Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Επιτάχυνση Υλικού για Αλγόριθμο Δένδρου Απόφασης σε Ετερογενές Υπολογιστικό Σύστημα

Διπλωματικη Εργασια

του

Άσιμ Ζουλκάρνι

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Αθήνα, Μάιος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Επιτάχυνση Υλικού για Αλγόριθμο Δένδρου Απόφασης σε Ετερογενές Υπολογιστικό Σύστημα

Διπλωματικη Εργασια

του

Άσιμ Ζουλκάρνι

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Μαΐου 2020.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επικ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2020

(Υπογραφή)

.....

Άσιμ Ζουλκάρνι
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι αλγόριθμοι μηχανικής μάθησης και συνεπώς οι μέθοδοι εξόρυξης δεδομένων, όπως οι αλγόριθμοι δένδρων απόφασης, χαίρουν αυξημένης δημοφιλίας σε εφαρμογές όπως οι ιατρικές διαγνώσεις, οι εκτιμήσεις ρίσκου έκδοσης πιστωτικών καρτών, η συμπεριφορά καταναλωτών κ.α. Ένα τόσο ευρύ φάσμα πεδίων στα οποία οι αλγόριθμοι των δένδρων απόφασης είναι εφαρμόσιμοι, σε συνδυασμό με την διαρκή αύξηση της διαθέσιμης πληροφορίας στην εποχή των Μεγάλων Δεδομένων, καθιστά επιθυμητή την αναζήτηση μεθόδων βελτιστοποίησης της απόδοσής τους σε ένα υπολογιστικό σύστημα τη στιγμή που πρόκειται για υψηλής υπολογιστικής απαιτητικότητας αλγορίθμους, που χρησιμοποιούν τα δεδομένα αυτά και συνιστούν ενδιάμεσες διαδικασίες πολυπλοκότερων αλγορίθμων όπως λ.χ. τα τυχαία δάση. Στις περιπτώσεις αυτές τα οφέλη της βελτιστοποίησης του μελετώμενου αλγορίθμου αναδεικνύονται ακόμα περισσότερο καθώς η επιτάχυνση των χρόνων εκτέλεσής τους ενισχύεται περαιτέρω.

Η ευελιξία και δημοφιλία των αλγορίθμων δένδρων απόφασης, σε συνδυασμό με την ευκολία χρήσης τους, αναδεικνύουν την ανάγκη για αναζήτηση λύσεων στο πρόβλημα της βελτιστοποίησης της απόδοσής τους σε μια σύγχρονη αρχιτεκτονική υλικού. Ως εκ τούτου, στην παρούσα διπλωματική εργασία επιχειρείται η προσέγγιση του ζητήματος αυτού μέσω εφαρμογής μεθόδων, που παρουσιάζονται αναλυτικότερα στη συνέχεια της εργασίας, με σκοπό τη βελτιστοποίηση μίας συγκεκριμένης υλοποίησης του αλγορίθμου δένδρου απόφασης (έκδοση C4.5) η οποία θα εκτελείται σε ετερογενές υπολογιστικό σύστημα το οποίο περιλαμβάνει συστοιχία επιτόπια προγραμματιζόμενων πυλών συνδυαστικά με κεντρική μονάδα επεξεργασίας.

Αρχικά, επιλέγεται και στη συνέχεια σχεδιάζεται ο υπολογιστικός πυρήνας που πρόκειται να εκτελείται σε συστοιχία επιτόπια προγραμματιζόμενων πυλών και αναλαμβάνει την υλοποίηση μέρους του αλγορίθμου ως συνάρτηση υλικού. Η σχεδίαση της συνάρτησης αυτής πραγματοποιείται με χρήση εργαλείων σύνθεσης υψηλού επιπέδου και σε γλώσσα προγραμματισμού C, για τους σκοπούς της οποίας επιχειρείται η καλύτερη δυνατή αξιοποίηση των πόρων του συστήματος ενώ μελετώνται πρωτόκολλα επικοινωνίας υλικού-λογισμικού. Τα αποτελέσματα που παρουσιάζονται αφορούν την υλοποίηση του αλγορίθμου στην πλατφόρμα ZedBoard της Digilent, της Zynq-7000 All Programmable SoC οικογένειας συσκευών.

Λέξεις Κλειδιά

Μηχανική Μάθηση, Εξόρυξη Δεδομένων, Δένδρο Απόφασης, C4.5, Σύνθεση Υψηλού Επιπέδου, Επιτάχυνση Υλικού.

Abstract

Machine learning algorithms and consequently data mining approaches such as Decision tree learning have been increasingly gaining popularity among fields such as medical diagnoses, risk evaluation of credit card application approval, consumers' behaviors, etc. Such a broad spectrum of fields where decision tree learning and classification are applicable along with the growth of available information to a massive extent in the era of big data make it desired to seek ways of optimizing a computationally wise heavy algorithm that can make use of this information and which alone is widely utilized and forms a single fraction of ensemble learning methods such as the random forests, etc. There, the benefits of an optimization in the former algorithm would be even further highlighted as the speedup is expected to be boosted by a significant degree.

Decision Tree Classification variants are among the most popular machine learning algorithms and have been applied in various fields with success. Their versatility and popularity, along with the ease to use, make it imperative that solutions be found regarding its performance optimization problem, hence in this project we tackle this issue by applying methods to optimize a Decision Tree Learning implementation (version C4.5), that will be executed in a heterogeneous computing system involving FPGA along with CPU, making use of the high-level synthesis oriented tools.

Initially, a profiling of the code is done, after which the computationally intensive part of the algorithm is determined, that is found to be the decision tree training part and within that part specifically, the Information Gain computations. Then the kernel has been developed as a hardware accelerated function that assumes the latter computations. The presented performance of the kernel was evaluated on a Zed platform integrated with Xilinx Zynq-7000 SoC in a FPGA-based heterogeneous system and it was shown that the accelerator can yield a significant kernel speedup in a single decision tree's training part, compared to an embedded ARM processor based implementation.

Keywords

Machine Learning, Data Mining, Decision Tree, C4.5, High-Level Synthesis, Hardware Acceleration.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή κ. Δημήτριο Σούντρη, για την ευκαιρία που μου έδωσε να ασχοληθώ με μία πολύ ενδιαφέρουσα εφαρμογή και σε ένα πολύ οργανωμένο και δημιουργικό περιβάλλον, όπως είναι το Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων (MicroLab) καθώς και για την εμπιστοσύνη που επέδειξε στο πρόσωπό μου. Επίσης ευχαριστώ ιδιαίτερα τον Μεταδιδακτορικό Ερευνητή κ. Χριστόφορο Κάχρη για την υπομονή του στις δύσκολες φάσεις της εργασίας, την καθοδήγηση και υποστήριξη που μου παρείχε με τις νουθεσίες του. Ακόμη, θα ήθελα να απευθύνω τις ευχαριστίες μου και σε όλο το προσωπικό του Εργαστηριού για την βοήθειά τους κατά τη διάρκεια εκπόνησης της διπλωματικής μου εργασίας, που υπήρξε πολύτιμη.

Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου, που με στήριξε καθ΄ όλη τη σταδιοδρομία μου στο Πανεπιστήμιο.

Contents

List of Figures

List of Tables

Chapter 1

Εκτεταμένη Περίληψη

1.1 Εισαγωγή

Ο επιστημονικός κλάδος της μηχανικής μάθησης εστιάζει στη μελέτη μηχανισμών μέσω των οποίων ευφυή συστήματα διαρκώς τείνουν να βελτιώνουν την απόδοσή τους. Η έρευνα στον τομέα αυτό περιλαμβάνει ανάπτυξη εφαρμογών που στην πλειοψηφία τους βασίζονται στην σύνταξη ενός μοντέλου γνώσης που χρησιμοποιείται ήδη από κάποια ανθρώπινη πορεία συλλογισμού. Σε ορισμένες τέτοιες περιπτώσεις, το ζητούμενο απαιτεί τη διεργασία κάποιας ταξινόμησης: την κατηγοριοποίηση αντικειμένων ανάλογα με κάποιες ιδιότητές τους. Τέτοια παραδείγματα περιλαμβάνουν μεταξύ άλλων περιπτώσεις έγκρισης πίστωσης χρημάτων, για την οποία εφαρμογή τα πρωταρχικά δεδομένα αφορούν στις λεπτομέρειες των μελετώμενων συναλλαγών καθώς και το ιστορικό πιστωτικών συναλλαγών του καταναλωτή, ενώ η προς ταξινόμηση απόφαση αφορά την έγκριση ή μη της συναλλαγής. Μία άλλη εφαρμογή θα μπορούσε να αφορά ένα σύστημα που συνεπικουρεί στην έκδοση προτάσεων σχετικά με τις κεφαλαιακές δαπάνες μίας εταιρίας, όπου κάθε προς εξέταση πρόταση δαπάνης ταξινομείται ως σύμφωνη ή μη με τις πολιτικές της εταιρίας. Εναλλακτική μελέτη περίπτωσης αφορά ιατρικές διαγνώσεις κάποιας ασθένειας, μια εφαρμογή για την οποία το σύστημα τροφοδοτείται από προηγούμενες διαγνώσεις μαζί με τα σχετικά σχόλια/λεπτομέρειες ανά περίπτωση, ενώ η ταξινόμηση μιας καινούργιας περίπτωσης κρίνεται καταφατική ή αρνητική αναλόγως με τις λεπτομέρειες που τη χαρακτηρίζουν.

Γίνεται φανερό, λοιπόν, πως ένα τόσο ευρύ φάσμα προβλημάτων στα οποία η προσέγγιση των δένδρων απόφασης/ταξινόμησης είναι εφαρμόσιμη, σε συνδυασμό με την ταχέως αυξανόμενη ποσότητα πληροφορίας προς διαχείριση, καθιστούν επιθυμητή την αναζήτηση μεθόδων για την βελτιστοποίηση ενός υπολογιστικά απαιτητικού αλγορίθμου. Οι ηλεκτρολόγοι μηχανικοί από την πλευρά τους, επιχειρούν την εύρεση μεθόδων για να επιτύχουν μία επιτάχυνση της απόδοσης του αλγορίθμου αυτού μέσω κατάλληλα προσαρμοσμένου υλικού στο οποίο θα εκτελούνται οι σχετικές διεργασίες που υλοποιούν τον αλγόριθμο αυτό. Επιθυμητό είναι δε, η υλοποίηση του αλγορίθμου, που αγνοεί το υλικό στο οποίο τρέχει, να μετατρέπεται με όσο το δυνατόν πιο εύκολο τρόπο σε υλοποίηση που το λαμβάνει υπόψιν προς αξιοποίηση των ειδικών δυνατοτήτων που αυτό έχει σχεδιαστεί να προσφέρει. Με αυτόν τον τρόπο προσεγγίζει

η παρούσα διπλωματική εργασία την επιτάχυνση υλικού ενός αλγορίθμου δένδρων απόφασης σε ένα ετερογενές υπολογιστικό σύστημα που περιλαμβάνει κεντρική μονάδα επεξεργασίας (CPU: Central Processing Unit) σε συνδυασμό με συστοιχία επιτόπια προγραμματιζόμενων πυλών (FPGA: Field-Programmable Gate Array).

Προσεγγίσεις σαν και αυτή πραγματοποιούνται προκειμένου να αυξήσουν την ενεργειακή απόδοση των υλοποιήσεων ευρέως διαδεδομένων αλγορίθμων μηχανικής μάθησης. Οι σύγχρονοι επεξεργαστές προηγμένης τεχνολογίας που θα χρησιμοποιούνταν εναλλακτικά για το σκοπό αυτό, παρά την υψηλή ταχύτητα που προσφέρουν, λειτουργούν σε πολύ υψηλές συχνότητες ρολογιού με συνεπαγόμενη μεγάλη κατανάλωση ενέργειας που κλιμακώνεται καθώς παράλληλες υλοποιήσεις χρησιμοποιούν επιπλέον επεξεργαστικούς πυρήνες. Τελευταία, η τεχνολογία των FPGA γίνεται όλο και πιο δημοφιλής μιας που προσφέρει τη δυνατότητα παράλληλης επεξεργασίας μεγάλης κλίμακας καθώς μοτίβα κώδικα εκτελούνται ταυτόχρονα σε ευέλικτες υπολογιστικές μονάδες χρησιμοποιώντας ιδανικά μόνο τα απαραίτητα στοιχεία, όπως λογικές πύλες, που απαιτούνται από συγκεκριμένους υπολογισμούς, σε αντίθεση με εναλλακτικές προκαθορισμένες αρχιτεκτονικές που απαιτούν την ενοποίηση όλων των επιμέρους στοιχείων τους για να λειτουργήσουν.

1.1.1 Συνεισφορά

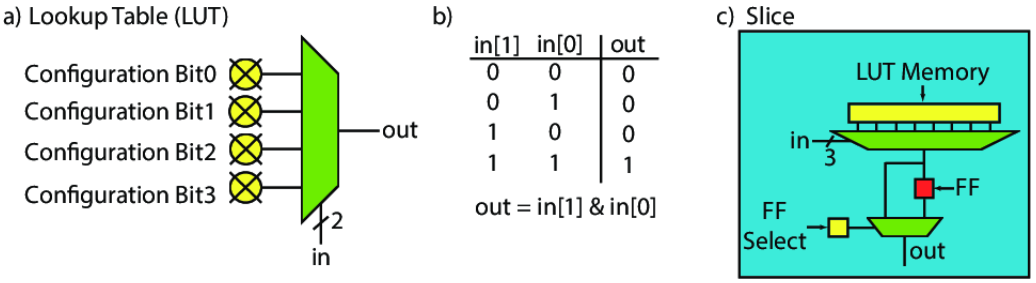
Η κύρια συνεισφορά της εργασίας αυτής περιλαμβάνει:

1. Παρουσίαση μιας καινοτόμας αρχιτεκτονικής υλικού για την επιτάχυνση αλγορίθμων δένδρων απόφασης.
2. Υλοποίηση αλγορίθμου μηχανικής μάθησης δένδρων ταξινόμησης.
3. Αποτίμηση αποτελεσμάτων και απόδοσης της υλοποίησης σε σύγκριση με αντίστοιχη σε κεντρική μονάδα επεξεργασίας.

1.2 Αρχιτεκτονική FPGA

Τα FPGAs (Field-Programmable Gate Arrays) είναι ημιαγώγιμες συσκευές που βασίζονται γύρω από μία μήτρα διαμορφώσιμων λογικών τμημάτων (CLBs: Configurable Logic Blocks) συνδεδεμένων μέσω προγραμματιζόμενων διασυνδέσεων. Οι συσκευές FPGA παρέχουν τη δυνατότητα επαναπρογραμματισμού μετά την παρασκευή τους ανάλογα με την επιθυμητή εφαρμογή ή λειτουργικότητα την οποία προορίζονται να επιτελούν. Με την πάροδο του χρόνου, τα FPGAs έχουν εξελιχθεί από μικρές συστοιχίες προγραμματιζόμενης λογικής και διασυνδέσεων σε τεράστιες συστοιχίες προγραμματιζόμενης λογικής και διασυνδέσεων με ενσωματωμένες μνήμες πάνω στην ψηφίδα, προσαρμοζόμενα μονοπάτια δεδομένων, υψηλές ταχύτητες εισόδου/εξόδου καθώς και μικροεπεξεργαστικούς πυρήνες, όλα στην ίδια ψηφίδα.

Τα FPGAs αποτελούνται από συστοιχίες τμημάτων λογικών πυλών και στοιχείων μνήμης συνδεδεμένα μέσω προγραμματιζόμενων διασυνδέσεων. Συνήθως, αυτά τα τμήματα λογικής υλοποιούνται ως πίνακες αναζήτησης (LUTs: Look-up Tables), μνήμες δηλαδή των οποίων τα



Σχήμα 1.1: Στο μέρος a) φαίνεται ένα LUT 2 εισόδων με τα 4-bits να διαμορφώνουν τη λειτουργικότητά του. Στο μέρος b) φαίνεται ο πίνακας αληθείας πύλης AND ως πιθανής λειτουργικότητας, Στο μέρος c) φαίνεται ένα απλό CLB (ή slice) με δυνατότητα αποθήκευσης της εξόδου σε FF. (Πηγή: [3])

εισόδια σήματα σχηματίζουν τις διευθύνσεις και τα εξόδια παρέχονται από τις καταχωρίσεις της μνήμης. Για την υλοποίηση μιας λειτουργικότητας με LUT, ο πίνακας αληθείας της συνάρτησης που την περιγράφει είναι αποθηκευμένος στη μνήμη και οι επιστρεφόμενες τιμές οδηγούνται από το υπόλοιπο κύκλωμα. Αυτοί οι πίνακες αποτελούν αποδοτική και ευέλικτη μέθοδο για σχετικά μικρές λογικές συναρτήσεις· η λειτουργικότητα εναλλάσσεται με προσαρμογή του περιγραφικού πίνακα αληθείας. Τα περισσότερα FPGAs χρησιμοποιούν LUT τεσσάρων ως έξι bit εισόδου ενώ τα μεγαλύτερα FPGAs περιλαμβάνουν εκατομμύρια τέτοια στοιχεία.

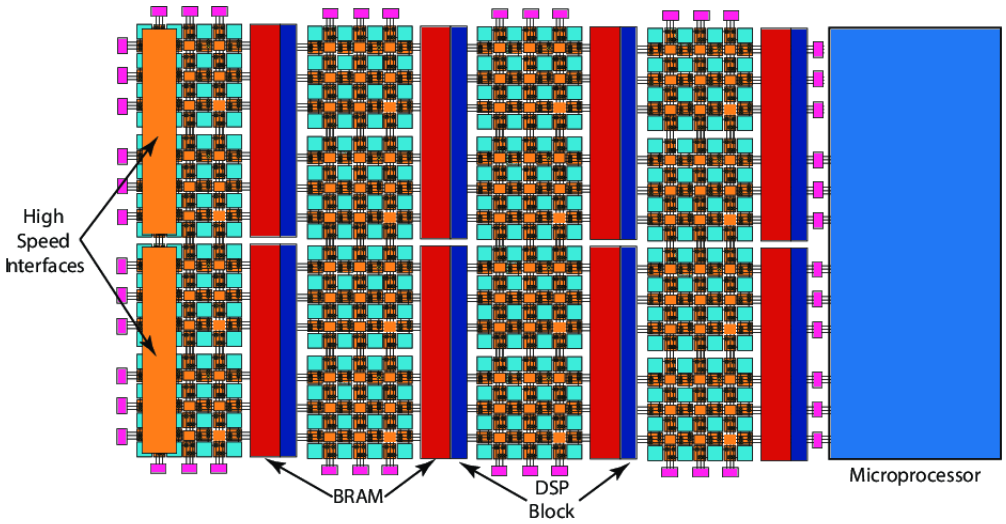
Το βασικό στοιχείο μνήμης ενός FPGA είναι το φλιπ-φλοπ (FF: Flip-Flop) και σύνολα αυτών συνήθως βρίσκονται τοποθετημένα μαζί με LUTs (βλ. Σχήμα 1.1). Πολλαπλά LUTs μπορούν να συνδυαστούν με FFs και άλλα κυκλώματα που προσφέρουν επιπρόσθετες λειτουργικότητες όπως λ.χ. έναν αθροιστή για το σχηματισμό ενός συνθετώτερου τμήματος λογικής που ονομάζεται διαμορφώσιμο λογικό τμήμα (CLB). Ένα CLB είναι ένας συνδυασμός από ένα μικρό αριθμό LUTs, FFs και πολυπλεκτών για το σχηματισμό ενός ισχυρότερου προγραμματιζόμενου λογικού στοιχείου, ενώ ο αριθμός των στοιχείων αυτών που διαμορφώνουν ένα CLB ποικίλλει από αρχιτεκτονική σε αρχιτεκτονική.

Ένα CLB μπορεί να αξιοποιεί και πολυπλοκότερες λογικές συναρτήσεις, όπως για παράδειγμα έναν ενσωματωμένο αθροιστή ως στοιχείο σύνθεσης. Στην περίπτωση αυτή ο αθροιστής δεν είναι προγραμματιζόμενη λογική, αλλά μπορεί να χρησιμοποιηθεί μόνο αυτούσιος και καθώς η λειτουργικότητα που επιτελεί είναι καίρια και ευρέως χρησιμοποιούμενη, είναι πιο αποδοτικό να χρησιμοποιείται ως τέτοιος παρά ως υλοποιημένος σε προγραμματιζόμενη λογική.

Καθώς δε το πλήθος των τρανζίστορ σε ένα FPGA συνεχίζει να αυξάνεται, οι αρχιτεκτονικές του τείνουν να χρησιμοποιούν όλο και περισσότερα στοιχεία που επιτελούν αυτόνομες λειτουργικότητες. Παραδείγματος χάριν, η ανάγκη πολλών εφαρμογών για πολλαπλούς υπολογισμούς προσθέσεων και πολλαπλασιασμών, οδήγησε σε FPGAs που περιέχουν πόρους που επιτελούν τις πράξεις αυτές όπως είναι τα στοιχεία DSPs: Digital Signal Processors που συν τοις άλλοις δίνουν τη δυνατότητα εκτέλεσης πράξεων συσσωρευτικού πολλαπλασιασμού πολύ πιο αποδοτικά από αντίστοιχη υλοποίηση προγραμματιζόμενης λογικής. Οι τμηματικές RAM (Random-Access Memory) (BRAM) (βλ. Σχήμα 1.2), από την άλλη μεριά είναι μονάδες διαμορφώσιμων μνημών τυχαίας προσπέλασης που υποστηρίζουν διάφορες διατάξεις, δυνατότητα

διευθυνσιοδότησης διαφορετικού μήκους λέξης, διεπαφές για διαύλους ή επεξεργαστές πάνω στην ψηφίδα και έτσι αποτρέπουν το κόστος χρόνου που θα συνεπαγόταν μια συνάμα σπάταλη από άποψη πόρων, αντίστοιχη υλοποίηση μνημών με FFs.

Τέλος, η προγραμματιζόμενη διασύνδεση είναι το άλλο σημείο κλειδί ενός FPGA, αφού παρέχει ένα ευέλικτο δίκτυο καλωδίων για τη σύνδεση μεταξύ των CLBs. Οι είσοδοι και οι έξοδοι του CLB συνδέονται σε ένα κανάλι δρομολόγησης που περιέχει τα bit διαμόρφωσης που οδηγούν την επικοινωνία με την προγραμματιζόμενη διασύνδεση και με τον τρόπο αυτό επιτυγχάνεται η επικοινωνία του FPGA με εξωτερικές συσκευές όπως έναν μικροεπεξεργαστή, μία μνήμη, έναν αισθητήρα κλπ. Αξίζει να σημειωθεί πως στις πλατφόρμες της εταιρίας Xilinx που βασίζονται σε ετερογενείς αρχιτεκτονικές με FPGAs, η σχεδίαση σύνθετων λειτουργικοτήτων και συναρτήσεων προσεγγίζεται με μεθοδολογίες σύνθεσης υψηλού επιπέδου (HLS: High-Level Synthesis) σε αναπτυξιακές πλατφόρμες ορισμού συστήματος ψηφίδας μέσω λογισμικού (SDSoC: Software-Defined System on Chip) που διευκολύνουν στην καλύτερη αξιοποίηση των πόρων του FPGA.



Σχήμα 1.2: Τα σύγχρονα FPGAs είναι ετερογενή περιλαμβάνοντας στοιχεία που επιτελούν προκαθορισμένες λειτουργικότητες, ενώ συχνά συνδυάζονται και με εξωτερικές μονάδες εξεργασίας μέσω διεπαφών πρωτοκόλλου AXI (Advanced eXtensible Interface). (Πηγή: [3])

1.3 Θεωρητικό Υπόβαθρο

1.3.1 Μοντέλο Ταξινόμησης

Οι αλγόριθμοι μηχανικής μάθησης όπως αυτοί των δένδρων απόφασης ή ταξινόμησης εφαρμόζονται ως μοντέλα σε ποικίλες περιπτώσεις εφαρμογών που απαιτούν πρόβλεψη ή ταξινόμηση. Πιο συγκεκριμένα, ο τρόπος με τον οποίο τα δένδρα απόφασης λειτουργούν είναι ο εξής: μία δενδρική δομή σχηματίζεται, αναπαριστώντας τα δεδομένα εκπαίδευσης του συστήματος με ένα συμπαγές σύνολο αν-τότε κανόνων, των λεγόμενων κανόνων ταξινόμησης.

Ακολούθως (βλ. Πίνακα 1.1) παρουσιάζεται ένα σύνολο δεδομένων (σ.δ.) εκπαίδευσης

δένδρου απόφασης (training data). Κάθε περίπτωση των δεδομένων περιλαμβάνει τιμές για ένα σύνολο χαρακτηριστικών που λαμβάνονται υπόψιν για την κατηγοριοποίησή της στην εκάστοτε κλάση. Η διαδικασία εκπαίδευσης, που αναλύεται στη συνέχεια, οδηγεί στην δόμηση μίας δενδρικής δομής που καλείται να αντιστοιχίσει περιπτώσεις των δεδομένων δοκιμής (testing data) σε κλάσεις (classification).

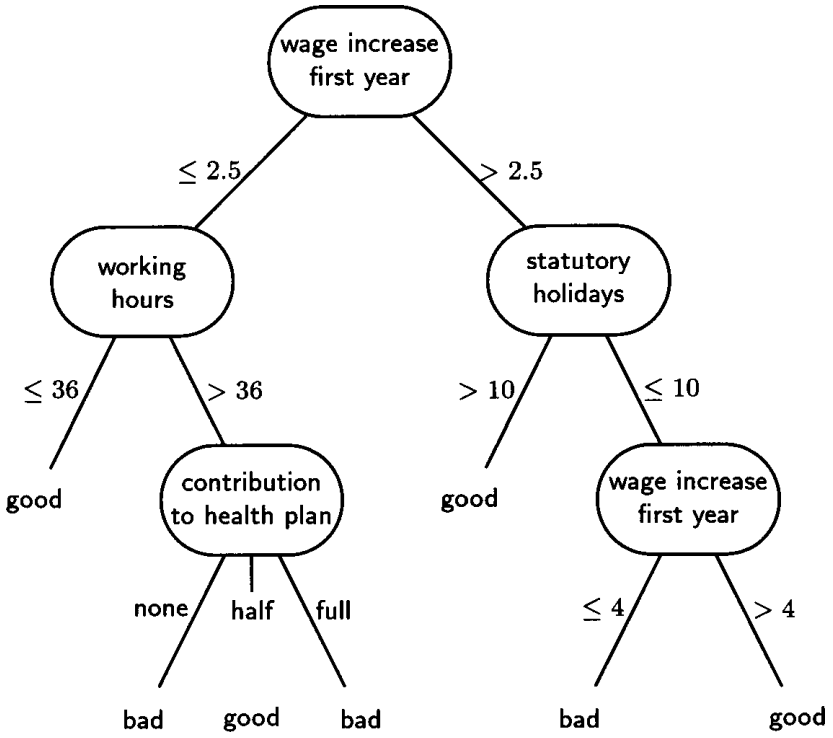
Outlook	Temp ($^\circ F$)	Humidity (%)	Windy	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

Πίνακας 1.1: Αναπαράσταση συνόλου δεδομένων εκπαίδευσης δένδρου απόφασης ως σύνολο περιπτώσεων τιμών χαρακτηριστικών, με πρωθύστερη αντιστοίχιση σε κλάσεις για τους σκοπούς της εκπαίδευσης. (Πηγή: [1])

Ο κάθε κόμβος του δένδρου ορίζει μία συνθήκη ελέγχου ενός δοσμένου χαρακτηριστικού των περιπτώσεων και το κάθε κλαδί που αναχωρεί από τον κόμβο αυτόν αντιστοιχεί σε μία διαφορετική διακριτή τιμή ενός συγκεκριμένου χαρακτηριστικού. Σε κάθε κόμβο η τιμή της περίπτωσης για το χαρακτηριστικό του κόμβου ελέγχεται και με τον τρόπο αυτό διαλέγεται το μονοπάτι που ακολουθείται. Στο Σχήμα 1.3 παρουσιάζεται μία δενδρική δομή που αναλύει διάφορα χαρακτηριστικά αξιολόγησης εργασιακών συνθηκών, ενώ η δυϊκή, στην περίπτωση αυτή, αποτίμηση της κλάσης γίνεται ανάλογα με την τιμή σε κάθε χαρακτηριστικό.

Η διαδικασία μέσω της οποίας κτίζεται ένα δένδρο απόφασης λέγεται εκπαίδευση του δένδρου. Σύμφωνα με την έκδοση C4.5 του αλγορίθμου εκπαίδευσης δένδρου απόφασης και παρομοίως με την προγενέστερη έκδοση επαναλαμβανόμενης διχοτόμησης (ID3: Iterative Dichotomiser 3), το δένδρο κτίζεται βάσει της αποτίμησης της μετρικής της εντροπίας πληροφορίας. Η διαδικασία είναι αναδρομική και το δένδρο δομείται από πάνω προς τα κάτω εκλέγοντας σε κάθε βήμα το διαχωριστικό χαρακτηριστικό που μεγιστοποιεί το κέρδος πληροφορίας. Μία επέκταση που προσφέρει ο C4.5 αλγόριθμος είναι η δυνατότητα διαχείρισης χαρακτηριστικών των οποίων οι τιμές κυμαίνονται σε συνεχές διάστημα λ.χ. αριθμών, ε-

φαρμόζοντας δυαδική διακριτοποίηση σε τιμές μικρότερες και μεγαλύτερες ή ίσες από κάποιο κατώφλι. Επιπροσθέτως, περιλαμβάνει μια διαδικασία κλαδέματος του παραχθέντος δένδρου αντικαθιστώντας παρακλάδια του με φύλλα που αντιστοιχούν στην επικρατούσα τιμή του υποδένδρου με στόχο την μείωση της υπερπροσαρμογής του δένδρου στα δεδομένα εκπαίδευσης. Το πρόβλημα αυτό είναι γνωστό ως overfitting και επιχειρείται η θεραπεία του από μεθόδους συνόλων δένδρων απόφασης, τα λεγόμενα τυχαία δάση. Τέλος, αναφορικά με απούσες τιμές στα σύνολα δεδομένων, αυτές αγνοούνται στο στάδιο της εκπαίδευσης ενώ στο στάδιο της ταξινόμησης επιστρέφεται η συχνότερη απόκριση στο σύνολο των αποκρίσεων, αν η απούσα τιμή ήταν να αντικατασταθεί με κάθε δυνατή διακριτή τιμή της μεταβλητής που αφορά.



Σχήμα 1.3: Γραφική αναπαράσταση δένδρου απόφασης αξιολόγησης εργασιακών συνθηκών. (Πηγή: [1])

1.3.2 Διακριτοποίηση Δεδομένων Εκπαίδευσης

Αναλόγως με τα δεδομένα εκπαίδευσης και με κριτήριο αν περιέχουν χαρακτηριστικά συνεχούς εύρους τιμών, εφαρμόζεται μία διαδικασία προεπεξεργασίας για την διακριτοποίησή τους· ακολούθως τα χαρακτηριστικά αντιμετωπίζονται σαν διακριτά δύο κατηγοριών, αυτών που υπερβαίνουν και αυτών που υποβαίνουν την τιμή του κατωφλίου. Το κατώφλι αυτό διαχωρίζει τα δεδομένα σε δύο κατηγορίες ως εξής. Έστω $S = \{a_1, a_2, ..., a_n\}$ οι τιμές εκπαίδευσης του χαρακτηριστικού από ένα συνεχές και διατεταγμένο σύνολο, που χωρίς βλάβη της γενικότητας θεωρούνται πραγματικοί αριθμοί και m το πλήθος των κλάσεων $\{C_1, C_2, ..., C_m\}$ του συνόλου των δεδομένων εκπαίδευσης. Εκ των $n-1$ δυνατών τιμών διαχωρισμού που αποτιμώνται, αυτή που ελαχιστοποιεί τη μετρική πληροφορίας επιλέγεται ως κατώφλι. Χρησιμοποιώντας μια σει-

ριακή διάταξη των τιμών, το μέγιστο κέρδος πληροφορίας βρίσκεται σε ένα πέρασμα. Η μετρική πληροφορίας που χρησιμοποιείται υπολογίζεται για κάθε πιθανό διαχωρισμό των δεδομένων, a_i ως εξής:

$$G(S) = -p_h \cdot E(S_h) - p_l \cdot E(S_l)$$

όπου

$$p_{h,l} = \frac{|S_{>a_i,\leqslant a_i}|}{|S|}$$

είναι τα κλάσματα των τιμών μεγαλύτερων (δείκτης h) και μικρότερων ή ίσων (δείκτης l) της ελεγχόμενης τιμής διαχωρισμού, αντίστοιχα και

$$E(S) = -\sum_{i=1}^{m} p_i \cdot log_2(p_i)$$

είναι η εντροπία πληροφορίας ενός συνόλου, όπου

$$p_i = \frac{freq(C_i, S)}{|S|}$$

είναι το κλάσμα των τιμών μέσα στο σύνολο που αντιστοιχούν στην κλάση υπ' αριθμόν i.

1.3.3 Εκπαίδευση Δένδρου Απόφασης

Έχοντας αντιμετωπίσει το ζήτημα των χαρακτηριστικών συνεχών τιμών, ο αλγόριθμος δόμησης του δένδρου απόφασης προβαίνει στη μεταχείρισή τους πλέον με καθολικό τρόπο. Εξασφαλισμένου ενός προσαρμοσμένου συνόλου δεδομένων εκπαίδευσης, έπονται τρία στάδια για τα οποία αποφαίνεται ο αλγόριθμος.

- Αν οι παρεχόμενες από το σύνολο δεδομένων περιπτώσεις ανήκουν σε μία κλάση, το δένδρο είναι φύλλο που αντιστοιχεί στην κλάση αυτή.
- Αν το σύνολο δεδομένων δεν περιέχει καμία περίπτωση, το δένδρο είναι φύλλο που αντιστοιχεί σε μία προκαθορισμένη (default) κλάση.
- Αν το σύνολο δεδομένων περιέχει περιπτώσεις που ανήκουν σε διάφορες κλάσεις, τότε το χαρακτηριστικό A με τιμές που περιγράφονται από το σύνολο $\{a_1, a_2, ..., a_n\}$ που μεγιστοποιεί τη μετρική κέρδους πληροφορίας υπολογίζεται για κάθε χαρακτηριστικό ως εξής:

  $$G(S, A) = E(S) - \sum_{i=1}^{n} p_i \cdot E(S_i) \tag{1.1}$$

  όπου $E(S)$ είναι η εντροπία πληροφορίας του υπό εξέταση χαρακτηριστικού και

  $$p_i = \frac{|S_i|}{|S|} \tag{1.2}$$

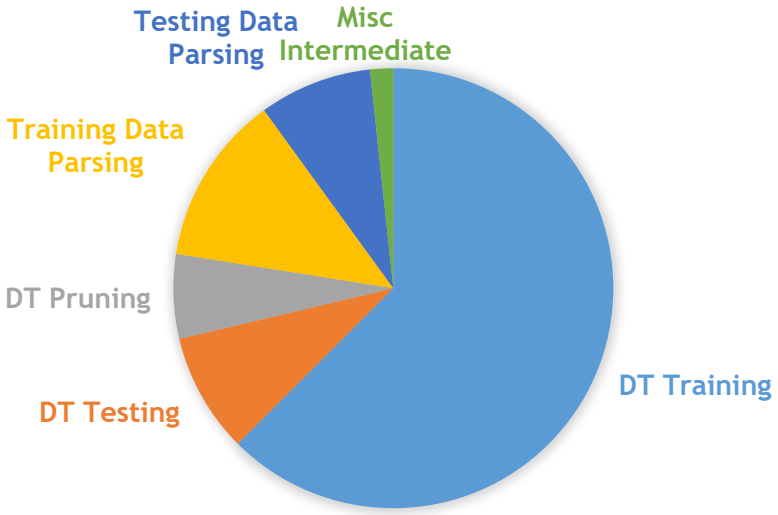
  όπου S_i είναι το νέο σύνολο δεδομένων που θα προέκυπτε αν ο διαχωρισμός γινόταν στην υπ' αριθμόν i τιμή χαρακτηριστικού. Ο αλγόριθμος συνεχίζει να δημιουργεί με τον τρόπο αυτό κάθε κόμβο του δένδρου αναδρομικά ενόσω υπολείπονται υποσύνολα δεδομένων εκπαίδευσης.

1.4 Σχεδιασμός Επιταχυντή Υλικού σε FPGA

Η συσκευή τύπου FPGA προσφέρει τη δυνατότητα εν παραλλήλω αποτίμησης συναρτήσεων που εναλλακτικά χρησιμοποιούν τους πόρους κεντρικής μονάδας επεξεργασίας (CPU), ωφελώντας σε απόδοση χάρη στην παράλληλη υλοποίηση στο κύκλωμα. Το περιβάλλον ανάπτυξης που προσφέρει τη δυνατότητα ορισμού συστήματος πάνω σε ψηφίδα μέσω λογισμικού δίνει τη δυνατότητα υλοποίησης του μελετώμενου αλγορίθμου σε γλώσσες υψηλού επιπέδου όπως είναι οι C/C++ γλώσσες προγραμματισμού, παράγοντας αυτομάτως την περιγραφή του αντιστοίχου κυκλώματος στην αντίστοιχη γλώσσα περιγραφής υλικού (HDL: Hardware Description Language). Στην παρούσα εργασία, εστιάζουμε στον εντοπισμό τμήματος ή τμημάτων του αλγορίθμου που κρίνονται υψηλών υπολογιστικών απαιτήσεων, ακολούθως ξεχωρίζοντάς το ή τα από το υπόλοιπο πρόγραμμα και επισημαίνοντας την οδηγία μεταγλώττισης απευθείας σε κύκλωμα στη συσκευή του FPGA. Αυτό το μοτίβο κώδικα που ακολουθείται για τους σκοπούς του διαχωρισμού της επιλεγμένης συνάρτησης, συνεπάγεται μία σειρά από περιορισμούς που επιβάλλονται αυτομάτως τόσο στον κώδικα που υλοποιείται σε υλικό όσο και τον υπόλοιπο κώδικα της υβριδικής αυτής υλοποίησης που εκτελείται στην κύρια μονάδα επεξεργασίας, καθώς η ροή εκτέλεσης του προγράμματος επιτάσσει την αναφορά στις ίδιες δομές δεδομένων που περιγράφουν τα δεδομένα εκπαίδευσης. Πρώτο βήμα λοιπόν, είναι η σκιαγράφηση της κατανομής του υπολογιστικού φόρτου ανά επιμέρους συναρτήσεις που συντελούν στην υλοποίηση του αλγορίθμου με βάση τις υπολογιστικές τους απαιτήσεις.

1.4.1 Ανάλυση Κατανομής Υπολογιστικού Φόρτου

Η μετρική που χρησιμοποιείται για την αποτίμηση του υπολογιστικού φόρτου είναι ο χρόνος και τα αποτελέσματα για την υλοποίηση του αλγορίθμου που προγραμματίστηκε ειδικά για τους σκοπούς της εργασίας ακολουθούν (βλ. Σχήμα 1.4 και αναλυτικά ποσοστά στον Πίνακα 1.2).



Σχήμα 1.4: Σκιαγράφηση του υπολογιστικού φόρτου (εφαρμογή για σ.δ. Adult [14]) ανά διαδικασία του αλγορίθμου δένδρου απόφασης (DT: Decision Tree). (Πηγή: [21])

Για το τμήμα δοκιμής του δένδρου απόφασης, το υπολογιστικό κόστος έγκειται στην προσπέλαση ενός μοναδικού μονοπατιού για έκαστη εκ των περιπτώσεων δοκιμής. Το δε τμήμα εκπαίδευσης του αλγορίθμου κρίνεται μακράν το πιο υπολογιστικά απαιτητικό συγκριτικά με το τμήμα δοκιμής και παρά τη δυνατότητα παραλληλοποίησης αυθαίρετα μεγάλου βαθμού που προσφέρει το δεύτερο, για τους σκοπούς της εργασίας αυτής επιλέγεται η βελτιστοποίηση υπό τους περιορισμούς του συστήματος της πλατφόρμας ZedBoard/Zynq-7000 All Programmable SoC του τμήματος εκπαίδευσης του δένδρου απόφασης.

Metric / Procedure	Time (%)
Decision Tree Training	62.5
Decision Tree Testing	8.3
Decision Tree Pruning	6.2
Training Data Parsing	12.5
Testing Data Parsing	8.8
Miscellaneous Intermediate	1.7

Πίνακας 1.2: Περιγραφή κατανομής υπολογιστικού φόρτου ανά επιμέρους διαδικασία.

Η διαδικασία δόμησης του δένδρου περιλαμβάνει μία σειρά από διαδικασίες που καταναλώνουν αξιοσημείωτα ποσοστά χρόνου, με ένα τμήμα εξ αυτών να αφορά σε διαδικασίες υψηλών απαιτήσεων μνήμης των οποίων η προσέγγιση της παρουσιαζόμενης μεθόδου επιτάχυνσης υλικού κρίνεται μη αποδοτική. Από την άλλη μεριά όμως, ενδιάμεσες διαδικασίες όπως ο υπολογισμός της μετρικής του κέρδους πληροφορίας καταναλώνουν αρκετό χρόνο ώστε να αξίζουν την επιμέρους επιτάχυνσή τους, ενώ οι απαιτήσεις μνήμης τμημάτων σαν και αυτό είναι δυνατόν να καλύπτονται από τους πόρους του συστήματος με το FPGA, σε αντίθεση με τα υπόλοιπα τμήματα της διαδικασίας εκπαίδευσης του δένδρου απόφασης, που απαιτούν την σύγχρονη επεξεργασία δεδομένων και πολλαπλών υποσυνόλων αυτών, καθώς ο αλγόριθμος προχωρά αναδρομικά.

Πιο συγκεκριμένα, από τη σκιαγράφηση της κατανομής του υπολογιστικού φόρτου ανά διαδικασία και για την μελετώμενη υλοποίηση του αλγορίθμου σε C++, το 62.5% του συνολικού χρόνου (συμπεριλαμβανομένου του χρόνου ανάγνωσης και διαχωρισμού των δεδομένων) καταναλώνεται από το τμήμα της εκπαίδευσης του δένδρου απόφασης. Το τμήμα δοκιμής του δένδρου εκπαίδευσης καταναλώνει μόλις το 8.3% του συνολικού χρόνου ενώ ο χρόνος που καταναλώνεται μόνο για την ανάλυση των δεδομένων ανέρχεται στο 8.8% του συνολικού χρόνου, ενώ τα δεδομένα αυτά αφορούν σύνολα δεδομένων εκπαίδευσης και δοκιμής με αναλογία περιπτώσεων κατ΄ αντιστοιχία 2:1. Καθώς δε η κλιμάκωση των δεδομένων δοκιμής οδηγεί σε κατ΄ αναλογία αύξηση χρόνου, λόγω της $O(1)$ πολυπλοκότητας επεξεργασίας μίας περίπτωσης σταθερού αριθμού χαρακτηριστικών, αν η αναλογία των δεδομένων ήταν να ισοσταθμιστεί σε 1:1, με διπλασιασμό του συνόλου περιπτώσεων δοκιμής, τα ποσοστά που θα περιέγραφαν την

νέα κατανομή είναι

$$p_{train} = 62.5\% \times \frac{100}{117.1} = 53.4\%$$

και

$$p_{test} = 2 \times 8.3\% \times \frac{100}{117.1} = 14.2\%$$

και με τον τρόπο αυτό ενισχύεται το επιχείρημα για τη μελέτη και βελτιστοποίηση του τμήματος εκπαίδευσης του δένδρου απόφασης· το ποσοστό του χρόνου που δαπανάται για την δόμηση ενός δέντρου απόφασης με ένα συγκεκριμένο πλήθος περιπτώσεων είναι ίσαμε 3.76 φορές περισσότερο από αυτό που δαπανάται για το ίδιο πλήθος περιπτώσεων του τμήματος δοκιμής, το οποίο, λαμβάνοντας υπόψιν μια εσωτερική όψη των υλοποιήσεων των τμημάτων αυτών, όπως αναφέρθηκε προηγουμένως κρίνεται ότι δεν είναι υπολογιστικά απαιτητικό, παρά απαιτητικό από άποψη μνήμης.

Στη συνέχεια, κρίνεται σκόπιμο να επικεντρωθεί το ενδιαφέρον στην εσωτερική δομή των συναρτήσεων που απαρτίζουν το τμήμα εκπαίδευσης του δένδρου απόφασης με στόχο να επιλεχθεί το μεγαλύτερο κατάλληλο τμήμα και να υλοποιηθεί σαν συνάρτηση υλικού (hardware function) που θα τρέχει σε FPGA. Από το 62.5% του χρόνου που καταναλώνεται για τη δόμηση του δένδρου, οι δύο ποσότητες των 37.5% και 25% απαιτούνται από τις διεργασίες που είναι υπεύθυνες για α) τη διαμόρφωση του υποσυνόλου των δεδομένων εκπαίδευσης επιλέγοντας τμήμα των περιπτώσεων εκπαίδευσης για το επόμενο βήμα της αναδρομικής προόδου του αλγορίθμου και β) την αποτίμηση του χαρακτηριστικού διαχωρισμού των δεδομένων σε κάθε αναδρομικό βήμα, αντίστοιχα.

1.4.2 Επιλογή Συνάρτησης για Υλοποίηση σε Υλικό

Το πρώτο μέρος του τμήματος εκπαίδευσης είναι αρκετά απαιτητικό από άποψη αναγκών μνήμης, οπότε αν και επιδέχεται παραλληλοποίηση, αυτή έγκειται στη δυνατότητα πρόσβασης της μνήμης τυχαίας προσπέλασης (RAM) με παράλληλο χαρακτήρα και ταυτοχρόνως ποσότητας δεδομένων που εύκολα υπερβαίνουν το ποσό της δυναμικής RAM (DRAM: Dynamic RAM) που μπορεί να παράσχει ένα ετερογενές σύστημα FPGA, ιδιαιτέρως τη στιγμή που οι περιπτώσεις του συνόλου δεδομένων εκπαίδευσης που προσπελάζονται, είναι απαραίτητο να συνυπάρχουν στην ίδια μνήμη καθώς το δένδρο κτίζεται αναδρομικά από πάνω προς τα κάτω. Από την άλλη μεριά όμως, το κομμάτι που αφορά την αποτίμηση του χαρακτηριστικού διαχωρισμού σε κάθε βήμα της αναδρομής απαιτεί την επεξεργασία του συνόλου των δεδομένων που περιλαμβάνονται σε κάθε υποσύνολο των περιπτώσεων που χρησιμοποιούνται για την εκπαίδευση του δένδρου. Κατά τη διάρκεια του κομματιού αυτού της επεξεργασίας των δεδομένων, το κέρδος πληροφορίας υπολογίζεται μέσω αποτίμησης των συχνοτήτων εμφάνισης των τιμών των χαρακτηριστικών και κατόπιν της εφαρμογής της μετρικής του κέρδους πληροφορίας κάθε φορά ώστε η συνάρτηση να επιστρέφει το χαρακτηριστικό που το μεγιστοποιεί.

Το υπό-τμήμα που αφορά στην αποτίμηση το χαρακτηριστικού διαχωρισμού διαλέγεται για υλοποίηση ως συνάρτηση υλικού και αυτό επιτυγχάνεται μέσω κατά σημεία μερικού και κατά

άλλα σημεία ολικού επανασχεδιασμού της υλοποίησης και αναθεώρηση των δομών δεδομένων που περιγράφουν τα υπό επεξεργασία από το FPGA δεδομένα επηρεάζοντας με τον τρόπο αυτό και διαδικασίες που τρέχουν στην κεντρική μονάδα επεξεργασίας, αλλά αναφέρονται στα ίδια δεδομένα.

1.5 Σχεδιασμός Συνάρτησης σε Υλικό

Το επόμενο βήμα της εργασίας αυτής είναι αναθεώρηση της υλοποίησης του αλγορίθμου δένδρου απόφασης από εξ ολοκλήρου προσέγγιση λογισμικού σε γλώσσα προγραμματισμού C++ σε τέτοια, ώστε η επιλεγμένη συνάρτηση που επελέγη με τα προαναφερθέντα κριτήρια να υλοποιηθεί απευθείας στο υλικό. Στο σημείο αυτό, γίνεται μια σύντομη περιγραφή των στοιχείων της σύνθεσης υψηλού επιπέδου που οδηγούν τη σχεδίαση της συνάρτησης υλικού.

Η σύνθεση υψηλού επιπέδου (HLS: High-Level Synthesis) προσφέρει ένα ακόμη στάδιο αφαιρετικότητας στην προσέγγιση της σχεδίασης του υλικού και δίνει, με τον τρόπο αυτό, τη δυνατότητα στο σχεδιαστή να επικεντρωθεί στα μεγαλύτερα σχεδιαστικά ζητήματα της υλοποίησης παρά στους επιμέρους καταχωρητές και τις λειτουργίες ανά κύκλο του ρολογιού. Τα εργαλεία σύνθεσης υψηλού επιπέδου αναλαμβάνουν την περιγραφή της μικροαρχιτεκτονικής επιπέδου μεταφοράς καταχωρητών (RTL: Register-Transfer Level) σε γλώσσα περιγραφής υλικού.

Συνοπτικά, τα εργαλεία σύνθεσης υψηλού επιπέδου αναλαμβάνουν την επιτέλεση διαδικασιών που εναλλακτικά θα έκανε ένας σχεδιαστής επιπέδου μεταφοράς καταχωρητών, όπως

- η ανάλυση και αξιοποίηση της δυνατότητας παράλληλης επεξεργασίας σε έναν αλγόριθμο.
- η εισαγωγή καταχωρητών καταλλήλως, ώστε να περιοριστεί το μέγεθος της κρίσιμης διαδρομής της σχεδίασης και να επιτευχθεί η λειτουργία του συστήματος στην επιθυμητή συχνότητα ρολογιού.
- η δημιουργία της λογικής ελέγχου που καθοδηγεί το μονοπάτι δεδομένων.
- η υλοποίηση των διεπαφών μέσω των οποίων το FPGA επικοινωνεί με το υπόλοιπο σύστημα.
- η απεικόνιση δεδομένων σε στοιχεία μνήμης για την καλύτερη αξιοποίηση των πόρων και του εύρους ζώνης επικοινωνίας.
- η απεικόνιση υπολογιστικών πυρήνων σε στοιχεία λογικής σχεδίασης επιτελώντας αυτόματα βελτιστοποιήσεις που υποδεικνύονται από τον χρήστη.

Στόχος της χρήσης σύνθεσης υψηλού επιπέδου είναι η κατά δύναμιν αυτοματοποίηση των παραπάνω διεργασιών βάσει υποδεικνυόμενων από τον χρήστη καταλλήλων μηνυμάτων. Το εργαλείο σύνθεσης υψηλού επιπέδου που αξιοποιείται στην παρούσα εργασία είναι το Vivado HLS της Xilinx, που μάλιστα προσφέρεται σε ολοκληρωμένο περιβάλλον προγραμματισμού και υλοποίησης κώδικα, το οποίο έχει επισημανθεί προηγουμένως ως SDSoC. Σε γενικές γραμμές,

απαιτούνται προσαρμογές του κώδικα και ανασχεδιασμός τμημάτων του ώστε να σέβεται τους κάτωθι περιορισμούς.

- Περιορισμός αναφορών σε αναφορές και λοιπών κλιμακούμενων αναφορών.
- Χρήση στατικής παρά δυναμικής διαχείρισης μνήμης από τη συνάρτηση υλικού.
- Γενικώτερος περιορισμός σχετικά με κλήσεις συστήματος.
- Περιορισμένος αριθμός τυπικών βιβλιοθηκών.
- Περιορισμός στοιχείων αντικειμενοστρέφειας της γλώσσας C++, όπως είναι οι δείκτες σε συναρτήσεις και οι εικονικές συναρτήσεις.
- Αδυναμία υποστήριξης αναδρομικών συναρτήσεων (μεταξύ άλλων περιορισμών, αφορά την εργασία αυτή σε κρίσιμο βαθμό σχετικά με την επιλογή μόνο τμήματος της διαδικασίας εκπαίδευσης που δεν είναι αναδρομικό παρά είναι ένα μεγάλο κομμάτι του αναδρομικού βήματος και κατά συνέπεια έχουμε κλήσεις της συνάρτησης υλικού όσες και τα βήματα αυτά).
- Η διεπαφή επικοινωνίας των δεδομένων από την εκτός ψηφίδας μνήμη πρέπει να υποδειχθεί ρητά από τον σχεδιαστή.

Το εργαλείο σύνθεσης υψηλού ως σύστημα απαιτεί είσοδο που περιλαμβάνει

- την επισήμανση της συνάρτησης που πληροί τις παραπάνω προδιαγραφές για υλοποίηση σε υλικό.
- μία δοκιμή σχεδίασης που για τους σκοπούς επαλήθευσης της ορθότητας της προς υλοποίηση συνάρτησης.
- την πλατφόρμα FPGA στόχο.
- την πρότυπη συχνότητα ρολογιού.
- επισημάνσεις οδηγιών για τη διαδικασία υλοποίησης.

και η έξοδός της σύνθεσης είναι μια σχεδίαση επιπέδου μεταφοράς καταχωρητών μαζί με εκτιμήσεις απόδοσης και αξιοποίησης πόρων. Συγκεκριμένα περιλαμβάνει

- περιγραφή του υλικού σε κατάλληλη γλώσσα (λ.χ. VHDL).
- προσομοιώσεις δοκιμής σχεδίασης επιπέδου μεταφοράς καταχωρητών.
- στατική ανάλυση απόδοσης και χρήσης πόρων του συστήματος
- σύνολο δεδομένων που περιγράφουν τα όρια της σχεδίασης για ευκολία σύνδεσης με το υπόλοιπο σύστημα

Έχοντας αποσαφηνίσει τους όρους και προϋποθέσεις υπό τις οποίες γίνεται η χρήση του εργαλείου σύνθεσης υψηλού επιπέδου της Vivado, μένει να γίνει η περιγραφή των αναθεωρήσεων και ανασχεδιασμών της υλοποίησης του αλγορίθμου, ώστε να επιτευχθεί το επιθυμητό αποτέλεσμα.

1.5.1 Αναπαράσταση δεδομένων στη μνήμη

Αρχικά, σημειώνεται πως τα δεδομένα εισόδου που έχουν τη μορφή χωρισμένων από κόμματα αλφαριθμητικών (Πίνακας 1.1), κατά το τμήμα της ανάλυσης των δεδομένων υφίστανται μια επεξεργασία για τη μετατροπή τους σε byte-κωδικοποιημένες τιμές. Αυτό γίνεται με την υλοποίηση λεξικού για κάθε χαρακτηριστικό, ώστε κάθε διαφορετική τιμή να αντιστοιχιστεί σε έναν αύξοντα αριθμό ανά χαρακτηριστικό. Με τον τρόπο αυτό γίνεται εξοικονόμηση σε bits σε σχέση με την υλοποίηση καθολικού λεξικού, αφού κάθε χαρακτηριστικό έχει το δικό του λεξικό και έτσι οι αριθμήσεις διαφορετικών χαρακτηριστικών μπορεί να επικαλύπτονται, αλλά με τη γνώση της στήλης στην οποία βρίσκονται (την ταυτότητα του χαρακτηριστικού) εξασφαλίζεται η κωδικοποίησή τους χωρίς απώλεια πληροφορίας με όσο πιο συμπαγή τρόπο γίνεται σε ένα byte.

Η αναπαράσταση ενός ακέραιου αριθμού σε ένα byte είναι η μικρότερη δυνατή ενιαία οντότητα που συνιστά τύπο δεδομένων σε γλώσσες προγραμματισμού όπως η C, ενώ ταυτόχρονα προσφέρει την απαραίτητη ευελιξία τής με τον ίδιο τρόπο αναφοράς από CPU και FPGA. Η ευελιξία του έγκειται στο ότι αν και στο FPGA μπορούν να χρησιμοποιηθούν λιγότερα από 8-bits για την κωδικοποίηση χαρακτηριστικού με λιγότερες από 256 διακριτές τιμές, δεν επιφέρει την επιβράδυνση που θα συνεπαγόταν η προσαρμογή του μεγέθους της μεταβλητής κατά την επικοινωνία FPGA-CPU. Για τα δε χαρακτηριστικά συνεχών τιμών, η κωδικοποίησή τους αφορά την μετά το στάδιο της προεπεξεργασίας (§1.3.2) δυϊκή αναπαράστασή τους.

1.5.2 Δομές και Μεταφορά Δεδομένων

Τα δεδομένα εκπαίδευσης αρχικά αναπαρίστανται σε δισδιάστατο πίνακα με την byte-κωδικοποιημένη τους μορφή όπως αναλύθηκε παραπάνω. Ο ανασχεδιασμός της δισδιάστατης δομής αυτής προβλέπει την ευθυγράμμισή τους τόσο σε προγραμματιστικό επίπεδο με τη μετατροπή της διπλής αναφοράς σε απλή αναφορά με κάθε γραμμή του πίνακα να συνδέεται σειριακά με την προηγούμενή της (βλ. πίνακες 1.3 και 1.4) όσο και σε φυσικό επίπεδο, αφού είναι απαραίτητο σύμφωνα με το πρωτόκολλο μεταφοράς που κρίνεται το πιο αποδοτικό για την περίσταση τα δεδομένα να είναι αποθηκευμένα σε φυσικά συνεχή μνήμη. Αυτό επιτυγχάνεται με την κλήση της συνάρτησης `sds_alloc()` σε αντίθεση με τις υπό τη συνάρτηση `malloc()` κλήσεις συστήματος που εξασφαλίζουν εικονικά συνεχή μνήμη.

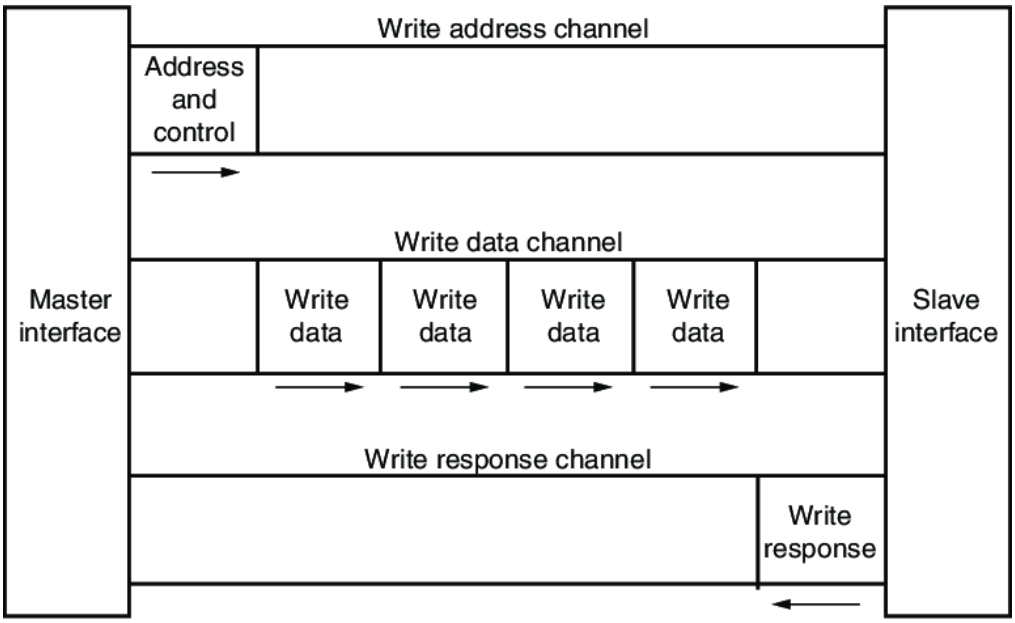
$(1,1)$	$(1,2)$	$(1,3)$
$(2,1)$	$(2,2)$	$(2,3)$
$(3,1)$	$(3,2)$	$(3,3)$

Πίνακας 1.3: Μητρική δομή 3×3 με επισήμανση δεικτών των στοιχείων.

$(1,1)$	$(1,2)$	$(1,3)$	$(2,1)$	$(2,2)$	$(2,3)$	$(3,1)$	$(3,2)$	$(3,3)$

Πίνακας 1.4: Ο προηγούμενος πίνακας έχοντας υποστεί ευθυγράμμιση.

Η μεταφορά των προσαρμοσμένων στις απαιτήσεις της σύνθεσης υψηλού επιπέδου δεδομένων γίνεται από είτε την κύρια ή την κρυφή μνήμη στο FPGA μέσω της διεπαφής προηγμένης επεκτάσιμης διασύνδεσης (AXI: Advanced eXtensible Interface), που συνιστά δίαυλο επικοινωνίας μεταξύ του μικροεπεξεργαστή και FPGA στην περίπτωση αυτή. Στην υλοποίηση της συνάρτησης αποτίμησης χαρακτηριστικού διαχωρισμού στο τμήμα δόμησης του δένδρου απόφασης, η ροή των δεδομένων είναι από τον επεξεργαστή και την επί-ψηφίδα μνήμη προς το FPGA, μόνο με την επιστρεφόμενη τιμή μήκους byte να ακολουθεί την αντίστροφη πορεία. Τα δεδομένα εκπαίδευσης που βρίσκονται αποθηκευμένα στη μνήμη αναφέρονται με τη διεύθυνση του πρώτου στοιχείου τους και κατευθύνονται προς το FPGA για επεξεργασία μέσω του ευέλικτου πρωτοκόλλου τέταρτης γενιάς AXI4 (βλ. Σχήμα 1.5), ενώ οι μεταβλητές με αναφορά σε τιμή μεταφέρονται μέσω του αντίστοιχου πρωτοκόλλου για μεταφορά λέξης σημείου-προς-σημείο AXI4-Lite.



Σχήμα 1.5: Διάγραμμα αρχιτεκτονικής πρωτοκόλλου AXI4 για την εγγραφή δεδομένων από μία διεπαφή αφέντη σε μία διεπαφή σκλάβο. (Πηγή: [5])

Έχοντας ήδη αναφερθεί στο πρωτόκολλο επικοινωνίας μέσω του οποίου μεταφέρονται τα δεδομένα, ακολούθως παρουσιάζονται οι ντιρεκτίβες `#pragma` τύπου SDS που άπτονται άμεσα της μελετώμενης εφαρμογής και μέσω των οποίων ο σχεδιαστής μπορεί να καθορίσει τις υπόλοιπες λεπτομέρειες της υλοποίησης της επικοινωνίας των δεδομένων από CPU σε FPGA, τα οποία δεδομένα εμφανίζονται σαν ορίσματα της συνάρτησης σε γλώσσα προγραμματισμού C/C++, και βάσει των ονομάτων των μεταβλητών-ορισμάτων συγκεκριμενοποιούνται.

- `access_pattern`: μέσω της οδηγίας αυτής καθορίζεται το μοτίβο προσπέλασης των δεδομένων αποθηκευμένων σε διανυσματική δομή και αναφερόμενων μέσω της διεύθυνσης του πρώτου στοιχείου τους. Αυτή μπορεί να είναι σειριακή, οπότε χρησιμοποιείται διεπαφή τύπου ουράς ή τυχαία, οπότε δημιουργείται διεπαφή τύπου μνήμης τυχαίας προ-

σπέλασης.

- `copy`: μέσω της οδηγίας αυτής διευκρινίζεται ρητά ότι τα δεδομένα αντιγράφονται από τη μνήμη του επεξεργαστή στην συνάρτηση υλικού.

- `data_mover`: με την οδηγία αυτή διευκρινίζεται η νουθεσία του σχεδιαστή προς τον εργαλείο σύνθεσης για τη δομή μέσω της οποίας υλοποιείται ο μεταφορέας δεδομένων· με χρήση ουράς, λίστας τύπου διασποράς-συλλογής ή απλής μεταφοράς.

- `mem_attribute`: μέσω της οδηγίας που χαρακτηρίζεται από αυτή τη λέξη κλειδί, διευκρινίζεται η φυσική ή μη συνέχεια της μνήμης στην οποία είναι αποθηκευμένη η προσδιοριζόμενη δομή δεδομένων. Σε περίπτωση μη διευκρίνισης της οδηγίας αυτής το εργαλείο SDSoC επιχειρεί να αναγνωρίσει από μόνο του τον τύπο συνέχειας της μνήμης.

- `sys_port`: με την οδηγία αυτή διευκρινίζεται ο τύπος της θύρας από την οποία διεκπεραιώνεται η επικοινωνία (λ.χ. θύρες υψηλής απόδοσης, για μη συνεκτικές αναφορές στην κρυφή μνήμη).

- `zero_copy`: με την οδηγία αυτή επισημαίνεται ότι η συνάρτηση υλικού έχει άμεση πρόσβαση στα δεδομένα που βρίσκονται σε κοινοποιημένη μνήμη, μέσω διεπαφής τύπου AXI.

Για την μελετώμενη εφαρμογή λοιπόν, ισχύει ότι τα δεδομένα δεν προσπελάζονται σειριακά σε σχέση με τον ευθυγραμμισμένο πίνακα, κάτι που οδηγεί σε τροποποίηση της σχεδίασης αναφορικά με τα δεδομένα που μεταδίδονται μέσω του AXI διαύλου διεπαφής. Πιο συγκεκριμένα, κάθε στήλη χαρακτηριστικού υπόκειται σε εφάπαξ επεξεργασία με σκοπό την καταμέτρηση των συχνοτήτων εμφάνισης των τιμών των χαρακτηριστικών, οι οποίες αποθηκεύονται ακολούθως σε BRAM για τους επακόλουθους υπολογισμούς, ενώ την ίδια στιγμή, η στήλη που αντιστοιχεί στις κλάσεις των μελετώμενων περιπτώσεων προσπελάζεται τόσες φορές, όσο είναι και το πλήθος των χαρακτηριστικών. Για τον λόγο αυτό, κρίνεται κατάλληλη η θεώρηση μιας νέας αναφοράς στην τελευταία στήλη που αφορά τις κλάσεις του συνόλου των δεδομένων, ώστε να μεταφέρεται ξεχωριστά από τις τιμές των υπολοίπων χαρακτηριστικών, αίροντας με τον τρόπο αυτό την παραβίαση της σειριακής προσπέλασης των επιμέρους δομών· επιδιώκεται σειριοποίηση των αναφορών ώστε να αξιοποιηθεί ο βέλτιστος μεταφορέας δεδομένων που έγκειται στην σειριακή προσπέλαση των επισημαινόμενων δομών.

Η ξεχωριστή αλλά ταυτόχρονη μεταφορά των δεδομένων βελτιώνει την του υπό υλοποίηση συστήματος διεκπεραιωτικότητα και επιφέρει καλύτερη αξιοποίηση του εύρους ζώνης μεταφοράς των δεδομένων καθώς η επικοινωνία των εναπομεινάντων δεδομένων είναι τακτική και η βελτιστοποίησή της καίριας σημασίας.

1.5.3 Παράλληλοι Υπολογισμοί

Στο μέρος αυτό μελετώνται βελτιστοποιήσεις άλλες από της σχετικές με την επικοινωνία επεξεργαστή-προγραμματιζόμενης λογικής. Πιο συγκεκριμένα μελετάται το σώμα του κώδικα

της συνάρτησης που οδεύει προς υλοποίηση σε υλικό και διερευνώνται οδηγίες προς το εργαλείο σύνθεσης υψηλού επιπέδου για την καλύτερη αξιοποίηση των πόρων του συστήματος και μέγιστη δυνατή παραλληλοποίηση για τους σκοπούς της αποτίμησης του χαρακτηριστικού διαχωρισμού στο τμήμα της εκπαίδευσης του δένδρου απόφασης. Ακολούθως παρουσιάζονται οι ντιρεκτίβες `#pragma` τύπου HLS που παρουσιάζουν ενδιαφέρον για την παρούσα εφαρμογή και μέσω των οποίων ο σχεδιαστής μπορεί να καθορίσει τις υπόλοιπες λεπτομέρειες της υλοποίησης με σκοπό τον βέλτιστο τρόπο διεκπεραίωσης των υπολογισμών της συνάρτησης υλικού. Λεπτομερέστερος τρόπος εφαρμογής των οδηγιών αυτών αναφέρεται παρακάτω, με τις προσαρμογές τους για τους σκοπούς της εφαρμογής.

- `array_partition`: με την οδηγία αυτή προκαλείται τμηματοποίηση μίας διανυσματικής δομής σε επιμέρους μικρότερα διανύσματα ή στοιχεία προσφέροντας περισσότερες και μικρότερες μνήμες, αυξάνοντας τις θύρες ανάγνωσης/εγγραφής, ανοίγοντας την πόρτα για παράλληλη επεξεργασία των περιεχόμενων δεδομένων.

- `loop_tripcount`: οι επαναληπτικοί βρόχοι που υλοποιούνται σε υλικό απαιτούν ως ένα βαθμό τη γνώση του αριθμού των επαναλήψεων· με την οδηγία αυτή μπορούν να επισημανθούν άνω και κάτω όρια πλήθους αυτών.

- `pipeline`: με την οδηγία αυτή επιτρέπεται σε λειτουργίες μέσα στο σώμα ενός επαναληπτικού βρόχου να εκτελούνται ταυτόχρονα με τη μέθοδο της σωλήνωσης και χρήσης πόρων με τον καλύτερο δυνατό τρόπο.

- `resource`: η οδηγία αυτή δίνει σωρεία δυνατοτήτων όσον αφορά την υλοποίηση μεταβλητών στο επίπεδο μεταφοράς καταχωρητών (λ.χ. BRAM μίας ή δύο θυρών εγγραφής/ανάγνωσης).

- `unroll`: η οδηγία αυτή επισημαίνει έναν επαναληπτικό βρόχο για ξεδίπλωμα σε ξεχωριστές οντότητες επεξεργασίας αντί σε μία συλλογή λειτουργιών. Με τον τρόπο αυτό τα ξεχωριστά υποσύνολα των επαναλήψεων εκτελούνται παράλληλα.

Επιστρέφοντας πίσω στην εφαρμογή των μεθόδων σύνθεσης υψηλού επιπέδου στην επιλεγμένη συνάρτηση για υλοποίηση σε υλικό, λαμβάνοντας υπόψιν της δυνατότητες που παρέχονται από το εργαλείο αυτό, παρουσιάζεται στη συνέχεια ένα σύνολο από τέτοιες επισημάνσεις (ντιρεκτίβες). Σε πρώτο στάδιο όμως, παρουσιάζεται ένα στιγμιότυπο (βλ. πίνακα 1.5) της αποτίμησης των συχνοτήτων των τιμών των χαρακτηριστικών κατ΄ αντιστοιχία με τις κλάσεις στις οποίες ανήκουν οι περιπτώσεις που περιγράφουν.

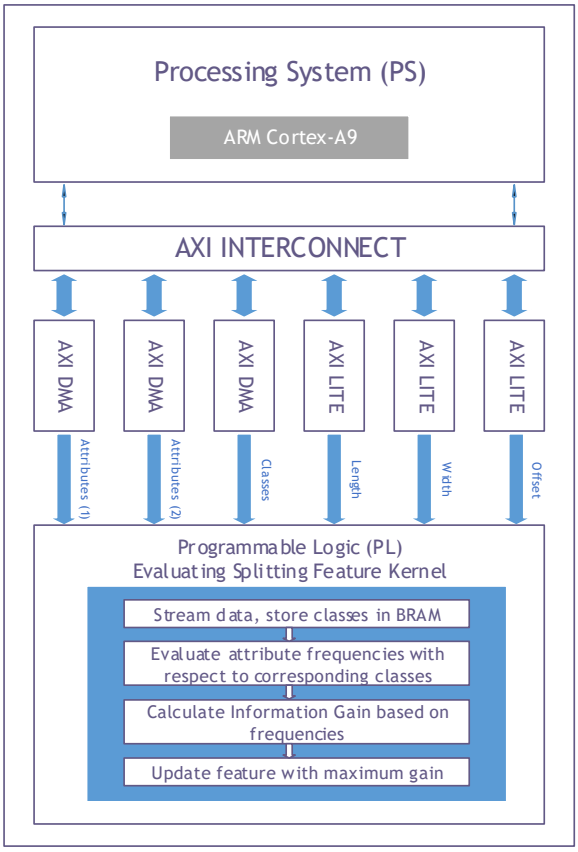
working class	**Income Classes**	
attributes	$\leq 50K$	$> 50K$
federal-gov	449	276
local-gov	1108	445
never-worked	5	0
private	12947	3569
self-emp-inc	363	454
self-emp-not-inc	1348	528
state-gov	683	256
without-pay	8	0

Πίνακας 1.5: Στον πίνακα αυτόν απεικονίζονται οι συχνότητες των τιμών ενός χαρακτηριστικού (τύπος εργασίας) ανά μισθολογική κλάση (σ.δ. Adult [14]) στην οποία αντιστοιχεί η περίπτωση που ανήκουν. (Πηγή: [21])

Ένας πίνακας συχνοτήτων σαν τον παραπάνω σχηματίζεται από μία μόνο στήλη τιμών ενός χαρακτηριστικού μαζί, φυσικά, με τη στήλη των κλάσεων και ένα τέτοιος δομείται για κάθε επανάληψη του εξώτερου επαναληπτικού βρόχου που συνιστά το σώμα της συνάρτησης υπολογισμού του χαρακτηριστικού διαχωρισμού. Μία μόνο επανάληψη ως προς τις τιμές του μελετώμενου χαρακτηριστικού, οι οποίες μεταδίδονται μέσω της AXI διεπαφής, και υπόκεινται σε σειριακή επεξεργασία, αρκεί για την αποτίμηση και αποθήκευση των συχνοτήτων αυτών σε μνήμη υλοποιημένη ως BRAM με την επισήμανση δύο θυρών. Η αξιοποίηση της δίθυρης BRAM γίνεται για τον λόγο ότι δίνει τη δυνατότητα μέτρησης συχνοτήτων δύο τιμών ταυτόχρονα, εξ ου και η αξιοποίηση δύο διαφορετικών ορισμάτων (βλ. Σχήμα 1.6) που συνιστούν αναφορές σε διαφορετικά σημεία του διανύσματος που φέρει τις τιμές ενός χαρακτηριστικού, μετρήσεις οι οποίες διεκπεραιώνονται ξεχωριστά χάριν των διαφορετικών αναφορών, διαδόσεων και συνάμα της δυνατότητας παράλληλης επεξεργασίας από το FPGA. Αυτή η αξιοποίηση επιτυγχάνεται με την επισήμανση της ντιρεκτίβας τύπου `unroll` και με παράγοντα δύο, σηματοδοτώντας τον διαχωρισμό του βρόχου σε δύο ανεξάρτητους βρόχους.

Επιπρόσθετα, επιχειρείται μία ακόμη βελτιστοποίηση για τον βρόχο που αφορά την προσπέλαση των τιμών των χαρακτηριστικών με σκοπό την αποτίμηση των συχνοτήτων. Αυτή επιτυγχάνεται με την επισήμανση της ντιρεκτίβας τύπου `pipeline` που επιφέρει τη σύγχρονη λειτουργία και εκτέλεση υπολογισμών από το σώμα του βρόχου και τούτο μέσω παράταξης των επαναλήψεων του βρόχου σε χρονικά επικαλυπτόμενα διαστήματα, οδηγώντας το διάστημα ενσωμάτωσης του βρόχου στο ελάχιστο δυνατό, διάστημα που προσδιορίζεται από το εργαλείο SDSoC κατά τη διάρκεια της σύνθεσης.

Το επόμενο βήμα εξεργασίας των δεδομένων λαμβάνει χώρα στον επεξεργαστικό πυρήνα του FPGA αφορά στην αξιοποίηση των με τον τρόπο που περιγράφηκε νωρίτερα αποτιμημένων συχνοτήτων για τους σκοπούς του υπολογισμού της του κέρδους πληροφορίας από τα υπό εξέταση δεδομένα. Ο υπολογισμός του κέρδους πληροφορίας απαιτεί υπολογισμούς συσσω-



Σχήμα 1.6: Παραπάνω παρουσιάζεται το απλοποιημένο διάγραμμα τμημάτων της ετερογενούς αρχιτεκτονικής που περιλαμβάνει συνδυασμό FPGA και CPU, προβάλλοντας δε την μεταξύ τους επικοινωνία. (Πηγή: [21])

ρευτικών αθροισμάτων σχετικά με τον υπολογισμό της εντροπίας πληροφορίας. Το τμήμα αυτό του αλγορίθμου υλοποιείται ως σειρά από εμφωλευμένους επαναληπτικούς βρόχους, των οποίων οι εσώτεροι, που αφορούν στην προσπέλαση των κλάσεων του συνόλου δεδομένων, υφίστανται τις προσαρμογές της επισήμανσης της σωλήνωσης, ενώ οι εξώτεροι τις υφίστανται στις επαναλήψεις που αφορούν στην προσπέλαση των τιμών των υπό εξέταση χαρακτηριστικών. Ο δεύτερος επισημαίνεται επιπλέον με ξεδίπλωμα βαθμού τέτοιου, ώστε να σέβεται τους περιορισμούς πόρων του συστήματος, αλλά ταυτόχρονα και να επιτρέπει τον στατιστικά μεγαλύτερο αριθμό αναγνώσεων των στοιχείων του πίνακα και την καλύτερη αξιοποίηση του δυνητικού εύρους ζώνης της μνήμης, που υλοποιεί τον πίνακα αυτό σε τμήματα χάρη σε σχετική ντιρεκτίβα τύπου `resource`, καθώς ανεξάρτητες πτυχές του βρόχου αναφέρονται στα ίδια δεδομένα. Ταυτοχρόνως, η τελευταία στήλη των δεδομένων εκπαίδευσης που συνιστά τη στήλη κλάσεων των εκπαιδευτικών περιπτώσεων, μεταδίδεται και αποθηκεύεται στην συσκευή του FPGA σε στοιχεία μνήμης τύπου δίθυρης BRAM, που δίδει τη δυνατότητα στον κατά δύο βαθμούς ξεδιπλωμένο βρόχο να προσπελάσει με τα σώματα βρόχου που προκύπτουν το ίδιο διάνυσμα των κλάσεων εξασφαλίζοντας ταυτόχρονα με τον τρόπο αυτό, την μεταξύ τους ανεξαρτησία.

Όσον δε αφορά το συσσωρευτικό άθροισμα, η εφαρμογή της σχέσης (1.1) σύμφωνα με

την οποία υπολογίζεται το κέρδος πληροφορίας της εκάστοτε στήλης (σύνολο τιμών χαρακτηριστικού) και για τον οποίο υπολογισμό μία σειρά από επιπρόσθετες προσαρμογές και ντιρεκτίβες επισημαίνονται, ώστε να διατίθενται οι κατάλληλοι πόροι για τις λοιπές λειτουργίες της συνάρτησης που περιλαμβάνουν προσθέσεις, πολλαπλασιασμούς καθώς και διαιρέσεις αριθμών κινητής υποδιαστολής. Αναφορικά με τους υπολογισμούς του κέρδους πληροφορίας, αλλά και των ενδιαμέσων μεταβλητών που συνεισφέρουν στον υπολογισμό του, το μήκος της μεταβλητής κινητής υποδιαστολής διαλέγεται σταθερό και ίσο με 32-bit, με ασήμαντες τελικές απώλειες ακρίβειας πρόβλεψης.

Συγκεκριμένα, οι πράξεις πρόσθεσης για λογαριασμό του συσσωρευτικού αθροίσματος υλοποιούνται σε αθροιστές κινητής υποδιαστολής τύπου DSP48, ενώ για τους πολλαπλασιασμούς των κλασμάτων της σχέση (1.2) με τις αντίστοιχες λογαριθμικές ποσότητες αξιοποιούνται παρομοίου των αθροιστών τύπου πολλαπλασιαστές κινητής υποδιαστολής, ενώ τέλος οι διαιρέσεις για τον υπολογισμό των κλασμάτων (1.2), αυτές λαμβάνουν χώρα σε διαιρέτες κινητής υποδιαστολής.

Το απλοποιημένο σχεδιάγραμμα τμημάτων που απεικονίζεται στο Σχήμα 1.6 αναδεικνύει την αρχιτεκτονική του συστήματος: περιλαμβάνει ένα σύστημα μικροεπεξεργαστή που αναλαμβάνει τον υπολογισμό των υπολοίπων εκ των ανατεθέντων στο FPGA υπολογισμών και διεργασιών, με το δεύτερο να αναλαμβάνει την αποτίμηση του χαρακτηριστικού διαχωρισμού βάσει του υπολογισμού του κέρδους πληροφορίας του. Τα δεδομένα που βρίσκονται σε διανυσματική μορφή μεταφέρονται μέσω άμεσης προσπέλασης μνήμης τύπου AXI, ενώ οι υπόλοιπες μεταβλητές μεταφέρονται μέσω της διεπαφής τύπου AXI4.

1.6 Αποτίμηση Αποτελεσμάτων

Οι μελέτες περιπτώσεων για την αποτίμηση της απόδοσης του συστήματος τόσο για το τμήμα εκπαίδευσης όσο και για το τμήμα δοκιμής του ταξινομητή δένδρου απόφασης, αφορούν στα σύνολα δεδομένων 'Adult' και 'Census-Income', πολυμεταβλητά σύνολα δεδομένων αντληθέντα από την αποθήκη μηχανικής μάθησης του UCI. Τα χαρακτηριστικά και των δύο συνόλων δεδομένων, περιλαμβάνουν τόσο κατηγορικά δεδομένα όσο και ακεραίους, με το δε τελευταίο να περιέχει και τιμές κινητής υποδιαστολής. Μία ποσοτική ανάλυση των δεδομένων αυτών δείχνει ότι το πρώτο σύνολο περιλαμβάνει 48842 περιπτώσεις συνολικά, 14 χαρακτηριστικών, 8 εκ των οποίων κατηγορικά και 6 συνεχών τιμών, στις οποίες εφαρμόζεται η μέθοδος διακριτοποίησης, όπως γίνεται και στα 7 χαρακτηριστικά συνεχών τιμών από τα συνολικά 40 χαρακτηριστικά στο δεύτερο σύνολο δεδομένων που περιλαμβάνει 199523 εκπαιδευτικές περιπτώσεις και 99762 δοκιμαστικές περιπτώσεις, με τα υπόλοιπα 33 να είναι κατηγορικά χαρακτηριστικά.

Ένας από τους σημαντικότερους παράγοντες που συνιστά τροχοπέδη για την επίτευξη ικανοποιητικής επιτάχυνσης της απόδοσης του συστήματος και το κέρδος από την υλοποίησης της συνάρτησης σε υλικό, είναι το κόστος της επικοινωνίας μεταξύ του επεξεργαστικού συστήματος (PS: Processing System) και προγραμματιζόμενης λογικής (PL: Programmable Logic). Υπό αυτές τις συνθήκες, στην παρούσα εργασία χρειάστηκε να συγκριθούν

μέθοδοι επικοινωνίας και να εξαχθεί ως συμπέρασμα η βέλτιστη υπό τους περιορισμούς του συστήματος μέθοδος επικοινωνίας PS-PL. Όπως αναφέρθηκε νωρίτερα, οι τιμές των κλάσεων προσπελάζονται περιοδικά, ανακόλουθα με τη σειριακή προσπέλαση των τιμών των υπόλοιπων χαρακτηριστικών. Το γεγονός αυτό οδήγησε στην εξέταση της μεταφοράς των δεδομένων με τη μέθοδο `zero_copy` είτε μόνο για τις κλάσεις ή και συνολικά για όλα τα δεδομένα, μία υλοποίηση που αποδείχθηκε να εντείνει το κόστος επικοινωνίας έτι περαιτέρω έναντι της υλοποίησης που οδήγησε στην τελική βελτίωση της απόδοσης.

Επιπρόσθετα, αξίζει να σημειωθεί πως οι περιορισμοί που επιβάλλονται μοιραία εξ αιτίας της απόφασης για υλοποίηση του αλγορίθμου στην υβριδική αυτή αρχιτεκτονική του συστήματος, που περιλαμβάνουν περισσότερες στατικές και εκ των προτέρων διατεθειμένες δομές δεδομένων, δρουν ενισχυτικά σε μια συνολική μείωση του χρόνου εκτέλεσης του τμήματος εκπαίδευσης του αλγορίθμου, ακόμα και σε συναρτήσεις που τρέχουν έξω από το FPGA, αλλά μοιράζονται τη φιλοσοφία της υλοποίησης, ίσως για ένα μικρό κόστος ευελιξίας.

Οι μετρήσεις που παρουσιάζονται ακολούθως στον πίνακα 1.6, αφορούν την εκτέλεση του αλγορίθμου για τα δύο προαναφερθέντα σύνολα δεδομένων, σε συχνότητα λειτουργίας και μεταφοράς δεδομένες ίδιες και ίσες με 142.86 MHz, με την χρήση των πόρων του συστήματος για τις συχνότητες αυτές να φαίνονται στον πίνακα 1.7. Προκειμένου να επιτευχθούν τα παρουσιαζόμενα αποτελέσματα, μία σειρά από παραμέτρους ήταν αντικείμενα πειραματισμού, όπως λ.χ. οι παράγοντες τμηματοποίησης πινάκων, οι παράγοντες ξεδιπλώματος επαναληπτικών βρόχων και πόροι στους οποίους λαμβάνουν χώρα οι διάφορες αριθμητικές πράξεις.

Ο κώδικας της μελετώμενης εφαρμογής σε C/C++ γλώσσα προγραμματισμού μαζί με της ντιρεκτίβες τύπου SDS και HLS, που οδήγησαν στην τελική αρχιτεκτονική του συστήματος, μεταγλωττίστηκαν σε ένα εκτελέσιμο αρχείο για τον μικροεπεξεργαστή (ARM) και ένα αρχείο μετάδοσης bit (bit-stream), που απαιτείται για τον προγραμματισμό του FPGA, μέσω της αναπτυξιακής πλατφόρμας SDSoC και επέτυχε τα ακόλουθα αποτελέσματα στο σύστημα της πλακέτας ZedBoard (Zynq-7000 All Programmable SoCs) με χρήση των πόρων όπως φαίνεται στον πίνακα 1.6.

Resource	**Hardware Function Resources**		
Name	***Used***	***Total***	% ***Utilization***
DSP	48	220	21.82
BRAM	75	140	53.57
LUT	35807	53200	67.31
FF	47608	106400	44.74

Πίνακας 1.6: Χρήση πόρων συστήματος ZedBoard για την αποτίμηση του χαρακτηριστικού διαχωρισμού. (Πηγή: [21])

Ενώ αναφορικά με τους χρόνους και τις ακρίβειες ταξινόμησης για τα δύο αναφερθέντα σύνολα δεδομένων αντίστοιχα τα αποτελέσματα παρουσιάζονται στον πίνακα 1.7.

Data set Name	**Implementation**			
	Software	***Hybrid***	***Speed-up***	***Accuracy***
Adult	2.33s	0.94s	2.48	82.54%
Census-Income	34.27s	15.5s	2.21	94.26%

Πίνακας 1.7: Σύγκριση αποτελεσμάτων χρόνου και ακρίβειας. (Πηγή: [21])

1.7 Συμπεράσματα

Στην εργασία αυτή, παρουσιάζεται μια εφαρμογή που προβάλλει την επιτάχυνση της απόδοσης μίας συνάρτησης, εν γένει σε γλώσσα προγραμματισμού C, που κατόπιν σκιαγράφησης της κατανομής του υπολογιστικού φόρτου του αλγορίθμου μάθησης δένδρου απόφασης, επιλέχθηκε να είναι για την παρούσα εφαρμογή, αυτή που αφορά την επιλογή του χαρακτηριστικού διαχωρισμού. Κατά το στάδιο επιλογής συνάρτησης για υλοποίηση σε υλικό, συναρτήσεις που θα προκαλούσαν επικοινωνιακή συμφόρηση στις αντίστοιχες διεπαφές και συνεπακόλουθο κόστος χρόνου, αφέθηκαν να εκτελούνται στην κύρια μονάδα επεξεργασίας, ενώ το τμήμα από το οποίο η υλοποίηση κρίθηκε ότι θα ωφεληθεί, σχεδιάστηκε σαν συνάρτηση υλικού.

Ενώ μία προσέγγιση για επιτάχυνση του τμήματος ταξινόμησης/δοκιμής ενδεχομένως να παρουσίαζε εντυπωσιακά αποτελέσματα, δύο κύριοι λόγοι επικέντρωσαν το ενδιαφέρον αυτής της εφαρμογής στο τμήμα της εκπαίδευσης του δένδρου απόφασης. Η εσωτερική απλότητα της υλοποίησης που αφορά στην ταξινόμηση βάσει ενός μοναδικού δένδρου, σε συνδυασμό με τον όγκο της έρευνας που διεξάγεται για τη βελτιστοποίησης της διαδικασίας ταξινόμησης βάσει συνόλου από δένδρα απόφασης, συνεισέφεραν στην επιλογή για επιχείρηση βελτιστοποίησης του τμήματος εκπαίδευσης.

Η επιτάχυνση εφαρμογών χρησιμοποιώντας σύγχρονες τεχνολογίες υλικού αποκτά όλο και περισσότερο ενδιαφέρον και παρουσιάζεται σαν ευκαιρία για επιτάχυνση της απόδοσης πλήθους από αλγορίθμους μηχανικής μάθησης, μεταξύ των οποίων τα δένδρα απόφασης και σύνολα αυτών, παρουσιάζουν μεγάλη δυναμική.

Chapter 2

Introduction

2.1 Introduction

The field of machine learning focuses on the study of mechanisms through which, intelligent systems improve their performance over time. Research on this field of computer science includes the development of applications, the majority of which involve some kind of method of knowledge modeling, as alternatively a human expert would attempt to tackle a similar issue. In some cases, the task requires classification of objects or instances in general; assigning them to categories based on certain features of those. Examples include cases of applications such as transaction approval, where features like the subject's credit history as well as other details related to the under examination transaction are taken into account. Another application would regard a system assisting in capital expenditure decisions, just as numerous other decision-regarding issues. In this case, every under examination proposal of expenditure is classified as accordant or not with a corporation's policies. Finally, a common set of interrelated case studies regard medical diagnoses; a machine learning system (such as a decision tree) is trained based on previous experience and patient history about features possibly leading to a disease and classifies new (testing) instances based on the very same features.

Machine learning algorithms and consequently data mining approaches such as Decision tree learning have been increasingly gaining popularity among fields such as medical diagnoses, risk evaluation of credit card application approval, consumers' behaviors, etc. Such a broad spectrum of fields where decision tree learning and classification are applicable along with the growth of available information to a massive extent in the era of big data make it desired to seek ways of optimizing a computationally wise heavy algorithm that can make use of this information and which alone is widely utilized and forms a single fraction of ensemble learning methods such as the random forests, etc. There, the benefits of an optimization in the former algorithm would be highlighted even further as the speedup is expected to be boosted by a significant degree.

Hardware engineers, from their perspective, make an effort to seek ways of making hardware acceleration methods accessible for a software designs of the algorithm that are

abstracted from the hardware they are going to run on, only to the highest degree it is still possible. This software approach of hardware acceleration involving Field Programmable Gate Arrays (FPGA)-based heterogeneous system is addressed in the presented work.

Such approaches are intended in order to improve the energy efficiency of broadly used machine learning algorithms' implementations. Modern day sophisticated processors that would be alternatively used for that purpose, though fast in high clock frequencies yield high energy consumption, escalating while more and more cores are utilized in parallel implementations. Recently, FPGAs are increasingly getting popular since they offer the option of highly parallelized implementations, where operations and code patterns are executed simultaneously in versatile computational units suiting only the necessary functions utilizing only as many elements such as logic gates required by the specific computations, in contrast to all alternative hardware, that have a fixed architecture requiring the integration of all of its components to operate.

2.2 Contributions

The main contributions of the project include:

- the presentation of a novel hardware architecture for acceleration of decision tree learning algorithms.
- a performance evaluation and comparison with Central Processing Unit (CPU) implementations.
- the possibility for up to approximately 2.5× speedup and lower energy consumption compared to typical CPU implementations.

Chapter 3

Related work

3.1 Approaches on Decision Tree Optimization

A lot of research has been carried out on how either decision tree classifiers or their ensembles such as random forests, other ensemble methods such as the so-called Bagging (Bootstrap Aggregation) or Boosting are to be implemented in some sort of computing system involving FPGAs. The fact that a decision tree constitutes a unit of the structures built by any of these methods indicates the importance of studying the optimization of a decision tree classifier. Any possible benefit obtained is transferable to an implementation involving multiple trees possibly over a cluster of FPGA computing nodes, where scaling the amount of nodes utilized causes increased performance gain; such an approach is made by [15], where software driven FPGA implementation of a decision tree ensemble classifier was proposed and the scalability over tree structures requiring more than the FPGA provided memory was examined. The classification part's performance was optimized in an implementation that offered the capability to deal with any size or number of decision trees determining whether FPGA alone execution or in combination with CPU is required to handle the size of the structure.

In [16] a different approach on performance optimization was presented, where decision tree ensembles of a determined amount of nodes were stored in an appropriate amount of FPGAs required for them to fit in the on-chip memory and communication protocols regrading their optimal inference were examined. Ensembles of shallow decision trees up to depth of 9 fit into a single FPGA, but trees of increased depth would require doubling of FPGA nodes utilized per depth increase of 1. Such are the trade-offs when attempting to optimize the decision tree ensemble classifier that requires the storage of the classifying structure into FPGA. In our work instead, FPGA is used to perform the computationally intensive training part without the need for the whole data set or decision tree to even fit into the usually restricted on-chip memory.

Another work as shown in [17], proposed an acceleration method of a decision tree classifier by means of exploiting the comparative ease with which FPGA performs calculations with integers over floating-point values. In their proposal the feature label should

require the minimum amount of bits to enumerate all labels and the threshold regarding the continuous values shall be stored in 32 bit floating-point variable as any less bits would compromise predictive accuracy. In our work, on the other hand, the optimal way to treat floating-point features would be to evaluate the threshold of every continuous attribute as a C/C++ floating point type at parse time by the CPU and store the testing data as byte-encoded values. This way, having the FPGA to processes any such floating-point types is circumvented, saving data streaming time along with floating-point comparison time, as a common processor might do this very job faster; byte is deemed the best data type that is the most flexible for both CPU and FPGA to deal with at the same time, avoiding redundant type conversions.

Among the several aspects as to how studies have sought to tackle the challenge of optimizing decision tree learning in hardware, comes the work of [18]. In it, the Hoeffding tree algorithm is studied and its hardware implementation is optimized through the fine tuning of parameters such as the number of quantiles required for the attribute learning, which despite having an impact on the resulting accuracy, helps achieve a significant speedup.

An additional proposal could be found at [19], where four different architectures were designed as an answer to the demand for decision tree classifying performance optimization, all of which were shown to out-speed equivalent software-only implementations. Each unit of the ensemble was implemented in different module in two possible ways; pipelined and sequential decision tree evaluation, for which modules speedups varied from 9.56 to 5188.74 times.

The approach that was closer to the current one was the one at [20]. Similarly a hybrid architecture was proposed, where the Gini Score computational weight was let for the FPGA to process, whereas the rest of the decision tree classifying algorithm was running on CPU, yielding again a speedup compared to a software-only implementation. The main differences between it and our work are on a theoretical level the metrics used for the decision tree building where we used the information gain instead and on a more technical level, the implementation and architecture design. The results were presented for an implementation that treated only binary classes, while indicating the extension to support more. Additionally, the binary nature of the classes was exploited to expand the Gini Score formula into a predetermined amount of sums and divisions, whereas in this work a more flexible approach is made. This by allowing the evaluation of the formula to be carried out as a for loop, making use of the high-level synthesis capability to transform C-style loops, as well as the rest of the hardware function code, into hardware seamlessly, without loss in accuracy compared to when the implementation runs on CPU only.

Chapter 4

Background

4.1 Decision Trees

Machine learning algorithms such as decision tree learning are applied as models to numerous cases that require classification and prediction. Specifically a tree structure is formed in order to represent the input data in a more compact way of if-then rules, the so called classification rules.

Below, (see Table 4.1) a data-set for the training of a decision tree is presented. Each training instance includes a set of features that take part in its classification into a target class. The training procedure that is further explained later, leads to the building of a tree structure which then matches each instance of the testing data to a class (classification).

Outlook	Temp ($^\circ F$)	Humidity (%)	Windy	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

Table 4.1: Representation of data as set of training instances. (Source: [1])

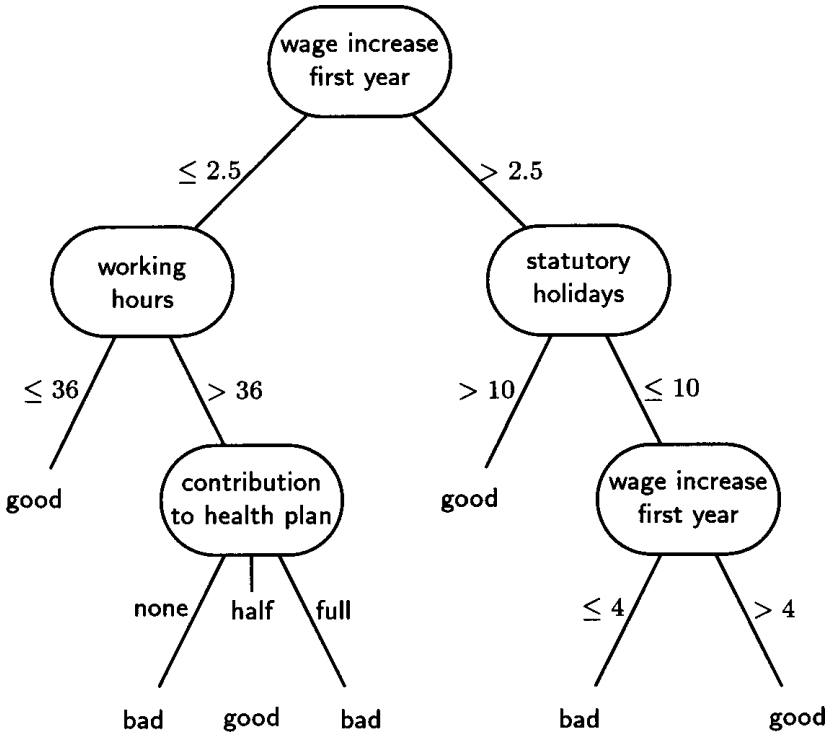
Every node in the tree defines a control condition of a given attribute of the instances and every branch that departs from that node corresponds to a different discrete value of the specific attribute. In every node, the case's value for the node's attribute is examined and the branch to follow is hence determined. The scheme following is an interpretation of the decision tree following it in Fig. 4.1, in the form of if-then rules.

The procedure during which the tree is build is called decision tree training. According to the C4.5 decision tree learning algorithm and similarly with the Iterative Dichotomiser 3 (ID3) [2] algorithm, a tree is built using the metric of information entropy. It works in a top-down manner recursively building the tree by finding the feature of the input data that will be used as splitting criteria at each point. An extension provided by the C4.5 algorithm is the capability to deal with continuous features, discretizing them in two categories below or equal and greater than a threshold.

Algorithm 1 Decision Tree 4.1 as if-then classifying rules

```
if wage increase first year ≤2.5 then

  if working hours ≤36 then
    class good
  else if working hours >36 then

    if contribution to health plan is none then
      class bad
    else if contribution to health plan is half then
      class good
    else if contribution to health plan is full then
      class bad
    end if
  end if
else if wage increase first year >2.5 then

  if statutory holidays >10 then
    class good
  else if statutory holidays ≤10 then

    if wage increase first year ≤4 then
      class bad
    else if wage increase first year >4 then
      class good
    end if
  end if
end if
```

Figure 4.1: A structure representing a tree which analyzes the various features that characterize the evaluation of labor conditions, is presented, where the binary class represents that evaluation. (Source: [1])



Additionally the C4.5 version of the algorithm, includes a procedure called pruning of decision tree as either post- or pre-pruning (the presented implementation applies post-pruning), replacing sub-trees with leaves corresponding to their most frequent values, in order to limit the over-fitting, an attribute known to characterize single decision trees. In addition, this version of the algorithm has a method for handling missing values by returning the most frequent class if the missing value were to be replaced with each possible value of the respective feature, but disregarding them in the training part.

4.2 Training Data Discretization

Depending on the training data set and whether it contains continuous features, a prepossessing for the binary discretization of the latter is applied; subsequently, the feature is treated as categorical with each instance evaluated as exceeding or subceeding an appropriately elected value. The threshold that divides the continuous feature (assuming it to be a real number) into two classes of the lower and higher values is calculated as follows. Let $S = \{a_1, a_2, ..., a_n\}$ be the training values of a continuous attribute in a sorted order and m the number of classes $\{C_1, C_2, ..., C_m\}$ of the data set. Out of the $n - 1$ possible splits evaluated, the one that minimizes the information metric is selected. Using the sorted order of the values, the maximum information conveyed is found in a single

pass. The information metric is calculated for each split a_i as:

$$G(S) = -p_h \cdot E(S_h) - p_l \cdot E(S_l)$$

where

$$p_{h,l} = \frac{|S_{>a_i, \leqslant a_i}|}{|S|}$$

are the fractions of the values greater and lower than or equal to the split, respectively and

$$E(S) = -\sum_{i=1}^{m} p_i \cdot log_2(p_i)$$

is the information entropy of a set, where

$$p_i = \frac{freq(C_i, S)}{|S|}$$

is the fraction of the values within the set that correspond to the class i.

4.3 Decision Tree Training

Having dealt with with continuous features, the decision tree building part treats attributes in a universal way for both continuous and categorical values. For a given set of training data there are the three following cases evaluated by the algorithm [1].

- If the instances provided by the data set belong to a single class, the tree is a leaf corresponding to that class.
- If the data set contains no instances, the tree is a leaf corresponding to a default class.
- If the data set contains instances belonging to various classes, then the feature A with attribute values $\{a_1, a_2, ..., a_n\}$ that maximizes the information gain calculated for each feature as

  $$G(S, A) = E(S) - \sum_{i=1}^{n} p_i \cdot E(S_i) \tag{4.1}$$

  where $E(S)$ is the entropy of the examined feature and

  $$p_i = \frac{|S_i|}{|S|} \tag{4.2}$$

  where S_i is the new data set that would be yielded if the split were to occur in the i-th attribute value. The algorithm proceeds to create every tree node recursively using subsets of the data.

To sum the training part up in simpler words and paraphrase the evaluation of the information gain [2]:

- Find the independent variable, which if used as splitting criteria, would yield to nodes as unrelated as possible to the dependent variable.
- Apply the splitting criteria.
- Repeat this procedure for every yielded node, while no further splitting is possible.

4.4 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks connected via programmable interconnects. FPGAs can be reprogrammed to satisfy the demands of an application or any functionality requirements after manufacturing [11]. Over time, FPGAs have evolved from small arrays of programmable logic to massive arrays of programmable logic and interconnect with on-chip memories, custom data paths high speed Input/Output, and microprocessor cores all co-located on the same chip.

An FPGA is an array of programmable logic blocks and memory elements that are connected together using programmable interconnect. Typically these logic blocks are implemented as a look-up table (LUT) a memory where the address signal are the inputs and the outputs are stored in the memory entries [3]. For the implementation of a function using a LUT, its truth table shall be stored in memory and the returned values will be driven by the rest of the circuit. These truth tables provide with an efficient and flexible method for relatively small functions' implementation; the functionality is altered by appropriate adjustments made to the descriptive truth table. Most FPGAs use LUT of four to six bits of input and modern FPGAs include millions of LUTs.

Ta basic memory element in an FPGA is the flip-flop and ensembles involving multiple of those are usually combined with LUTs (Fig. 4.2). Multiple LUTs can, in turn, be combined with FFs and other circuits as well that offer alternative functionalities, such as a full-adder, for the formation of a more complex logic block, which is called a configurable logic block (CLB). A CLB is a combination of a small number of LUTs, FFs and multiplexers in order to form a more powerful element of programmable logic, while the number of such elements that form a CLB varies depending on the FPGA architecture.

A CLB can utilize more complex logic functions as elements, for example a full-adder as an element of synthesis. In this case, the full-adder is not programmable logic, but can only used as is and while its functionality is crucial and popular, it is more efficient to use it this way, rather than implemented in programmable logic; this is a typical example of flexibility-performance trade-off, in favor of performance.

As the number of transistors on a FPGA continues to grow, FPGA architectures lean towards using more and more elements that operate as autonomous entities. For

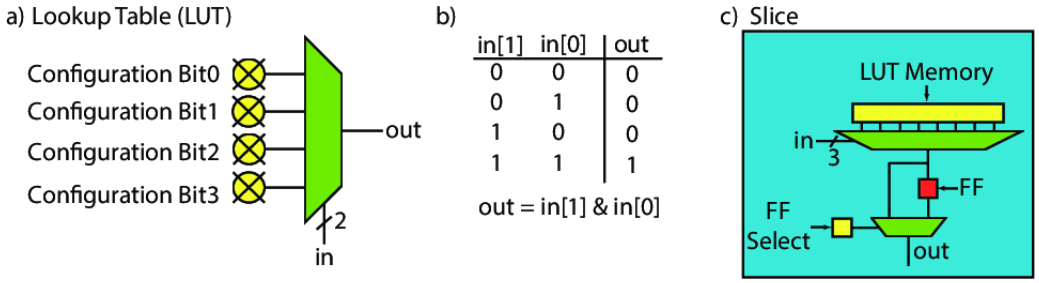


Figure 4.2: In part a) a dual-input LUT is presented with the 4-bits driving its functionality. In part b) the truth table of a possible implemented function is presented (AND-gate). In part c) a simple CLB (alternatively referred to as slice) with the possibility of storing the output in FF is shown. (Source: [3])

example, the tendency of many applications to an excessive utilization of addition and multiplication units, led to FGPAs including more autonomous resources carrying out such operations, along with the capability of yielding the better performing of programmable logic units designed to offer e.g. cumulative sum functionality in what are called digital signal processors (DSPs). Block Random-Access Memories (BRAMs) (Fig. 4.3) on the other hand, are elements of Random-Access Memories (RAMs) with the ability of offering variable length addressing, supporting various interfaces or microprocessor buses in a way that performs much better as compared to equivalent FF-based implementations on programmable logic.

Last but not least, programmable interconnect is another essential for the FPGA, for it offers the flexible wire network for the interconnection of the CLBs (or slices). The inputs

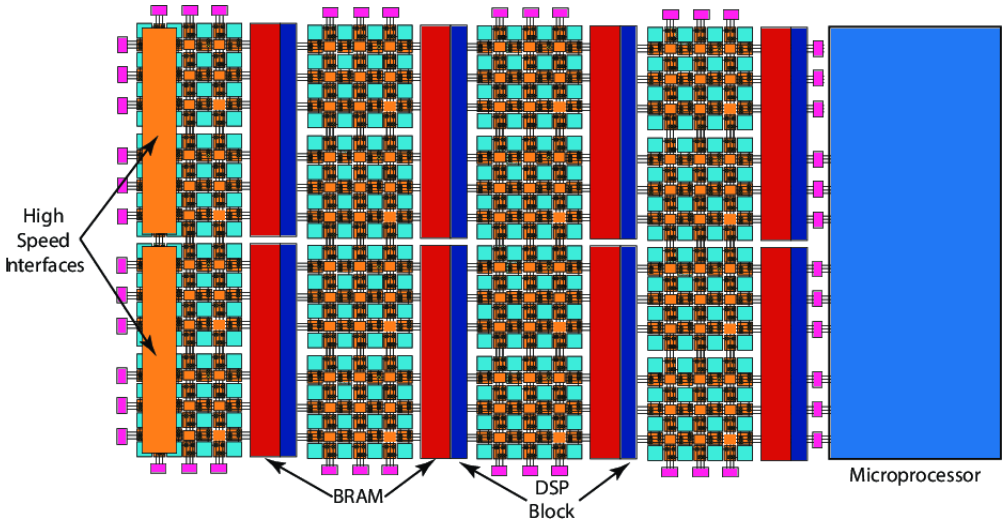


Figure 4.3: Modern FPGAs are heterogeneous including more complex elements performing predefined operations, while often combined with other on-chip resources through interconnections using interfaces such as the Advanced eXtensible Interfaces (AXI). (Source: [3])

and the outputs of the CLB are connected through a routing channel that includes the configuration bits which can enable the communication of the CLB with the interconnect in a programmable manner (Fig. 4.4). This is the method by which the communication of the FPGA with other peripherals is accomplished. At this point, it should be mentioned, that regarding the Xilinx platforms deploying heterogeneous architectures involving FPGAs, the design process of a functionality in hardware is approached by the more abstract method of High-Level Synthesis (HLS) in Software-Defined System on Chip (SDSoC) development platforms that along with the high-level design luxury, result in better resource utilization of the FPGA.

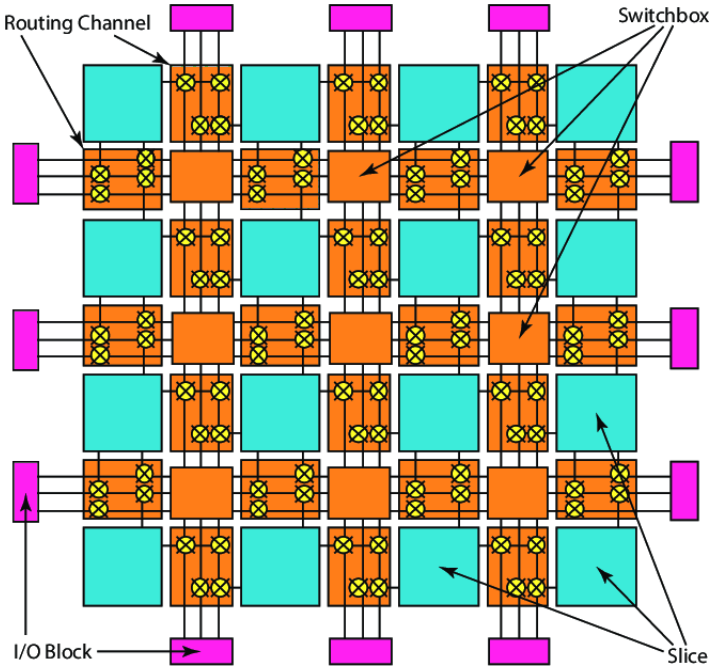


Figure 4.4: The structure of FPGA: Programmable Logic and memory resources are interconnected, while I/O Blocks offer FPGA interconnection with other on-chip resources. (Source: [3])

4.5 Zynq-7000 SoC Architecture

The Xilinx Zynq-7000 architecture enables implementation of custom logic in the Programmable Logic (PL) and custom software in the Processing System (PS). It allows for the realization of unique and differentiated system functions. The integration of the PS with the PL allows levels of performance that two-chip solutions, e.g. an application-specific integrated circuit with an FPGA, cannot match due to their limited I/O bandwidth, latency, and power budgets.

The inclusion of an application processor enables high-level operating system support, e.g., Linux. Other standard operating systems used with the Cortex-A9 processor are also available for the Zynq-7000 family. Regarding the programmable logic, the available

resources of Zedboard Zynq-7000 SoC are briefly described in Table 4.2.

Resource Name			
DSP	BRAM	LUT	FF
220	140	53200	106400

Table 4.2: Available resources for Zedboard Zynq-7000 SoC.

The PS and the PL are on separate power domains, enabling the user of these devices to power down the PL for power management if required. The processors in the PS always boot first, allowing a software-wise approach for PL configuration. PL configuration is managed by software running on the CPU, so it boots similar to an application-specific integrated circuit [10]. The architecture of the Zynq-7000 System-on-Chip design is presented in Fig 4.5.

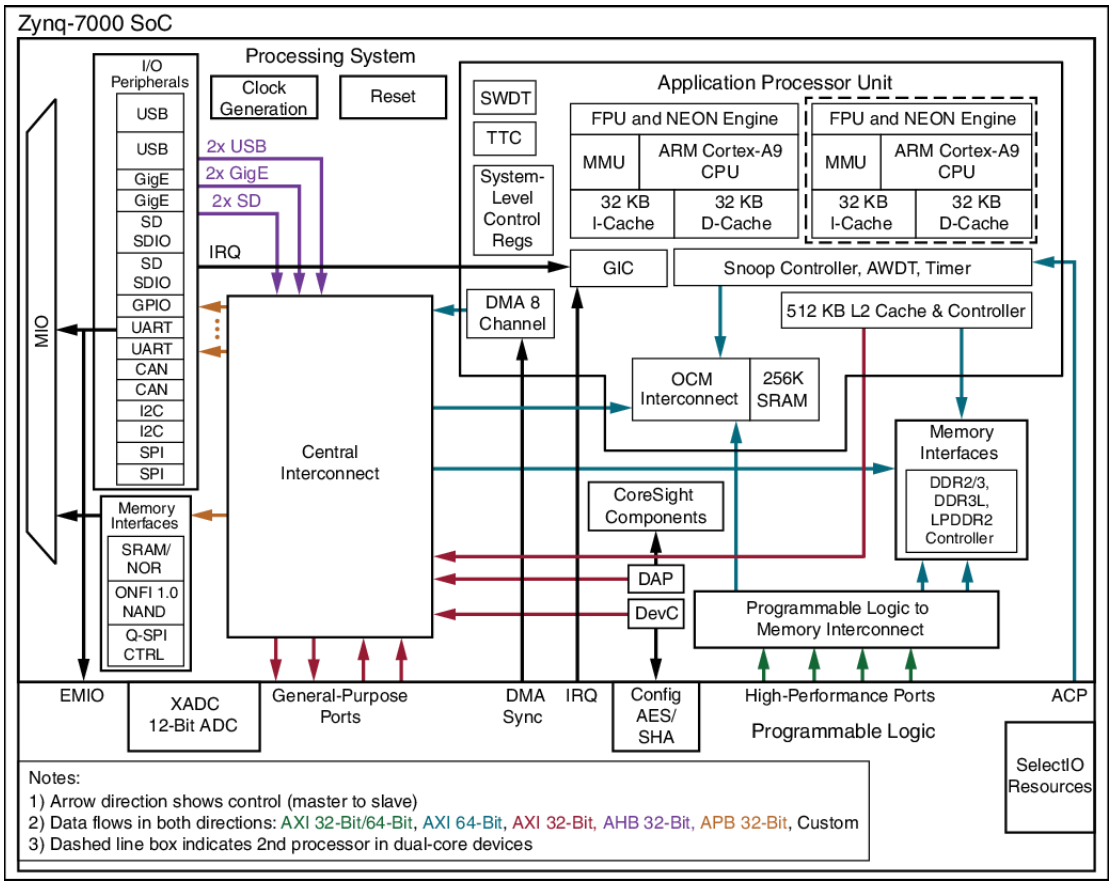


Figure 4.5: Zynq-7000 Architecture Overview. (Source: [10])

4.6 SDSoC Development Platform

The Software-Defined System-on-Chip (SDSoC) environment provides a framework for developing and delivering hardware accelerated embedded processor applications using standard programming languages. It includes a familiar embedded processor development flow with an Eclipse-based integrated development environment (IDE), compilers for the embedded processor application and for hardware functions implemented on the programmable logic resources of the Xilinx device [7]. The `sds++` system compiler analyzes a program to determine the data flow between software and hardware functions, generating an application-specific system on-chip supporting bare metal, Linux, and FreeRTOS as the target operating system. The `sds++` system compiler generates hardware intellectual property (IP) and software control code that automatically implements data transfers and synchronizes hardware accelerators and application software, pipelining communication and computation.

The concept of a platform is integral to the SDSoC environment, as it defines the hardware and software components as well as the meta-data on which SDSoC applications are built [8]. Figure 4.6 illustrates the platform and its components. An SDSoC platform defines a base hardware/software architecture and an application context including the processing system, external memory interfaces, custom input/output, a software runtime with an operating system, boot-loaders, drivers for platform peripherals and a root file system. Every project created with the SDx IDE targets a specific platform and is customized with application-specific hardware accelerators and data motion networks.

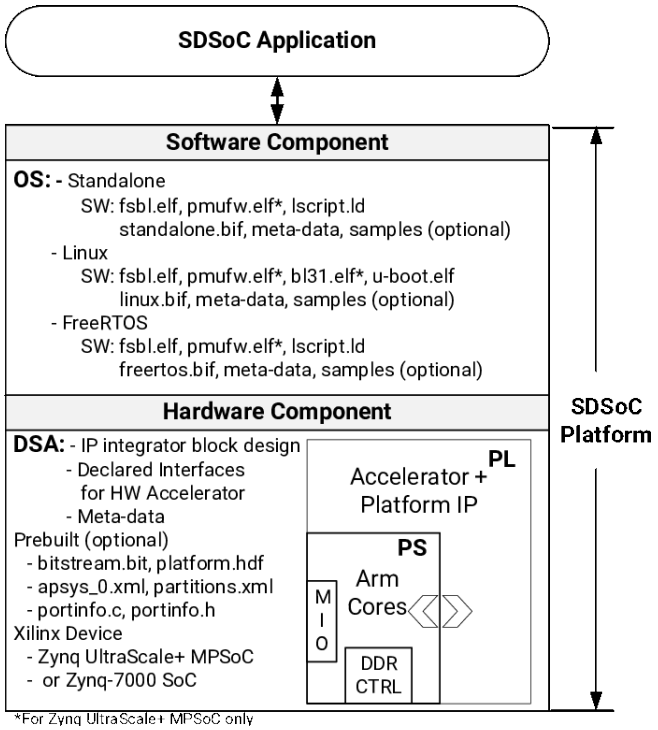


Figure 4.6: The SDSoC Development Platform. (Source: [8])

In conclusion the FPGA offers the capability of in-parallel executing functions that could be run by the CPU, resulting in the benefits obtained by the parallel implementation as a circuit. The SDSoC development environment, on the other hand, is the tool that provides the opportunity to design code in higher level languages such as C/C++ and automatically produce the VHDL description that implements the indicated function in hardware.

4.7 Target Platform Description

For the purposes of the presented work and the implementation of hardware acceleration of a decision tree learning algorithm, the ZedBoard system was made use of. ZedBoard is a low-cost development board for the Xilinx Zynq-7000 All Programmable SoC (AP SoC). While it contains everything necessary to create a Linux, Android, Windows, or other OS/RTOS based design, the SDSoC development platform is, in the present work, specified to create a Linux based design. Additionally, several expansion connectors expose the processing system and programmable logic I/Os for easy user access. The Zynq-7000 SoCs feature a tightly coupled ARM processing system and 7 series programmable logic that allow powerful designs with ZedBoard. Target applications include video processing, motor control, software acceleration, Linux/Android/RTOS development, embedded ARM processing, and general Zynq-7000 SoC prototyping. Key features and elements of the presented platform include [12]:

- PS & PL I/O expansion (FMC, Pmod, XADC)
- Analog Devices ADV7511 High Performance 225 MHz HDMI Transmitter (1080p HDMI, 8-bit VGA, 128x32 OLED)
- Analog Devices ADAU1761 SigmaDSP Stereo, Low Power, 96 kHz, 24-Bit Audio Codec
- USB OTG 2.0 and USB-UART
- 10/100/1000 Ethernet
- Onboard USB-JTAG Programming
- 4 GB SD card
- 256 MB Quad-SPI Flash
- 512 MB DDR3
- Dual-core ARM Cortex-A9
- Xilinx Zynq-7000 AP SoC XC7Z020-CLG484
- General Zynq-7000 AP SoC prototyping

- Embedded ARM processing
- Linux/Android/RTOS development
- Software acceleration
- Motor control
- Video processing

Chapter 5

Software Acceleration

5.1 Introduction

In this project, we focus on identifying a function that is highly computationally intensive, separating it from the rest of the code and let it be compiled as a circuit. This code pattern yields a number of restrictions for both the code selected to be hardware accelerated as well as the rest of the code sharing structures and other variables with the target function. The profiling of the algorithm execution is needed in advance to help us identify that function.

5.2 Algorithm Profiling

A custom implementation of the C4.5 decision tree algorithm in C++ is designed, developed and then profiled as to what percentage of the time is consumed by each sub-procedure in order to help us identify the most highly computationally intensive one. The results are presented in the diagram below.

In the testing part, a single path along the decision tree is traversed for each of the testing instances. The training part of the algorithm is by far the most time consuming

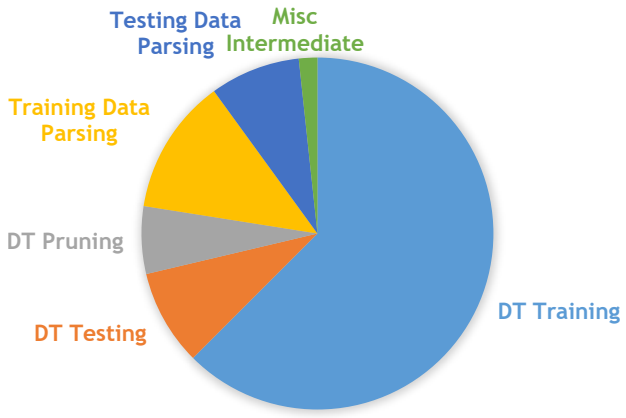


Figure 5.1: Profiling of Decision Tree (DT) Classifier for Adult [14] data set.

compared to with the testing part and although the classification part has the ability to be parallelizable by any desired factor regarding the independent testing of the input instances, in this work we focus on accelerating the most time consuming part which is the training one. The building of the decision tree involves a series of highly time consuming procedures some of them being mainly memory intensive requiring high memory usage and the hardware acceleration of such functions is deemed inefficient. On the other hand, the part regarding the information gain computation is time consuming enough to be worth being accelerated separately, requiring memory utilization capable of being provided by the FPGA resources, unlike the rest of the training part that requires processing the data set and its subsets simultaneously and in a recursive manner.

Metric / Procedure	Time (%)
Decision Tree Training	62.5
Decision Tree Testing	8.3
Decision Tree Pruning	6.2
Training Data Parsing	12.5
Testing Data Parsing	8.8
Miscellaneous Intermediate	1.7

Table 5.1: Detailed percentage description of load distribution per function.

Specifically, from the profiling carried out for a C++ implementation of the C4.5 algorithm and according to the detailed percentage description in Table 5.1 (obtained through the GNU `gprof` performance profiling tool), 62.5% of the total time consumed for the whole procedure, including the time required to load and process the raw data from the input files, and the whole training and testing parts of the algorithm, was required by the training part. The testing part consumed only the 8.3% of the total time while the time required to parse these data consumed 8.8% of the total time and when the ratio of the instances for the two parts of the algorithm (training and testing) respectively was 2:1. Since the scaling of the testing data yields commensurate time increase to the respective procedure, due to the $O(1)$ complexity of a single instance processing for a constant number of features, the new percentages of the time consumed for each part, if the ratio training to testing data were to become 1:1, by increasing the amount of the testing data to match that of the training data, would be

$$p_{train} = 62.5\% \times \frac{100}{117.1} = 53.4\%$$

and

$$p_{test} = 2 \times 8.3\% \times \frac{100}{117.1} = 14.2\%$$

and these percentages justify our decision to attempt an optimization on the decision tree training part, since the amount of time spent on building a decision tree with a specific amount of instances takes up to 3.76 times more than it would for the same amount of instances of the testing part, which, including an inner sight of the implementations of these parts as mentioned earlier, is deemed computationally not intensive.

Having said that, we might now want to focus on what nested functions of the training part would be time-wise worth of being implemented as hardware functions that would run on FPGA. Of the 62.5% of the time that is consumed for the building of the tree, its two portions of 37.5% and 25%, are required by the processes of a) forming the reduced data set that provides the subset of instances that the next step on the recursive building of the tree will use, and b) the evaluation of the splitting feature at each step of the building, respectively.

5.3 Hardware Function Selection

The first sub-part of the training is highly memory intensive, and though parallelizable, this characteristic lies upon accessing random-access memory (RAM) in parallel manner, of data that very easily exceed the amount of dynamic-RAM (DRAM) provided by an FPGA-based heterogeneous system, especially when these instances of data-subsets are required to co-exist in the RAM while the tree is built recursively, in a top-down manner. On the other hand, the part regarding the evaluation of the splitting feature at each step of the recursion, requires the processing of the whole data included in each subset of training instances. During this part, the information gain is calculated for each feature by means of evaluating the frequencies of the attributes and applying of the metric each time, and the feature maximizing the information gain is its output.

The sub-part of the splitting feature evaluation is implemented as hardware function through special re-design of the code with adjustments regarding the structures used to describe the data processed by the FPGA as well as the rest of the algorithm that refers to the very same data.

Chapter 6

Acceleration Design

6.1 High-Level Synthesis

The High-Level Synthesis approach in the design of the hardware accelerator adds yet another layer of abstraction to the design of the hardware accelerator and this way allows for the designer's concentration on the main issues of the algorithm and its implementation. The high-level synthesis tools are assigned the task of describing the architecture of the hardware function in a Register-Transfer Level hardware description language.

Briefly, the high-level synthesis tools assume the following tasks, that alternatively would be up to a human to design in register-transfer level:

- the analysis and exploitation of the concurrent possibility of processing observed in an algorithm.
- the appropriate deployment of registers in order to reduce the design's volume of the critical path and to achieve the system operation at the desired clock frequency.
- the creation of the control logic that yields the data path design.
- the implementation of the interfaces through which FPGA communicates with the rest of the system.
- the mapping of data into memory elements for the better utilization of the resources and bandwidth.
- the mapping of computational kernels into logic-gate components design by automatically performing optimizations that are indicated by the user.

The main objective of making use of the high-level synthesis is the automatization to the higher extent possible, of the aforementioned procedures based on user directive messages. The high-level synthesis tool that is used for the purposes of this project is the Vivado HLS (Xilinx) which is offered integrated with the SDSoC IDE. In general, the necessary adjustments to a C/C++ language implementation are subject to the following restrictions.

- Restrictions in reference of references and other nested references.
- Use of static rather than dynamic memory allocation by hardware function.
- General restriction on system calls.
- Limited number of standard libraries offered.
- Restriction of some object-oriented elements of the language C/C++, such as function pointers and virtual functions.
- Inability to support recursive functions (among other restrictions, this one affects the presented application with its restriction in only optimizing parts of the training process that are not recursive and while the training procedure is mostly recursive, the hardware accelerator calls are as many as the steps of the recursion).
- The communication interface of the data between the PS-PL memories must be explicitly indicated by the designer.

The high-level synthesis tool seen as a system requires input that includes:

- the indication of the function that meets the aforementioned criteria for a hardware implementation.
- a test bench for the verification purposes of the function to be implemented.
- the FPGA target platform.
- the desired clock frequency.
- directives for the implementation process.

and the output of the synthesis tool is a register-transfer level design accompanied by estimates of performance and utilization of resources. Specifically it includes:

- description of the hardware in a specialized language (e.g. VHDL).
- register-transfer level design testing simulations.
- static analysis of system performance and use of resources.
- meta-data regarding the design boundaries for making the function's integration with the rest of the system easier.

Having clarified the restrictions under which the Vivado high-level synthesis tool is subjected, we proceed to describe the revisions and redesign of the algorithm's implementation in order to achieve the desired outcome.

6.2 Data Representation in Memory

First, it should be mentioned that the input data are represented in the CSV (Comma-Separated Values) format as strings (Table 4.1) and undergo a parsing processing during which they are compressed into byte-encoded values. This is achieved through the implementation of a dictionary per each feature do as to have every attribute value mapped into an identifying sequential number. In this way, the data are represented in much more compact form as compared to with what they would have alternatively required in terms of bits to be represented if a single dictionary was used. With each feature having its own dictionary, enumeration of attributes belonging to different features may overlap, but the combined knowledge of the column (feature) which they belong to, assures that the compression is loss-less.

The representation of an integer in a byte-long word is the lightest way of possibly representing it, that accounts for a standardized C/C++ data type, providing this way the necessary flexibility of being referred to in the same way by both PS and PL. The flexibility lies upon its being able to offer the lightest common data type that doesn't yield any communication overhead per se, when there are possible representations of integers requiring less than 8-bits in an FPGA (when the attributes' values are less than or equal to 256), a case in which the casting overhead would be detrimental to the overall system performance. Regarding the continuous valued attributes, their encoding is achieved after the pre-processing resulting in their binary representation (§4.2).

6.3 Structure and Streaming of Data

The training data set is initially represented in a two-dimensional matrix, in a byte-encoded manner, as analyzed earlier. The redesign of this 2D structure leads to a linearized array of data (Tables 6.1 and 6.2) in a from a programming approach dereference by one level, as well as in a hardware mapped linearization, a necessary condition of making use of the more advantageous, from a performance point-of-view, streaming protocol that will be analyzed later.

The contiguity of the memory on a physical level is achieved through the use of the

$(1,1)$	$(1,2)$	$(1,3)$
$(2,1)$	$(2,2)$	$(2,3)$
$(3,1)$	$(3,2)$	$(3,3)$

Table 6.1: Exemplary 3×3 matrix with indices specified.

$(1,1)$	$(1,2)$	$(1,3)$	$(2,1)$	$(2,2)$	$(2,3)$	$(3,1)$	$(3,2)$	$(3,3)$

Table 6.2: Table above, having undergone linearization.

`sds_alloc()` function call, declared in library `sds_lib.h`, which guarantees physically contiguous memory, unlike the C standard `malloc()` function call, whose underlying system calls provide for contiguity only regarding the virtual memory.

The streaming of the data, which have undergone the aforementioned adjustments to meet the high-level synthesis method's criteria, takes place between either the main or the cache memory of the processing system and the FPGA and through the advanced extensible interface, the communication bus between the processing system and the programmable logic. In the implementation of the function that is assigned to carry out the splitting feature evaluation for the purposes of building the decision tree, the flow of data is from the processing system and towards the programmable logic, while only the returning function byte-sized value, follows the other way round. The training data stored in memory are referred to by their first element's address and are directed to the FPGA for further processing, through the flexible fourth generation protocol AXI4 (Fig. 6.1), while the remaining variables referred to by their values are transferred through the equivalent point-to-point word streaming protocol.

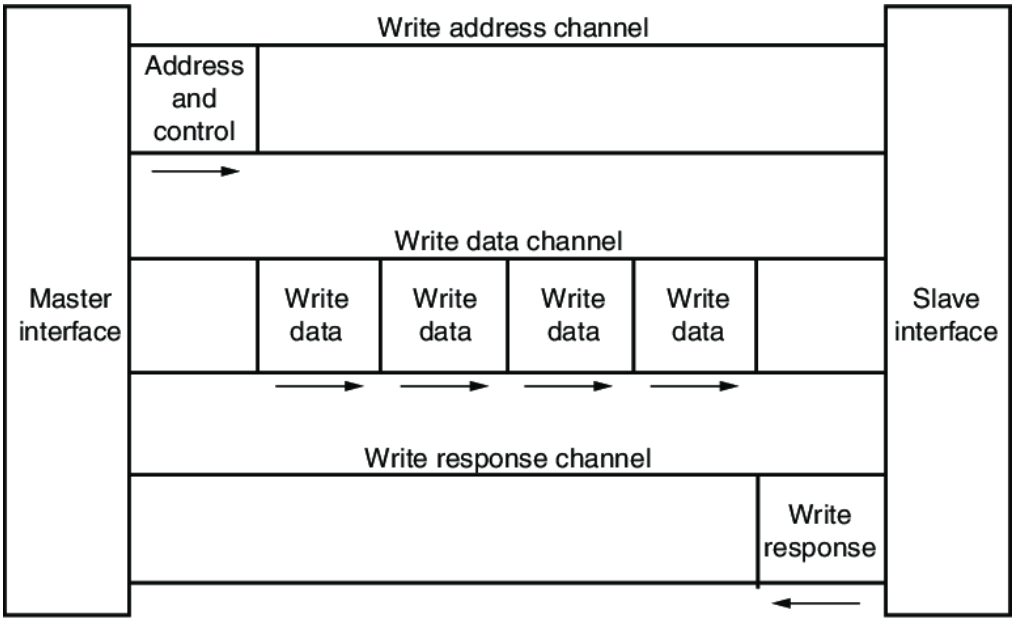


Figure 6.1: A diagram illustrating how the AXI4 protocol for writing data from a master to a slave interface, works. (Source: [5])

Having already referred to the protocol followed by the system to implement the streaming of the data, we proceed to present the `#pragma` [13] style directives with which the designed can specify the details surrounding the PS-PL communication in a C-style code. These variables are specified above the function declaration and refer to the function's arguments. The ones that are pertinent to the application studied presently, are briefly defined below.

- `access_pattern`: it specifies the data access pattern in the hardware function. The

SDSoC tool checks the value of this directive to determine the hardware interface to synthesize. If the access pattern is sequential, a streaming interface will be generated. Otherwise, with random access pattern, a RAM interface will be generated.

- `copy`: the copy directive implies that data is explicitly copied between the host processor memory and the hardware function.
- `data_mover`: it can be used to override the compiler choice of the type of the data mover made automatically by analysis of the code. It specifies the hardware IP type used to transfer an array argument.
- `mem_attribute`: the directive which includes this key word, informs the `sds++` compiler about the physical contiguity or not of the data refereed to by the specified argument. In its absence, SDSoC ascribes a default value based on code analysis.
- `sys_port`: through this directive the designer indicates about the desired memory port type; overrides default, e.g. Accelerator Coherency Port (ACP), High Performance Port (HPC).
- `zero_copy`: usage of this directive means that the hardware function accesses the data directly from shared memory through an AXI master bus interface.

For the present application, the fact that the data are not accessed in a sequential manner with regards to the serialized array, leads us to design a different way with which the data will be streamed through the Advanced Extensible Interface (AXI) interconnect core. Every attribute column is processed once in order to have the attributes' frequencies counted and stored in block-RAM (BRAM) for the proceeding calculations, while at the same time the class column of the data which corresponds to the class that each instance targets is traversed as many times as the number of features is. For this purpose an extra reference on the last column of the training data streamed along with each column of attributes. This way sequential access of each column is achieved with the kernel accessing different features' values as different references that are streamed simultaneously. This optimizes the throughput of the communication required each time the hardware function is utilized and the data selected by the host are streamed for further processing.

6.4 Hardware Function Body Design

Before we proceed to the next step of our work, we shall analyze optimizations other than those related to the communication between PS and PL. Namely, there is only a number of directives germane to the optimization of the code in the body of the function elected to be implemented in hardware, that are offered by the high-level synthesis tool and yield the better resource utilization possible, while giving the opportunity for a high degree of concurrency in the operations carried out for the purposes of the splitting feature evaluation, of the decision tree building part. Below, a subset of the `#pragma` [13] style

related directives parameters provided by the high-level synthesis tool is presented, that provide interesting ways to optimize the specific function's body as far as the betterment of how mostly arithmetic and BRAM operations are carried out is concerned.

- `array_partition`: it partitions an array into smaller arrays or individual elements, resulting in RTL with multiple small memories or multiple registers instead of one large memory, effective increase of the amount of read and write ports for the storage, potentially improving the throughput of the design, requiring more memory instances or registers.

- `loop_tripcount`: it is required to know to some extent the number of iterations of a loop; it may depend on the value of variables used in the loop expression e.g. x<y, or depend on control statements used inside the loop. In some cases Vivado HLS cannot determine the trip count, and the latency is unknown.

- `pipeline`: it reduces the initiation interval for a function or loop by allowing the concurrent execution of operations. A pipelined function or loop can process new inputs every n clock cycles, where n is the initiation interval (II) of the loop or function. The default initiation interval for the pipeline directive is 1, which processes a new input every clock cycle. See Figures 6.2 and 6.3 below for a graphical explanation of how loop pipelining works.

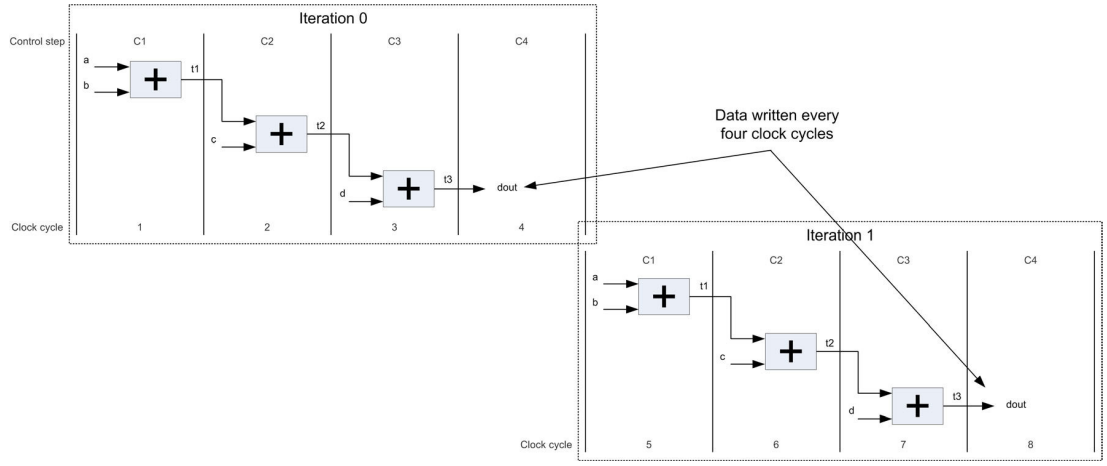


Figure 6.2: An example of a loop before any pipelining. (Source: [3])

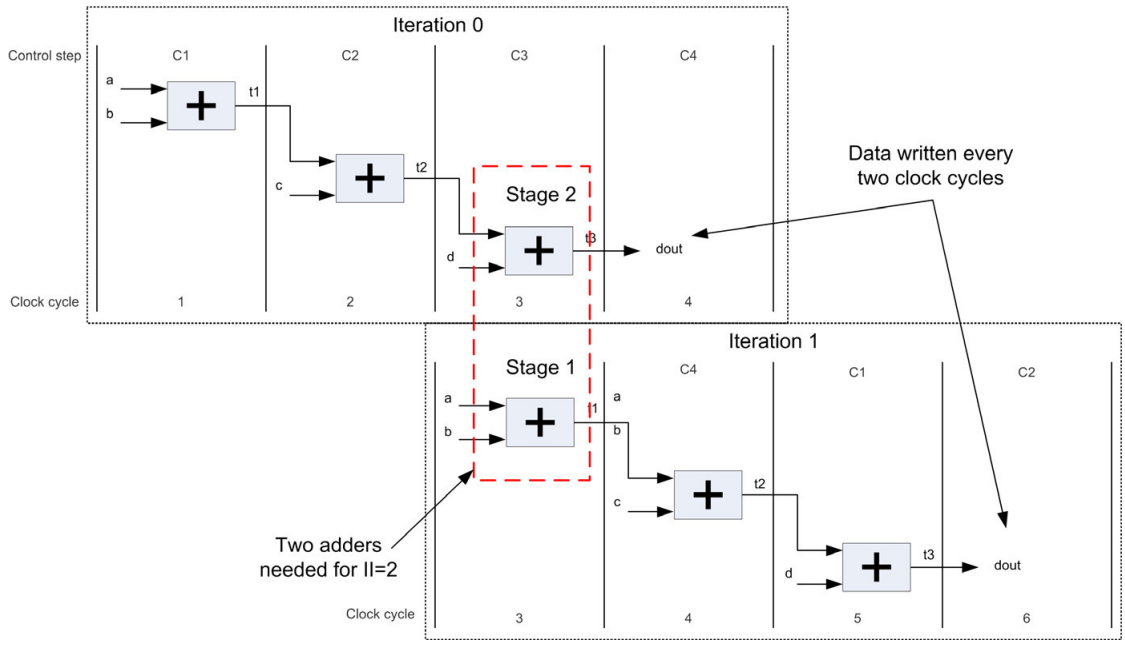


Figure 6.3: The loop above, applying pipelining with an II=1 results in a new iteration started every clock cycle. Iteration one is started in C2 and iteration 2 is started in C3. (Source: [3])

- `resource`: Specify that a specific library resource is used to implement a variable such as an array, an arithmetic operation or function argument in the RTL. If the resource directive is not specified, Vivado HLS determines the resource to use. See bullets below for a detailed description of how some resources have been specified for the purposes of the present application.
  - `RAM_2P_BRAM`: A dual-port RAM implemented with a block RAM that allows read operations on one port and both read and write operations on the other port.
  - `FMul_fulldsp`: Floating-point multiplier implemented with only DSP48 primitives.
  - `FAddSub_fulldsp`: Floating-point adder or subtractor implemented using only DSP48s primitives.
  - `FDiv`: Floating-point divider.
- `unroll`: it creates multiple independent operations rather than a single collection of operations. The unroll directive transforms loops by creating multiples copies of the loop body in the RTL design, which allows some or all loop iterations to occur in parallel. Loops in the C/C++ functions are kept rolled by default. When loops are rolled, synthesis creates the logic for one iteration of the loop, and the RTL design executes this logic for each iteration of the loop in sequence. A loop is executed for the number of iterations specified by the loop induction variable. Using the unroll

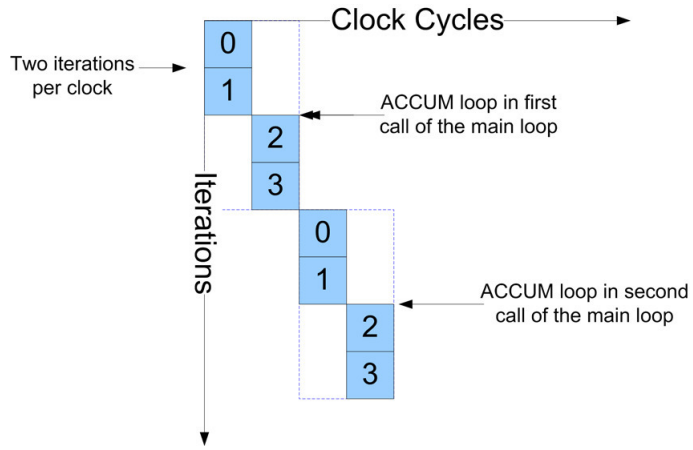


Figure 6.4: Unrolling a loop by a factor of two, yields concurrent execution of some operations, doubling the amount of resources used. (Source: [3])

directive you can unroll loops to increase data access and throughput. See Fig. 6.4.

The next step in our work is to implement the selected function in hardware. For this purpose, we make use of the SDSoC development environment that allows the designer to control the C/C++ synthesis through optimization directives (`#pragma` [9]).

First, the initially two-dimensional matrix containing the byte-encoded attributes, is re-designed as a serialized array that is stored in a physically contiguous memory to achieve the optimal kernel-processor streaming strategy. Additional characteristics inherent to the splitting feature evaluation function and the data structures involved in it are taken into account as the hardware design using high-level directives is implemented.

Second, a number of pragma directives were added to specify an implementation of

working class attributes	**Income Classes**	
	$\leq 50K$	$> 50K$
federal-gov	449	276
local-gov	1108	445
never-worked	5	0
private	12947	3569
self-emp-inc	363	454
self-emp-not-inc	1348	528
state-gov	683	256
without-pay	8	0

Table 6.3: Instance of the measured frequencies of a feature's values from the Adult [14] data-set, with respect to the corresponding classes. (Source: [21])

the maximum information gain evaluation as well as the metric's values calculations. To proceed in the further explanation of the accelerator architecture regarding the selected function, we shall explain in more detail the work carried out by that function.

For each feature column the frequencies of the present attributes with respect to the corresponding classes that the training instance targets are calculated and stored in a two-dimensional frequency matrix as demonstrated in Table 6.3.

Such a frequency matrix is formed by a single column of attributes corresponding to just one feature of the training data and one is formed for every external loop of the maximum information gain evaluation. A single iteration of the respective feature column whose data are streamed through the AXI interface and are processed sequentially is enough for its items' frequencies to be stored in the dual-port BRAM in the kernel. The dual port BRAM is selected for it offers the opportunity to evaluate the attribute's frequencies with respect to the corresponding classes of two features at a time (hence the two attribute arguments in Fig. 6.5; the offset is the address difference of the two references) making use of the independent nature of the processing on the FPGA. This is accomplished by specifying the unroll pragma directive with a factor of two to the loop regarding the iterations over the number of features.

Additionally, an attempt to optimize the loop regarding the iteration over the input data that measures the aforementioned frequencies is made. This is achieved by the use of the pipeline pragma directive, the effect of which essentially allows concurrent operation executions of the loop's body by means of deploying loop iterations in overlapping time intervals, yielding a respective drop to the parameter of the loop's initiation interval to the minimum possible for the specific loop and which is determined by SDSoC during the synthesis.

The next step of the processing carried out at the kernel regards the processing of the recently measured frequencies and the application of the information gain formula with the calculation of the respective cumulative sums of entropy portions. This part of the algorithm is implemented as a series of double nested loops of which the innermost have the pipeline applied to their iterations over the number of classes of the data set and the outermost have it applied to the iterations over the number of attribute values per examined feature. The latter is furthermore unrolled by a factor that would statistically allow the throughput of the frequencies matrix reads to be maximized as the accesses of its elements is done by the independent unrolled loop parts that can refer to the different blocks of the array simultaneously, for which the respective block partition pragma is specified for the linearized matrix. At the same time, the last column of the training data that corresponds to the target class of the training instances has been streamed and stored in dual-port BRAM in the FPGA, that allows the unrolled by a factor of two loops that iterate over equal amounts of the features' attributes (split in half) and enables the access of the classes values at the same time while guaranteeing the independence of the execution of these two unrolled loop parts.

As far as the cumulative sum calculation is concerned, the application of the formula

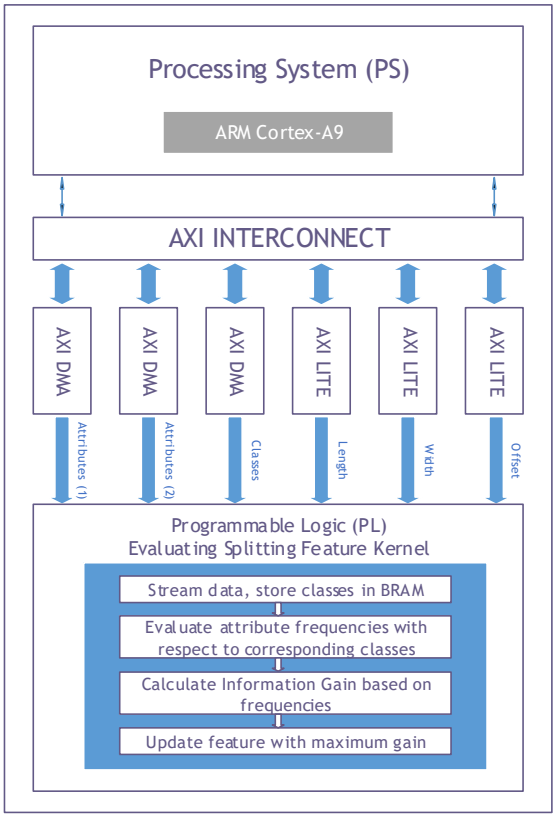


Figure 6.5: Above is the simplified block design of the heterogeneous architecture involving FPGA and CPU, featuring therebetween communication.

(4.1) that calculates the information gain of the column/feature is utilized and for which a number of adjustments and directives are applied yielding the respective resource allocation where the additions, multiplications and divisions of floating point numbers are going to take place. The additions for the purposes of the cumulation of the sum of the entropy are implemented in floating-point adders that use only DSP48 [6] primitives, whereas the multiplications of the fractions (4.2) with their respective logarithm portions are calculated in floating-point multiplier using similar primitives, while the divisions that are required for the fractions (4.2) calculations are carried out at floating-point divider.

A simplified block design is shown in Fig. 6.5 that illustrates the system architecture design. It includes a processing system that carries out the rest of the calculations assigned to the FPGA implemented kernel, which performs the task of the splitting feature evaluation, through the calculation of the information gain metric. Data stored in arrays are transferred through AXI direct memory access (DMA) while the rest integer type variables are transferred through the AXI4-Lite interface.

Chapter 7

Performance Evaluation

The case studies considered for the performance evaluation of the training and testing parts of the decision tree classifier were the Adult and Census-Income, multivariate data sets provided by the UCI machine learning repository [14]; features of both data sets included both categorical and integer type attributes with the latter including floating-point type attributes as well. A quantitative profiling of the data sets shows that the former includes 48842 total instances of 14 features, 8 of which are of categorical attributes and 6 are of continuous values upon which the discretization prepossessing was applied, as was done for the 7 continuous features of a total of 40 features, in the Census-Income database of 199523 training instances and 99762 testing ones, with the rest 33 being categorical value features.

As it has been demonstrated in the previous chapter, the optimized function of the algorithm regards the evaluation of the splitting feature at each step of the recursive building of the decision tree. The main computational load sought to be optimized in a hybrid implementation where these computations are to be carried out by the FPGA, is caused by the triply nested loop over the number of features, number of attribute values per feature, number of classes, all in that sequence, besides the loop over the instances. The loop over the features is unrolled by a factor two and the loop over the number of attribute values is unrolled by a factor of four, while the rest of the loops are pipelined. This means that a data-set requiring more iterations over one of the unrolled loops primarily and out of the pipelined loops as well, gets more out of the accelerator as a total speedup compared to with a sequential execution. Data-sets like the Census-Income that include more continuous attributes which for the purposes of the training part are binarized, make less of the accelerator despite having more features. In general, the higher the average branching factor of a tree, the higher is the gain of the execution in the presented accelerator, as that would imply more average attribute values, enhancing the gain made by the fourfold unrolled loop. Generally the speedup is highlighted by the better utilization of the inner and executed in a more concurrent manner loops that make the most of more attribute values and classes and an example of a data-set that makes less of it, only compared to with the Adult data-set, is the Connect-4 [14] data-set with

three possible attribute values over 42 features, when Adult's features of working class, education, marital status, occupation, relationship and native country have 7, 15, 6, 13, 5, 40 different values of attributes, respectively, yielding higher average branching factor.

One of the major factors that could hinder the speedup and the overall gain of a hardware function implementation on FPGA is the communication overhead caused by the need to transfer the data between the Processing System (PS) and Programmable Logic (PL). Such being the case, in this work we had to compare and deduce the most appropriate means of achieving the PS-PL communication. As mentioned earlier, the classes' attributes are accessed in a periodic manner which violates the sequential access pattern characterizing the rest of the attributes. That led us to evaluate the use of zero-copy for the data streamed, either partially for the classes, or for the whole data set streamed, an overall implementation that was observed to yield higher communication overhead compared to the implementation that resulted in the final kernel speedup. Moreover, it should be noticed that the hybrid implementation imposals of more pre-allocated structures preference over the classic dynamic ones that are typically used, further boost the system speedup of the training part as a whole, even through functions running out of FPGA. The measurements presented were observed for the Adult and Census-Income data sets at data motion and operating clock frequencies both being at 142.86 MHz, the resource utilization for which by the hardware function is shown in Table 7.1 (DSP: digital signal processors, BRAM: block random-access memory, LUT: look-up tables, FF: flip-flops). In order to achieve the performance of the implementation presented a number of parameters were subjects to fine tuning, such as those of the array partition factors, loop-unrolling factors and resources where operations such as additions, multiplications and divisions were carried out.

Table 7.1: Resource Utilization for Splitting Feature Evaluation. Target Platform: ZedBoard Zynq-7000 SoC.

Resource	**Hardware Function Resources**		
Name	***Used***	***Total***	% ***Utilization***
DSP	48	220	21.82
BRAM	75	140	53.57
LUT	35807	53200	67.31
FF	47608	106400	44.74

The C/C++ code of the algorithm along with the software-defined system (SDS) and high-level synthesis (HLS) directives that result in the presented architecture, was compiled into an executable file that along with the bit-stream generated by the SDSoC development platform required for the FPGA configuration, was able to run on the ZedBoard (Zynq-7000 All Programmable SoCs) system yielding the following results in time

and accuracy, for both mentioned sets respectively, as shown in Table 7.2 and graphically with a bar-chart in Fig. 7.1.

Table 7.2: Time and Accuracy Comparison.

Data set Name	**Implementation**			
	Software	***Hybrid***	***Speed-up***	***Accuracy***
Adult	2.33s	0.94s	2.48	82.54%
Census-Income	34.27s	15.5s	2.21	94.26%

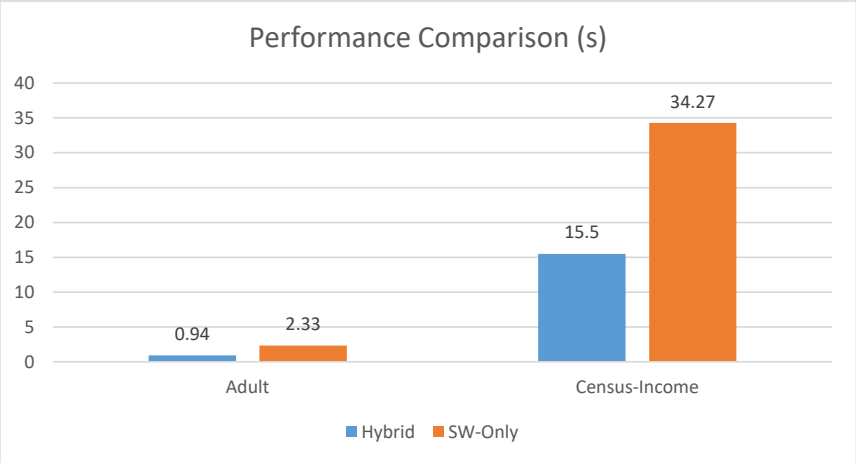


Figure 7.1: The bar-chart above, graphically represents the results shown in Table 7.2.

Chapter 8

Conclusion

8.1 Summary and conclusions

In this work, an implementation that featured hardware acceleration of a decision tree classifier was presented, where functions that posed bottlenecks for the performance optimization were let to run on a common processor (Dual-core ARM Cortex-A9) and functions that their hardware implementation was deemed beneficial, had their architecture-involving FPGA-designed in a novel way. While attempting to accelerate the classification part of the algorithm might as well have given impressive results, two main reasons focused our interest on the training part. The intrinsic simplicity of an implementation regarding the single decision tree classifying part along with the bulk of research carried out already regarding classification using decision tree ensembles, contributed to our decision to attempt an optimization on the learning procedure instead.

Acceleration using state-of-the-art hardware is increasingly getting popular and is sought as a resort for performance optimization for various machine learning algorithms among which decision trees and their ensembles have great potential. Having tackled the issue of implementing a hardware function that carries out the evaluation of the splitting feature procedure during the training prat of the decision tree, future extensions could include more kernels that would allow for bigger parts of the training part to be optimized, as the need for more resources would be covered from more FPGA nodes. Besides covering a wider subset of the building of the tree one could include different aspects of the algorithm in the hardware acceleration effort, such as the testing part, and compare the results with alternative application-specific standard products performances or even alternative hardware accelerators (e.g. GPU) on similar tasks.

8.2 Publication

A. Zoulkarni, C. Kachris and D. Soudris. *"Hardware Acceleration of Decision Tree Learning Algorithm"*, 2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST).

Bibliography

[1] J. Ross Quinlan, "C4.5. Programs for Machine Learning", Elsevier Inc. (1993).

[2] I. Vlahavas, P. Kefalas, N. Bassiliades, I. Refanidis, F. Kokkoras, I. Sakellariou, "Artificial Intelligence (in Greek - Τεχνητή Νοημοσύνη)", Book, Gartaganis Publications, ISBN 960-7013-28-X, 2002.

[3] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer, "Parallel Programming for FPGAs", ArXiv e-prints (2018).

[4] Michael Fingeroff, "High-Level Synthesis Blue Book", Mentor Graphics Corporation (2010).

[5] Xilinx Inc., "AXI Reference Guide UG761", March 7, 2011, pp. 7.

[6] Xilinx Inc., "Vivado Design Suite User Guide High-Level Synthesis UG902", December 20, 2018, pp. 167-171.

[7] Xilinx Inc., "SDSoC Environment User Guide UG1027", December 5, 2018.

[8] Xilinx Inc., "SDSoC Environment Platform Development Guide UG1146", January 24, 2019.

[9] Xilinx Inc., "SDx Pragma Reference Guide UG1253", January 24, 2019, pp. 33-62.

[10] Xilinx Inc., "Zynq-7000 SoC Data Sheet: Overview DS190", July 2, 2018.

[11] Xilinx.com, "What is an FPGA? Field Programmable Gate Array", [Online]. Available: `https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html`

[12] Xilinx.com, "ZedBoard Zynq-7000 ARM/FPGA SoC Development Board", [Online]. Available: `https://www.xilinx.com/products/boards-and-kits/1-elhabt.html`

[13] Xilinx.com, "SDAccel Development Environment Help", [Online]. Available: `https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc`

[14] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository", 2017. [Online]. Available: `http://archive.ics.uci.edu/ml`

[15] M. Owaida, H. Zhang, C. Zhang and G. Alonso, "Scalable inference of decision tree ensembles: Flexible design for CPU-FPGA platforms", 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, 2017, pp. 1-8.

[16] M. Owaida and G. Alonso, "Application Partitioning on FPGA Clusters: Inference over Decision Tree Ensembles", 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, 2018, pp. 295-2955.

[17] S. Zhao, Y. Sun and S. Chen, "A Discretization Method for Floating-Point Number in FPGA-based Decision Tree Accelerator", 2018 IEEE 4th International Conference on Computer and Communications (ICCC), Chengdu, China, 2018, pp. 2698-2703.

[18] Z. Lin, S. Sinha and W. Zhang, "Towards Efficient and Scalable Acceleration of Online Decision Tree Learning on FPGA", 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, 2019, pp. 172-180.

[19] R. Struharik, "Decision tree ensemble hardware accelerators for embedded applications", 2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, 2015, pp. 101-106.

[20] R. Narayanan, D. Honbo, G. Memik, A. Choudhary and J. Zambreno, "An FPGA Implementation of Decision Tree Classification", 2007 Design, Automation & Test in Europe Conference & Exhibition, Nice, 2007, pp. 1-6.

[21] A. Zoulkarni, C. Kachris and D. Soudris. "Hardware Acceleration of Decision Tree Learning Algorithm", 2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST).