμόζονται, σε τι μας ωφελούν στην επιστήμη και στην καθημερινότητα των ανθρώπων και να δημιουργήσουμε ένα απλό νευρωνικό δίκτυο κυκλωματικά, να το εξετάσουμε και να βγάλουμε αποτελέσματα και συμπεράσματα για τη συμπεριφορά που περιμένουμε να βγάλει σε σύγκριση με αυτήν που ξέρουμε ότι πρέπει να βγάλει. Πιο συγκεκριμένα, θα εξετάσουμε τη συμπεριφορά ενός απλού νευρωνικού δικτύου με πράξεις ακεραίων αριθμών.

1.2 Νευρωνικά Δίκτυα

Τα Νευρωνικά Δίκτυα είναι εμπνευσμένα από το Κεντρικό Νευρικό Σύστημα (ΚΝΣ) που υπάρχει σε κάθε ζωντανό οργανισμό. Ο στόχος των Νευρωνικών Δικτύων είναι να προσομοιώσουν αυτό το σύστημα με τη βοήθεια κυκλωμάτων συνδεδεμένων μεταξύ τους. Τα κυκλώματα αυτά ονομάζονται νευρώνες. [1]

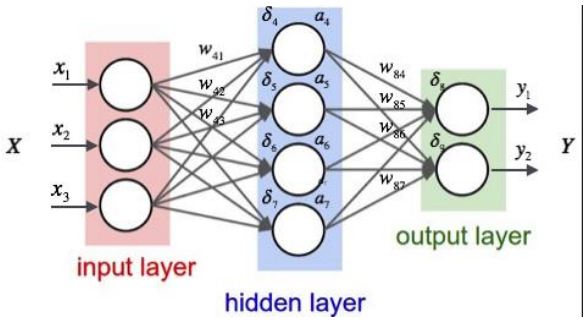
Στην επόμενη ενότητα θα μελετήσουμε τις τεχνολογίες που θα εφαρμόσουμε (FPGA, VHDL, Quartus Prime και Modelsim) στην εργασία αυτή

2.Περιγραφή Τεχνολογιών

Σε αυτή την ενότητα θα μελετήσουμε και θα αναλύσουμε την τεχνολογία περιγραφής υλικού (FPGA), καθώς και την γλώσσα περιγραφής υλικού που θα ασχοληθούμε σε αυτή την εργασία (VHDL) και τα δύο εργαλεία υλοποίησης (Quartus Prime, Modelsim).

Παρακάτω θα εξηγήσουμε σε βάθος τη λειτουργία των νευρωνικών δικτύων, τα είδη που υπάρχουν, λειτουργίες που εκτελεί το κάθε είδος, θετικά και αρνητικά για κάθε νευρωνικό δίκτυο, όπως και κάποιες εφαρμογές τους.

Οι νευρώνες ή αλλιώς κόμβοι του δικτύου είναι τα δομικά στοιχεία για την λειτουργία των δικτύων και τους χωρίζουμε σε τρία είδη: νευρώνες εισόδου, νευρώνες εξόδου και υπολογιστικούς νευρώνες ή αλλιώς κρυμμένοι νευρώνες. Οι κόμβοι αυτοί δουλεύουν ξεχωριστά και παράλληλα μεταξύ τους και ο καθένας τους δέχονται ένα σύνολο από αριθμητικές εισόδους από διάφορες πηγές (είτε από εξωτερικούς παράγοντες, είτε από άλλες διασυνδέσεις που έχουν μεταξύ τους), συνιστούν σε έναν υπολογισμό με βάση των εισόδων αυτών και παράγεται μια έξοδος.



Εικόνα 2.1 Διάγραμμα ενός Νευρωνικού Δικτύου

Η έξοδος αυτή μετέπειτα θα κατευθυνθεί είτε στο περιβάλλον (π.χ. οθόνη), είτε ως τροφοδοσία ή είσοδος σε άλλους νευρώνες του δικτύου.

Τα είδη των νευρώνων που αναφέραμε παραπάνω εκτελούν τις εξής λειτουργίες:

Νευρώνες εισόδου: Δεν παίρνουν μέρος σε κανέναν από τους υπολογισμούς που εκτελούνται κατά τη διάρκεια λειτουργίας του δικτύου, είναι απλοί μεσολαβητές στο περιβάλλον (π.χ. οθόνη) και στους υπολογιστικούς νευρώνες του δικτύου. [2]

Υπολογιστικοί Νευρώνες: Η λειτουργία αυτών των νευρώνων είναι να πολλαπλασιάζουν κάθε είσοδό τους με τα αντίστοιχα βάρη και να υπολογίζουν το ολικό άθροισμα των γινομένων (π.χ. για ένα δίκτυο με μια είσοδο x και τρεις νευρώνες a, b και c, η είσοδος πολλαπλασιάζεται με αυτές τις εισόδους και στην έξοδο προστίθενται τα αποτελέσματα των κάθε νευρώνων μεταξύ τους). Το άθροισμα αυτό τροφοδοτεί μια συνάρτηση ενεργοποίησης, η οποία υλοποιείται εσωτερικά σε κάθε κόμβο. Για τις τρέχουσες εισόδους και βάρη του νευρώνα, η τιμή που λαμβάνει για το όρισμα η συνάρτηση είναι και η έξοδός του. [2]

Νευρώνες Εξόδου: Κατευθύνουν τις τελικές αριθμητικές εξόδους του δικτύου στο περιβάλλον. [2] [3]

Συνήθως, η συνάρτηση που λαμβάνεται και ως όρισμα εξόδου ενός νευρώνα για τις τρέχουσες τιμές και βάρη είναι της μορφής:

$$y_k = \phi\left(\sum_{i=0}^{N} x_{ki} w_{ki}\right)$$

[2]

Όπου:

χ_{ki}: i-οστή είσοδος του k νευρώνα

w_{ki}: i-οστά αντίστοιχα βάρη του k νευρώνα

$\varphi(\cdot)$: η συνάρτηση ενεργοποίησης του νευρωνικού δικτύου

y_k: η έξοδος για τον νευρώνα k

Κάθε νευρωνικό δίκτυο έχει τη δική του συνάρτηση ενεργοποίησης, για τη λειτουργία που προορίζεται, και τις χωρίζουμε σε 4 είδη: βηματική, γραμμική, μη-γραμμική και στοχαστική.

1. Η ποιο συνηθισμένη μορφή της βηματικής συνάρτησης είναι:

$$\phi(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$ [2]

Αυτού του είδους η συνάρτηση ενεργοποίησης δεν θεωρείται πολύ χρήσιμη, γιατί το κύριο πρόβλημα της είναι ότι η παράγωγός της απειρίζεται.

2. Η γραμμική συνάρτηση ενεργοποίησης είναι της μορφής:

$$\phi(x) = x$$ [2]

Ό,τι εισέλθει σαν είσοδος, θα εξαχθεί και σαν έξοδος. Δεν είναι πολύ χρήσιμη, λόγω ότι το δίκτυο δεν είναι σε θέση να εκπαιδευτεί από τις προηγούμενες καταστάσεις.

3. Οι μη-γραμμικές συναρτήσεις ενεργοποίησης συνήθως ονομάζονται και σιγμοειδής συναρτήσεις και είναι αυτές που χρησιμοποιούμαι κυριότερα. Οι ποιο τυπικές είναι:

Λογιστική σιγμοειδής (η συνάρηση που θα χρησιμοποίησουμε):

$$\phi(x) = \frac{1}{1 + e^{-x}}$$ [2]

Υπερβολική εφαπτομένη:

$$\phi(x) = \tanh x$$

Οι στοχαστικές συναρτήσεις ενεργοποίησης έχουν σαν στόχο να εισάγουν στο δίκτυο μια τυχαιότητα στη διαδικασία εκμάθησης του δικτύου, για τη βελτίωση της απόδοσης και της ικανότητας γενίκευσης του νευρωνικού δικτύου. Η πιο γνωστή συνάρτηση είναι η **Dropout**.[2]

Σε μια σύνοψη τα νευρωνικά δίκτυα αποτελούνται από (εικόνα 2.1, σελίδα 13):

Βάρη: Είναι μηχανικές αριθμητικές τιμές που πολλαπλασιάζονται με τις εισόδους του δικτύου, μαθαίνονται από το δίκτυο και αυτοπροσαρμόζονται ανάλογα με τη διαφορά μεταξύ των προβλεπόμενων εξόδων έναντι των εισόδων εκπαίδευσης. [2] [3]

Συνάρτηση ενεργοποίησης: Είναι η μαθηματική εξίσωση που επιτρέπει την ανάλογη εναλλαγή κατάστασης του νευρώνα ON/OFF. [2] [3]

Επίπεδο εισόδου: Αντιπροσωπεύει τις διαστάσεις του διανύσματος εισόδου. [2] [3]

Κρυφό επίπεδο: Είναι οι ενδιάμεση κόμβοι και χωρίζουν τον χώρο της εισόδου σε περιοχές με όρια. Αυτό το επίπεδο κάνει και μια ακόμα λειτουργία που είναι να λαμβάνει ένα σύνολο σταθμισμένων εισόδων και παράγει μια έξοδο μέσω μιας συνάρτηση ενεργοποίησης.[2] [3]

Επίπεδο εξόδου: Αντιπροσωπεύει την έξοδο/ους του νευρωνικού δικτύου.[2] [3]

Πως γίνεται η εκπαίδευση των Νευρωνικών Δικτύων;

Η πιο βασική ιδιότητα των νευρωνικών δικτύων είναι η ικανότητα εκπαίδευσής τους. Η εν λόγω εκπαίδευση γίνεται μέσω μιας βαθμιαίας σύλληψης πληροφορίας, που αποσκοπεί στην ανταλλαγή των τιμών και βαρών μέσα στο δίκτυο, και στη συνέχεια η πληροφορία αυτή γίνεται διαθέσιμη προς ανάκτηση για περαιτέρω επεξεργασία. Η εκμάθηση των δικτύων γίνεται με δύο τρόπους: μάθηση με επίβλεψη και μάθηση χωρίς επίβλεψη. [2] [3]

Μάθηση με επίβλεψη: Αυτή η κατηγορία εκμάθησης έχει έναν εξωτερικό εκπαιδευτή (συνήθως έναν χρήστη, έναν προγραμματιστή ή τον δημιουργό του δικτύου), που επεξεργάζεται την πληροφορία που θα εισέλθει στο νευρωνικό δίκτυο και το εκπαιδεύει αναλόγως. Κάποιες από τις μεθόδους που ανήκουν σε αυτή την κατηγορία είναι η μάθηση με διόρθωση σφάλματος και η στοχαστική μάθηση. Κάποια παραδείγματα για αυτή την κατηγορία είναι η διακοπή της διαδικασίας εκμάθησης, τα πρότυπα εκπαίδευσης, η παρουσίαση προόδου δικτύου και η συχνότητα δικτύου. Η κατηγορία αυτή χωρίζεται σε δύο υποκατηγορίες: τη δομική και την προσωρινή εκμάθηση. Για τη δομική εκμάθηση, οι αλγόριθμοι χρησιμοποιούνται για την εύρεση της καλύτερης σχέσης μεταξύ εισόδου – εξόδου, για κάθε ξεχωριστό ζευγάρι προτύπων. Κάποια παραδείγματα των δύο υποκατηγοριών είναι ότι στη δομική εκμάθηση περιλαμβάνεται η αναγνώριση και κατηγοριοποίηση των προτύπων, ενώ στη προσωρινή εκμάθηση είναι η πρόβλεψη και ο έλεγχος. [2] [3]

Μάθηση χωρίς επίβλεψη: Οι αλγόριθμοι αυτής της μάθησης δεν απαιτούν έναν «εξωτερικό» επιβλέποντα, γιατί οι αλγόριθμοι που χρησιμοποιούνται είναι αυτό-οργανώμενοι (με βοήθεια κάποιου άλλου αλγόριθμου ταξινόμησης) και βασίζονται σε

τοπική πληροφορία σε όλη τη διάρκεια εκπαίδευσης του δικτύου. Παραδείγματα αλγορίθμων μάθησης χωρίς επίβλεψη είναι ο αλγόριθμος Hebbian, ο διαφορικός αλγόριθμος Hebbian και ο Min-Max αλγόριθμος. [2] [3]

Συνήθως, οι περισσότερες διαδικασίες εκμάθησης είναι offline. Όταν το δείγμα προτύπων για την τροποποίηση των τιμών και των βαρών χρησιμοποιείται όλο, πριν την τελική χρήση του δικτύου ως εφαρμογή, τότε την ονομάζουμε offline εκμάθηση. Αυτά τα είδη των αλγορίθμων έχουν σαν απαίτηση να βρίσκονται όλα τα πρότυπα παρόντα στην εκμάθηση του δικτύου. Αυτό το γεγονός αποκλείει την πιθανότητα να εισέλθουν νέες πληροφορίες μέσω νέων προτύπων. Υπάρχουν νευρωνικά δίκτυα τα οποία δεν αποκλείουν την εισαγωγή νέας πληροφορίας, μετά την τελική τους μοντελοποίηση. Οι offline διαδικασίες εκμάθησης έχουν το πλεονέκτημα ότι επικεντρώνονται κυρίως στην δυνατότητα να δίνουν καλύτερες λύσεις σε δύσκολα προβλήματα. [2] [3]

<u>Πόσα είδη Νευρωνικών Δικτύων υπάρχουν;</u>

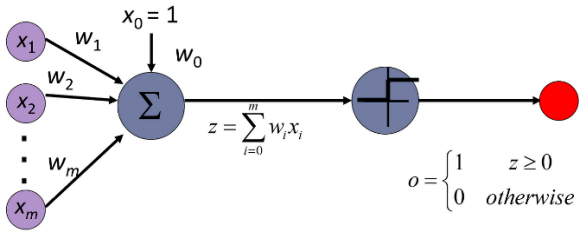
Υπάρχουν πολλά είδη νευρωνικών δικτύων διαθέσιμα, τα οποία μπορεί να είναι ήδη ανεπτυγμένα ή να βρίσκονται στο στάδιο ανάπτυξης. Τα νευρωνικά δίκτυα μπορούν να ταξινομηθούν ανάλογα με τα εξής χαρακτηριστικά: Δομή, Ροή δεδομένων, Νευρώνες που χρησιμοποιούνται, την πυκνότητά τους, επίπεδα και φίλτρα ενεργοποίησης βάθους κ.λπ. Παρακάτω θα μελετήσουμε τα βασικά είδη των νευρωνικών δικτύων που υπάρχουν.

Έχουμε εννέα γνωστά είδη νευρωνικών δικτύων και είναι τα εξής:

1. Perceptron[4]
2. Νευρωνικό δίκτυο τροφοδοσίας (Feed forward neural network) [4]
3. Πολυεπιπεδικό Perceptron (Multilayer Perceptron) [4]
4. Νευρωνικό δίκτυο συνέλιξης (Convolutional neural network) [4]
5. Νευρωνικό δίκτυο συνάρτησης ακτινικής βάσης (Radial basis functional neural network) [4]
6. Μακροχρόνια - Βραχυπρόθεσμη μνήμη (LSTM – long short – term memory) [4]
7. Επαναλαμβανόμενο νευρωνικό δίκτυο (Recurrent neural network) [4]
8. Μοντέλα ακολουθίας σε ακολουθία (Sequence to sequence models) [4]
9. Αρθρωτό νευρωνικό δίκτυο (Modular neural network) [4]

Perceptron

Το μοντέλο αυτό, προτεινόμενο από τους Marvin Minsky και Seymour Papert, είναι ένα από τα πιο απλά και παλιότερα μοντέλα ενός νευρώνα. Είναι η μικρότερη μονάδα νευρωνικού δικτύου που κάνει ορισμένους υπολογισμούς για την ανίχνευση χαρακτηριστικών ή επιχειρηματικής ευφυΐας στα δεδομένα εισόδου. Δέχεται σταθμισμένες εισόδους και εφαρμόζει τη συνάρτηση ενεργοποίησης για να λάβει την έξοδο ως τελικό αποτέλεσμα. Το Perceptron είναι γνωστό και ως TLU – Threshold Logic Unit (Λογική Μονάδα Κατωφλίου). [2] [4]



Εικόνα 2.2 Αναπαράσταση ενός Perceptron

Το Perceptron ανήκει στην κατηγορία της μάθησης με επίβλεψη και είναι ένας αλγόριθμος μάθησης που ταξινομεί τα δεδομένα σε δύο κατηγορίες, άρα επομένως ένα Perceptron ονομάζεται και δυαδικός ταξινομητής. [2] [4]

Θετικά [2] [4]

Το Perceptron μπορεί να εφαρμόσει λογικές πύλες όπως την AND, OR και NAND.

Αρνητικά [2] [4]

Τα perceptrons μπορούν να μάθουν μόνο γραμμικά διαχωρίσιμα προβλήματα, όπως το boolean AND πρόβλημα. Για μη γραμμικά προβλήματα, όπως το πρόβλημα boolean XOR, δεν λειτουργούν. [3]

Νευρωνικό δίκτυο τροφοδοσίας (Feed forward neural network)

Είναι ένας από τους δύο γενικούς τύπους νευρωνικών δικτύων, χαρακτηρίζεται από την κατεύθυνση της πληροφορίας μέσα στα επίπεδα. Η κατεύθυνση που παίρνει η πληροφορία είναι μονής (έχει μόνο τη διάδοση προς τα εμπρός) σε σύγκριση με το επαναλαμβανόμενο νευρωνικό δίκτυο, πράγμα που σημαίνει ότι η πληροφορία θα εισέλθει στην είσοδο, θα

πάει στο κρυφό επίπεδο (εάν έχει) και θα βγει στην έξοδο χωρίς διακλαδώσεις ή οπισθοδρόμηση στο δίκτυο [2] [4] (Παράδειγμα νευρωνικού δικτύου τροφοδοσίας, εικόνα 2.1, σελίδα 12).

Αυτό το είδος νευρωνικού δικτύου μπορεί να χαρακτηριστεί ως πολυεπιπεδικό ή μονοεπιπεδικό, ανάλογα με το πόσους νευρώνες θέλουμε να χρησιμοποιήσουμε, όπου τα επίπεδα καθορίζουν και την πολυπλοκότητα της συνάρτησης ενεργοποίησης. Αυτό σημαίνει ότι όσο πιο πολλοί νευρώνες, τόσο πιο περίπλοκη θα είναι η συνάρτηση και ανάποδα. [2] [4]

Εφαρμογές [2] [4]

1. Απλή ταξινόμηση, όπου οι παραδοσιακοί αλγόριθμοι ταξινόμησης που βασίζονται στη μηχανική μάθηση έχουν περιορισμούς.
2. Αναγνώριση προσώπου, απλή άμεση επεξεργασία εικόνας.
3. Αναγνώριση ομιλίας κ.ά.

Θετικά [2] [4]

1. Εύκολα στην κατασκευή, όχι πολύ περίπλοκα και εύκολα διαχειρίσιμα.

2. Γρήγορα στις διεργασίες που τους ανατίθενται, λόγω της μονόδρομης κατεύθυνσης της πληροφορίας.

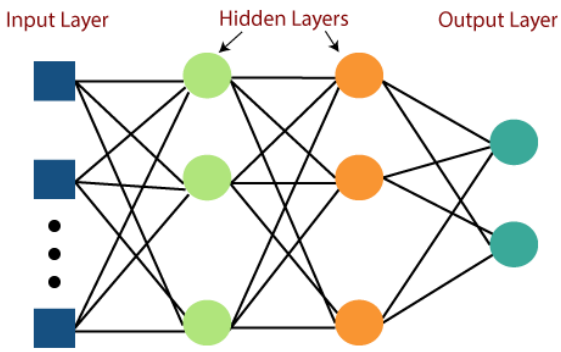
3. Διαχειρίζονται εύκολα την πληροφορία με πολύ θόρυβο.

Αρνητικά [2] [4]

Δεν μπορεί να χρησιμοποιηθεί στην βαθιά εκμάθηση (deep learning), λόγω απουσίας πυκνών επιπέδων και της πίσω διάδοσης της πληροφορίας.

Πολυεπιπεδικό Perceptron (Multilayer Perceptron)

Είναι ένα από τα πρώτα περίπλοκα νευρωνικά δίκτυα, όπου όλοι οι κόμβοι (νευρώνες) είναι συνδεδεμένοι μεταξύ τους, το οποίο το κάνει ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο. Η πληροφορία είναι διπλής κατεύθυνσης (έχει και διάδοση προς τα εμπρός αλλά και διάδοση προς τα πίσω). [2] [4]



Εικόνα 2.3 Αναπαράσταση ενός Πολυεπιπεδικού Perceptron (Multilayer Perceptron)

Η είσοδος πολλαπλασιάζεται με τα βάρη και τροφοδοτεί τη συνάρτηση ενεργοποίησης και στην πίσω διάδοση, ενώ τα βάρη τροποποιούνται αναλόγως για να μειωθεί η απώλεια πληροφορίας. Με λίγα λόγια, τα βάρη είναι μηχανικά μαθημένες τιμές από τα νευρωνικά δίκτυα. Αυτά τα δίκτυα αυτοπροσαρμόζονται ανάλογα με τη διαφορά μεταξύ των προβλεπόμενων εξόδων σε σχέση με τις εισόδους εκπαίδευσης. [2] [4]

Χρησιμοποιούνται μη-γραμμικές εξισώσεις, ακολουθούμενες από εξισώσεις ενεργοποίησης SoftMax (υπολογίζει την εκθετική e-δύναμη της δεδομένης τιμής εισόδου και το σύνολο των εκθετικών τιμών όλων των τιμών στις εισόδους) στο επίπεδο εξόδου. [2] [4]

Εφαρμογές [2] [4]

1. Αναγνώριση ομιλίας
2. Αυτόματη μετάφραση
3. Σύνθετη ταξινόμηση

Θετικά [2] [4]

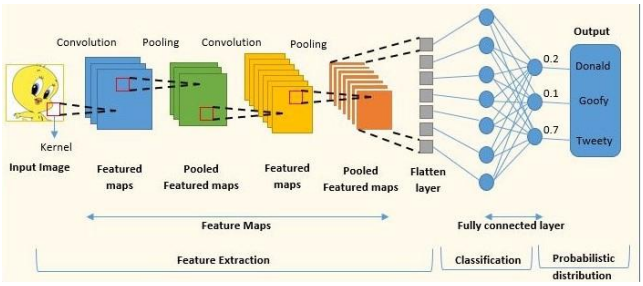
Χρησιμοποιείται στη βαθιά εκμάθηση (deep learning), λόγω της ύπαρξης πυκνών και πλήρως συνδεδεμένων επιπέδων και της πίσω διάδοσης.

Αρνητικά [2] [4]

1. Σε σχέση με άλλα δίκτυα, είναι πιο περίπλοκο στη σχεδίαση και συντήρηση.
2. Εξαρτώμενο από τον αριθμό των κρυφών επιπέδων, είναι σχετικά πιο αργό ως δίκτυο.

Νευρωνικό δίκτυο συνέλιξης (Convolutional neural network)

Αυτό το είδος δικτύου, αντί για την τυπική δυσδιάστατη διάταξη των νευρώνων, περιέχει μια τρισδιάστατη διάταξη των νευρώνων του. Η πληροφορία που επεξεργάζεται κυριότερα αυτό το δίκτυο είναι οι ψηφιακές εικόνες. [2] [4]



Εικόνα 2.4 Απεικόνιση ενός νευρωνικού δικτύου συνέλιξης

Το πρώτο επίπεδο του νευρωνικού δικτύου ονομάζεται συνελικτικό επίπεδο. Σε αυτό το επίπεδο, κάθε νευρώνας επεξεργάζεται μόνο μια μικρή περιοχή της εικόνας, όχι ολόκληρο το οπτικό πεδίο. Αυτή η επεξεργασία γίνεται με τη χρήση φίλτρων, τα οποία εφαρμόζονται σε διάφορα τμήματα της εικόνας. Το δίκτυο μαθαίνει να αντιλαμβάνεται την εικόνα σε μικρές μονάδες και μπορεί να επεξεργαστεί αυτές τις μικρές πληροφορίες πολλές φορές για να δημιουργήσει μια πλήρη αναπαράσταση της εικόνας. Η διαδικασία επεξεργασίας συμπεριλαμβάνει τη μετατροπή της εικόνας από τον χρωματικό χώρο RGB ή HSI σε κλίμακα του γκρι. Επιπλέον, ενδέχεται να γίνουν περαιτέρω τροποποιήσεις στις τιμές των pixel, προκειμένου να ενισχυθεί η ανίχνευση των άκρων, και οι εικόνες μπορούν να ταξινομηθούν σε διάφορες κατηγορίες. [2] [4]

Η ανάλυση αυτή εστιάζει στη χρήση των συνελικτικών νευρωνικών δικτύων (CNN) για την αναγνώριση και ταξινόμηση εικόνων. Συγκεκριμένα, το CNN αποτελείται από συνελικτικά στρώματα που επεξεργάζονται τμήματα της εικόνας με χρήση φίλτρων, ακολουθούμενα από στρώματα ομαδοποίησης. Στη συνέχεια, η έξοδος του τελευταίου στρώματος συνέλιξης τροφοδοτείται σε ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο (MLP) για την ταξινόμηση των εικόνων. [2] [4]

Τα φίλτρα χρησιμοποιούνται για την εξαγωγή σημαντικών χαρακτηριστικών από την εικόνα. Στο MLP, οι είσοδοι πολλαπλασιάζονται με βάρη και περνούν από μια λειτουργία ενεργοποίησης. Το επίπεδο συνέλιξης χρησιμοποιεί τη συνάρτηση ενεργοποίησης ReLU (Rectified Linear Unit), η οποία εισάγει μη γραμμικότητα στο δίκτυο και βοηθά στην

αποτελεσματική εκμάθηση χαρακτηριστικών, ενώ το MLP χρησιμοποιεί μη γραμμική συνάρτηση ενεργοποίησης (συνήθως ReLU ή άλλη) που ακολουθείται από τη συνάρτηση SoftMax για την ταξινόμηση των εξόδων σε κατηγορίες.[2] [4]

Τα συνελικτικά νευρωνικά δίκτυα έχουν αποδειχθεί αποτελεσματικά σε πολλές εφαρμογές, όπως η αναγνώριση εικόνας και βίντεο, η σημασιολογική ανάλυση και η ανίχνευση παράφρασης. [2] [4]

Εφαρμογές [2] [4]

1. Επεξεργασία εικόνας
2. Αναγνώριση ομιλίας
3. Αυτόματη μετάφραση
4. Όραση υπολογιστή

Θετικά [2] [4]

1. Χρησιμοποιείται στην βαθιά εκμάθηση (deep learning) με λίγες παραμέτρους.
2. Λίγοι παράμετροι για εκμάθηση σε σχέση με ένα πλήρες συνδεδεμένο επίπεδο.

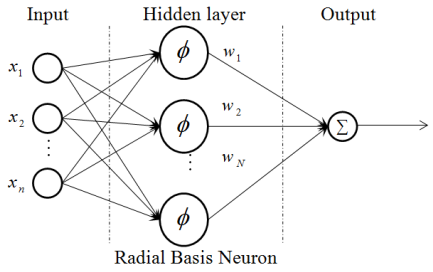
Αρνητικά [2] [4]

1. Σχετικά δύσκολο στην σχεδίαση και στην συντήρηση.
2. Σχετικά αργό (εξαρτάται από τον αριθμό των κρυμμένων επιπέδων).

Νευρωνικά δίκτυα συνάρτησης ακτινικής βάσης (Radial basis functional neural network)

Τα δίκτυα ακτινικής συνάρτησης βάσης (RBF) είναι ένας συγκεκριμένος τύπος νευρωνικού δικτύου που χρησιμοποιείται για προβλήματα προσέγγισης συναρτήσεων. Τα δίκτυα RBF (Radial Basis Function – Συνάρτηση Ακτινικής Βάσης) διαφέρουν από άλλα νευρωνικά δίκτυα ως προς την αρχιτεκτονική τους, που περιλαμβάνει τρία επίπεδα, την καθολική προσέγγιση και τη μεγαλύτερη ταχύτητα εκμάθησης. [2] [4]

Το νευρωνικό δίκτυο συνάρτησης ακτινικής βάσης περιλαμβάνει τρία κύρια στρώματα: το επίπεδο εισόδου που δέχεται ένα διάνυσμα εισόδου, το στρώμα των νευρώνων RBF και το επίπεδο εξόδου που περιλαμβάνει έναν κόμβο ανά κατηγορία για την ταξινόμηση. Η διαδικασία ταξινόμησης βασίζεται στον υπολογισμό της ομοιότητας μεταξύ της εισόδου και των πρωτοτύπων που αποθηκεύονται στο σύνολο εκπαίδευσης. [2] [4]



Εικόνα 2.4 Απεικόνηση ενός νευρονικού δικτύου συνάρτησης ακτινικής βάσης

Όταν πρέπει να ταξινομηθεί ένα νέο διάνυσμα εισόδου, κάθε νευρώνας υπολογίζει την Ευκλείδεια απόσταση ανάμεσα στην είσοδο και το πρωτότυπό του. Για παράδειγμα, αν έχουμε δύο κλάσεις (A και B), το νέο διάνυσμα εισόδου ταξινομείται στην κατηγορία που έχει τα πιο κοντινά πρωτότυπα. [2] [4]

Κάθε νευρώνας RBF παράγει μια τιμή από 0 έως 1 καθώς συγκρίνει την είσοδο με το πρωτότυπό του. Όταν η είσοδος είναι πανομοιότυπη με το πρωτότυπο, η έξοδος του νευρώνα RBF είναι 1, και όσο αυξάνεται η απόσταση μεταξύ της εισόδου και του πρωτότυπου, η απόκριση του νευρώνα μειώνεται εκθετικά προς το 0. Η απόκριση αυτή παριστάνει μια καμπύλη που θυμίζει καμπάνα. Το επίπεδο εξόδου περιλαμβάνει έναν νευρώνα για κάθε κατηγορία. [2] [4]

<u>Θετικά</u> [2] [4]

Τα δίκτυα με λειτουργία ακτινικής βάσης (RBF) έχουν πλεονεκτήματα εύκολης σχεδίασης, καλής γενίκευσης, ισχυρής ανοχής στον θόρυβο εισόδου και ικανότητας διαδικτυακής εκμάθησης. Οι ιδιότητες των δικτύων RBF καθιστούν πολύ κατάλληλο το σχεδιασμό ευέλικτων συστημάτων ελέγχου.

<u>Αρνητικά</u> [2] [4]

Τα δίκτυα RBF έχουν το μειονέκτημα ότι απαιτούν καλή κάλυψη του χώρου εισόδου από συναρτήσεις ακτινικής βάσης. Τα κέντρα δικτύου RBF προσδιορίζονται με αναφορά στην κατανομή των δεδομένων εισόδου, αλλά χωρίς αναφορά στην εργασία πρόβλεψης.

Μακροχρόνια – Βραχυπρόθεσμη μνήμη (LSTM – Long Short Term Memory)

Τα δίκτυα LSTM είναι ένας τύπος Αναδρομικού Νευρωνικού Δικτύου (RNN) που χρησιμοποιεί ειδικές μονάδες, εκτός από τις τυπικές μονάδες. Οι μονάδες LSTM περιλαμβάνουν ένα 'κυψέλη μνήμης' που μπορεί να διατηρεί πληροφορίες στη μνήμη για μεγάλα χρονικά διαστήματα. Χρησιμοποιείται ένα σύνολο από πύλες για να ελέγχει πότε η πληροφορία εισέρχεται στη μνήμη, πότε εξέρχεται και πότε λησμονείται. Υπάρχουν τρία είδη πυλών, δηλαδή η πύλη εισόδου, η πύλη εξόδου και η πύλη λησμόνησης. Η πύλη εισόδου αποφασίζει πόσες πληροφορίες από τον τελευταίο δείγμα θα διατηρηθούν στη μνήμη. Η πύλη εξόδου ρυθμίζει το ποσό δεδομένων που περνά στο επόμενο επίπεδο, και οι πύλες λησμόνησης ελέγχουν το ρυθμό κατακόρυφης επεξεργασίας της αποθηκευμένης μνήμης. Αυτή η αρχιτεκτονική τους επιτρέπει να μάθουν μακροπρόθεσμες εξαρτήσεις [2] [4] (Δες εικόνα 2.5 σελίδα 24).

Αυτή είναι μία από τις υλοποιήσεις των κυττάρων LSTM, υπάρχουν πολλές άλλες αρχιτεκτονικές. [2] [4]

Θετικά [2] [4]

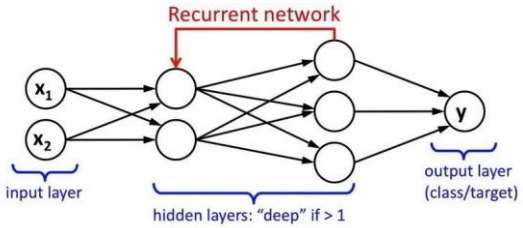
Έχουν μια μνήμη που τους επιτρέπει να αποθηκεύουν και να επαναχρησιμοποιούν πληροφορίες από προηγούμενες εισόδους.

Αρνητικά [2] [4]

1. Πρώτον, είναι πιο περίπλοκα από τα παραδοσιακά RNN και απαιτούν περισσότερα δεδομένα εκπαίδευσης προκειμένου να μάθουν αποτελεσματικά.
2. Δεύτερον, δεν είναι κατάλληλα για εργασίες διαδικτυακής μάθησης, όπως εργασίες πρόβλεψης ή ταξινόμησης όπου τα δεδομένα εισόδου δεν είναι μια ακολουθία.
3. Τα LSTM χρειάζονται περισσότερο χρόνο για να εκπαιδευτούν.
4. Τα LSTM απαιτούν περισσότερη μνήμη για εκπαίδευση.
5. Τα LSTM είναι εύκολο να υπερπροσαρμοστούν.
6. Το dropout είναι πολύ πιο δύσκολο να εφαρμοστεί σε LSTM.
7. Τα LSTM είναι ευαίσθητα σε διαφορετικές αρχικοποιήσεις τυχαίου βάρους.

Επαναλαμβανόμενο νευρωνικό δίκτυο (Recurrent Neural Network - RNN)

Το Επαναλαμβανόμενο Νευρωνικό Δίκτυο (Recurrent Neural Network - RNN) έχει σχεδιαστεί με σκοπό να διατηρεί την πληροφορία που παράγεται από ένα επίπεδο και να την ανατροφοδοτεί στην είσοδο, προκειμένου να βοηθήσει στην πρόβλεψη των αποτελεσμάτων του. Το Επαναλαμβανόμενο Νευρωνικό Δίκτυο (RNN) είναι ένας τύπος νευρωνικού δικτύου που χρησιμοποιεί διαδοχικά δεδομένα ή δεδομένα χρονοσειρών. Συνήθως, το πρώτο επίπεδο είναι ένα feedforward νευρωνικό δίκτυο, το οποίο ακολουθείται από ένα επαναλαμβανόμενο στρώμα νευρωνικού δικτύου. [2] [4]



Εικόνα 2.5 Απεικόνηση ενός επαναλαμβανομενου νευρωνικου δικτυου RNN

Στο επαναλαμβανόμενο στρώμα, ορισμένες πληροφορίες που δημιουργούνται σε προηγούμενα χρονικά βήματα αποθηκεύονται σε μια συνάρτηση μνήμης. Κατά την προώθηση προς τα εμπρός, αυτές οι αποθηκευμένες πληροφορίες λαμβάνονται υπόψη, προκειμένου να συμβάλουν στην πρόβλεψη. Εάν η πρόβλεψη δεν είναι σωστή, χρησιμοποιείται ο ρυθμός μάθησης για να διορθώσει μικρές αλλαγές στο δίκτυο. Αυτό έχει ως αποτέλεσμα την αύξηση της ακρίβειας των προβλέψεων κατά την οπισθοδιάδοση, καθιστώντας το δίκτυο ικανότερο να προβλέπει σωστά στο μέλλον. [2] [4]

Εφαρμογές [2] [4]

- Επεξεργασία κειμένου όπως αυτόματη πρόταση, γραμματικοί έλεγχοι κ.λπ.
- Επεξεργασία κειμένου σε ομιλία
- Tagger εικόνας
- Ανάλυση Συναισθήματος
- Μετάφραση

Θετικά [2] [4]

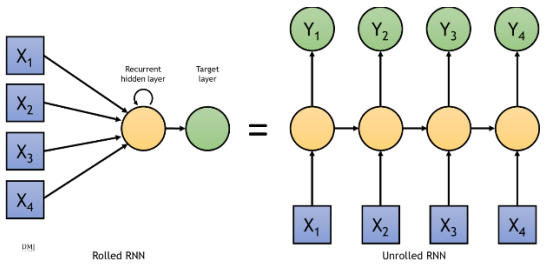
1. Ένα από τα πλεονεκτήματα μοντελοποίησης διαδοχικών δεδομένων είναι η δυνατότητα να θεωρηθεί ότι κάθε δείγμα εξαρτάται από το ιστορικό του.
2. Χρησιμοποιείται με στρώματα συνέλιξης για επέκταση της αποτελεσματικότητας των εικονοστοιχείων.

Αρνητικά [2] [4]

1. Προβλήματα εξασθένησης και εκρηκτικής αύξησης της κλίσης.
2. Η εκπαίδευση ενός επαναλαμβανόμενου νευρωνικού δικτύου θα μπορούσε να είναι μια δύσκολη εργασία.
3. Δύσκολη η επεξεργασία μεγάλων διαδοχικών δεδομένων χρησιμοποιώντας το ReLU ως λειτουργία ενεργοποίησης.

Μοντέλα ακολουθίας σε ακολουθία (Sequence to sequence models)

Ένα μοντέλο ακολουθίας προς ακολουθία αποτελείται από δύο Αναδρομικά Νευρωνικά Δίκτυα. Σε αυτό, υπάρχει ένας κωδικοποιητής που επεξεργάζεται την είσοδο και ένας αποκωδικοποιητής που επεξεργάζεται την έξοδο. Οι δύο αυτές μονάδες λειτουργούν ταυτόχρονα, είτε χρησιμοποιώντας τις ίδιες παραμέτρους είτε διαφορετικές. Αυτό το μοντέλο, αντίθετα από το συμβατικό RNN, είναι εξαιρετικά χρήσιμο σε περιπτώσεις όπου το μήκος των δεδομένων εισόδου είναι ίσο με το μήκος των δεδομένων εξόδου. Παρόλο που έχουν παρόμοια πλεονεκτήματα και περιορισμούς με το RNN, αυτά τα μοντέλα εφαρμόζονται συνήθως κυρίως σε chatbots, μηχανές μεταφράσεων και συστήματα απάντησης σε ερωτήσεις. [2] [4]



Εικόνα 2.6 Απεικόνιση ενός μοντέλου ακολούθιας σε ακολουθεία

Θετικά [2] [4]

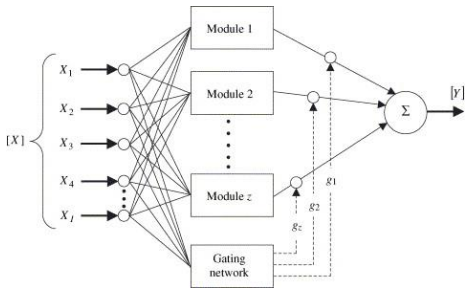
Μπορούν να χειριστούν ένα ευρύ φάσμα εργασιών όπως η αυτόματη μετάφραση, η σύνοψη κειμένου και η δημιουργία λεζάντας εικόνων, καθώς και ακολουθίες εισόδου και εξόδου μεταβλητού μήκους.

Αρνητικά [2] [4]

Συχνά δυσκολεύονται να καταγράψουν μακροπρόθεσμες εξαρτήσεις και πληροφορίες περιβάλλοντος, ειδικά όταν οι ακολουθίες είναι πολύ μεγάλες ή πολύπλοκες.

Αρθρωτό νευρωνικό δίκτυο (Modular neural network)

Ένα αρθρωτό νευρωνικό δίκτυο αποτελείται από πολλαπλά ανεξάρτητα λειτουργικά δίκτυα, τα οποία εκτελούν υπό-εργασίες χωρίς καμία αλληλεπίδραση ή συνεργασία κατά τη διάρκεια της υπολογιστικής διαδικασίας. Κάθε δίκτυο αυτόνομα συμβάλλει στην επίτευξη του τελικού αποτελέσματος. [2] [4]



Εικόνα 2.7 Απεικόνιση ενός αρθρωτού νευρωνικού δικτυού

Η διάσπαση μιας πολύπλοκης υπολογιστικής διαδικασίας σε ανεξάρτητα στοιχεία επιφέρει σημαντική επιτάχυνση στην εκτέλεσή της. Αυτό συμβαίνει λόγω της απουσίας αλληλεπίδρασης και σύνδεσης μεταξύ των στοιχείων, προσφέροντας έτσι αυξημένη υπολογιστική ταχύτητα. [2] [4]

Εφαρμογές [2] [4]

1. Συστήματα πρόβλεψης χρηματιστηρίου
2. Προσαρμοστικό MNN για αναγνώριση χαρακτήρων
3. Συμπίεση δεδομένων εισόδου υψηλού επιπέδου

Θετικά [2] [4]

1. Αποτελεσματικός
2. Ανεξάρτητη εκπαίδευση
3. Ευρωστία

Αρνητικά [2] [4]

Προβλήματα κινούμενου στόχου

2.1 VHDL–VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

Η VHDL είναι μια γλώσσα προγραμματισμού γενικής χρήσης περιγραφής υλικού, βελτιστοποιημένη για τον σχεδιασμό ηλεκτρονικών κυκλωμάτων ή συστημάτων. Η VHDL προκαθορίζεται για τη σύνθεση και προσομοίωση κυκλωμάτων. Παρ' όλο που οι κώδικες στην VHDL με σωστή σύνταξη μπορούν να προσομοιωθούν, δεν είναι όλες οι δομές, όμως, συνθέσιμες στην πραγματικότητα. [2] [5]

Η VHDL πήρε το όνομά της από το πρόγραμμα του Υπουργείου Άμυνας των Ηνωμένων Πολιτειών που το δημιούργησε, το Πρόγραμμα Ολοκληρωμένων Κυκλωμάτων Πολύ Υψηλής Ταχύτητας (VHSIC). Στις αρχές της δεκαετίας του 1980, το Πρόγραμμα VHSIC αναζήτησε μια νέα HDL για χρήση στο σχεδιασμό των ολοκληρωμένων κυκλωμάτων που σκόπευε να αναπτύξει. Το προϊόν αυτής της προσπάθειας ήταν η VHDL Version 7.2, που κυκλοφόρησε το 1985. Η προσπάθεια τυποποίησής του ως πρότυπο IEEE ξεκίνησε τον επόμενο χρόνο. [2] [5]

Το 1983, η VHDL αναπτύχθηκε αρχικά κατόπιν εντολής του Υπουργείου Άμυνας των ΗΠΑ προκειμένου να τεκμηριωθεί η συμπεριφορά των ASIC που περιλάμβαναν οι εταιρείες προμηθευτών στον εξοπλισμό. [2] [5]

Η ιδέα της δυνατότητας προσομοίωσης των ASIC από τις πληροφορίες αυτής της τεκμηρίωσης ήταν τόσο προφανώς ελκυστική, που αναπτύχθηκαν λογικοί προσομοιωτές που μπορούσαν να διαβάσουν τα αρχεία VHDL. Το επόμενο βήμα ήταν η ανάπτυξη εργαλείων λογικής σύνθεσης που διαβάζουν το VHDL και εξάγουν έναν ορισμό της φυσικής υλοποίησης του κυκλώματος. [2] [5]

Λόγω του ότι το Υπουργείο Άμυνας απαιτεί όσο το δυνατόν μεγαλύτερο μέρος της σύνταξης να βασίζεται στο Ada, προκειμένου να αποφευχθεί η επανεφεύρεση εννοιών που είχαν ήδη δοκιμαστεί διεξοδικά στην ανάπτυξη του Ada, το VHDL δανείζεται σε μεγάλο βαθμό από τη γλώσσα προγραμματισμού Ada τόσο στην έννοια όσο και στη σύνταξη. [2][5] Η VHDL χρησιμοποιείται γενικά για τη σύνταξη μοντέλων κειμένου που περιγράφουν ένα λογικό κύκλωμα. Ένα τέτοιο μοντέλο επεξεργάζεται από ένα πρόγραμμα σύνθεσης, μόνο εάν είναι μέρος του λογικού σχεδιασμού. Ένα πρόγραμμα προσομοίωσης χρησιμοποιείται για τον έλεγχο του λογικού σχεδιασμού, χρησιμοποιώντας μοντέλα προσομοίωσης για την αναπαράσταση των λογικών κυκλωμάτων που διασυνδέονται με το σχέδιο. Αυτή η συλλογή μοντέλων προσομοίωσης ονομάζεται συνήθως πάγκος δοκιμών. [2][5] Ένας προσομοιωτής VHDL είναι συνήθως ένας προσομοιωτής που βασίζεται σε γεγονότα. Αυτό σημαίνει ότι κάθε συναλλαγή προστίθεται σε μια ουρά συμβάντων για μια συγκεκριμένη προγραμματισμένη ώρα. Π.χ. εάν γίνει εκχώρηση σήματος μετά από 1 νανοδευτερόλεπτο, το συμβάν προστίθεται στην ουρά για το χρόνο +1 ns. Επιτρέπεται επίσης μηδενική καθυστέρηση, αλλά πρέπει να προγραμματιστεί ακόμα: για αυτές τις περιπτώσεις χρησιμοποιείται καθυστέρηση δέλτα, η οποία αντιπροσωπεύει ένα απείρως μικρό χρονικό βήμα. Η προσομοίωση αλλάζει μεταξύ δύο τρόπων: εκτέλεσης δηλώσεων, όπου αξιολογούνται οι δηλώσεις ενεργοποίησης, και επεξεργασίας συμβάντων, όπου επεξεργάζονται συμβάντα στην ουρά. [2][5]

Το VHDL έχει κατασκευές για να χειρίζεται τον παραλληλισμό που είναι εγγενής στα σχέδια υλικού, αλλά αυτές οι δομές (διαδικασίες) διαφέρουν στη σύνταξη από τις παράλληλες κατασκευές στο Ada (εργασίες). Όπως και η Ada, η VHDL πληκτρολογείται έντονα και δεν κάνει διάκριση πεζών-κεφαλαίων. Για την άμεση αναπαράσταση

λειτουργιών που είναι κοινές στο υλικό, υπάρχουν πολλά χαρακτηριστικά του VHDL που δεν βρίσκονται στο Ada, όπως ένα εκτεταμένο σύνολο τελεστών Boole, συμπεριλαμβανομένων των nand και nor. [2][5]

Το VHDL έχει δυνατότητες εισαγωγής και εξόδου αρχείων και μπορεί να χρησιμοποιηθεί ως γλώσσα γενικής χρήσης για την επεξεργασία κειμένου, αλλά τα αρχεία χρησιμοποιούνται πιο συχνά από έναν πάγκο δοκιμών προσομοίωσης για δεδομένα ερεθίσματος ή επαλήθευσης. Υπάρχουν ορισμένοι μεταγλωττιστές VHDL που δημιουργούν εκτελέσιμα δυαδικά αρχεία. Σε αυτήν την περίπτωση, μπορεί να είναι δυνατό να χρησιμοποιηθεί η VHDL για τη σύνταξη ενός δοκιμαστικού πάγκου για την επαλήθευση της λειτουργικότητας του σχεδίου, χρησιμοποιώντας αρχεία στον κεντρικό υπολογιστή για τον ορισμό ερεθισμάτων, την αλληλεπίδραση με τον χρήστη και τη σύγκριση των αποτελεσμάτων με τα αναμενόμενα. Ωστόσο, οι περισσότεροι σχεδιαστές αφήνουν αυτή τη δουλειά στον προσομοιωτή. [2][5]

Είναι σχετικά εύκολο για έναν άπειρο προγραμματιστή να παράγει κώδικα που προσομοιώνεται με επιτυχία αλλά δεν μπορεί να συντεθεί σε μια πραγματική συσκευή ή είναι πολύ μεγάλος για να είναι πρακτικός. Μια ιδιαίτερη παγίδα είναι η κατά λάθος παραγωγή διαφανών μανδαλώσεων αντί για flip-flops τύπου D ως στοιχεία αποθήκευσης. [2][5]

Κάποιος μπορεί να σχεδιάσει υλικό σε ένα VHDL IDE (για υλοποίηση FPGA, όπως Xilinx ISE, Altera Quartus, Synopsys Synplify ή Mentor Graphics HDL Designer) για να παράγει το σχηματικό RTL του επιθυμητού κυκλώματος. Μετά από αυτό, το παραγόμενο σχηματικό μπορεί να επαληθευτεί χρησιμοποιώντας λογισμικό προσομοίωσης, το οποίο δείχνει τις κυματομορφές των εισόδων και εξόδων του κυκλώματος μετά τη δημιουργία του κατάλληλου πάγκου δοκιμών. Για να δημιουργηθεί ένας κατάλληλος πάγκος δοκιμών για ένα συγκεκριμένο κύκλωμα ή κώδικα VHDL, οι είσοδοι πρέπει να οριστούν σωστά. Για παράδειγμα, για την είσοδο ρολογιού απαιτείται μια διαδικασία βρόχου ή μια επαναληπτική δήλωση.[2][5]

Ένα τελευταίο σημείο είναι ότι, όταν ένα μοντέλο VHDL μεταφράζεται στις "πύλες και καλώδια" που αντιστοιχίζονται σε μια προγραμματιζόμενη λογική συσκευή όπως CPLD ή

FPGA, τότε διαμορφώνεται το πραγματικό υλικό, αντί να "εκτελείται" ο κώδικας VHDL "σαν κάποια μορφή τσιπ επεξεργαστή". [2] [5]

Το βασικό πλεονέκτημα της VHDL, όταν χρησιμοποιείται για το σχεδιασμό συστημάτων, είναι ότι επιτρέπει την περιγραφή (μοντελοποίηση) και την επαλήθευση (προσομοίωση) της συμπεριφοράς του απαιτούμενου συστήματος, προτού τα εργαλεία σύνθεσης μεταφράσουν το σχέδιο σε πραγματικό υλικό (πύλες και καλώδια). [2][5]

Ένα άλλο πλεονέκτημα είναι ότι η VHDL επιτρέπει την περιγραφή ενός ταυτόχρονου συστήματος. Η VHDL είναι μια γλώσσα ροής δεδομένων, στην οποία κάθε πρόταση λαμβάνεται υπόψη για εκτέλεση ταυτόχρονα, σε αντίθεση με τις διαδικαστικές γλώσσες υπολογιστών όπως οι BASIC, C και ο κώδικας συναρμολόγησης, όπου μια ακολουθία εντολών εκτελείται διαδοχικά, μία εντολή τη φορά. [2][5]

Ένα έργο VHDL είναι πολλαπλών χρήσεων. Όταν δημιουργείται μία φορά, ένα μπλοκ υπολογισμού μπορεί να χρησιμοποιηθεί σε πολλά άλλα έργα. Ωστόσο, πολλές διαμορφωτικές και λειτουργικές παράμετροι του μπλοκ μπορούν να συντονιστούν (παράμετροι χωρητικότητας, μέγεθος μνήμης, βάση στοιχείου, σύνθεση μπλοκ και δομή διασύνδεσης). [2][5]

Ένα έργο VHDL είναι φορητό. Έχοντας δημιουργηθεί για μια βάση στοιχείων, ένα έργο υπολογιστικής συσκευής μπορεί να μεταφερθεί σε μια άλλη βάση στοιχείων, για παράδειγμα σε VLSI με διάφορες τεχνολογίες. [2][5]

Ένα μεγάλο πλεονέκτημα της VHDL σε σύγκριση με την αρχική Verilog είναι ότι η VHDL διαθέτει σύστημα πλήρους τύπου. Οι σχεδιαστές μπορούν να χρησιμοποιήσουν το σύστημα τύπων για να γράψουν πολύ πιο δομημένο κώδικα (ειδικά δηλώνοντας τύπους εγγραφών). [2] [5]

2.2 FPGA (Field Programmable Gate Array)

Το FPGA είναι μια διάταξη από πύλες που έχουν τη δυνατότητα να προγραμματίζονται επιτόπου· αυτός είναι ένας τύπος ολοκληρωμένου κυκλώματος που έχει τη δυνατότητα να προγραμματιστεί ή να επαναπρογραμματιστεί μετά την κατασκευή του. Αποτελείται από μια σειρά προγραμματιζόμενων λογικών μπλοκ και διασυνδέσεων που ρυθμίζονται με τέτοιο τρόπο ώστε να εκτελούν διάφορες ψηφιακές λειτουργίες. Τα FPGA συνήθως χρησιμοποιούνται σε εφαρμογές όπου απαιτείται ευελιξία, ταχύτητα και δυνατότητα παράλληλης επεξεργασίας, όπως στις τηλεπικοινωνίες, την αυτοκινητοβιομηχανία, την αεροδιαστημική και τη βιομηχανία. [2][5]

Η διαμόρφωση ενός FPGA καθορίζεται χρησιμοποιώντας μια γλώσσα περιγραφής υλικού (HDL – Hardware Description Language), παρόμοια με αυτή που χρησιμοποιείται για ένα ολοκληρωμένο κύκλωμα συγκεκριμένης εφαρμογής (ASIC – Application Specific Integrated Circuit). Τα διαγράμματα κυκλωμάτων χρησιμοποιούνταν προηγουμένως για τον καθορισμό της διαμόρφωσης, αλλά αυτό είναι όλο και πιο σπάνιο λόγω της εμφάνισης των ηλεκτρονικών εργαλείων αυτοματισμού σχεδιασμού. [2][5]

Τα λογικά μπλοκ ενός FPGA μπορούν να ρυθμιστούν έτσι ώστε να εκτελούν σύνθετες συνδυαστικές λειτουργίες ή να λειτουργούν ως απλές λογικές πύλες όπως AND και XOR. Στα περισσότερα FPGA, τα λογικά μπλοκ περιλαμβάνουν και στοιχεία μνήμης, τα οποία μπορεί να είναι απλά flip-flops ή πιο ολοκληρωμένα μπλοκ μνήμης. Πολλά FPGA μπορούν να επαναπρογραμματιστούν για να υλοποιήσουν διαφορετικές λογικές λειτουργίες, επιτρέποντας έτσι ευέλικτους επαναδιαμορφώσιμους υπολογισμούς όπως εκτελούνται σε λογισμικό υπολογιστή. [2][5]

Τα FPGA διαδραματίζουν επίσης ρόλο στην ανάπτυξη ενσωματωμένων συστημάτων λόγω της ικανότητάς τους να ξεκινούν την ανάπτυξη λογισμικού συστήματος ταυτόχρονα με το υλικό, να επιτρέπουν προσομοιώσεις απόδοσης συστήματος σε πολύ πρώιμο στάδιο ανάπτυξης και να επιτρέπουν διάφορες δοκιμές συστήματος και επαναλήψεις σχεδίασης πριν από την οριστικοποίηση της αρχιτεκτονικής του συστήματος. [2] [5]

2.2.1 Σχεδιασμός των FPGA

Τα σύγχρονα FPGA διαθέτουν άφθονες λογικές πύλες και μπλοκ RAM για την υλοποίηση πολύπλοκων ψηφιακών υπολογισμών. Μπορούν να εκτελέσουν οποιαδήποτε λογική λειτουργία που θα μπορούσε να υλοποιηθεί σε ASIC, ενώ προσφέρουν πλεονεκτήματα όπως η δυνατότητα ενημέρωσης της λειτουργικότητας μετά την αποστολή, η μερική επαναδιαμόρφωση μέρους του σχεδιασμού και το χαμηλό μη επαναλαμβανόμενο κόστος ανάπτυξης, παρά το γενικά υψηλότερο κόστος ανά μονάδα. [2][5]

Καθώς τα σχέδια FPGA χρησιμοποιούν υψηλές ταχύτητες εισόδου/εξόδου και αμφίδρομους διαύλους δεδομένων, η διασφάλιση του σωστού χρονισμού - εντός των ορίων εγκατάστασης και αναμονής - αποτελεί πρόκληση. Ο σχεδιασμός δαπέδου συμβάλλει στη βέλτιστη κατανομή πόρων ώστε να ικανοποιούνται αυτοί οι χρονικοί περιορισμοί. [2][5]

Ορισμένα FPGA ενσωματώνουν και αναλογικά χαρακτηριστικά πέρα από τις ψηφιακές λειτουργίες. Το πιο συνηθισμένο είναι ο προγραμματιζόμενος ρυθμός ανόδου/καθόδου σε κάθε ακροδέκτη εξόδου, επιτρέποντας τη μείωση των θορύβων (ringing) σε ελαφρώς φορτωμένους ακροδέκτες και την αύξηση της ταχύτητας σε βαριά φορτωμένα κανάλια. Επιπλέον, διαθέτουν κυκλώματα οδήγησης ταλαντωτών χαλαζία, ταλαντωτές RC στο τσιπ και βρόχους κλειδώματος φάσης (PLL) με ενσωματωμένους ταλαντωτές ελεγχόμενης τάσης, χρήσιμους για τη διαχείριση των ρολογιών και των σειριακών διασυνδέσεων υψηλής ταχύτητας (SERDES). Συνηθισμένοι είναι και οι διαφορικοί συγκριτές σε ακίδες εισόδου, σχεδιασμένοι για διαφορική σηματοδότηση. Μερικά FPGA μικτού σήματος διαθέτουν επίσης ενσωματωμένους ADC και DAC με αναλογικά μπλοκ ρύθμισης σήματος, λειτουργώντας ως ολοκληρωμένα συστήματα σε τσιπ (SoC). Έτσι, η διαχωριστική γραμμή μεταξύ των κλασικών FPGA και των FPAA (Field Programmable Analog Array) αρχίζει να θολώνει. [2][5]

Η βασική αρχιτεκτονική των FPGA αποτελείται από μία διάταξη λογικών μπλοκ, γνωστών ως ρυθμιζόμενα λογικά μπλοκ (CLB) ή μπλοκ λογικής διάταξης (LAB), ανάλογα με τον

προμηθευτή, καθώς και από επιθέματα εισόδου/εξόδου και κανάλια δρομολόγησης. Συνήθως, τα κανάλια δρομολόγησης έχουν ομοιόμορφο πλάτος (σε αριθμό σημάτων), ενώ πολλαπλά επιθέματα I/O μπορούν να χωρέσουν στο ύψος μιας γραμμής ή στο πλάτος μιας στήλης στον πίνακα. [2][5]

Για να υλοποιηθεί ένα κύκλωμα σε FPGA, πρέπει να υπάρχει επάρκεια σε πόρους. Ο απαιτούμενος αριθμός λογικών μπλοκ και ακίδων I/O είναι συνήθως εύκολο να υπολογιστεί, όμως ο αριθμός των απαιτούμενων διαύλων δρομολόγησης μπορεί να διαφέρει σημαντικά, ακόμη και ανάμεσα σε σχέδια με την ίδια ποσότητα λογικής. Για παράδειγμα, ένας διακόπτης τύπου crossbar απαιτεί πολύ περισσότερη δρομολόγηση από μία συστολική συστοιχία. Καθώς τα αχρησιμοποίητα κανάλια αυξάνουν το κόστος και μειώνουν την απόδοση χωρίς να προσφέρουν όφελος, οι κατασκευαστές επιδιώκουν να παρέχουν τόσους διαύλους ώστε να δρομολογούνται αποτελεσματικά τα περισσότερα σχέδια. Η επιλογή του αριθμού διαύλων βασίζεται σε εκτιμήσεις όπως ο κανόνας του Rent ή σε εμπειρικά δεδομένα. [2][5]

Ένα λογικό μπλοκ περιέχει συνήθως μερικά λογικά κελιά. Ένα τυπικό κύτταρο περιλαμβάνει ένα LUT τεσσάρων εισόδων, έναν πλήρη αθροιστή (FA) και ένα flip-flop τύπου D. Το LUT μπορεί να διασπαστεί σε δύο μικρότερα LUT τριών εισόδων. Στη συνδυαστική λειτουργία, αυτά συνδυάζονται μέσω ενός πολυπλέκτη σε ένα τετραεισαγωγικό LUT. Στην αριθμητική λειτουργία, οι έξοδοί τους τροφοδοτούν τον αθροιστή, με τη λειτουργία να επιλέγεται μέσω άλλου πολυπλέκτη. Τέλος, η έξοδος μπορεί να είναι είτε σύγχρονη είτε ασύγχρονη, ανάλογα με τον τρίτο πολυπλέκτη. Σε πολλές περιπτώσεις, τμήματα του αθροιστή υλοποιούνται απευθείας στα LUT για εξοικονόμηση χώρου. [2][5]

Οι σύγχρονες οικογένειες FPGA ενσωματώνουν ακόμα περισσότερες λειτουργίες σε επίπεδο πυριτίου, μειώνοντας τον απαιτούμενο χώρο και αυξάνοντας την απόδοση σε σχέση με την υλοποίηση τους μέσω πρωτόγονων λογικών στοιχείων. Τέτοιες λειτουργίες περιλαμβάνουν πολλαπλασιαστές, μπλοκ DSP, ενσωματωμένους επεξεργαστές, λογικές I/O υψηλής ταχύτητας και μνήμες.[2][5]

Τα κορυφαία FPGA ενσωματώνουν επίσης πομποδέκτες πολλαπλών Gigabit και "σκληρούς" πυρήνες IP, όπως πυρήνες επεξεργαστών, ελεγκτές Ethernet MAC, PCI ή PCI Express, και ελεγκτές εξωτερικής μνήμης. Οι πυρήνες αυτοί, κατασκευασμένοι από ειδικά τρανζίστορ και όχι από LUT, προσφέρουν επιδόσεις και κατανάλωση ενέργειας συγκρίσιμες με ASIC, χωρίς να δεσμεύουν μεγάλο μέρος του προγραμματιζόμενου υφάσματος. Οι πομποδέκτες περιλαμβάνουν επίσης κυκλώματα ρύθμισης σήματος υψηλής απόδοσης, σειριοποιητές και αποσειριοποιητές, λειτουργίες αδύνατες να υλοποιηθούν με LUT. Η λειτουργικότητα φυσικού επιπέδου (PHY), όπως η κωδικοποίηση γραμμής, μπορεί να υλοποιείται είτε σε σκληρή λογική είτε να αφήνεται στον χρήστη, ανάλογα με το FPGA. [2][5]

2.2.2 Προγραμματισμός των FPGA

Για να οριστεί η συμπεριφορά του FPGA, ο χρήστης παρέχει ένα σχέδιο σε γλώσσα περιγραφής υλικού (HDL) ή ως σχηματικό σχεδιασμό. Η φόρμα HDL είναι πιο κατάλληλη για εργασία με μεγάλες δομές, επειδή είναι δυνατό να καθοριστεί λειτουργική συμπεριφορά υψηλού επιπέδου, αντί να σχεδιάζεται κάθε κομμάτι με το χέρι. Ωστόσο, η σχηματική καταχώριση μπορεί να επιτρέψει την ευκολότερη οπτικοποιήση ενός σχεδίου και των λειτουργικών μονάδων του. [2][5]

Χρησιμοποιώντας ένα ηλεκτρονικό εργαλείο αυτοματισμού σχεδιασμού, δημιουργείται μια δικτυακή λίστα χαρτογράφησης τεχνολογίας. Η netlist μπορεί στη συνέχεια να προσαρμοστεί στην πραγματική αρχιτεκτονική FPGA χρησιμοποιώντας μια διαδικασία που ονομάζεται *place-and-route*, η οποία συνήθως εκτελείται από το ιδιόκτητο λογισμικό *place-and-route* της εταιρείας FPGA. Ο χρήστης θα επικυρώσει τα αποτελέσματα του χάρτη, του τόπου και της διαδρομής μέσω ανάλυσης χρονισμού, προσομοίωσης και άλλων μεθοδολογιών επαλήθευσης και επικύρωσης. Μόλις ολοκληρωθεί η διαδικασία σχεδίασης και επικύρωσης, το δυαδικό αρχείο που δημιουργείται —συνήθως χρησιμοποιώντας το ιδιόκτητο λογισμικό του προμηθευτή FPGA— χρησιμοποιείται για την (εκ νέου) διαμόρφωση του FPGA. Αυτό το αρχείο μεταφέρεται στο FPGA/CPLD μέσω μιας σειριακής διεπαφής (JTAG) ή σε μια εξωτερική συσκευή μνήμης, όπως μια EEPROM (*Electrically Erasable Programmable Read-Only Memory*). [2][5]

Τα πιο κοινά HDL είναι τα VHDL και Verilog, καθώς και επεκτάσεις όπως το SystemVerilog. Ωστόσο, σε μια προσπάθεια να μειωθεί η πολυπλοκότητα του σχεδιασμού σε HDL, οι οποίες έχουν συγκριθεί με το αντίστοιχο των γλωσσών assembly, υπάρχουν κινήσεις (από ποιον;) για να αυξηθεί το επίπεδο αφαίρεσης μέσω της εισαγωγής εναλλακτικών γλωσσών. Η γλώσσα προγραμματισμού γραφικών LabVIEW της National Instruments (μερικές φορές αναφέρεται ως G) διαθέτει μια πρόσθετη μονάδα FPGA, διαθέσιμη για στόχευση και προγραμματισμό υλικού FPGA. Η Verilog δημιουργήθηκε για να απλοποιήσει τη διαδικασία, καθιστώντας την HDL πιο ισχυρή και ευέλικτη. Η Verilog είναι αυτή τη στιγμή η πιο δημοφιλής. Δημιουργεί ένα επίπεδο αφαίρεσης για να κρύψει τις λεπτομέρειες της υλοποίησής της. Η Verilog έχει σύνταξη τύπου C, σε αντίθεση με το VHDL. [2][5]

Για να απλοποιηθεί ο σχεδιασμός πολύπλοκων συστημάτων στα FPGA, υπάρχουν βιβλιοθήκες με προκαθορισμένες πολύπλοκες συναρτήσεις και κυκλώματα που έχουν δοκιμαστεί και βελτιστοποιηθεί για να επιταχύνουν τη διαδικασία σχεδιασμού. Αυτά τα προκαθορισμένα κυκλώματα ονομάζονται συνήθως πυρήνες πνευματικής ιδιοκτησίας (IP) και είναι διαθέσιμα από προμηθευτές FPGA και τρίτους προμηθευτές IP. Σπάνια είναι δωρεάν και συνήθως κυκλοφορούν με ιδιόκτητες άδειες. Άλλα προκαθορισμένα κυκλώματα είναι διαθέσιμα από κοινότητες προγραμματιστών, όπως το OpenCores (συνήθως κυκλοφορούν με άδειες ελεύθερου και ανοιχτού κώδικα, όπως η άδεια GPL, BSD ή παρόμοια άδεια) και άλλες πηγές. Τέτοια σχέδια είναι γνωστά ως υλικό ανοιχτού κώδικα. [2][5]

Σε μια τυπική ροή σχεδίασης, ένας προγραμματιστής εφαρμογών FPGA θα προσομοιώσει τη σχεδίαση σε πολλαπλά στάδια σε όλη τη διαδικασία σχεδιασμού. Αρχικά, η περιγραφή RTL σε VHDL ή Verilog προσομοιώνεται, δημιουργώντας πάγκους δοκιμών για την προσομοίωση του συστήματος και την παρατήρηση των αποτελεσμάτων. Στη συνέχεια, αφού ο κινητήρας σύνθεσης έχει αντιστοιχίσει το σχέδιο σε μια netlist, η netlist μεταφράζεται σε μια περιγραφή σε επίπεδο πύλης, όπου επαναλαμβάνεται η προσομοίωση για να επιβεβαιωθεί ότι η σύνθεση έγινε χωρίς σφάλματα. Τέλος, η σχεδίαση παρουσιάζεται στο FPGA, στο οποίο σημείο μπορούν να προστεθούν καθυστερήσεις διάδοσης και η προσομοίωση εκτελείται ξανά, με αυτές τις τιμές να σχολιάζονται ξανά στη λίστα δικτύου. [2][5]

Πιο πρόσφατα, η OpenCL (*Open Computing Language*) χρησιμοποιείται από προγραμματιστές για να επωφεληθούν από την απόδοση και την αποδοτικότητα ισχύος που παρέχουν τα FPGA. Το OpenCL επιτρέπει στους προγραμματιστές να αναπτύσσουν κώδικα στη γλώσσα προγραμματισμού C και να στοχεύουν λειτουργίες FPGA ως πυρήνες OpenCL, χρησιμοποιώντας κατασκευές OpenCL. Για περισσότερες πληροφορίες, δείτε *σύνθεση υψηλού επιπέδου* και *C σε HDL*. [2][5]

Τα περισσότερα FPGA βασίζονται σε μια προσέγγιση που βασίζεται σε SRAM για να προγραμματιστούν. Αυτά τα FPGA μπορούν να προγραμματιστούν και να επαναπρογραμματιστούν εντός του συστήματος, αλλά απαιτούν εξωτερικές συσκευές εκκίνησης. Για παράδειγμα, η μνήμη flash ή οι συσκευές EEPROM μπορεί συχνά να φορτώνουν περιεχόμενο σε εσωτερική SRAM, η οποία ελέγχει τη δρομολόγηση και τη λογική. Η προσέγγιση SRAM βασίζεται στο CMOS.[2] [5]

2.2.3 Εφαρμογές των FPGA

Ένα FPGA μπορεί να χρησιμοποιηθεί για την επίλυση οποιουδήποτε προβλήματος που είναι υπολογίσιμο. Αυτό αποδεικνύεται ασήμαντα από το γεγονός ότι τα FPGA μπορούν να χρησιμοποιηθούν για την υλοποίηση ενός μαλακού μικροεπεξεργαστή, όπως ο Xilinx MicroBlaze ή ο Altera Nios II. Το πλεονέκτημά τους έγκειται στο ότι είναι σημαντικά ταχύτερα για ορισμένες εφαρμογές, λόγω της παράλληλης φύσης τους και της βέλτιστής τους όσον αφορά τον αριθμό των πυλών που χρησιμοποιούνται για ορισμένες διαδικασίες.[2][5]

Τα FPGA ξεκίνησαν αρχικά ως ανταγωνιστές των CPLD για την εφαρμογή λογικής κόλλας για πλακέτες τυπωμένων κυκλωμάτων. Καθώς το μέγεθος, οι δυνατότητες και η ταχύτητά τους αυξάνονταν, τα FPGA ανέλαβαν πρόσθετες λειτουργίες, σε σημείο που ορισμένα διατίθενται πλέον στην αγορά ως πλήρη συστήματα σε τσιπ (SoC). Ιδιαίτερα με την εισαγωγή αποκλειστικών πολλαπλασιαστών στις αρχιτεκτονικές FPGA στα τέλη της δεκαετίας του 1990, οι εφαρμογές που παραδοσιακά αποτελούσαν το μοναδικό εφεδρικό υλικό του ψηφιακού επεξεργαστή σήματος (DSP) άρχισαν να ενσωματώνουν FPGA. [2][5]

Η εξέλιξη των FPGA οδήγησε στην αύξηση της χρήσης αυτών των συσκευών, των οποίων η αρχιτεκτονική επιτρέπει την ανάπτυξη λύσεων υλικού βελτιστοποιημένων για πολύπλοκες εργασίες, όπως τμηματοποίηση εικόνας 3D MRI, 3D διακριτός μετασχηματισμός κυματιδίων, ανακατασκευή τομογραφικής εικόνας ή συστήματα PET/MRI. Οι λύσεις που αναπτύχθηκαν μπορούν να εκτελούν εντατικές εργασίες υπολογισμού με παράλληλη επεξεργασία, είναι δυναμικά επαναπρογραμματιζόμενες και έχουν χαμηλό κόστος, ενώ πληρούν τις σκληρές απαιτήσεις σε πραγματικό χρόνο που σχετίζονται με την ιατρική απεικόνιση. [2][5]

Μια άλλη τάση στη χρήση των FPGA είναι η επιτάχυνση υλικού, όπου μπορεί κανείς να χρησιμοποιήσει το FPGA για να επιταχύνει ορισμένα μέρη ενός αλγορίθμου και να μοιραστεί μέρος του υπολογισμού μεταξύ του FPGA και ενός γενικού επεξεργαστή. [2][5]

Παραδοσιακά, τα FPGA έχουν δεσμευτεί για συγκεκριμένες κάθετες εφαρμογές, όπου ο όγκος παραγωγής είναι μικρός. Για αυτές τις εφαρμογές χαμηλού όγκου, το ασφάλιστρο που πληρώνουν οι εταιρείες σε κόστος υλικού ανά μονάδα για ένα προγραμματιζόμενο τσιπ είναι πιο προσιτό από τους πόρους ανάπτυξης που δαπανώνται για τη δημιουργία ενός ASIC. Από το 2017, η νέα δυναμική κόστους και απόδοσης έχει διευρύνει το φάσμα των βιώσιμων εφαρμογών.[2][5]

2.2.4 Ασφάλεια

Τα FPGA έχουν τόσο πλεονεκτήματα όσο και μειονεκτήματα σε σύγκριση με τα ASIC ή τους ασφαλείς μικροεπεξεργαστές, όσον αφορά την ασφάλεια υλικού. Η ευελιξία των FPGA καθιστά τις κακόβουλες τροποποιήσεις κατά την κατασκευή μικρότερο κίνδυνο. Προηγουμένως, για πολλά FPGA, η ροή δυαδικών ψηφίων σχεδίασης ήταν εκτεθειμένη ενώ το FPGA τη φορτώνει από εξωτερική μνήμη (συνήθως σε κάθε ενεργοποίηση). Όλοι οι μεγάλοι προμηθευτές FPGA προσφέρουν πλέον ένα φάσμα λύσεων ασφαλείας στους σχεδιαστές, όπως κρυπτογράφηση και έλεγχος ταυτότητας bitstream. Για παράδειγμα, τα Altera και Xilinx προσφέρουν κρυπτογράφηση AES (έως 256-bit) για ροές bit που είναι αποθηκευμένα σε εξωτερική μνήμη flash. Οι φυσικές μη κλωνοποιήσιμες λειτουργίες (PUF – Physical Unclonable Function) είναι ολοκληρωμένα κυκλώματα που έχουν τις δικές τους μοναδικές υπογραφές, λόγω επεξεργασίας, και μπορούν επίσης να

χρησιμοποιηθούν για την ασφάλεια των FPGA ενώ καταλαμβάνουν πολύ λίγο χώρο υλικού. [2] [5]

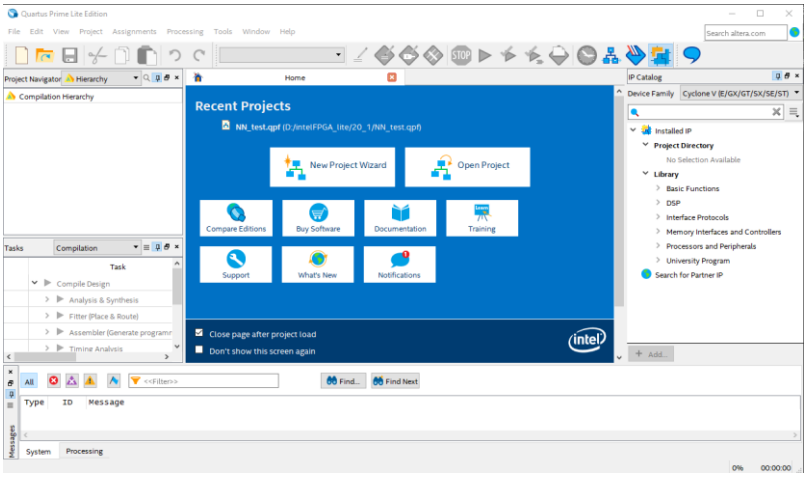
Τα FPGA που αποθηκεύουν τη διαμόρφωσή τους εσωτερικά σε μη πτητική μνήμη flash, δεν εκθέτουν τη ροή bit και δεν χρειάζονται κρυπτογράφηση. Επιπλέον, η μνήμη flash για έναν πίνακα αναζήτησης παρέχει προστασία από ανατροπή ενός μόνο συμβάντος για εφαρμογές χώρου. Οι πελάτες που επιθυμούν υψηλότερη εγγύηση αντοχής σε παραβίαση μπορούν να χρησιμοποιήσουν FPGA με δυνατότητα εγγραφής μίας φοράς, antifuse από προμηθευτές όπως η Microsemi. [2] [5]

2.3 Εργαλεία Υλοποίησης

Παρακάτω θα δώσουμε μια αναλυτική επεξήγηση για τα εργαλεία που χρησιμοποιήθηκαν, στην προκειμένη περίπτωση τα εργαλεία προσομοίωσης Quartus και Modelsim.

2.3.1 Quartus

Το Intel Quartus Prime είναι λογισμικό σχεδιασμού προγραμματιζόμενων λογικών συσκευών που παράγεται από την Intel. Πριν από την εξαγορά της Altera από την Intel, το εργαλείο ονομαζόταν Altera Quartus Prime, παλαιότερα Altera Quartus II. [2] [6]



Εικόνα 2.3.1.1 Quartus Lite Edition αρχική οθόνη

Το Intel Quartus Prime είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης για προγραμματιζόμενες λογικές συσκευές, όπως FPGA και CPLD (Complex Programmable Logic Device). Το λογισμικό αυτό επιτρέπει στους χρήστες να σχεδιάσουν, να συνθέσουν, να αναλύσουν και να προγραμματίσουν τα σχέδιά τους χρησιμοποιώντας γλώσσες περιγραφής υλικού όπως VHDL και Verilog. Ακολουθούν μερικές από τις κύριες δυνατότητες του Quartus Prime: [2] [6]

1. **Ανάλυση και Σύνθεση Σχεδίων HDL**: Το Quartus Prime προσφέρει ισχυρά εργαλεία για τη σύνταξη κώδικα σε HDL (Hardware Description Languages) και τη μετατροπή του σε λογικά κυκλώματα που μπορούν να προγραμματιστούν σε FPGA ή CPLD.
2. **Ανάλυση Χρονισμού**: Το εργαλείο παρέχει δυνατότητες για την ανάλυση του χρονισμού του σχεδίου, επιτρέποντας στους χρήστες να διασφαλίσουν ότι το σχέδιό τους πληροί τις απαιτήσεις χρονισμού για την σωστή λειτουργία.
3. **Διαγράμματα RTL**: Το Quartus Prime περιλαμβάνει εργαλεία για την εξέταση των διαγραμμάτων RTL (Register Transfer Level), που βοηθούν τους χρήστες να κατανοήσουν τη λειτουργία του σχεδίου τους σε επίπεδο καταχωρητών και συνδέσεων.
4. **Προσομοίωση**: Οι χρήστες μπορούν να προσομοιώσουν την αντίδραση του σχεδίου τους σε διάφορα ερεθίσματα, επιτρέποντας την επιβεβαίωση της σωστής λειτουργίας πριν από την φυσική υλοποίηση.
5. **Προγραμματισμός Συσκευών**: Το εργαλείο περιλαμβάνει δυνατότητες για τη διαμόρφωση και τον προγραμματισμό της τελικής συσκευής, επιτρέποντας την άμεση φόρτωση του σχεδίου στην προγραμματιζόμενη λογική συσκευή.
6. **Υλοποίηση VHDL και Verilog**: Το Quartus Prime υποστηρίζει πλήρως τις γλώσσες VHDL και Verilog, παρέχοντας εργαλεία για τη σύνταξη, επεξεργασία και επαλήθευση κώδικα σε αυτές τις γλώσσες.
7. **Οπτική Επεξεργασία Λογικών Κυκλωμάτων**: Η πλατφόρμα προσφέρει γραφικά εργαλεία για την οπτική δημιουργία και επεξεργασία λογικών κυκλωμάτων, διευκολύνοντας τους χρήστες που προτιμούν έναν πιο οπτικό τρόπο σχεδίασης.
8. **Διανυσματική Προσομοίωση Κυματομορφών**: Η δυνατότητα αυτή επιτρέπει την προσομοίωση της λειτουργίας του σχεδίου και την ανάλυση των κυματομορφών των σημάτων για την επιβεβαίωση της σωστής λειτουργίας. [2] [6]

Χαρακτηριστηκά του Quartus

Quartus Prime Pro Edition: [2] [6]

- Σχεδιασμένο για προηγμένες συσκευές όπως τα Intel Agilex και Stratix 10.
- Υποστηρίζει προηγμένες λειτουργίες όπως η μερική επαναδιαμόρφωση (Partial Reconfiguration), η γρήγορη ανασύνταξη (Rapid Recompile), και ο μπλοκικός σχεδιασμός (Block-Based Design).
- Περιλαμβάνει το Intel HLS Compiler, το DSP Builder, και το Nios II Embedded Design Suite.
- Προσφέρει εργαλεία ανάλυσης ισχύος και θερμότητας, καθώς και το Signal Tap Logic Analyzer για την παρακολούθηση και αποσφαλμάτωση σημάτων σε πραγματικό χρόνο.

Quartus Prime Standard Edition: [2] [6]

- Υποστηρίζει FPGA συσκευές όπως τα Intel Arria 10 και Cyclone 10 LP.
- Παρέχει ολοκληρωμένα εργαλεία σχεδίασης υψηλού επιπέδου, συμπεριλαμβανομένων των εργαλείων εισαγωγής σχεδίων βάσει C, συστημάτων/IP, και μοντέλων.
- Περιλαμβάνει εργαλεία προσομοίωσης και ανάλυσης χρονισμού για την επαλήθευση της απόδοσης των σχεδίων.

Quartus Prime Lite Edition: [2] [6]

- Μια δωρεάν έκδοση που προσφέρει βασικά εργαλεία σχεδίασης και προγραμματισμού για συσκευές χαμηλού κόστους και χαμηλής ισχύος όπως τα Intel Cyclone 10 LP και MAX series.
- Δεν υποστηρίζει προηγμένες λειτουργίες όπως η μερική επαναδιαμόρφωση και η γρήγορη ανασύνταξη.

Υποστηριζόμενες συσκευές [2] [6]

Κάθε έκδοση του Quartus Prime υποστηρίζει διαφορετικές συσκευές. Η Pro Edition υποστηρίζει συσκευές υψηλής απόδοσης όπως τα Intel Agilex και Stratix 10, ενώ η Standard Edition καλύπτει συσκευές όπως τα Intel Arria 10 και Cyclone 10 LP. Η Lite

Edition είναι ιδανική για συσκευές χαμηλού κόστους και χαμηλής ισχύος όπως τα Cyclone 10 LP και MAX series.

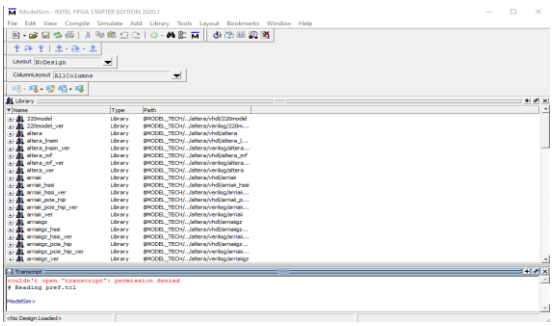
Βασικά εργαλεία και λειτουργίες [2] [6]

- **Platform Designer:** Αυτό το εργαλείο ενσωμάτωσης συστημάτων δημιουργεί αυτόματα λογική διασύνδεσης για τη σύνδεση IP λειτουργιών και υποσυστημάτων, εξοικονομώντας χρόνο και προσπάθεια κατά τη σχεδίαση FPGA.
- **Timing Analyzer:** Εργαλείο ανάλυσης χρονισμού που επικυρώνει την απόδοση του χρονισμού όλων των λογικών στοιχείων στο σχέδιο χρησιμοποιώντας μια βιομηχανικά τυποποιημένη μεθοδολογία περιορισμών, ανάλυσης και αναφοράς.
- **Signal Tap Logic Analyzer:** Εργαλείο ανάλυσης και αποσφαλμάτωσης σημάτων σε πραγματικό χρόνο χωρίς την ανάγκη εξωτερικών I/O pins ή εξοπλισμού εργαστηρίου.

Το Intel Quartus Prime είναι ένα ισχυρό εργαλείο που καλύπτει όλες τις ανάγκες σχεδίασης και ανάπτυξης για προγραμματιζόμενες λογικές συσκευές, παρέχοντας μια ολοκληρωμένη πλατφόρμα για τους μηχανικούς ηλεκτρονικών και τους σχεδιαστές ψηφιακών συστημάτων. [2] [6]

2.3.2 ModelSim και ModelSim Altera

Το **ModelSim** είναι ένα ισχυρό εργαλείο που παρέχει ένα ολοκληρωμένο περιβάλλον για την ανάπτυξη και προσομοίωση ψηφιακών κυκλωμάτων. Χρησιμοποιείται κυρίως για τον έλεγχο της λογικής των ψηφιακών σχεδίων πριν αυτά περάσουν στο στάδιο της υλοποίησης σε FPGA ή ASIC. [2] [7]



Εικόνα 2.3.2.1 Modelsim Atera κύρια οθόνη

Ας δούμε μερικές από τις κύριες δυνατότητες και χαρακτηριστικά του ModelSim [2] [7]:

1. <u>Προσομοίωση</u>:

- Το ModelSim υποστηρίζει προσομοίωση σε επίπεδο συμπεριφοράς, σε επίπεδο χρονισμού, και σε επίπεδο RTL (Register Transfer Level). Αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί τόσο για την πρώιμη ανάλυση της λογικής του σχεδίου όσο και για την προσομοίωση με χρονισμούς που αντανακλούν το πραγματικό υλικό.

- Επιτρέπει τη μικτή προσομοίωση, δηλαδή την ταυτόχρονη προσομοίωση μοντέλων που περιγράφονται τόσο σε VHDL όσο και σε Verilog, κάτι που είναι ιδιαίτερα χρήσιμο σε σχέδια που χρησιμοποιούν διαφορετικά HDL.

2. <u>Διασύνδεση με άλλα εργαλεία</u>:

- Το ModelSim μπορεί να ενσωματωθεί με άλλα εργαλεία EDA (Electronic Design Automation), όπως το *Synopsys* και το *Xilinx ISE/Vivado*, διευκολύνοντας την ολοκληρωμένη ροή εργασίας από το σχεδιασμό μέχρι την υλοποίηση.

- Υποστηρίζει επίσης εξωτερικά σενάρια και αυτοματισμούς μέσω TCL scripting, παρέχοντας ευελιξία στους χρήστες για την προσαρμογή της λειτουργίας του στις ανάγκες του έργου τους.

3. <u>Παρακολούθηση και Ανάλυση Κυματομορφών</u>:

- Το ModelSim διαθέτει ένα εργαλείο κυματομορφών που επιτρέπει την παρακολούθηση των σημάτων του κυκλώματος σε πραγματικό χρόνο κατά την προσομοίωση. Αυτό βοηθά στην ανάλυση της συμπεριφοράς του κυκλώματος και στην ανίχνευση προβλημάτων.

- Οι κυματομορφές μπορούν να αποθηκευτούν και να επαναχρησιμοποιηθούν για περαιτέρω ανάλυση ή τεκμηρίωση.

4. <u>Δημιουργία και Επαλήθευση Testbenches</u>:

- Οι χρήστες μπορούν να δημιουργούν πολύπλοκα testbenches για να αυτοματοποιήσουν την επαλήθευση του σχεδίου τους. Τα testbenches μπορούν να περιλαμβάνουν ερεθίσματα (stimuli) για τα σήματα εισόδου, καθώς και ελέγχους (assertions) για να διασφαλίσουν ότι το κύκλωμα λειτουργεί όπως αναμένεται.

5. <u>Δυνατότητες Εντοπισμού Σφαλμάτων</u>:

- Το ModelSim προσφέρει εργαλεία για την ανίχνευση και επιδιόρθωση σφαλμάτων (debugging). Αυτά περιλαμβάνουν τη δυνατότητα για παύση της προσομοίωσης, τη χρήση σημείων ελέγχου (breakpoints), και την επεξεργασία του κώδικα σε πραγματικό χρόνο.

- Υπάρχουν επίσης δυνατότητες παρακολούθησης σημάτων μέσω κυματομορφών και αλλαγών, επιτρέποντας την εύκολη ανίχνευση και ανάλυση των σφαλμάτων που εμφανίζονται κατά την προσομοίωση.

6. <u>Συμβατότητα και Επεκτασιμότητα</u>:

- Το ModelSim υποστηρίζει πολλές εκδόσεις των HDL, καθώς και διαφορετικές τυποποιήσεις και βιβλιοθήκες, καθιστώντας το συμβατό με ένα ευρύ φάσμα έργων και σχεδίων.

- Μπορεί να επεκταθεί με πρόσθετα (plugins) και εργαλεία τρίτων για επιπλέον λειτουργίες.

7. <u>Εκδόσεις του ModelSim</u>:

- Το ModelSim διατίθεται σε διάφορες εκδόσεις, όπως το ModelSim PE (Personal Edition) και το ModelSim SE (Standard Edition). Η PE είναι πιο προσιτή, ενώ η SE παρέχει πιο προηγμένες δυνατότητες και επεκτασιμότητα για επαγγελματική χρήση.

Το ModelSim είναι ουσιαστικά ένα εργαλείο για τους επαγγελματίες στον τομέα της σχεδίασης ψηφιακών κυκλωμάτων που απαιτεί προηγμένες γνώσεις στις HDL και στη διαδικασία της προσομοίωσης. Παρέχει όλα τα απαραίτητα εργαλεία για την επαλήθευση

και βελτιστοποίηση των σχεδίων πριν αυτά περάσουν στην παραγωγή, διασφαλίζοντας ότι τα κυκλώματα λειτουργούν σωστά και αποδοτικά. [2] [7]

Από την άλλη το Modelsim Altera είναι μια παραλλαγή του ModelSim, ειδικά βελτιστοποιημένη για χρήση με FPGA και CPLD της εταιρείας Altera (που πλέον ανήκει στην Intel). Είναι ουσιαστικά μια έκδοση του ModelSim προσαρμοσμένη για να υποστηρίζει άμεσα τις βιβλιοθήκες και τα εργαλεία της Altera. [2] [7]

Αυτή η έκδοση συνήθως προσφέρεται σε δύο μορφές [2] [7]:

1. **ModelSim-Altera Starter Edition**: Μια δωρεάν έκδοση που είναι περιορισμένη σε σχέση με την πλήρη έκδοση. Έχει περιορισμούς όσον αφορά το μέγεθος του σχεδίου που μπορεί να προσομοιωθεί και τη λειτουργικότητα.

2. **ModelSim-Altera Edition**: Μια πληρωμένη έκδοση που προσφέρει πλήρη υποστήριξη για όλα τα εργαλεία της Altera και επιτρέπει την προσομοίωση μεγαλύτερων και πιο περίπλοκων σχεδίων.

Αυτή η έκδοση του ModelSim είναι στενά ενσωματωμένη με το Quartus II (ή το νεότερο Quartus Prime), που το βασικό περιβάλλον ανάπτυξης της Altera/Intel, διευκολύνοντας την προσομοίωση και την επαλήθευση των σχεδίων απευθείας από το περιβάλλον ανάπτυξης. [2] [7]

Κύρια Χαρακτηριστικά του ModelSim Altera [2] [7]

- **Προσομοίωση FPGA**: Βελτιστοποιημένο για προσομοιώσεις FPGA, εξασφαλίζοντας μεγαλύτερη ταχύτητα και ακρίβεια στις προσομοιώσεις των σχεδίων.

- **Ενσωμάτωση με Quartus**: Ομαλή ενσωμάτωση με το Quartus II/Prime για την ανάπτυξη, σύνθεση και προσομοίωση ψηφιακών σχεδίων σε FPGA.

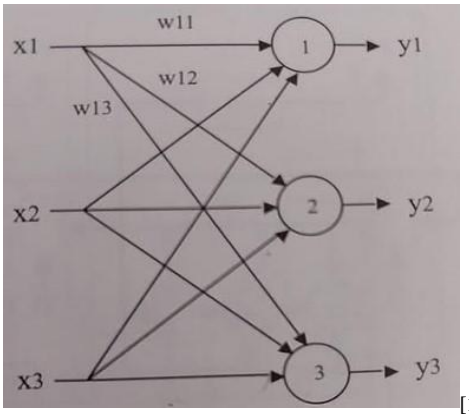
- **Υποστήριξη Ειδικών Βιβλιοθηκών**: Περιλαμβάνει τις βιβλιοθήκες της Altera/Intel για την προσομοίωση συγκεκριμένων στοιχείων FPGA.

Με λίγα λόγια, το ModelSim είναι ένα ισχυρό εργαλείο προσομοίωσης για ψηφιακά σχέδια, ενώ το ModelSim Altera αποτελεί μια εξειδικευμένη έκδοση του ModelSim, ιδανική για τους χρήστες FPGA και CPLD της Altera/Intel. [2] [7]

Στην επόμενη ενότητα θα μιλήσουμε για την μεθοδολογία, σχεδιασμό και την υλοποίηση του πρακτικού μέρους αυτής της πτυχιακής εργασίας, που είναι το χτίσιμο του νευρωνικού μας δικτύου.

3.Μεθοδολογία

Σε αυτό το κεφάλαιο θα μελετήσουμε διάφορες μεθόδους αρχιτεκτονικής και θα αναπτύξουμε παραδείγματα συμπεριφοράς νευρωνικών δικτύων, καθώς και θα εξηγήσουμε παρακάτω τις λειτουργίες των εργαλείων και του περιβάλλοντος μας.



[1]

Εικόνα 3.1 Απεικόνιση του νευρωνικού δικτύου της εργασίας

3.1 Εισαγωγή

Στα προηγούμενα κεφάλαια εξηγήσαμε τι είναι ένα νευρωνικό δίκτυο, τι λειτουργείες έχει, τι είδη υπάρχουν, τους διάφορους τρόπους που μπορούμε να εκπαιδεύσουμε τα νευρωνικά δίκτυα, την ιστορία τους, πως εμφανίστηκε η ιδέα για έναν τεχνητό εγκέφαλο. Σε αυτό το κεφάλαιο θα αναπτύξουμε, καθώς και θα εξηγήσουμε την λειτουργεία ενός δικού μας νευρωνικού δικτύου.

3.2 Ανάλυση

Το νευρωνικό δίκτυο που θα εξηγήσουμε παρακάτω είναι ένα απλό δίκτυο τριών εισόδων και εξόδων. Το Quartus Prime Lite Edition, αν και είναι το εργαλείο το οποίο θα παρουσιάσουμε τον κώδικα μας είναι πολύ περιορισμένο στο τι μπορεί να μας υπολογίσει

σαν αποτελέσματα και μπορεί να μας μπερδέψει αυτό το πράγμα, για ευνοϊκούς λόγους θα χρησιμοποιήσουμε και το Modelsim Altera για την αναπαράσταση των αποτελεσμάτων.

Όπως εξηγήσαμε στο προηγούμενο κεφάλαιο, για να δημιουργήσουμε ένα νευρωνικό δίκτυο χρειαζόμαστε εισόδους για την αρχική κατάσταση (τιμές π.χ. γράμματα, αριθμητικές πράξεις κλπ.) που θα δίνουμε, τα βάρη ή τα κρυφά επίπεδα στα οποία θα δίνουμε και σε αυτά μια αρχική κατάσταση για να γίνονται οι πράξεις μεταξύ των εισόδων και των βαρών, μια συνάρτηση ενεργοποίησης (δες κεφάλαιο 2 σελίδα 14) για τον σωστό υπολογισμό των νευρώνων (αντιστοιχία και σωστές πράξεις μεταξύ εισόδων με βάρη), καθώς και τις ανάλογες εξόδους μας για να μπορούμε να δούμε και να ελέγχουμε τα αποτελέσματα μας.

Στο παράρτημα 1 έχουμε την δομή του κύριου προγράμματος μας, παρατηρούμε στην αρχή τις βιβλιοθήκες μας και επειδή είμαστε στο περιβάλλον του Quartus οι βασικές βιβλιοθήκες θα είναι από την IEEE. Η πρώτη βιβλιοθήκη *IEEE.STD_LOGIC_1164.ALL* είναι η βασική μας βιβλιοθήκη για να μπορεί πρόγραμμα μας να καταλαβαίνει τύπους λογικής *STD_LOGIC* (π.χ. όλες τις εισόδους, εξόδους, καταστάσεις κλπ.)και *STD_LOGIC_VECTOR* (π.χ. όλους τους **τύπους** ανάγνωσης δεδομένων που θα έχουν μια αρχή και ένα τέλος). Η δεύτερη βιβλιοθήκη *IEEE.STD_LOGIIC_ARITH.ALL* ασχολείται καθαρά μόνο με τις αριθμητικές πράξεις του προγράμματος (για ευκολία θα ασχοληθούμε μόνο με αριθμούς). Η *IEEE.STD_LOGIC_SIGNED.ALL* ασχολείται με το εύρος (επειδή οι αριθμοί στο πρόγραμμα μεταφράζονται στο δυαδικό σύστημα, δηλαδή 0 και 1 μας δίνει τιμές από την μικρότερη αρνητική ως και την μεγαλύτερη θετική τιμή που του επιτρέπουμε συμπεριλαμβανομένου και το 0) των αριθμών για να μην έχουμε λογικά λάθη στις πράξεις μας (χωρίς αυτήν το πρόγραμμα διαβάζει μόνο θετικούς ακεραίους, μας επιτρέπει να κάνουμε και πράξεις και με αρνητικούς ακεραίους). Και η *IEEE.STD_LOGIC_NUMERIC.ALL* ασχολείται με τις αριθμητικές πράξεις τύπου vector (STD_LOGIC_VECTOR). Στην αρχικοποίηση των μεταβλητών έχουμε την ονομασία του προγράμματος ως ENTITY.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
--USE IEEE.STD_LOGI_NUMERIC.ALL;
```

Εικόνα 3.2.1 Βιβλιοθήκες κύριου προγράμματος

ΠΡΟΣΟΧΗ! ότι δώσουμε σαν ονομασία στο πρόγραμμα μας μετά το ENTITY πρέπει και στην αποθήκευση του να έχει αυτή την ονομασία, διαφορετικά δεν αναγνωρίζεται κατά την εκκίνηση του προγράμματος και μας δείχνει σφάλμα.

```
ENTITY nn_signed_wfsm IS
```

Εικόνα 3.2.2 Ονομασία προγράμματος

Παρακάτω δηλώνουμε τις μεταβλητές n (αριθμό νευρώνων), m (αριθμό βαρών για νευρώνα), b (πόσα bits θα έχουμε για είσοδο ή βαρών). Μετά παρατηρούμε x_1, x_2, x_3, καθώς και y_1, y_2, y_3, (x για είσοδο και y για έξοδο), άρα μπορούμε να πούμε πως το πρόγραμμα μας θα έχει τρεις εισόδους και αντίστοιχα θα παράξει για την κάθε είσοδο και από μια έξοδο ή αποτέλεσμα. Το clk ή αλλιώς ρολόι έχει ως σκοπό την ενημέρωση μιας τυχόν αλλαγής κατάστασης μέσα στο πρόγραμμα. Η μεταβλητή start έχει ως σκοπό την αρχικοποίηση των μεταβλητών του προγράμματος (δηλαδή δίνοντας μια πρώτη τιμή για να ξεκινήσει η λειτουργεία του προγράμματος). Η reset έχει ως σκοπό κατά την εκκίνηση του προγράμματος να μηδενίσει τις αρχικές τιμές που έχουν πάρει οι μεταβλητές από την εντολή start, με σκοπό, να βγάλουμε σωστά αποτελέσματα στις εξόδους του προγράμματος. Η valuesready είναι σημαία ολοκλήρωσης που υποδεικνύει ότι το κύκλωμα έχει τελειώσει με την επεξεργασία δεδομένων και τα αποτελέσματα είναι έτοιμα για ανάγνωση. Και τέλος έχουμε το s1, s2, s3 που αυτές θα είναι οι έξοδοι που παράγονται από την πράξη της συνάρτησης ενεργοποίησης με τις εξόδους y.

```
GENERIC (
   n: INTEGER := 3; -- # of neurons
   m: INTEGER := 3; -- # of inputs or weights per neuron
   b: INTEGER := 16 -- # of bits per input or weight
);

PORT (
   x1:    IN SIGNED(b-1 DOWNTO 0);
   x2:    IN SIGNED(b-1 DOWNTO 0);
   x3:    IN SIGNED(b-1 DOWNTO 0);

   w:     IN SIGNED(b-1 DOWNTO 0);

   clk, start, reset: IN STD_LOGIC;

   valuesready : OUT STD_LOGIC;
   test: OUT SIGNED(b-1 DOWNTO 0); -- register test output

   y1: OUT SIGNED(2*b-1 DOWNTO 0);
   y2: OUT SIGNED(2*b-1 DOWNTO 0);
   y3: OUT SIGNED(2*b-1 DOWNTO 0);

   s1: OUT SIGNED(2*b-1 DOWNTO 0);
   s2: OUT SIGNED(2*b-1 DOWNTO 0);
   s3: OUT SIGNED(2*b-1 DOWNTO 0)
);
```

Εικόνα 3.2.3 Δήλωση μεταβλητών προγράμματος

Στην συνέχεια θα εξηγήσουμε την αρχιτεκτονική του προγράμματος, δηλαδή το πως είναι δομημένο το δίκτυο (ή αλλιώς το τσιπάκη/πλακετούλα) σε ηλεκτρονική μορφή. Ξεκινώντας δηλώνουμε τους τύπους των δεδομένων που θα αναπαρίστανται οι τιμές από τις μεταβλητές του ENTITY.

```
ARCHITECTURE neural OF nn_signed_wfsm IS
    TYPE weights IS ARRAY (1 TO n*m) OF SIGNED(b-1 DOWNTO 0);
    TYPE inputs IS ARRAY (1 TO m) OF SIGNED(b-1 DOWNTO 0);
    TYPE outputs IS ARRAY (1 TO m) OF SIGNED(2*b-1 DOWNTO 0);
    ----------LUT Types--------------
    TYPE addresses IS ARRAY (1 TO m) OF STD_LOGIC_VECTOR(11 DOWNTO 0);
    TYPE LUT_outputs IS ARRAY (1 TO m) OF STD_LOGIC_VECTOR(2*b-1 DOWNTO 0);
    TYPE state IS (S00, S01, S10, S11);
```

Εικόνα 3.2.4 Δήλωση τύπων δεδομένων

Παρακάτω όπως φαίνεται στο παράρτημα 1 εφαρμόζουμε μεταβλητές για τον τρόπο διαβάσματος και εγγραφής των τιμών που θα δούμε στην έξοδο. Αυτά τα εφαρμόζουμε φτιάχνοντας μια απλή μνήμη (παράρτημα 3 LUTmemory.vhd) χρησιμοποιώντας την εντολή component που μας επιτρέπει να δανειστούμε αυτές τις μεταβλητές από αυτή την μνήμη και να τις εφαρμόσουμε μέσα στο πρωτεύων πρόγραμμα.

```
component LUTmemory is
   PORT(
         read_data            : OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
         address              : IN  STD_LOGIC_VECTOR( 11 DOWNTO 0 );
         write_data           : IN  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
         MemRead, Memwrite    : IN  STD_LOGIC;
         clock, reset         : IN  STD_LOGIC
   );
end component;
```

Εικόνα 3.2.5 Δέσμευση μεταβλητών παραρτήματος 3 LUTmemory.vhd

Οι παρακάτω δηλώσεις **SIGNAL** χρησιμοποιούνται για την αποθήκευση και τη διαχείριση των δεδομένων και σημάτων ελέγχου που απαιτούνται για την εκτέλεση λειτουργιών, όπως η εφαρμογή της σιγμοειδής συνάρτησης ενεργοποίησης και η μετάβαση των καταστάσεων της μηχανής καταστάσεων FSM.

```
--------LUT Table Signals------------
SIGNAL MemRead, Memwrite: std_logic;
SIGNAL read_data, write_data : STD_LOGIC_VECTOR( 31 DOWNTO 0 );
SIGNAL addr : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL sig_result: STD_LOGIC_VECTOR(2*b-1 DOWNTO 0);
SIGNAL LUT_output: LUT_outputs;
SIGNAL index: INTEGER RANGE 0 TO 15 := 0;
---FSM signals-------------------------
SIGNAL Moore_state: state;
SIGNAL sum, sigmoid, ready: std_logic;
```

Εικόνα 3.2.6 Σήματα/Λυχνίες λειτουργείας προγράμματος

Αφού έχουμε δηλώσει τους τύπους, τα σήματα και τον τρόπο εγγραφής και ανάγνωσης ξεκινάμε με την εντολή BEGIN και η πρώτη ενέργεια που πρέπει να κάνουμε είναι να δώσουμε αρχικές ή μηδενικές τιμές στις εισόδους, σήματα, καταστάσεις και δίνοντας τα κατάλληλα εύρη τιμών (λόγου της βιβλιοθήκης SIGNED) για να μην υπάρξουν σφάλματα.

```
BEGIN
   -------------LUT--------------
   MemRead <= '1';
   Memwrite <= '0';
   write_data <= x"00000000";
   LUT: LUTmemory port map(sig_result, addr, write_data, MemRead, Memwrite, clk, reset);
```

Εικόνα 3.2.7 Αρχικοποίηση τιμών

Παρακάτω αρχικοποιούμε την μηχανή καταστάσεων με ονομασία FSM:PROCESS με τους τύπους καταστάσεων/states S00, S01, S10 και S11. Η ροή των καταστάσεων στο πρόγραμμα έχει ως εξής:

1. S00: Αρχικοποίηση.

2. S01: Συσσώρευση.

3. S10: Εφαρμογή της σιγμοειδής συνάρτησης ενεργοποίησης.

4. S11: Ολοκλήρωση επεξεργασίας και προετοιμασία για επόμενη διαδικασία.

Η αλλαγή των καταστάσεων εξαρτάται αποκλειστικά μόνο από την ακμοπυροδότηση του clk'event (αν είναι 0 παραμένει στην ίδια κατάσταση και αν είναι 1 θα αλλάζει στην προκειμένη περίπτωση).

Το FSM:PROCESS συντονίζει τη ροή των λειτουργιών του κυκλώματος. Καθορίζει πότε εκτελούνται οι βασικές λειτουργίες, όπως η συσσώρευση και η εφαρμογή της σιγμοειδούς συνάρτησης ενεργοποίησης, και πότε τα αποτελέσματα είναι έτοιμα.

Η PROCESS διαχειρίζεται όλες τις βασικές αριθμητικές πράξεις (MAC και sigmoid), εισάγει και αποθηκεύει τα δεδομένα, και παράγει τα τελικά αποτελέσματα που είναι έτοιμα για χρήση από άλλα υποσυστήματα. Στη προκειμένη περίπτωση κάνει την αντιστοίχιση των μεταβλητών της αρχιτεκτονικής (ARCHTECTURE) με της οντότητας (ENTITY) και της μνήμης LUT (παράρτημα 3), την λειτουργεία ενός καταχωριτή ολίσθησης όπου όταν η μεταβλητή reset είναι 1 τότε οι έξοδοι του προγράμματος ανεξαρτήτως των τιμών των εισόδων θα είναι μηδενικές.

Και τέλος μέσα στην PROCESS έχουμε τον υπολογισμό της multiply-accumulate (MAC) και σιγμοειδούς συνάρτησης ενεργοποίησης μέσω της LUT, όπου η βασική της λειτουργεία είναι:

1. **Multiply-Accumulate (MAC):**

    - Για κάθε νευρώνα, υπολογίζονται τα αθροίσματα των προϊόντων εισόδων και βαρών (MAC).

    - Εφαρμόζεται έλεγχος υπερχείλισης για ακρίβεια.

2. **Sigmoid μέσω LUT:**

    - Η MAC έξοδος περνάει από έναν πίνακα αναζήτησης (LUT) για να υπολογιστεί η τιμή της συνάρτησης sigmoid, χρησιμοποιώντας προϋπολογισμένα αποτελέσματα για ταχύτητα.

Αυτό το τμήμα του κώδικα είναι η βασική υπολογιστική μονάδα ενός τεχνητού νευρωνικού δικτύου, υπολογίζοντας τις γραμμικές συναρτήσεις (MAC) και μη γραμμικές συναρτήσεις (sigmoid) που απαιτούνται για τη λειτουργία ενός νευρώνα.

Το παράρτημα 2 είναι ένα πρόγραμμα δοκιμής, το οποίο είναι απαραίτητο στην εκτέλεση του προγράμματος για το παράρτημα 1 στο περιβάλλον του Modelsim (δες σελίδα 42), για τον λόγο ότι το περιβάλλον του Quartus μας περιορίζει αρκετά σε θέματα χρονισμού με αποτέλεσμα να παίρνουμε λανθασμένα αποτελέσματα.

Η δουλειά του προγράμματος δοκιμής είναι να δίνει τις πρώτες τιμές για επεξεργασία στο κύριο πρόγραμμα (ανάθεση και αντιστοίχιση όλων των εισόδων, εξόδων, βαρυτήτων και χρόνων με τις μεταβλητές του προγράμματος δοκιμής), κάτω από μια ονομασία που θέτουμε στις ρυθμίσεις του Quartus (στην προκειμένη περίπτωση η ονομασία είναι uut), αφού του αρχικοποιήσουμε μέσα σε αυτό χρονισμούς (αρχικές και τελικές τιμές χρόνου, κάθε πότε θα γίνεται αλλαγή κατάστασης κ.α.), μεγέθη (π.χ. πόσα bits θα διαβάζει και θα εγγράφει).

Η δημιουργία του προγράμματος δοκιμής (testbench program) είναι παρόμοια με του κυρίου προγράμματος, με την διαφορά ότι στο τμήμα της οντότητας (ENTITY) δεν δηλώνουμε τίποτα και τα δηλώνουμε στην αρχιτεκτονική του ως components δανειζόμενα από το κύριο πρόγραμμα.

Το παράρτημα 3 είναι ένα πρόγραμμα μνήμης που η βασική του λειτουργεία είναι η εγγραφή και ο πεπερασμός του εύρους τιμών που έχουμε αναθέσει στο κύριο πρόγραμμα, στην προκειμένη περίπτωση το εύρος είναι από -2,48 ως και 2,47 αλλά επειδή έχουμε δώσει ως παράμετρο (στις βιβλιοθήκες) να διαχειριζόμαστε μόνο ακέραιες τιμές δεν θα πάρουμε τα επιθυμητά αποτελέσματα και στην έξοδο θα βγάλει σφάλμα, οπότε μια τελευταία λειτουργεία που αναθέτουμε στο πρόγραμμα αυτό είναι να πολλαπλασιάσουμε το εύρος των τιμών με x1000, έτσι ώστε να έχουμε μόνο ακέραιες τιμές και βάζουμε ως

υποσημείωση για τον απλό χρήστη, ότι για να βρει τις πραγματικές τιμές πρέπει να διαιρέσει τα αποτελέσματα με /1000.

Τέλος το πρόγραμμα δοκιμής της μνήμης στο παράρτημα 4 κάνει την αρχικοποίηση των τιμών και εγγραφή σε πίνακα για χρήση, για το κύριο πρόγραμμα.

Στην επόμενη ενότητα θα γίνει επεξήγηση των αναφερόμενων αποτελεσμάτων σε μια ποιο μεγάλη ανάλυση.

4.Αποτελέσματα

Στο προηγούμενο κεφάλαιο εξηγήσαμε την λειτουργεία του κάθε αναφερομένου προγράμματος σε βάθος. Σε αυτό το κεφάλαιο θα εξηγήσουμε τα αποτελέσματα των προγραμμάτων και γιατί χρειαζόμαστε και προγράμματα δοκιμής για την ανάλυση που θέλουμε να κάνουμε.

4.1 Επεξήγηση προγραμμάτων δοκιμής (Testbench Programs)

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο χρησιμοποιούμαι δύο προγράμματα για εγγραφή κώδικα το Quartus Prime Lite Edition και το Modelsim. Για ευκολία κατανόησης κώδικα χρησιμοποιούμαι το Quartus και για την εκτέλεσή του χρησιμοποιούμαι το Modelsim, ο λόγος είναι ότι το Quartus δεν μας δίνει στην ελεύθερη έκδοση του, τους χρόνους που χρειαζόμαστε για να παρακολουθήσουμε την σωστή εκτέλεση των προγραμμάτων (οι χρόνοι κυμαίνονται από 0 ως και 900 ms για τις τελικές κυματομορφές), ενώ το Modelsim δεν έχει αυτό το πρόβλημα και για αυτό χρησιμοποιούμαι προγράμματα δοκιμής (Testbench Programs), για να μπορούμε να εντάξουμε τα προγράμματα του Quartus στο Modelsim χωρίς την εγγραφή ποιο περίπλοκου κώδικα.

4.2 Αποτελέσματα προγραμμάτων

Ξεκινώντας με το παράρτημα 3 και 4 LUTmemory.vhd και LUTmemory_tb.vhd (σελίδες 72 και 74), όπως αναφέρεται στο προηγούμενο κεφάλαιο, η κύρια του λειτουργεία είναι η εγγραφή και αποθήκευση τιμών για χρήση από το κύριο πρόγραμμα (σε αυτή τη περίπτωση 4096 τιμές από -2048 ως 2047 συμπεριλαμβανομένου και το 0) και για λόγους ευκολίας τα νούμερα που θα φαίνονται στα αποτελέσματα θα είναι μόνο θετικοί αριθμοί.

Στον παρακάτω πίνακα παρατηρούμαι κάποιες από τις αναφερόμενες τιμές του LUTmemory.vhd:

WIDTH=32; DEPTH=4096; ADDRESS_RADIX=HEX; DATA_RADIX=BIN; CONTENT BEGIN	
000	00000000000000000000000111110100;
001	00000000000000000000000111110100;
002	00000000000000000000000111110101;
003	00000000000000000000000111110101;
. . . .	
FFC	00000000000000000000000111110011;
FFD	00000000000000000000000111110011;
FFE	00000000000000000000000111110100;
FFF	00000000000000000000000111110100;
END;	

Πίνακας 4.2.1 Αποτελέσματα εγγραφής μνήμης

Στην παρακάτω εικόνα (εικόνα 4.2.1) απεικονίζεται η λειτουργεία του προγράμματος που θέλουμε να δείξουμε στην εργασία αυτή. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο πολλαπλασιάσαμε τα νούμερα με x1000 για την διαχείριση ακεραίων αριθμών στο πρόγραμμα μας.

Παρατηρούμε στην εικόνα ότι κάθε φορά που βγαίνει το τελευταίο αποτέλεσμα κατάστασης (το S3), το σήμα READY αλλάζει και γίνεται 1 και όταν επανέρχεται στην αρχική του κατάσταση ενεργοποιείται το σήμα START και οι τιμές των εισόδων ολισθαίνουν κατά μια θέση και ξεκινάει η διαδικασία ξανά, εφαρμόζοντας τα βάρη και την συνάρτηση ενεργοποίησης και τέλος το tb/I (ή αλλιώς testbench/i) είναι πόσες επαναλήψεις επιθυμούμε εμείς να κάνει το πρόγραμμα.

Παρατηρούμε στην εικόνα 4.2.1 το *tb/w* (ή αλλιώς testbench/weight) στην αρχικοποίηση των μεταβλητών βλέπουμε ότι πριν πάρει τιμές για να ξεκινήσει τις πράξεις η αρχική του τιμή είναι αδιάφορη και γι’ αυτό είναι με κόκκινο χρώμα και αναπαρίσταται και με το γράμμα x.

Στη συνέχεια μετά την αρχικοποίηση των βαρών εκτελούνται κανονικά οι πράξεις για εννέα κύκλους ρολογιού (αυτό είναι καθαρά μόνο για την εργασία μπορεί να είναι παραπάνω ή λιγότεροι κύκλοι ρολογιών).

ΠΡΟΣΟΧΗ! Η σειρά με την οποία τα βάρη εισήλθαν είναι: w_1=9, w_2=8, w_3=7, w_4=6, w_5=5, w_6=4, w_7=3, w_8=2, w_9=1 και το w=0 επειδή δεν το χρησιμοποιούμε μέσα στις πράξεις μας το αγνοούμε.

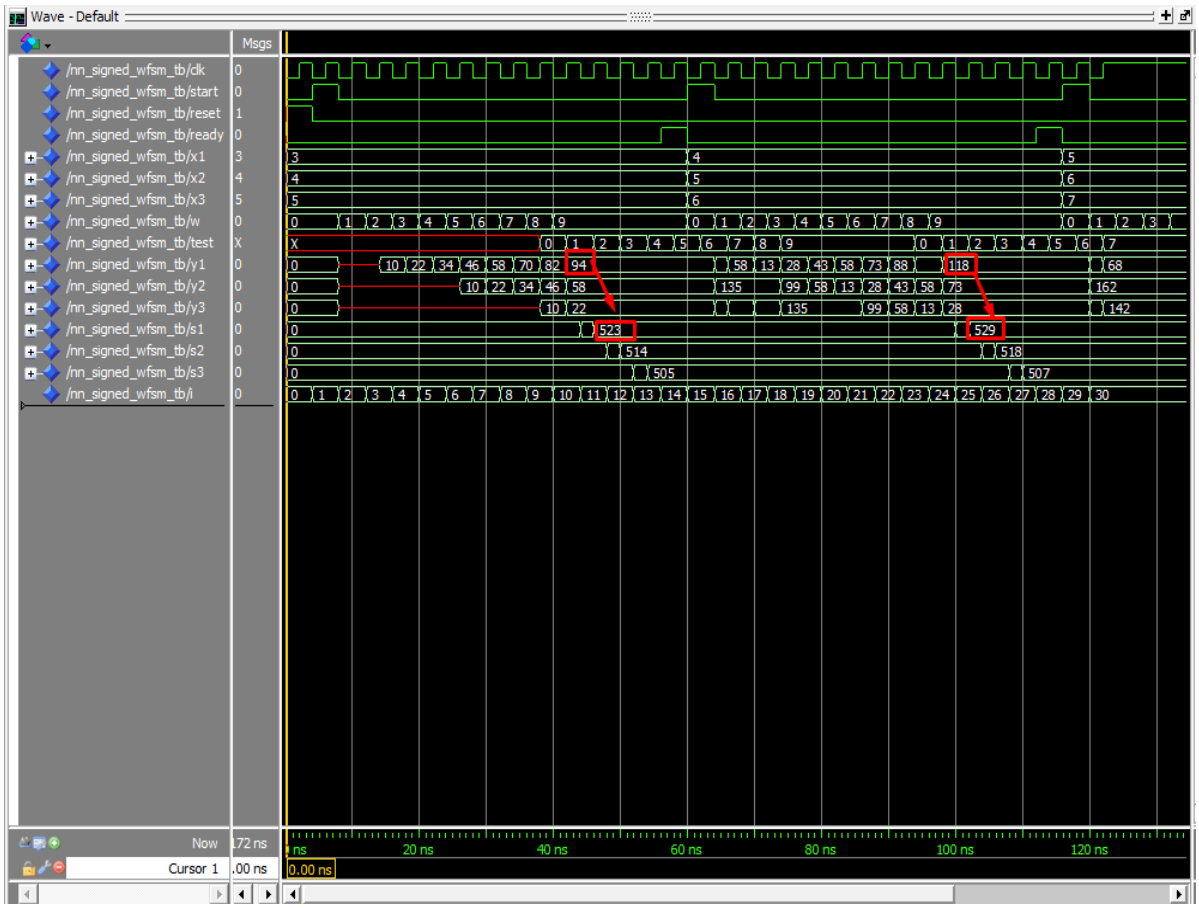
Για κάθε έξοδο y, οι μαθηματικές πράξεις για τον πρώτο κόμβο έχουν ως εξής:

- $y_1= (x_1*w_1)+ (x_2*w_2)+ (x_3*w_3)= (3*9)+ (4*8)+ (5*7)=(27)+(32)+(35)=94$
- $y_1= (x_1*w_4)+ (x_2*w_5)+ (x_3*w_6)= (3*6)+ (4*5)+ (5*4)=(18)+(20)+(20)=58$
- $y_1= (x_1*w_7)+ (x_2*w_8)+ (x_3*w_9)= (3*3)+ (4*2)+ (5*1)=(9)+(8)+(5)=22$

Και αντίστοιχα για κάθε έξοδο s, η κάθε έξοδος y πολλαπλασιάζεται με την συνάρτηση ενεργοποίησης.

Αυτές οι πράξεις θα επαναλαμβάνονται για κάθε εννέα κύκλους ρολογιού με την μόνη διαφορά ότι οι είσοδοι θα ολισθαίνονται κατά μια θέση προς τα πάνω.

Και τέλος στην εικόνα απεικονίζονται τα αποτελέσματα των τιμών εξόδων y να μετατρέπονται στις εξόδους καταστάσεων s, με την βοήθεια της σιγμοειδούς συνάρτησης ενεργοποίησης μέσω του LUTmemory.vhd (Βιβλιογραφία 6 σελίδα 77).



Εικόνα 4.2.1 Λειτουργεία προγραμμάτων και αντιστοίχιση τιμών εξόδων με τις καταστάσεις σε δεκαδική μορφή

Στην επόμενη ενότητα θα μιλήσουμε για κάποια συμπεράσματα και προσωπικές γνώμες για την εμπειρία αυτής της πτυχιακής εργασίας.

5.Συμπεράσματα

Η εργασία είχε ως σκοπό την επεξήγηση της έννοιας του νευρωνικού δικτύου (Neural Network), την ανάλυση, εφαρμογές στην καθημερινότητά μας και την ανάπτυξη παραδειγμάτων για την κατανόηση του πώς φτιάχνουμε ένα νευρωνικό δίκτυο. Η εργασία και τα παραδείγματα που αναπτύχθηκαν είχαν σκοπό να δείξουν μια απλή εκδοχή από τις πολλές αναπαραστάσεις των νευρωνικών δικτύων.

Τα νευρωνικά δίκτυα μπορούν πολύ απλά να αλλάξουν και να ξαναλλάξουν τα αποτελέσματα για μια άλλη διεργασία που το απαιτεί. Αυτός είναι και ο σκοπός τους: να προσομοιώσουν τον ανθρώπινο εγκέφαλο και να εξελίσσονται και να εκπαιδεύονται αναλόγως. Μια αλλαγή, για παραδείγματος χάρη, για αυτή την εργασία θα μπορούσε, αντί για αποτελέσματα να έχει μόνο ακέραιους αριθμούς, να έχει αριθμούς κινητής υποδιαστολής, χαρακτήρες, οι είσοδοι να πολλαπλασιάζονται χιαστί με τα βάρη κ.λπ., και βεβαίως θα πρέπει να γίνουν οι κατάλληλες αλλαγές για να μπορέσουμε να βγάλουμε τα επιθυμητά αποτελέσματα στην οθόνη μας. Τα νευρωνικά δίκτυα είναι μία καινοτόμος τεχνολογία που μπορεί να βοηθήσει την ανθρωπότητα στις καθημερινές της δυσκολίες ή και για τις απορίες της που μπορεί να υπάρχουν, να προσομοιώσουν μια κατάσταση κ.λπ., και είναι ακόμα σε συνεχή εξέλιξη και ανάπτυξη με τις συνεχείς καινούργιες πληροφορίες που εμφανίζονται.

Επιπλέον, τα νευρωνικά δίκτυα χρησιμοποιούνται πλέον ευρέως σε τομείς όπως η ιατρική διάγνωση, η ανάλυση εικόνας και φωνής, η αυτόνομη οδήγηση, αλλά και στην εξυπηρέτηση πελατών μέσω έξυπνων βοηθών. Η ευελιξία τους τα καθιστά ιδανικά εργαλεία για την αντιμετώπιση προβλημάτων που παλαιότερα απαιτούσαν ανθρώπινη σκέψη και παρέμβαση. Είναι σημαντικό να κατανοήσουμε ότι όσο τα δεδομένα αυξάνονται και οι ανάγκες για αυτοματοποίηση διευρύνονται, τόσο περισσότερο η σημασία των νευρωνικών δικτύων θα ενισχύεται. Με τη σωστή εκπαίδευση και παραμετροποίηση, μπορούν να επιτυγχάνουν εξαιρετικά επίπεδα ακρίβειας και απόδοσης, προσφέροντας λύσεις σε προκλήσεις του σύγχρονου κόσμου.

ΠΑΡΑΡΤΗΜΑΤΑ

1. nn_signed_wfsm.vhd[1]

```
--CONSIDER FIXED POINT ARITHMETIC WITH SCALING 1/100

--Remember to divide simulator result with 1000 since we use fixed point arithmetic

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.STD_LOGIC_ARITH.ALL;

USE IEEE.STD_LOGIC_SIGNED.ALL;

--USE IEEE.STD_LOGIC_NUMERIC.ALL;

ENTITY nn_signed_wfsm IS

	GENERIC (

		n: INTEGER := 3; -- # of neurons

		m: INTEGER := 3; -- # of inputs or weights per neuron

		b: INTEGER := 16 -- # of bits per input or weight);

	PORT (

		x1:	IN SIGNED(b-1 DOWNTO 0);

		x2:	IN SIGNED(b-1 DOWNTO 0);

		x3:	IN SIGNED(b-1 DOWNTO 0);

		w:	IN SIGNED(b-1 DOWNTO 0);

		clk, start, reset: IN STD_LOGIC;

		Valuesready : OUT STD_LOGIC;

		test: OUT SIGNED(b-1 DOWNTO 0); -- register test output
```

```
        y1: OUT SIGNED(2*b-1 DOWNTO 0);
        y2: OUT SIGNED(2*b-1 DOWNTO 0);
        y3: OUT SIGNED(2*b-1 DOWNTO 0);

        s1: OUT SIGNED(2*b-1 DOWNTO 0);
        s2: OUT SIGNED(2*b-1 DOWNTO 0);
        s3: OUT SIGNED(2*b-1 DOWNTO 0));
END nn_signed_wfsm;
ARCHITECTURE neural OF nn_signed_wfsm IS
    TYPE weights IS ARRAY (1 TO n*m) OF SIGNED(b-1 DOWNTO 0);
    TYPE inputs IS ARRAY (1 TO m) OF SIGNED(b-1 DOWNTO 0);
    TYPE outputs IS ARRAY (1 TO m) OF SIGNED(2*b-1 DOWNTO 0);
    ----------LUT Types----------
    TYPE addresses IS ARRAY (1 TO m) OF STD_LOGIC_VECTOR(11 DOWNTO 0);
    TYPE LUT_outputs IS ARRAY (1 TO m) OF STD_LOGIC_VECTOR(2*b-1 DOWNTO
0);
    TYPE state IS (S00, S01, S10, S11);
    component LUTmemory is
        PORT(
        read_data: OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
        address : IN STD_LOGIC_VECTOR( 11 DOWNTO 0 );
        write_data : IN  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
```

```
        MemRead, Memwrite : IN STD_LOGIC;
        clock, reset : IN STD_LOGIC );
    end component;
    --------LUT Table Signals------------
    SIGNAL MemRead, Memwrite: std_logic;
    SIGNAL read_data, write_data : STD_LOGIC_VECTOR( 31 DOWNTO 0 );
    SIGNAL addr : STD_LOGIC_VECTOR(11 DOWNTO 0);
    SIGNAL sig_result: STD_LOGIC_VECTOR(2*b-1 DOWNTO 0);
    SIGNAL LUT_output: LUT_outputs;
    SIGNAL index: INTEGER RANGE 0 TO 15 := 0;
    ---FSM signals---------------------------
    SIGNAL Moore_state: state;
    SIGNAL sum, sigmoid, ready: std_logic;
    BEGIN
        -------------LUT--------------
        MemRead <= '1';
        Memwrite <= '0';
        write_data <= x"00000000";
        LUT: LUTmemory port map(sig_result, addr, write_data, MemRead, Memwrite,
clk, reset);
        STEPS: PROCESS(clk)
            VARIABLE temp: INTEGER RANGE 0 TO 15 := 0;
            BEGIN
```

```
        IF (clk'EVENT AND clk='1') THEN
                temp := temp + 1;
                IF (ready = '1' OR reset = '1') THEN
                        temp := 0;
                END IF;
        END IF;
        index <= temp;
END PROCESS;
FSM: PROCESS (clk, reset)
    BEGIN
        IF(reset = '1') THEN
            Moore_state <= S00;
            sum <= '0';
            sigmoid <= '0';
        ELSIF (clk = '1' AND clk'event) THEN
            CASE Moore_state IS
                WHEN S00 =>
                    IF index = 0 THEN
                        Moore_state <= S01;
                        sum <= '1';
                        sigmoid <= '0';
                    ELSE
                        Moore_state <= S00;
```

```
                sum <= '0';
                sigmoid <= '0';
            END IF;
        WHEN S01 =>
            IF index <= 8 THEN
                Moore_state <= S01;
                sum <= '1';
                sigmoid <= '0';
            ELSE
                Moore_state <= S10;
                sum <= '0';
                sigmoid <= '1';
            END IF;
        WHEN S10 =>
            IF index <= 11 THEN
                Moore_state <= S10;
                sum <= '0';
                sigmoid <= '1';
            ELSE
                Moore_state <= S11;
                sum <= '0';
                sigmoid <= '0';
            END IF;
```

```
                    WHEN S11 =>
                        IF index > 12 THEN
                            Moore_state <= S00;
                        END IF;
                END CASE;
        END IF;
END PROCESS;
ready <= '1' WHEN Moore_state = S11 ELSE '0';
PROCESS (clk, w, x1, x2, x3)
        VARIABLE weight: weights;
        VARIABLE input: inputs;
        VARIABLE output: outputs;
        VARIABLE prod, acc: SIGNED(2*b-1 DOWNTO 0);
        VARIABLE sign: STD_LOGIC;
        ----------LUT Variables----------
        VARIABLE LUT_output: LUT_outputs;
        BEGIN
                ----- shift register inference: ----------
                IF (reset = '1') THEN
                        LUT_output(1) := X"00000000";
                        LUT_output(2) := X"00000000";
                        LUT_output(3) := X"00000000";
                        output(1) := X"00000000";
```

```
                output(2) := X"00000000";
                output(3) := X"00000000";
                addr <= "000000000000";
            ELSIF (clk'EVENT AND clk='1') THEN
                weight := w & weight(1 TO n*m-1);
            END IF;
            -------- initialization: ----------
            input(1) := x1;
            input(2) := x2;
            input(3) := x3;
            ------ multiply-accumulate: ----------
            IF sum = '1' THEN
                Ll: FOR i IN 1 TO n LOOP
                    acc := (OTHERS => '0');
                    L2: FOR j IN 1 TO m LOOP
                        prod := input(j)*weight(m*(i-1)+j);
                        sign := acc(acc'LEFT);
                        acc := acc + prod;
                        ---- overflow check: ----------
                        IF (sign=prod(prod'left)) AND
(acc(acc'left) /= sign) THEN
                            acc := (acc'LEFT => sign,
OTHERS => NOT sign);
```

```
                    END IF;

                END LOOP L2;

                output(i) := acc;

            END LOOP Ll;

        ELSIF sigmoid = '1' THEN

            addr <= std_logic_vector(output(index - 9)(11 downto 0));

            LUT_output(index - 9) := sig_result;

        END IF;

    ------ outputs: -------------------------

        Valuesready <= ready;

        test <= weight(n*m);

        y1 <= output(1);

        y2 <= output(2);

        y3 <= output(3);



        s1 <= signed(LUT_output(1));

        s2 <= signed(LUT_output(2));

        s3 <= signed(LUT_output(3));

    END PROCESS;

END neural;
```

2. nn_SIGNED_wfsm_tb.vhd[1]

```
library ieee;
use ieee.std_logic_1164.all;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
--use ieee.numeric_std.all;
entity nn_SIGNED_wfsm_tb is
end nn_SIGNED_wfsm_tb;
architecture arch of nn_SIGNED_wfsm_tb is
	component nn_signed_wfsm is
		GENERIC (
			n: INTEGER := 3; -- # of neurons
			m: INTEGER := 3; -- # of inputs or weights per neuron
			b: INTEGER := 16 -- # of bits per input or weight
		);
		PORT (
			x1:	IN SIGNED(b-1 DOWNTO 0);
			x2:	IN SIGNED(b-1 DOWNTO 0);
			x3:	IN SIGNED(b-1 DOWNTO 0);
			w:	IN SIGNED(b-1 DOWNTO 0);

			clk, start, reset : IN STD_LOGIC;
```

```
                Valuesready : OUT STD_LOGIC;
                test: OUT SIGNED(b-1 DOWNTO 0); -- register test output

                y1: OUT SIGNED(2*b-1 DOWNTO 0);
                y2: OUT SIGNED(2*b-1 DOWNTO 0);
                y3: OUT SIGNED(2*b-1 DOWNTO 0);

                s1: OUT SIGNED(2*b-1 DOWNTO 0);
                s2: OUT SIGNED(2*b-1 DOWNTO 0);
                s3: OUT SIGNED(2*b-1 DOWNTO 0)
        );
end component;
constant T : time := 500 ns;
constant T : time := 4 ns;
signal clk, start, reset, ready : std_logic :='0';
signal x1, x2, x3, w, test : SIGNED( 15 DOWNTO 0 ) := x"0000";
signal y1, y2, y3, s1, s2, s3 : SIGNED( 31 DOWNTO 0 ) := x"00000000";

constant num_of_clocks : integer := 30;
signal i : integer := 0;
begin
```

```
uut: nn_signed_wfsm port map(x1 => x1, x2 => x2, x3 => x3, w => w, clk => clk, start => start, reset => reset, Valuesready => ready, test => test, y1 => y1, y2 => y2, y3 => y3, s1 => s1, s2 => s2, s3 => s3);

-- continuous clock

process

    begin

        clk <= '0';

        wait for T/2;

        clk <= '1';

        wait for T/2;

        if (i = num_of_clocks) then

            wait;

        else

            i <= i + 1;

        end if;

end process;

Force: process

begin

    IF i = 0 THEN

        x1 <= x"0003";

        x2 <= x"0004";

        x3 <= x"0005";

        start <= '0';
```

```
            reset <= '1';
            wait for T;
END IF;
start <= '1';
reset <= '0';
wait for T;
start <= '0';
w  <= x"0001";
wait for T;
w  <= x"0002";
wait for T;
w  <= x"0003";
wait for T;
w  <= x"0004";
wait for T;
w  <= x"0005";
wait for T;
w  <= x"0006";
wait for T;
w  <= x"0007";
wait for T;
w  <= x"0008";
wait for T;
```

```
                w  <= x"0009";
                wait for T;
                wait for 4*T;
                if (ready = '1') then
                        w <= x"0000";
                        x1 <= x1 + 1;
                        x2 <= x2 + 1;
                        x3 <= x3 + 1;
                end if;
                if (i = num_of_clocks) then
                        wait;
                end if;
        end process;
end arch;
```

3. LUTmemory.vhd[1]

```
-- Dmemory module (implements the data memory for MIPS)
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;
ENTITY LUTmemory IS
		PORT(
			read_data: OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
			address : IN STD_LOGIC_VECTOR( 11 DOWNTO 0 );
			write_data : IN  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
			MemRead, Memwrite : IN STD_LOGIC;
			clock, reset: IN  STD_LOGIC );
END LUTmemory;
ARCHITECTURE behavior OF LUTmemory IS
		SIGNAL write_clock : STD_LOGIC;
BEGIN
	LUT_memory: altsyncram
	GENERIC MAP (
```

```
                operation_mode => "SINGLE_PORT",
                width_a => 32,
                widthad_a => 12,
                lpm_type => "altsyncram",
                outdata_reg_a => "UNREGISTERED",
                        -- Reads in mif file for initial data memory values
                init_file => "sigmoidLUT.mif",
                intended_device_family => "Cyclone")
        PORT MAP (
                wren_a => memwrite,
                clock0 => write_clock,
                address_a => address,
                data_a => write_data,
                q_a => read_data);
                        -- Load memory address & data register with write clock
                write_clock <= NOT clock;
END behavior;
```

4. LUTmemory_tb.vhd[1]

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity LUTmemory_tb is
end LUTmemory_tb;
architecture arch of LUTmemory_tb is
	component LUTmemory is
			PORT(
				read_data: OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
				address : IN STD_LOGIC_VECTOR( 11 DOWNTO 0 );
				write_data : IN  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
				MemRead, Memwrite : IN STD_LOGIC;
				clock, reset: IN  STD_LOGIC );
	end component;
	constant T : time := 4 ns;
	signal MemRead, Memwrite, clock, reset : std_logic :='0';
	signal read_data, write_data : STD_LOGIC_VECTOR( 31 DOWNTO 0 ) := x"00000000";
	signal address : STD_LOGIC_VECTOR( 11 DOWNTO 0 ) := x"000";
	constant num_of_clocks : integer := 5;
	signal i : integer := 0;
	begin
```

```
uut: LUTmemory port map(read_data => read_data, address => address, write_data => write_data, MemRead => MemRead, Memwrite => Memwrite, clock => clock, reset => reset);

-- continuous clock

process

    begin

        clock <= '0';

        wait for T/2;

        clock <= '1';

        wait for T/2;

        if (i = num_of_clocks) then

            --file_close(output_buf);

            wait;

        else

            i <= i + 1;

        end if;

end process;

Force: process

begin

    reset <= '1';

    wait for T;

    reset <= '0';

    wait for T;
```

```
                address <= x"001";
                wait for T;
                address <= x"005";
                wait for T;
                address <= x"009";
                wait for T;
                address <= x"00D";
                wait for T;
                if (i = num_of_clocks) then
                        wait;
                end if;
        end process;
end arch;
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Βιβλίο: Εκδόσεις Κλειδάριθμος Σχεδιασμός κυκλωμάτων με τη VHDL Volnei A. Pedroni
2. ChatGPT

https://chatgpt.com

3. Είδη νευρωνικών δικτύων

https://www.mygreatlearning.com/blog/types-of-neural-networks/

4. Έννοια νευρωνικών δικτύων

https://el.wikipedia.org/wiki/Νευρωνικό_δίκτυο

5. FPGA

https://en.wikipedia.org/wiki/Field-programmable_gate_array

6. Quartus

https://en.wikipedia.org/wiki/Quartus_Prime

7. Modelsim

https://en.wikipedia.org/wiki/ModelSim

8. υλοποίηση σιγμοειδής συνάρτησης vCalc

https://www.vcalc.com/wiki/vCalc/Sigmoid%20Function