Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Μελέτη Συνεπεξεργαστών για Κβαντικούς Υπολογιστές με Χρήση Αναδιατασσόμενης Λογικής

Ζαμπετάκης Μιχαήλ

<u>Επιτροπή:</u>
Δόλλας Απόστολος (Επιβλέπων)
Σαμολαδάς Βασίλειος
Έλληνας Δημοσθένης

Φεβρουάριος 2008

Σύνοψη

Οι κβαντικοί Υπολογιστές είναι συσκευές για υπολογισμούς, οι οποίες κάνουν απ’ ευθείας χρήση κάποιων φαινόμενων κβαντικής μηχανικής, όπως την υπέρθεση και την διεμπλοκή καταστάσεων, προκειμένου να κάνουν υπολογισμούς σε δεδομένα. Αν και οι κβαντικοί υπολογισμοί είναι ακόμα στα πρώτα βήματά τους, πειράματα έχουν διεξαχθεί όπου έχουν εκτελεστεί κβαντικοί υπολογισμοί μικρής κλίμακας. Η έρευνα, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο, διεξάγεται με γοργούς ρυθμούς και πολλές κυβερνήσεις και στρατιωτικά ταμεία στηρίζουν την έρευνα των κβαντικών υπολογιστών για ιδιωτική και κρατική χρήση. Η κρυπτανάλυση, η επιστήμη των υλικών, η νανοτεχνολογία και άλλοι τομείς θεωρείται πως θα γνωρίσουν τεράστια πρόοδο με την εξέλιξη τον κβαντικών υπολογιστών.

Με τη σημερινή τεχνολογία είναι αδύνατο να κατασκευαστούν μεγάλης κλίμακας κβαντικοί υπολογιστές προκειμένου να εκτελεστούν αλγόριθμοι. Έτσι σε αυτή την εργασία μελετάται η λειτουργία των κβαντικών επεξεργαστών προκειμένου να δημιουργηθεί ένας εξομοιωτής με χρήση του ήδη υπάρχοντος υλικού των υπολογιστών. Αρχικά μοντελοποιήθηκε το πρόβλημα με χρήση αλγορίθμου και έπειτα υλοποιήθηκαν δυο διαφορετικές σχεδιάσεις με χρήση αναδιατασσόμενης λογικής. Τα συστήματα τα οποία αναπτύχθηκαν βοηθούν στην εξομοίωση κβαντικών υπολογισμών προκειμένου να δίνεται η δυνατότητα να εκτελεστούν κβαντικοί αλγόριθμοι με τη χρήση του συμβατικού υλικού.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω

τους επιβλέποντες καθηγητές μου για την πολύτιμη βοήθειά τους σε νέα θέματα για εμένα και για τις γνώσεις που μου προσέφεραν.

τους καθηγητές μου και τους φίλους μου στο Πολυτεχνείο Κρήτης για τη βοήθειά τους κατά τη διάρκεια των σπουδών μου.

τους φίλους μου και τους γονείς μου για την πολύτιμη προσφορά και υποστήριξή τους.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Πρόλογος

Με την σύλληψη της ιδέας των κβαντικών υπολογιστών φάνηκαν οι δυνατότητες που θα μπορούσαν να προσφέρουν στην ανθρωπότητα. Οι κβαντικοί υπολογιστές κάνουν χρήση φαινόμενων κβαντικής μηχανικής προκειμένου να επεξεργαστούν τα δεδομένα. Πιστεύεται ευρέως πως αν μπορούσαν να κατασκευαστούν μεγάλης κλίμακας κβαντικοί υπολογιστές θα ήταν ικανοί να λύσουν συγκεκριμένα προβλήματα πολύ γρηγορότερα από κάθε κλασικό υπολογιστή. Οι κβαντικοί υπολογιστές διαφέρουν κατά πολύ από τους κλασικούς υπολογιστές, όπως τους DNA υπολογιστές, ή τους υπολογιστές που βασίζονται σε τρανζίστορ, ακόμα και από αυτούς που χρησιμοποιούν κάποια επακόλουθα της κβαντικής μηχανικής (π.χ. ομοιοπολικοί δεσμοί). Κάποιες αρχιτεκτονικές υπολογιστών, όπως οι οπτικοί υπολογιστές, μπορεί να χρησιμοποιούν την κλασική υπέρθεση των ηλεκτρομαγνητικών κυμάτων, όμως χωρίς κάποιους συγκεκριμένους κβαντομηχανικούς πόρους όπως την υπέρθεση και την διεμπλοκή καταστάσεων και έτσι δεν αποκτούν τις δυνατότητες για υπολογιστική επιτάχυνση που επιτυγχάνουν οι κβαντικοί υπολογιστές.

Σε αυτή την εργασία έγινε εκτεταμένη χρήση αναδιατασσόμενης λογικής η οποία συνδυάζει την ευελιξία του λογισμικού με τις υψηλές επιδόσεις του υλικού με το να επεξεργάζεται ευέλικτα και γρήγορα τα δεδομένα με τη χρήση FPGAs. Η κύρια διαφορά με τους κλασικούς επεξεργαστές είναι η ικανότητα να κάνουν ουσιώδεις αλλαγές στο data path αντί για έλεγχο ροής. Μια Field Programmable Gate Array (FPGA) είναι μια συσκευή η οποία περιέχει αναπρογραμματιζόμενα λογικά στοιχεία τα οποία καλούνται λογικά block καθώς και αναπρογραμματιζόμενες λογικές διασυνδέσεις. Με τον κατάλληλο προγραμματισμό και ένωση αυτών επιτυγχάνεται το επιθυμητό αποτέλεσμα κάθε φορά. Συχνά στις FPGAs υπάρχουν ολοκληρωμένα στοιχεία μνήμης (flip-flops) τα οποία μπορούν να χρησιμοποιηθούν άμεσα.

Οι FPGAs είναι συνήθως πιο αργές από τις συσκευές ASIC (Application Specific Integrated Circuit) και παράλληλα δεν μπορούν να διαχειριστούν σύνθετες σχεδιάσεις. Τα προτερήματά τους περιλαμβάνουν μικρότερο χρόνο time to market, δυνατότητα επαναπρογραμματισμού για διόρθωση σφαλμάτων και μικρότερο, μη επαναλαμβανόμενο κόστος. Τα CPLD (Complex Programmable Logic Devices) είναι συσκευές για πιο απλές σχεδιάσεις.

1.2 Υλοποίηση

Σε αυτή την εργασία μελετήθηκε και υλοποιήθηκε η δημιουργία ενός εξομοιωτή κβαντικών υπολογιστών με χρήση της σημερινής τεχνολογίας υπολογιστών. Αρχικά μελετήθηκε η λειτουργία των κβαντικών υπολογιστών και μοντελοποιήθηκε η λειτουργία τους με τη χρήση ενός αλγορίθμου στη γλώσσα προγραμματισμού C προκειμένου να μελετηθεί και να γίνει περισσότερο αντιληπτή η λειτουργία τους. Έπειτα μελετήθηκαν και σχεδιάστηκαν δυο αρχιτεκτονικές προσεγγίσεις, με διαφορετικά χαρακτηριστικά, οι οποίες επιτρέπουν την εξομοίωση κβαντικών επεξεργαστών. Τέλος μελετήθηκαν και συγκρίθηκαν τα αποτελέσματα τα οποία προέκυψαν από κάθε μια σχεδίαση και συγκρίθηκαν για ορθότητα και αποτελεσματικότητα με βάση τον αλγόριθμο μοντελοποίησης.

Στο 1^{ο} κεφάλαιο γίνεται μια εισαγωγή στο θέμα της εργασίας. Στο 2^{ο} γίνεται μια εισαγωγή στους κβαντικούς υπολογιστές μιας και δεν είναι μια ευρέως διαδεδομένη τεχνολογία. Στο 3^{ο} κεφάλαιο φαίνεται πως μοντελοποιήθηκε το πρόβλημα και πως αναπτύχθηκε ο αλγόριθμος προσομοίωσης κβαντικών υπολογιστών καθώς και τα αποτελέσματα που προέκυψαν. Η μελέτη, η σχεδίαση και τα αποτελέσματα της πρώτης και δεύτερης αρχιτεκτονικής, με χρήση αναδιατασσόμενων πόρων, που σχεδιάστηκαν παρουσιάζονται στο 3^{ο} και 4^{ο} κεφάλαιο αντίστοιχα. Στο 6^{ο} φαίνεται πως έγινε η πιστοποίηση των αποτελεσμάτων για τις δυο αρχιτεκτονικές ενώ στο 7^{ο} παρατίθενται τα αποτελέσματα της εργασίας. Τέλος, στο 8^{ο} κεφάλαιο παρουσιάζονται σύντομα τα συμπεράσματα και κάποιες μελλοντικές προτάσεις για τα συστήματα τα οποία μελετήθηκαν και δημιουργήθηκαν. Ακολουθεί η βιβλιογραφία στην οποία παραπέμπεται ο αναγνώστης σε ορισμένες περιπτώσεις.

1.3 Συνεισφορά

Η εργασία αυτή έχει ως σκοπό τη μελέτη των κβαντικών υπολογιστών με σκοπό τη δημιουργία ενός εξομοιωτή κβαντικών επεξεργαστών για ανάπτυξη και εκτέλεση αλγορίθμων για κβαντικούς υπολογιστές με χρήση της σημερινής τεχνολογίας των υπολογιστών. Αυτό θα δώσει τη δυνατότητα της κατανόησης της λειτουργίας και των δυνατοτήτων των κβαντικών επεξεργαστών, αλλά και τη δυνατότητα να αναπτυχθούν και να εξομοιωθούν αλγόριθμοι για κβαντικούς επεξεργαστές χωρίς η χρήση αυτών να είναι απαραίτητη. Συγκεκριμένα στην παρούσα εργασία έγιναν τα εξής:

- Μελέτη τρόπου λειτουργίας και αρχών κβαντικών επεξεργαστών
- Εξομοίωση τρόπου λειτουργίας κβαντικού επεξεργαστή με τη χρήση λογισμικού
- Συλλογή και μελέτη αποτελεσμάτων της εξομοίωσης του επεξεργαστή με τη χρήση αλγορίθμου

- Μελέτη αναγκών και δυνατοτήτων 1ου αρχιτεκτονικού συστήματος εξομοίωσης κβαντικών επεξεργαστών
- Ανάπτυξη 1ου αρχιτεκτονικού συστήματος με χρήση αναδιατασσόμενης λογικής
- Πιστοποίηση ορθής και πλήρης λειτουργίας του 1ου αρχιτεκτονικού συστήματος
- Εξαγωγή αποτελεσμάτων συστήματος, σύγκριση και σχολιασμός
- Μελέτη αναγκών και δυνατοτήτων 2ου αρχιτεκτονικού συστήματος εξομοίωσης κβαντικών επεξεργαστών
- Ανάπτυξη 2ου αρχιτεκτονικού συστήματος με χρήση αναδιατασσόμενης λογικής
- Πιστοποίηση ορθής και πλήρης λειτουργίας του 2ου αρχιτεκτονικού συστήματος
- Εξαγωγή αποτελεσμάτων συστήματος, σύγκριση και σχολιασμός
- Σύγκριση αρχιτεκτονικών συστημάτων μεταξύ τους και με τον αλγόριθμο εξομοίωσης και εξαγωγή συμπερασμάτων ως προς επιδόσεις δυνατότητες και αδυναμίες
- Μελέτη για περαιτέρω βελτιώσεις συστημάτων

1.4 Σχετική Έρευνα

Η έρευνα σε αυτόν το τομέα κάνει ακόμα τα πρώτα της βήματα. Όμως υπάρχουν ήδη εξαίρετες μελέτες σε αντίστοιχα θέματα οι οποίες δείχνουν το ενδιαφέρον και τη σημασία των κβαντικών επεξεργαστών.

Μια ενδιαφέρουσα αναφορά που αξίζει να γίνει είναι σε μια δημοσίευση των Niwa, Matsumoto και Imai με θέμα “General-Purpose parallel simulator for quantum computing” [1]. Σε αυτή τη δημοσίευση παρουσιάστηκε ένας εξομοιωτής σε γλώσσα προγραμματισμού C ο οποίος έκανε χρήση ενός υπερυπολογιστή Sun Enterprise 4500 ο οποίος είχε στη διάθεσή του 8 επεξεργαστές και 10GB μνήμης Ram. Ο συγκεκριμένος εξομοιωτής έκανε χρήση όλων των διαθέσιμων επεξεργαστών προκειμένου να βρεθούν τα επιθυμητά αποτελέσματα.

Μια άλλη εξίσου ενδιαφέρουσα δημοσίευση από τους Viamontes, Rajagopalan, Markov και Hayes με θέμα “Gate-Level Simulation of Quantum Circuits” έχει γίνει πρόσφατα [2]. Αυτή η εργασία περιλαμβάνει τη δημιουργία ενός εξομοιωτή με τη χρήση ενός εργαλείου εξομοίωσης με το όνομα QuIDD (Quantum Information Decision Diagram) και αποτιμήθηκε με τη χρήση του αγόριθμου του Grover για κβαντική αναζήτηση. Το εργαλείο QuIDD κάνει χρήση των Binary Decision Diagrams [3].

Ακόμα, σημαντική είναι η δουλειά που έγινε από τους Fujishima, Saito, Onouchi και Hoh με θέμα “High-speed processor for quantum-computing emulation and its applications” στην οποία αναπτύχθηκε ένα σύστημα εξομοίωσης για κβαντική συστήματα 16 qubit [4]. Ο συγκεκριμένος εξομοιωτής είχε αρκετά καλή απόδοση όμως είχε τη δυνατότητα εκτέλεσης αποκλειστικά για συγκεκριμένες πύλες (Hadamard και Not) και μπορούσε να χρησιμοποιηθεί σε συγκεκριμένες μόνο εφαρμογές.

Αξίζει να σημειωθεί και η δημοσίευση των A. Gothandaraman, G.Lee Warren, G. D. Patterson και R.J. Harrison πάνω στη δημιουργία ενός συστήματος επιτάχυνσης μιας κβαντικής Monte Carlo (QMC) εφαρμογής χημείας με τη χρήση δυο FPGA.[5] Με τη δουλειά τους (Hardware Acceleration of a Quantum Monte Carlo Application) πέτυχαν τριπλασιασμό σε επιδόσεις ταχύτητας για QMC εφαρμογές. Έτσι το σύστημά τους κατάφερε να εξομοιώσει 4000 άτομα με τη χρήση QMC μεθόδων.

Οι Michael T. Niemier, Arun F. Ridrigues και Peter M. Kogge με την εργασία τους, "A Potentially Implementable FPGA for Quantum Dot Cellular Automata", μελέτησαν και σχεδίασαν ένα κύκλωμα για quantum Dot Cellular Automat a με τη χρήση μιας QCA FPGA.[6] Σε αυτή την FPGA κάνοντας την κατάλληλη δρομολόγηση κάθε φορά επιτύγχαναν τα η εξομοίωση ενός αυτόματου που για κβαντικούς υπολογισμούς.

Η εργασία με θέμα "Combined Integer and Floating Point Multiplication Architecture(CIFM) for FPGAs and Its Reversible Logic Implementation" των H. Thapliyal, H. Arabnia και A. Vinod πρότεινε τη δημιουργία ενσωματωμένων μονάδων κινητής υποδιαστολής πολλ/σμού 24x24 σε FPGAs έναντι των 18x18 που υπάρχουν σήμερα.[7] Πρότεινε να αποτελούνται από μικρότερους πολλ/στές οι οποίοι θα μπορούν να δουλεύουν και ανεξάρτητα και επίσης όλοι θα μπορούν να αναστρέψουν τις διαδικασίες τους κάνοντας χρήση φαινόμενων των κβαντικών υπολογισμών.

Μια ακόμα σημαντική δημοσίευση με θέμα "FPGA Emulation of Quantum Circuits" από τους Ahmed Usman Khalid, Zeljko Zilic και Katarzyna Radecka στην οποία χρησιμοποιώντας μια FPGA εξομοίωσαν διαφόρων τύπων κβαντικά κυκλώματα.[8] Το σύστημά τους έχει τη δυνατότητα να εκτελεί μερικές συγκεκριμένες πύλες αλλά και το σύστημα να περιέχει ελάχιστα qubits μιας και το όλο σύστημα αυξάνει εκθετικά σε μέγεθος με την αύξηση των qubits. Με το σύστημά τους πέτυχαν 1000 φορές ταχύτερη εκτέλεση των αλγορίθμων του Grover και του μετασχηματισμού Fourier συγκρινόμενο με τους αλγόριθμους της βιβλιοθήκης "libquantum"[9].

Στην εργασία των Owen Callanan, Andy Nisbet, Emre Özer, James Sexton και David Gregg με θέμα "FPGA Implementation of a Lattice Quantum Chromodynamics Algorithm Using Logarithmic Arithmetic" αναπτύχθηκε σε μια Virtex II FPGA ένα πλέγμα Quantum Chromodynamics.[10] Ένα τέτοιο πλέγμα περιγράφει τις αλληλεπιδράσεις διαφόρων υποατομικών σωματιδίων. Σκοπός τους είναι να ερευνήσουν αν η χρήση FPGA και η αριθμητική λογαρίθμων μπορεί να επιτύχει υψηλές αποδόσεις σε θέματα του συγκεκριμένου τομέα. Έτσι επιτυγχάνουν μεγάλο αριθμό υπολογισμών και υψηλό χρονισμό του συστήματός τους.

Τέλος, η εργασία με θέμα "A High Speed Quantum Communication Testbed" των Carl J. Williams, Xiao Tang, Mikko Hiekkero, Julie Rouzaud, Richang Lu, Andreas Goedecke, Alan Migdall, Alan Mink, Anastase Nakassis, Leticia Pibida, Jesse Wen, Edward Hagley, and Charles W. Clark διαπραγματεύεται θέματα που αφορούν τη χρήση κβαντικής κρυπτογραφίας για εμπορική χρήση. Η συγκεκριμένη τεχνική η οποία ερευνήθηκε στη συγκεκριμένη δημοσίευση υλοποιήθηκε με τη χρήση αναδιατασσόμενης λογικής και PCB προκειμένου να ερευνηθούν περαιτέρω θέματα όπως η λειτουργικότητα και η διεπικοινωνία.

Κεφάλαιο 2

Λειτουργία των Κβαντικών Υπολογιστών

Οι κβαντικοί Υπολογιστές είναι συσκευές για υπολογισμούς, οι οποίες κάνουν απ' ευθείας χρήση κάποιων φαινόμενων κβαντικής μηχανικής, όπως την υπέρθεση και την διεμπλοκή καταστάσεων, προκειμένου να κάνουν υπολογισμούς σε δεδομένα. Στον κλασικό υπολογιστή, τα δεδομένα μετρούνται με bits, ενώ σε ένα κβαντικό υπολογιστή με qubits (**qu**antum **bits**). Η βασική αρχή των κβαντικών υπολογιστών είναι πως οι κβαντικές ιδιότητες των σωματιδίων μπορούν να χρησιμοποιηθούν για την αναπαράσταση και τη δόμηση δεδομένων, και πως οι κβαντικοί μηχανισμοί μπορούν να κατασκευάσουν ώστε να διεξάγουν λειτουργίες με αυτά τα δεδομένα. [12]

Αυτό το κεφάλαιο αποσκοπεί στην εισαγωγή του αναγνώστη στις βασικές αρχές των κβαντικών υπολογιστών. Δεν θα αναφερθούμε στο πως κατασκευάζονται τέτοιοι επεξεργαστές αλλά στο πως μπορούν να δουλέψουν αποτελεσματικά με σκοπό να μας επιφέρουν τα επιθυμητά αποτελέσαμτα.

2.1 Εισαγωγή στους Κβαντικούς Υπολογιστές

2.1.1 Τα Βασικά των Κβαντικών Υπολογιστών

Στην κβαντομηχανική, η κατάσταση ενός φυσικού συστήματος (όπως ένα ηλεκτρόνιο ή ένα φωτόνιο) περιγράφεται από ένα άνυσμα μαθηματικής μορφής και καλείται χώρος Hilbert (Hilbert space). Η αντίληψη του χώρου Hilbert εξαρτάται από το κάθε συγκεκριμένο σύστημα. Για παράδειγμα, στην περίπτωση ενός συστήματος ενός και μόνο σωματιδίου στις τρεις διαστάσεις, η κατάσταση μπορεί να περιγραφεί από μια συνάρτηση μιγαδικών στον R^3 (τρισδιάστατο χώρο) και καλείται συνάρτηση κύματος (wave function) . Ιδιαίτερη σημασία έχει το ότι οι κβαντικές καταστάσεις μπορούν να βρίσκονται σε υπέρθεση των βασικών καταστάσεων. Η χρονική εξέλιξη του συστήματος είναι μοναδιαία κάτι το οποίο σημαίνει πως ολόκληρη η διαδικασία είναι αντιστρέψιμη.

Ένας κλασικός υπολογιστής έχει μνήμη η οποία απαρτίζεται από bits, οπού κάθε bit έχει την τιμή 1 ή 0. Η συσκευή υπολογίζει με το να διαχειρίζεται αυτά τα bits. Για παράδειγμα με το να μεταφέρει αυτά τα bit σε λογικές πύλες και να τα γράφει πάλι πίσω στη μνήμη. Ένας κβαντικός υπολογιστής συγκρατεί ένα σύνολο από qubits. Το qubit μπορεί να έχει την τιμή 1, 0 ή μια υπέρθεση αυτών. Ο κβαντικός υπολογιστής λειτουργεί με το να διαχειρίζεται αυτά τα qubits. Για παράδειγμα μεταφέροντας τα qubit από τη μνήμη σε κβαντικές πύλες και πάλι πίσω.

Τα qubits για ένα κβαντικό υπολογιστή μπορούν να υλοποιηθούν χρησιμοποιώντας σωματίδια με δυο καταστάσεις spin «πάνω» και «κάτω» (τυπικά συμβολίζονται $|0\rangle$ και $|1\rangle$) Σε ένα κλασικό υπολογιστή με ένα καταχωρητή 3 bit σε κάθε χρονική στιγμή τα bit του καταχωρητή θα βρίσκονται σε μια καθορισμένη κατάσταση (π.χ. «101»). Σε ένα κβαντικό υπολογιστή τα qubits μπορούν να βρίσκονται σε υπέρθεση όλων των κλασικά επιτρεπτών καταστάσεων. Στην πραγματικότητα, ο καταχωρητής περιγράφεται από μια συνάρτηση κύματος (wave function):

$$|\psi\rangle = a|000\rangle + b|001\rangle + c|010\rangle + d|011\rangle + e|100\rangle + f|101\rangle + g|110\rangle + h|111\rangle$$

Όπου οι συντελεστές *a, b, c, …* είναι μιγαδικοί αριθμοί των οποίων τα μέτρα τους είναι η πιθανότητα να μετρηθούν τα qubits σε κάθε κατάσταση. Συνεπώς, $|c|^2$ είναι η πιθανότητα να μετρηθεί το qubit στην κατάσταση 010. Το ότι αυτοί οι αριθμοί είναι μιγαδικοί είναι μεγάλης σημασίας γιατί οι φάσεις των αριθμών μπορούν εποικοδομητικά ή καταστροφικά να επηρεάσουν το ο ένας τον άλλο, κάτι που αποτελεί μια σημαντική ιδιότητα των κβαντικών αλγορίθμων. [13]

Ένας *n* qubit καταχωρητής απαιτεί 2^n μιγαδικούς αριθμούς για να καθοριστεί η κατάστασή του. Σαν αποτέλεσμα, ο αριθμός των κλασικών καταστάσεων αυξάνει εκθετικά με τον αριθμό των qubit. Επίσης οι συντελεστές δεν είναι ανεξάρτητοι, αφού οι πιθανότητες πρέπει να αθροίζουν στο 1. Η αναπαράσταση δεν είναι και αυτή μοναδική αφού δεν υπάρχει φυσικός διαχωρισμός μεταξύ ενός συγκεκριμένου καταχωρητή και ενός όμοιου όπου όλα τα πλάτη έχουν πολλαπλασιαστεί με την ίδια φάση.

2.1.2 Υπολογισμοί Κβαντικών Υπολογιστών

Στο παραπάνω παράδειγμα τα περιεχόμενα του καταχωρητή με τα qubit μπορούμε να τα φανταστούμε σαν ένα 8-διάστατο άνυσμα μιγαδικών. Ένας αλγόριθμος για ένα κβαντικό υπολογιστή πρέπει να αρχικοποιεί το άνυσμα αυτό σε μια συγκεκριμένη μορφή. Σε κάθε βήμα του αλγόριθμου αυτό το άνυσμα πολλαπλασιάζεται με ένα μοναδιαίο πίνακα Η μοναδιάια φύση του πίνακα εξασφαλίζει το ότι ο πίνακας θα είναι αντιστρέψιμος και έτσι όλο το βήμα θα μπορεί να αντιστραφεί.

Κατά τον τερματισμό του αλγορίθμου το 8-διάστατο άνυσμα θα αποθηκευτεί στον καταχωρητή και πρέπει με κάποιο τρόπο να διαβαστεί από τον κβαντικό καταχωρητή με μια κβαντική μέτρηση (quantum measurement). Όμως με βάση τους νόμους της κβαντομηχανικής, αυτή η μέτρηση θα αποδώσει μια τυχαία 3-bit ποσότητα και θα καταστρέψει την αποθηκευμένη κατάσταση.

Ένας κβαντικός αλγόριθμος δημιουργείται με την κατάλληλη τοποθέτηση μοναδιαίων λειτουργιών. Για ένα δεδομένο αλγόριθμο, οι λειτουργίες αυτές θα εκτελεστούν πάντα στην ίδια ακριβή σειρά μιας και δεν υπάρχουν προτάσεις “If Then” για να εναλλάσσουν τη ροή αφού δεν γίνεται να διαβαστεί η κατάσταση πριν την τελευταία μέτρηση. Παρόλα αυτά, υπάρχει η δυνατότητα να εφαρμοστούν πύλες υπό όρους (conditional gate) όπως η controlled NOT πύλη, ή αλλιώς CNOT και άλλες.

2.1.3 Η δύναμη των Κβαντικών Υπολογιστών

Η παραγοντοποίησει ακεραίων πιστεύεται πως είναι μη πρακτική με ένα συνηθισμένο υπολογιστή για μεγάλους αριθμούς. Σε σύγκριση, ένας κβαντικός επεξεργαστής θα μπορούσε να λύσει αυτό το πρόβλημα σχετικά εύκολα με τη χρήση του αλγόριθμου του Shor [14]. Μπορεί επίσης να λύσει εύκολα το πρόβλημα που καλείται “discrete logarithm”. Αυτή η δυνατότητα θα επέτρεπε σε ένα κβαντικό υπολογιστή να «σπάσει» πολλά κρυπτογραφικά συστήματα που χρησιμοποιούνται σήμερα τα οποία χρησιμοποιούνται για να προστατέψουν ασφαλές σελίδες στο διαδίκτυο, κρυπτογραφημένα e-mail και πολλούς άλλους τύπους δεδομένων. Ο μοναδικός τρόπος αύξησης της ασφάλειας ενός αλγορίθμου, όπως αυτός του RSA, είναι η αύξηση του μεγέθους του κλειδιού και η ελπίδα να μην κατασκευάσει κάποιος έναν ισχυρό κβαντικό υπολογιστή. Ο μοναδικός τρόπος για να ξεπεραστεί αυτό το δίλλημα είναι η χρήση κάποιου είδους κβαντικής κρυπτογραφίας. Άλλες κοινές εφαρμογές οι οποίες έχουν αναπτυχθεί είναι τα κβαντικά πεπερασμένα αυτόματα, οι κβαντικά μηχανές Turing, οι γενικές μηχανές Turing, η κβαντική παραγωγή κλειδιών για κρυπτογραφία, κβαντικά πρωτόκολλα κρυπτογραφίας, αλγόριθμοι για μέτρηση κα αναζήτηση, και πολλοί άλλοι [15].

Τα Ιδιαίτερα προτερήματα των κβαντικών υπολογιστών που έχουν ανακαλυφθεί μέχρι σήμερα έναντι των κλασσικών υπολογιστών, αφορούν μόνο το πρόβλημα της παραγοντοποήησης και του διακριτού λογάριθμου. Παρόλα αυτά δεν υπάρχει απόδειξη πως αυτό το προτέρημα είναι πραγματικό μιας και ένας όμοιας ταχύτητας αλγόριθμος μπορεί να ανακαλυφθεί για τους κλασικούς υπολογιστές. Υπάρχει ένα πρόβλημα στο οποίο οι κβαντικοί υπολογιστές έχουν ένα μικρό όμως σημαντικό προτέρημα. Αυτό είναι στην αναζήτηση σε κβαντικές βάσεις δεδομένων το οποίο μπορεί να επιλυθεί με τη χρήση του αλγόριθμου του Grover [16]. Σε αυτή την περίπτωση τα οφέλη των κβαντικών υπολογιστών είναι προφανή. Αυτό αποδεικνύει πως οι κβαντικοί υπολογιστές είναι καλύτεροι από τους κλασικός σε τουλάχιστον ένα πρόβλημα.

Μια σύγκριση μεταξύ των κλασσικών κλάσεων πολυπλοκότητας [18] θα δείξει καλύτερα το πόσο διαφέρει η ισχύς μεταξύ κλασικών και κβαντικών υπολογιστών:

Classical Complexity Class	Intuitive Meaning	Examples
P (ή PTIME)	Polynomial time: the running time of the algorithm is, in the worst case, a polynomial in the size of the input. All problems in P are tractable.	Multiplication
NP	Nondeterministic polynomial time: a candidate answer can be checked for correctness in polynomial time.	Factoring Complex Numbers
NP-complete	A subset of problems in NP that can be mapped into one another in polynomial time. If just one of the problems in this class is shown to be tractable, then they must all be tractable.	Scheduling Satisfiability Traveling Salesman Problem
ZPP	Can be solved, with certainty, by PTMs in average case polynomial time.	
BPP	Can be solved in polynomial time by PTMs with probability > 2/3. Probability of success can be made arbitrary close to 1 by iterating the algorithm a certain number of times.	Problems in BPP are tractable

Πίνακας 1.1: *Μερικές κλασικές κλάσεις πολυπλοκότητας και οι ορισμοί τους.*

Quantum Class	Class in Computational Problems that can…	Relationship to Classical Complexity Classes
QP	… be solved, with certainty, in worst case polynomial time by a quantum computer.	$P \subset QP$ (the quantum computer can solve more problems in worst-case polynomial time than the classical computer).
BQP	… be solved in worst case polynomial time by a quantum computer with probability > 2/3 (i.e., he probability of error is bounded, hence the "B" in "BQP")	BPP $\subseteq$ BQP $\subseteq$ PSPACE (i.e., it is not known whether QTMs are more powerful than PTMs). Shor proved factoring is in BQP.
ZQP	… be solved with zero error probability in expected polynomial time by a quantum computer.	ZPP $\subset$ ZQP

Πίνακας 1.2: *Μερικές νέες, κλάσεις κβαντικής πολυπλοκότητας, οι ορισμοί τους και οι σχέσεις τους με τις κλασικές κλάσης*.

2.2 Η σημειογραφία Bra/Ket

Η σημειογραφία Bra/Ket είναι ο σύνηθες τρόπος για να περιγραφούν οι κβαντικές καταστάσεις στην κβαντομηχανική. Αναφέρεται έτσι αφού το εσωτερικό γινόμενο δυο καταστάσεων δηλώνεται από ένα bracket $\langle x|\,|y\rangle$, το οποίο αποτελείται από ένα αριστερό μέρος, $\langle x|$ το οποίο καλείται Bra, και ένα δεξιό, $|y\rangle$, το οποίο καλείται Ket. Η σημειογραφία επινοήθηκε από τον Paul Dirac, και είναι γνωστή και ως η σημειογραφία του Dirac [17]. Τα ποιο συχνά χρησιμοποιούμενα ανύσματα είναι:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Και το εσωτερικό γινόμενο δυο ανυσμάτων $|x\rangle$ και $|y\rangle$ συμβολίζεται με:

$$\langle x \| y \rangle \text{ ή } \langle x | y \rangle$$

Και το εξωτερικό γινόμενο δυο ανυσμάτων $|x\rangle$ και $|y\rangle$ συμβολίζεται με:

$$|x\rangle \; \langle y|$$

Το εσωτερικό και το εξωτερικό γινόμενο δυο ανυσμάτων βρίσκεται με απλό πολλαπλασιασμό πινάκων:

$$\langle 0|1\rangle = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 \cdot 0 + 0 \cdot 1 = 0$$

$$|0\rangle\langle 1| = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

2.3 Το Κβαντικό Bit (Qubit)

Ένα κβαντικό bit (quantum bit), ή qubit (ή και qbit) είναι μια μονάδα κβαντικής πληροφορίας. Αυτή η πληροφορία περιγράφεται από ένα δυο επιπέδων άνυσμα κατάστασης (state vector), το οποίο είναι ισοδύναμο με ένα δυσδιάστατο άνυσμα στους μιγαδικούς αριθμούς.

Το bit είναι η βάση της υπολογιστικής πληροφορίας. Ανεξάρτητα από τη φυσική του υπόσταση, έχει πάντοτε την τιμή 0 ή 1. Αυτό ομοιάζει με ένα διακόπτη ο οποίος είναι είτε στη θέση on είτε στη θέση off. Το qubit έχει μερικές ομοιότητες με το bit, αλλά γενικά διαφέρει κατά πολύ. Όπως και το bit, το qubit μπορεί να έχει δυο πιθανές τιμές, το 0 και το 1. Η διαφορά είναι πως ενώ το bit πρέπει να έχει ακριβώς μια από τις δυο αυτές τιμές, το qubit μπορεί να έχει την τιμή 0, ή την 1 ή μια υπέρθεση αυτών των δυο.

Οι καταστάσεις στις οποίες ένα qubit μπορεί να μετρηθεί είναι γνωστές ως βασικές καταστάσεις. Όπως συνηθίζεται σε κάθε κβαντική κατάσταση η σημειογραφεία του Dirac ή Bra/Ket χρησιμοποιείται για την αναπαράστασή τους. Αυτό σημαίνει πως οι δυο υπολογιστικές βάσεις κατάστασης γράφονται $|0\rangle$ και $|1\rangle$ (“ket 0” and “ket 1”).

Μια ακέραια κβαντική κατάσταση είναι μια γραμμική υπέρθεση των δυο παραπάνω καταστάσεων. Αυτό σημαίνει πως ένα qubit μπορεί να αναπαρασταθεί ως ένας γραμμικός συνδυασμός των $|0\rangle$ και $|1\rangle$. Δηλαδή:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \text{ ή } \psi = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Όπου a και b είναι πιθανότητες και μπορούν να είναι μιγαδικοί αριθμοί. Κατά τη μέτρηση ενός qubit, η πιθανότητα να μετρήσουμε την τιμή $|0\rangle$ είναι $|\alpha|^2$ και η πιθανότητα να μετρήσουμε $|1\rangle$ είναι $|\beta|^2$. Αφού τα τετράγωνα των πλατών είναι πιθανότητες θα πρέπει να αθροίζουν στο ένα. Άρα για τα a και b θα πρέπει να ισχύει πως:

$$|\alpha|^2 + |\beta|^2 = 1$$

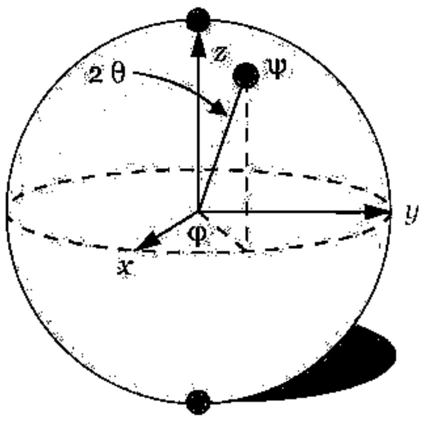
έτσι ώστε να εξασφαλισθεί πως θα μετρηθεί είτε η μια τιμή είτε η άλλη.

Αυτή η ικανότητα των κβαντικών συστημάτων να βρίσκονται ταυτόχρονα σε μια μίξη όλων των πιθανών καταστάσεων καλείται αρχή της υπέρθεσης. Αυτό μπορεί να φανεί εύκολα με τη χρήση της σφαίρας του Bloch. Ο χώρος καταστάσεων ενός καταχωρητή qubit μπορεί να αναπαρασταθεί γεωμετρικά από μια σφαίρα του Bloch. Κάθε κατάσταση ψ μπορεί να γραφεί ως μια υπέρθεση των ανυσμάτων ket $|0\rangle$ και $|1\rangle$. Αφού οι φάσεις δεν επιδρούν με τη φυσικά κατάσταση μπορούμε να την αναπαραστήσουμε έτσι ώστε ο παράγων $|0\rangle$ να είναι πραγματικός και μη αρνητικός. [15]

Έτσι τελικά το ψ μπορεί να γραφτεί ως:

$$\psi = \cos\theta|0\rangle + \varepsilon^{i\phi}\sin\theta = \cos\theta|0\rangle + (\cos\phi + i\sin\phi)\sin\theta|1\rangle$$

$$\text{με} \quad 0 \le \theta \prec \pi/2, 0 \le \phi \prec 2\pi$$



Εικόνα 2.1: *Η σφαίρα του Bloch*

Όλες οι καταστάσεις αυτές είναι γραμμικά ανύσματα στο χώρο. Αυτό έχει ως αποτέλεσμα να υπακούν σε όλες τις ιδιότητες των ανυσμάτων [18]. Έτσι είναι για τα ανύσματα $|a\rangle, |b\rangle, |\gamma\rangle$ Ε V^n και $|a\rangle, |b\rangle, |\gamma\rangle \subseteq C^n$ με

$|\alpha\rangle = (\alpha_1, ..., \alpha_n)$

$|\beta\rangle = (\beta_1, ..., \beta_n)$

$|\gamma\rangle = (\gamma_1, ..., \gamma_n)$

Ισχύουν:

$$|\gamma\rangle = |\alpha\rangle + |\beta\rangle \Leftrightarrow \gamma_i = \alpha_\iota + \beta_i \ , \ \mu\varepsilon \ i = 1, ..., n$$

$$|\alpha\rangle + |\beta\rangle = |\beta\rangle + |\alpha\rangle$$

$$|\alpha\rangle + (|\beta\rangle + |\gamma\rangle) = (|\alpha\rangle + |\beta\rangle) + |\gamma\rangle$$

Για c, d Ε C ισχύουν:

$$c(|\alpha\rangle + |\beta\rangle) = c|\alpha\rangle + c|\beta\rangle$$

$$(c+d)|\alpha\rangle = c|\alpha\rangle + d|\alpha\rangle$$

$$(cd)|\alpha\rangle = c(d|\alpha\rangle)$$

Αντίστοιχα με τα παραπάνω ορίζονται και συστήματα με παραπάνω των δύο καταστάσεων οπού κάνουν χρήση πολλαπλών qubits. Για παράδειγμα ένα άνυσμα κατάστασης ψ με δυο qubits περιγράφεται ως εξής:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

Με τα

$$|00\rangle = |0\rangle|0\rangle = |0,0\rangle$$

$$|01\rangle = |0\rangle|1\rangle = |0,1\rangle$$

$$|10\rangle = |1\rangle|0\rangle = |1,0\rangle$$

$$|11\rangle = |1\rangle|1\rangle = |1,1\rangle$$

Και

$$\alpha_{00} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \ , \ \alpha_{01} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \ , \ \alpha_{10} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \ , \ \alpha_{11} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Και εδώ όπως και πριν οι πιθανότητες θα πρέπει να αθροίζουν στο 100% ή στο 1, δηλαδή $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$. Τώρα η πιθανότητα να μετρήσουμε το πρώτο qubit και να λάβουμε την τιμή 0 είναι $|\alpha_{00}|^2 + |\alpha_{01}|^2$ ενώ για το δεύτερο qubit την τιμή 0 η πιθανότητα θα είναι $|\alpha_{00}|^2 + |\alpha_{10}|^2$ κ.ο.κ.

2.4 Κβαντικές Πύλες

Μέχρις στιγμής έχουμε δει πως περιγράφεται μια κβαντική κατάσταση, αλλά όχι πως κατασκευάζουμε μια επιθυμητή κατάσταση. Έτσι οι κβαντικές πύλες (ή λογικές κβαντικές πύλες) είναι η αναλογία των λογικών πυλών των κλασικών υπολογιστών στους κβαντικούς όμως υπολογιστές. Οι κβαντικές λογικές πύλες, σε αντίθεση με τις κλασικές λογικές πύλες, είναι αντιστρέψιμες και έτσι η όλη διαδικασία μπορεί να αντιστραφεί. Η πιο απλή αλλά και βασική πύλη που θα δείξουμε είναι αυτή της αντιστροφής. Αυτή η πύλη μετασχηματίζει το $|0\rangle$ σε $|1\rangle$ και αντίστροφα. Δηλαδή η πύλη X θα είναι:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Πολλαπλασιάζοντας το X με τα $|0\rangle$ και $|1\rangle$ θα έχουμε:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

Από τα δυο παραπάνω βλέπουμε πως η πύλη X κάνει την αντιστροφή όπως θέλαμε. Αυτή ονομάζεται κβαντική πύλη της αντιστροφής (quantum NOT gate). Στην πραγματικότητα αυτή η πύλη δρα γραμμικά. Δηλαδή μετασχηματίζει την κατάσταση $\alpha|0\rangle + \beta|1\rangle$ σε $\alpha|1\rangle + \beta|0\rangle$:

$$\mathrm{X}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Μπορούμε να συμβολίσουμε την παραπάνω διαδικασία ως εξής:

$$\alpha|1\rangle + \beta|0\rangle \xrightarrow{\mathrm{X}} \alpha|0\rangle + \beta|1\rangle$$

Μπορούμε να διαπιστώσουμε πως ακόμη και μετά την εφαρμογή της πύλης θα ισχύει πως $|\alpha|^2 + |\beta|^2 = 1$. Αυτό οφείλεται στο γεγονός ότι η πύλη X είναι μοναδιαία, δηλαδή:

$$\mathrm{X}^{\mathrm{T}}\mathrm{X} = \mathrm{X}\mathrm{X}^{\mathrm{T}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathrm{I}$$

Αυτό συμβαίνει με κάθε κβαντική πύλη U προκειμένου να διατηρούνται οι ιδιότητες της κβαντομηχανικής:

$$U^{\mathrm{T}}U = UU^{\mathrm{T}} = \mathrm{I}$$

Ακόμα μια Βασική πύλη είναι η πύλη Hadamard

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

η οποία είναι μοναδιαία αφού $H = H^T$ και $H^2 = I$. Αυτή η πύλη δημιουργεί μια κατάσταση ίσιων υπερθέσεων όταν εφαρμόζεται στις βασικές καταστάσεις. Αυτό φένεται παρακάτω, μιας και οι πιθανότητες να μετρήσουμε 0 και 1 είναι ίσες.

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

Υπάρχουν πολλές πύλες που εκτελούν διαφόρων ειδών πράξεις. Μερικές από αυτές παρουσιάζονται παρακάτω:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \alpha|0\rangle + \beta|1\rangle \xrightarrow{I} \alpha|0\rangle + \beta|1\rangle$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \alpha|0\rangle + \beta|1\rangle \xrightarrow{Z} \alpha|0\rangle - \beta|1\rangle$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \alpha|0\rangle + \beta|1\rangle \xrightarrow{Y} bi|0\rangle + ai|1\rangle$$

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad \alpha|0\rangle + \beta|1\rangle \xrightarrow{S} \alpha|0\rangle + \beta i|1\rangle$$

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \quad \alpha|0\rangle + \beta|1\rangle \xrightarrow{T} \alpha|0\rangle + \beta e^{i\pi/4}|1\rangle$$

2.5 Ελεγχόμενες Πύλες

Παραπάνω αναφέραμε πως αφού στους κβαντικούς αλγορίθμους δεν υπάρχει η δυνατότητα ελέγχου ροής, θα πρέπει να υπάρχουν άλλες μέθοδοι για τον έλεγχο ροής. Αυτές είναι οι ελεγχόμενες πύλες. Αν για παράδειγμα θέλουμε, σε ένα σύστημα δυο qubit, να εφαρμόσουμε μια πύλη στο δεύτερο qubit, ανάλογα με την τιμή του πρώτου qubit. Αν θέλουμε να εφαρμόσουμε μια κβαντική πύλη NOT στο δεύτερο qubit μόνο αν το πρώτο βρίσκεται στην κατάσταση $|1\rangle$ τότε μπορούμε να χρησιμοποιήσουμε την πύλη ελεγχόμενη NOT (controlled NOT gate ή CNOT). Η CNOT περιγράφεται από τον πίνακα:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Είναι εύκολο να διαπιστωθεί πως η παραπάνω πύλη CNOT επιτελεί αυτό το μετασχηματισμό:

$$|00\rangle \to |00\rangle,\ |01\rangle \to |01\rangle,\ |10\rangle \to |11\rangle,\ |11\rangle \to |10\rangle$$

Γενικά μπορούμε να μετατρέψουμε οποιαδήποτε απλή πύλη U, σε ελεγχόμενη πύλη CU (controlled-U), αν το πρώτο qubit είναι στην κατάσταση $|1\rangle$, με τη χρήση ενός μοναδιαίου πίνακα *I*. Δηλαδή η πύλη CU κατασκευάζεται ως εξής:

$$CU = \begin{bmatrix} I & 0 \\ 0 & U \end{bmatrix}$$

Με *I* το μοναδιαίο 2x2 πίνακα, το U μοναδιαία 2x2 πύλη και τα «0» ένα 2x2 πίνακα με όλα τα στοιχεία μηδενικά. Η πύλη CU πρέπει είναι και αυτή, όπως και οι απλές κβαντικές πύλες, μοναδιαία κάτι το οποίο συμβαίνει αφού και η U είναι μοναδιαία.

2.6 Η Πράξη της Μέτρησης

Νωρίτερα αναφέραμε πως η πράξη της μέτρησης σε ένα κβαντικό σύστημα διαφέρει κατά πολύ σε σχέση με μια κλασική μέτρηση όπως έχουμε συνηθίσει. Μια μέτρηση, σε ένα τέτοιο σύστημα, το κάνει πλέον μη κλειστό το οποίο σημαίνει πως δεν είναι πλέον αποτέλεσμα μοναδιαίων βημάτων. Άρα η μέτρηση σε ένα κβαντικό σύστημα δεν είναι μοναδιαία.

Οι κβαντικές μετρήσεις περιγράφονται από μια συλλογή $\{M_m\}$ πράξεων μέτρησης. Ο δείκτης m αναφέρεται στο αποτέλεσμα της μέτρησης που μπορεί να προκύψει. Αν η κατάσταση του συστήματος πριν τη μέτρηση ήταν $|\psi\rangle$, τότε η πιθανότητα να μετρήσουμε m είναι:

$$p(m) = \langle\psi|M_m^T M_m|\psi\rangle$$

και η κατάσταση μετά τη μέτρηση θα είναι:

$$\frac{M_m|\psi\rangle}{\sqrt{p(m)}} = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^T M_m|\psi\rangle}}$$

Επίσης θα πρέπει να ισχύει πως:

$$\sum_m M_m^T M_m = 1$$

Ένα απλό αλλά κατανοητό παράδειγμα είναι η μέτρηση σε ένα κβαντικό σύστημα ενός qubit το οποίο βρίσκεται στην κατάσταση $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$.

$$M_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$M_0 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Οι πιθανότητες να μετρήσουμε 0 και 1 είναι αντίστοιχα:

$$p(0) = \langle\psi|M_0^T M_0|\psi\rangle = \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = |\alpha|^2$$

$$p(1) = \langle\psi|M_1^T M_1|\psi\rangle = \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = |\beta|^2$$

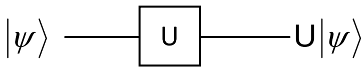
Οι καταστάσεις μετά τη μέτρηση θα είναι με βάση τα παραπάνω:

$$\frac{M_0|\psi\rangle}{\sqrt{p(0)}} = \frac{1}{|\alpha|} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{|\alpha|} \begin{bmatrix} \alpha \\ 0 \end{bmatrix} = \frac{\alpha}{|\alpha|}|0\rangle$$

$$\frac{M_1|\psi\rangle}{\sqrt{p(1)}} = \frac{1}{|\beta|} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{|\beta|} \begin{bmatrix} 0 \\ \beta \end{bmatrix} = \frac{\beta}{|\beta|}|1\rangle$$

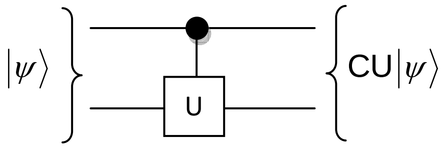
2.7 Κβαντικά Κυκλώματα

Όπως οι σχεδιαστές κλασικών κυκλωμάτων χρησιμοποιούν ειδικά σύμβολα για τις βασικές λογικές πύλες για να σχεδιάζουν στο χαρτί, έτσι συμβαίνει και για τους σχεδιαστές κβαντικών κυκλωμάτων. Ένα κβαντικό κύκλωμα είναι μια συλλογή από κβαντικές πύλες ακυκλικά συνδεμένες με «κβαντικά καλώδια». Μια κβαντική πύλη συμβολίζεται με το όνομα της πύλης μέσα σε ένα τετράγωνο κουτί. Για παράδειγμα αν θέλουμε να εφαρμόσουμε την πύλη U σε ένα qubit του οποίου η κατάσταση ήταν η $|\psi\rangle$ αυτό συμβολίζεται με:



Εικόνα 2.2: *Εφαρμογή της πύλης U σε ένα qubit σε κατάσταση* $|\psi\rangle$

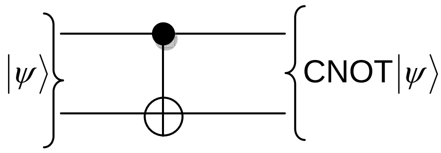
Μια πύλη controlled-U (CU) που δρα σε ένα σύστημα με δυο qubits που βρίσκεται στην κατάσταση $|\psi\rangle$ συμβολίζεται ως εξής:



Εικόνα 2.3: *Εφαρμογή της πύλης controlled-U σε ένα σύστημα δυο qubit σε κατάσταση* $|\psi\rangle$

Το qubit με τη μαύρη κουκίδα είναι το qubit ελέγχου (control qubit), ενώ το άλλο το qubit στόχου. Το ότι η κουκίδα είναι γεμάτη σημαίνει πως η πύλη εκτελείται αν το qubit ελέγχου έχει την τιμή 1, αλλιώς αν η κουκίδα ήταν λευκή τότε θα σήμαινα πως η πύλη εκτελείται αν το qubit ελέγχου είναι 0.

Επειδή η πύλη CNOT είναι ιδιαίτερης σημασίας έχει ένα ειδικό συμβολισμό:

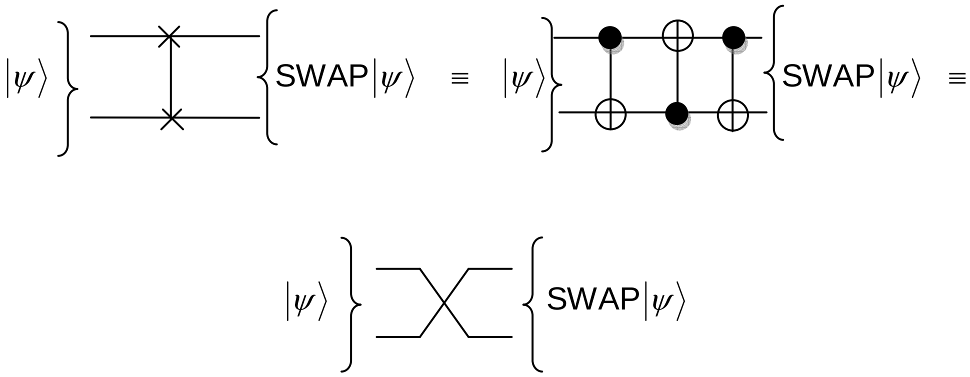


Εικόνα 2.4: *Συμβολισμός πύλης CNOT*

Μια άλλη πύλη είναι αυτή της αντιστροφής δυο qubit. Ο πίνακας της πύλης αυτής SWAP είναι:

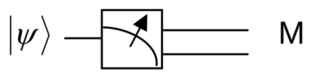
$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Καθώς και το κύκλωμά της είναι:



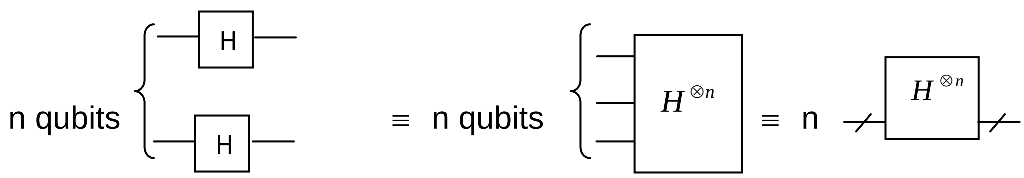
Εικόνα 2.5: *Η πύλη SWAP*

Χρειαζόμαστε επίσης να συμβολίσουμε και την πράξη της μέτρησης ενός qubit σε ένα κβαντικό κύκλωμα. Χρειαζόμαστε επίσης να διαχωρίσουμε το κλασικό bit με το qubit με το να το σχεδιάζουμε με μια διπλή γραμμή:

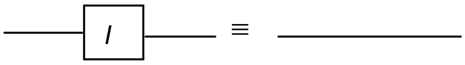


Εικόνα 2.6: *Το κυκλωματικό σύμβολο της πράξης της μέτρησης*

Μπορούμε να κάνουμε την αναπαράσταση ενός κυκλώματος ακόμα πιο συμπαγή με διάφορους τρόπους όπως φαίνονται εδώ:

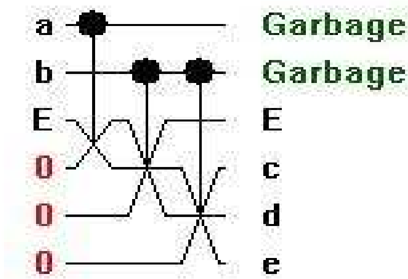


Εικόνα 2.7: *Σύντομη περιγραφή πολλαπλών πυλών H*

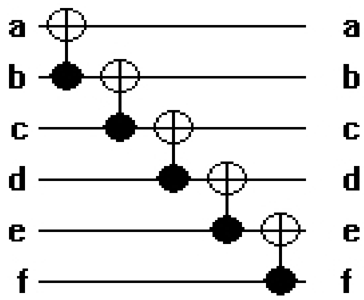


Εικόνα 2.8: *Σύντομη περιγραφή της ταυτοτικής πύλης*

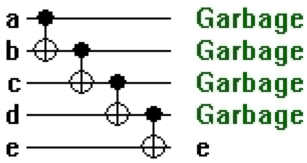
Διάφορα κβαντικά κυκλώματα με τη λειτουργία τους παρουσιάζονται εδώ [19]:



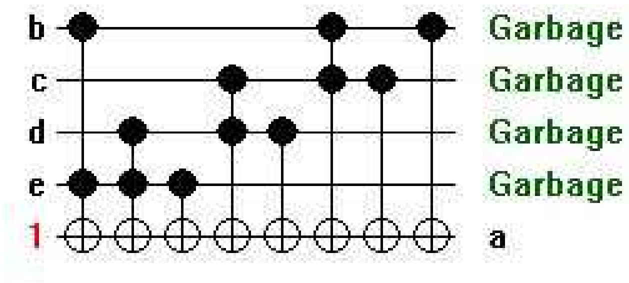
Εικόνα 2.9*: 2-προς-4 αποκωδικοποιητής*



Εικόνα 2.10: *Κωδικοποίηση Gray*



Εικόνα 2.11: *Κλασική πύλη EXOR*



Εικόνα 2.12: *Αλγόριθμος του Groove (ο a είναι 1 αν και μόνον αν ο δυαδικός αριθμός είναι ακέραιο πολλαπλάσιο του 5)*

Κεφάλαιο 3

Μοντελοποίηση Συστήματος με χρήση Αλγορίθμου

Η μοντελοποίηση ενός συστήματος πριν την κατασκευή του μπορεί να προσφέρει χρήσιμες πληροφορίες για τη συμπεριφορά, τον τρόπο λειτουργίας αλλά και τις δυνατότητες ενός συστήματος. Έτσι πριν την ανάπτυξη του πραγματικού συστήματος, η μοντελοποίηση είναι ένα πολύ βασικό στάδιο, από το οποίο θα πρέπει να προκύπτουν χρήσιμα συμπεράσματα.

Για την βοήθεια στην κατανόηση του τρόπου λειτουργίας των κβαντικών επεξεργαστών αλλά και για τον έλεγχο ορθότητας των αρχιτεκτονικών συστημάτων που σχεδιάστηκαν κατασκευάστηκε ένας αλγόριθμος εξομοίωσης κβαντικών υπολογισμών. Αυτός ο αλγόριθμος μοντελοποιεί το πρόβλημα και προσφέρει πολύτιμες πληροφορίες για τις ανάγκες σε υπολογιστικούς πόρους της εξομοίωσης.

3.1 Προδιαγραφές Συστήματος Μοντελοποίησης

Το σύστημα που θα μοντελοποιηθεί με αλγόριθμο θα πρέπει να είναι σε θέση να εξομοιώνει πολλές από τις λειτουργίες ενός κβαντικού επεξεργαστή. Επίσης θα πρέπει να είναι εύκολα παραμετροποιήσιμο για τη μελέτη διαφόρων συστημάτων και να μπορεί να διαβάζει μεγάλες ποσότητες δεδομένων από κάποιο αρχείο για εκτέλεση λειτουργιών.

Η αρχικοποίηση θα γίνεται εύκολα αφού συνήθως το άνυσμα καταστάσεων έχει σε όλες τις θέσεις 0 εκτός από μια που έχει την τιμή 1. Η πύλες που θα πρέπει να εκτελεστούν θα πρέπει να είναι αποθηκευμένες σε ένα αρχείου με συγκεκριμένη μορφή και το πρόγραμμα θα πρέπει να τις διαβάζει και να τις εκτελέσει σειριακά, αφού πρώτα τις ελέγξει για την ορθότητά τους. Η μορφή που θα πρέπει να έχει είναι η εξής:

Τύπος πύλης (Controlled Gate (c), Hadamard (h) …)
Πύλη (8 αριθμοί double από το 0 έως το 1 διαχωριζόμενοι από ένα χαρακτήρα κενού διαστήματος)
Κύκλωμα πύλης (Συμβολοσειρά η οποία αποτελείται αποκλειστικά από τους χαρακτήρες “0”, “1”, “X”, “G” και έχει μήκος όσο το πλήθος των qubit του συστήματος που προσομοιώνουμε)

Για παράδειγμα ένα κύκλωμα όπως αυτό του σχήματος 2.10 θα μπορούσε, με βάση την παραπάνω κωδικοποίηση, να αναπαρασταθεί ως εξής:

c
0.00.0 1.0 0.0 1.0 0.0 0.0 0.0
G 1 X X X X
c
0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
X G 1 X X X
c
0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
X X G 1 X X
c
0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
X X X G 1 X
c
0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
X X X X G 1

Ένα τυχαίο κύκλωμα με πύλες Hadamard και Control θα είχε την εξής περιγραφή:

c
0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
X 1 G 1 0 X
h
0.707106781 0.0 0.707106781 0.0 0.707106781 0.0 -0.707106781 0.0
X X G X X X
h
0.707106781 0.0 0.707106781 0.0 0.707106781 0.0 -0.707106781 0.0
X X X G X X
c
0.00.0 1.0 0.0 1.0 0.0 0.0 0.0
X 1 X 1 G X
c
0.00.0 1.0 0.0 1.0 0.0 0.0 0.0
X 1 0 X 0 G

Στην περίπτωση της πράξης της μέτρησης τοποθετείται ως γράμμα πύλης το “m” και στην επόμενη γραμμή τοποθετείται ο αριθμός του qubit το οποίο θέλουμε να μετρηθεί. Για παράδειγμα για ένα σύστημα με 4 qubit:

m
3
m
0
m
1

Το σύστημα που θα μοντελοποιηθεί θα πρέπει να εκτελεί όλες τις πύλες που υπάρχουν σε συγκεκριμένο αρχείο στην σειρά στην οποία βρίσκονται και να εντοπίζει τυχόν λάθη. Επίσης, θα μπορεί να δείχνει κατά τη διάρκεια εκτέλεσης τι εκτελείται κάθε στιγμή και στο τέλος να εξάγει σε αρχείο τα αποτελέσματα. Τέλος, ο αλγόριθμος που θα χρησιμοποιηθεί για την μοντελοποίηση θα πρέπει να είναι απλός, κατανοητός, αλλά και να εκτελείται γρήγορα προκειμένου να μπορούμε αργότερα να τον συγκρίνουμε με το σύστημα που θα δημιουργήσουμε, καθώς και να έχει αναπτυχθεί σε μια ευρέως γνωστή γλώσσα προγραμματισμού η οποία θα τρέχει γρήγορα.

3.2 Μεθοδολογία Υλοποίησης Συστήματος Μοντελοποίησης

Για την υλοποίηση του αλγόριθμου εξομοίωσης ενός κβαντικού επεξεργαστή θα πρέπει να χρησιμοποιήσουμε ένα αλγόριθμο αρκετά γρήγορο προκειμένου να έχουμε έναν αλγόριθμο ο οποίος θα μπορεί να τρέξει κυκλώματα μεγάλου μεγέθους και πολλών qubit σε ένα κλασικό υπολογιστή. Έτσι διαλέξαμε τον αναδρομικό αλγόριθμο που παρουσίασε ο V.Samoladas για την διαχείριση της αναπαράστασης των κβαντικών καταστάσεων [20]. Ο βασικός αλγόριθμος περιγράφεται από τις παρακάτω εξισώσεις:

$$\{s, U, s'\}Q = T(s, 1, [\varepsilon, U, s'], Q)$$

$$T(c \cdot s, d, A, Q_0 \,\&\, Q_1) = \begin{cases} T(s, d, A, Q_0) \,\&\, T(s, d, A, Q_1) & if \;\; c = X \\ T(s, d, A, Q_0) \,\&\, dQ_1 & if \;\; c = 0 \\ dQ_0 \,\&\, T(s, d, A, Q_1) & if \;\; c = 1 \end{cases}$$

$$[\varepsilon, U, s'](Q_0 \,\&\, Q_1) = \left(T(s', 1, u_0, Q_0) + T(s', 0, u_1, Q_1)\right) \& \left(T(s', 0, u_2, Q_0) + T(s', 1, u_3, Q_1)\right)$$

$$T(\varepsilon, d, A, Q) = AQ$$

Q : κατάσταση
s,s’ : μέρος συμβολοσειράς που περιγράφει το κύκλωμα
ε : κενή συμβολοσειρά
c : χαρακτήρας συμβολοσειράς ελέγχου
U : η πύλη
u_1, u_2, u_3, u_4 : τα στοιχεία της πύλης U
T : διαδικασία

Με βάση τον παραπάνω αλγόριθμο Από την αρχική κατάσταση Q εφαρμόζοντας την πύλη U με συμβολοσειρά ελέγχου την s,s’ θα μεταβούμε στην κατάσταση $T(s, 1, [\varepsilon, U, s'], Q)$. Από αυτή, όπως φαίνεται από τη δεύτερη εξίσωση, θα ελέγξουμε τον πρώτο χαρακτήρα και θα κάνουμε την ανάλογη διεργασία. Θα καλέσουμε την ίδια συνάρτηση αναδρομικά ($T(s, d, A, Q_0)$) ή και θα υπολογίσουμε μια νέα τιμή (dQ_1). Αφού η συμβολοσειρά s μείνει κενή με βάση την τρίτη εξίσωση συνεχίζουμε με μια νέα συνάρτηση η οποία είναι πάλι αναδρομική και επιλύεται με βάση τη δεύτερη εξίσωση πάλι. Αφού τελικά τα s και s’ μείνουν κενά έχουμε επεξεργαστεί το σύστημα και έχουμε υπολογίσει την πύλη, όπως φαίνεται και από την τέταρτη εξίσωση.

Η διαδικασία της μέτρησης ξεκινά αφού ζητηθεί η μέτρηση ενός συγκεκριμένου qubit. Αρχικά βρίσκεται η τιμή μέτρησης. Με βάση τις τιμές του qubit στις διάφορες καταστάσεις και έναν τυχαίο αριθμό βρίσκεται η τιμή επιστροφής (0 ή 1). Έπειτα υπολογίζονται οι νέες τιμές του ανύσματος καταστάσεων με βάση την τιμή επιστροφής, όπως παρουσιάστηκαν στην ενότητα 2.6 και τέλος επιστρέφεται η τιμή μέτρησης.

Το άνυσμα καταστάσεων συνήθως δεν έχει πολλά διαφορετικά στοιχεία, μιας και οι πύλες που εκτελούνται είναι συχνά ίδιες και απλά αντιμεταθέτουν στοιχεία. Αυτό μπορούμε να το εκμεταλλευτούμε έτσι ώστε αντί να αποθηκεύουμε το άνυσμα καταστάσεων στη μνήμη του υπολογιστή σαν ένα κανονικό άνυσμα, να το αποθηκεύσουμε σαν μια δενδρική δομή στην οποία δεν θα υπάρχουν κοινά στοιχεία.

Αυτή η δενδρική δομή θα αποθηκέυει τις τιμές του ανύσματος καταστάσεων στους κόμβους του δένδρου ταξινομημένες με βάση το μέτρο των στοιχείων. Η ταξινόμηση θα βοηθήσει αργότερα ώστε να μπορούμε να αναζητήσουμε κάποιο στοιχείο πολύ γρήγορα. Το κανονικό άνυσμα καταστάσεων θα έχει δείκτες προς τους κόμβους αυτού του δένδρου με τη δυνατότητα πάνω από μια διαφορετικές καταστάσεις να μπορούν να δείχνουν στον ίδιο κόμβο του δένδρου. Με αυτόν τον τρόπο εξοικονομούμε χώρο αφού το άνυσμα καταστάσεων αυξάνει εκθετικά με τον αριθμό των qubit και η κάθε καταχώρηση του ανύσματος καταλαμβάνει πολύ χώρο στη μνήμη.

Το δένδρο θα έχει στη ρίζα του μια ενδεικτική τιμή (π.χ. 0,2) και κάθε νέο στοιχείο που θα εισέρχεται θα προστίθεται στην θέση που του αντιστοιχεί με βάση το μέτρο του. Για παράδειγμα το στοιχείο με τιμές 0,1 και 0,1 θα τοποθετηθεί στον αριστερό παιδί-κόμβο από τη ρίζα μιας και το μέτρο του (0,1) είναι μικρότερο από αυτό της ρίζας (0,2). Κάθε στοιχείο θα αποθηκεύεται μαζί με τις δυο τιμές του (πραγματικό και φανταστικό μέρος του μιγαδικού) και μαζί με το μέτρο του έτσι ώστε κάθε φορά που χρειάζεται να μην πρέπει να υπολογιστεί εκ νέου. Επίσης κάθε κόμβος του δένδρου θα αποθηκεύει και το πόσοι δείκτες δείχνουν προς αυτόν έτσι ώστε όταν πλέον δεν υπάρχει κανένας να απομακρύνεται από το δένδρο.

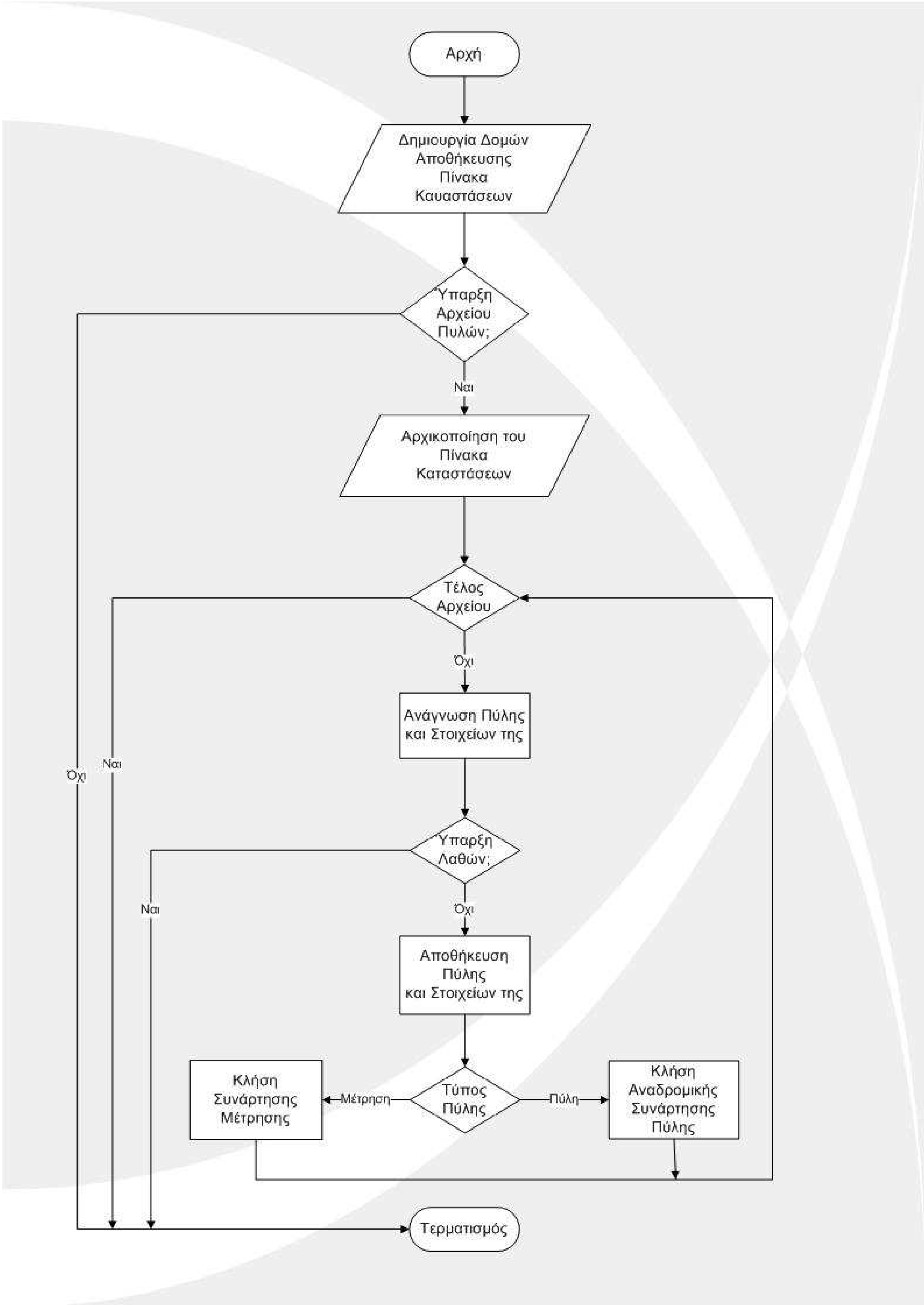
3.3 Υλοποίηση Συστήματος Μοντελοποίησης

Με βάση τις παραπάνω προδιαγραφές και την μελέτη της μεθοδολογίας της υλοποίησης του συστήματος του συστήματος μοντελοποίησης ξεκινήσαμε την υλοποίησή του. Επιλέξαμε τη γλώσσα προγραμματισμού C μιας και είναι ιδιαίτερα κατανοητή, γρήγορα εκτελέσιμή και μας έδινε επαρκείς δυνατότητες για να κάνουμε ότι χρειαζόμασταν. Ο compiler που επιλέξαμε ήταν ο gcc v 4.1.2 ο οποίος είναι ιδιαίτερα αποδοτικός κυρίως σε θέματα αναδρομών.

Αρχικά δηλώσαμε μεταβλητές για τη δομή του δένδρου οι οποίες είναι τρεις αριθμοί double (64 bit-11bit exponent, 51bit mantissa 1bit sign) εύρους +/-1,7^{E}308 ένα για το πραγματικό και ένα για το φανταστικό μέρος του μιγαδικού της τιμής της κατάστασης και ένα για το μέτρο του. Ακόμη, περιλαμβάνει ένα ακέραιο ο οποίος χρησιμοποιείται ως μετρητής των δεικτών που δείχνουν προς το συγκεκριμένο κόμβο και τέλος περιλαμβάνει δυο δείκτες ένα για κάθε απόγονο του κόμβου. Επειδή η ακρίβεια των αριθμών double είναι πολύ μεγάλη και συνήθως χάνεται μεταξύ των πράξεων θεωρήσαμε πως δυο αριθμοί double είναι ίδιοι αν έχουν ίδια ψηφία μέχρι και το 6^o δεκαδικό τους ψηφίο. Έτσι με βάση αυτή τη συνθήκη θα εισάγουμε στο δένδρο καταστάσεων δυο καταστάσεις στον ίδιο κόμβο. Η δομή του ανύσματος καταστάσεων, όπως αναφέρθηκε προηγουμένως, είναι ένα άνυσμα μεγέθους 2^{qubits} με δείκτες προς τη δενδρική δομή. Η δομή της πύλης αποτελείται από 8 double αριθμούς (4 μιγαδικούς) και της συμβολοσειράς ελέγχου για τις ελεγχόμενες πύλες μήκους όσο το πλήθος των qubits μειούμενο κατά ένα (τη θέση της πύλης).

Στην αρχή του αλγόριθμου ελέγχεται αν υπάρχει το αρχείο με τις πύλες, gates.txt, και αν όχι ενημερώνει με μήνυμα σφάλματος και τερματίζει. Σε αντίθετη περίπτωση, ανοίγει το αρχείο για ανάγνωση και διαβάζει τον τύπο της πύλης. Αν δεν είναι γνωστός ενημερώνει το χρήστη με το κατάλληλο μήνυμα και τερματίζει. Σε περίπτωση που είναι πύλη control ή Hadamard διαβάζει και αποθηκεύει την πύλη και έπειτα το κύκλωμα ελέγχου στις αντίστοιχες μεταβλητές. Στην περίπτωση της πράξης της μέτρησης διαβάζει μόνο το qubit το οποίο θα πρέπει να μετρηθεί. Καθ' όλη τη διάρκεια της ανάγνωσης του αρχείου γίνονται έλεγχοι για την ορθότητα των δεδομένων και αν υπάρξει κάποιο σφάλμα, τότε προβάλλεται στο χρήστη το αντίστοιχο μήνυμα και το πρόγραμμα τερματίζεται.

Η πύλη η οποία θα αναγνωρισθεί, εκτελείται με βάση τον αντίστοιχο αλγόριθμο που παρουσιάστηκε στην παραπάνω ενότητα. Μετά το πέρας της εκτέλεσης κάθε πύλης τυπώνει ότι εκτελέστηκε, διαβάζεται μια νέα εκτελείται κ.ο.κ.. Όταν πλέον δεν θα υπάρχουν νέες πύλες το πρόγραμμα προβάλει τα στοιχεία του ανύσματος καταστάσεων και τερματίζει. Ένα σχηματικό διάγραμμα του αλγόριθμου που αναπτύξαμε παρουσιάζεται εδώ:



Εικόνα 3.1: *Σχηματικό διάγραμμα Αλγόριθμου Μοντελοποίησης Συστήματος*

3.4 Πιστοποίηση Συστήματος Μοντελοποίησης

Για το έλεγχο της ορθότητας του αλγορίθμου που αναπτύξαμε, χρησιμοποιήσαμε κάποιες δοκιμασίες τις οποίες εντοπίσαμε στο διαδίκτυο. Η πρώτη δοκιμασία έγινε με κάποια τυχαία παραγόμενα κυκλώματα για να ελέγξουμε πως το σύστημα δουλεύει χωρίς να έχει απρόβλεπτες συμπεριφορές. Η δεύτερη περιλάμβανε κάποια κυκλώματα τα οποία έδινα γνωστά αποτελέσματα, τα οποία πιστοποίησαν την ορθή λειτουργία του συστήματος μοντελοποίησης.

Για την πρώτη από τις δυο πιστοποιήσεις, εντοπίσαμε ένα αλγόριθμο του Scott Aaronson για “Random Clifford-Group Quantum Circuit Generator” [21]. Αυτός ο αλγόριθμος δεχόταν τα ορίσματα για τις προδιαγραφές του τυχαίου συστήματος που θα παρήγαγε. Δηλαδή δεχόταν το πλήθος τον qubit που θα είχε το σύστημα, το πόσες πύλες θα παρήγαγε, και τι ποσοστό των πυλών θα ήταν CNOT, Hadamard, Phase και Measurement. Αυτό είχε ως έξοδο μια συγκεκριμένη μορφή περιγραφής κβαντικών κυκλωμάτων την οποία και χρησιμοποιούσαμε κάθε φορά προκειμένου να ελέγξουμε την αξιοπιστία αλλά και τη ταχύτητα του συστήματος. Για τη μετατροπή από τη μια μορφή εξόδου περιγραφής κβαντικών κυκλωμάτων , που έδινε το πρόγραμμα, στην επιθυμητή μορφή εισόδου του αλγόριθμου που αναπτύξαμε, αναπτύξαμε και χρησιμοποιήσαμε ένα αλγόριθμο ο οποίος κάνει αυτοματοποιημένα αυτή την εργασία.

Για τη δεύτερη πιστοποίηση, χρησιμοποιήσαμε κάποιες έτοιμες δοκιμασίες από τη σελίδα στο διαδίκτυο του πανεπιστημίου της Βικτώριας στον Καναδά [22]. Αυτά περιλαμβάνουν τη περιγραφή κάποιων κβαντικών κυκλωμάτων μαζί με τη λειτουργία τους. Συνήθως κάνουν κάποιες ευρέως γνωστές διεργασίες όπως πρόσθεση, κωδικοποίηση, αποκωδικοποίηση κ.α.. Στη συγκεκριμένη σελίδα υπάρχουν διάφορα κυκλώματα, για κάθε ένα από τα οποία υπάρχει σύνοψη περιγραφή, η κωδικοποίησή του σε κβαντικό κύκλωμα, ένα κβαντικό κύκλωμα και οι προδιαγραφές του [23].

Χρησιμοποιώντας τις δυο παραπάνω μεθόδους πιστοποιήσαμε την ορθή και πλήρη λειτουργία του συστήματος μοντελοποίησης το οποίο αναπτύξαμε. Έτσι ώστε αργότερα να μπορούμε να το χρησιμοποιήσουμε ως σύστημα δοκιμασίας αλλά και σύγκρισης του τελικού συστήματος που αναπτύξαμε.

```
Gate 1 is c
Gate 2 is h
Gate 3 is c
Gate 4 is m
Gate 5 is c
Gate 6 is c
Gate 7 is h
Gate 8 is h
Gate 9 is c
Gate 10 is h
Simulation completed succesfully
```

Εικόνα 3.2: *Έξοδος κώδικα κατά τη διάρκεια εκτέλεσης*

```
State vector is:
================
Q[0] is: Re=0.667959, Img= 0.306503
Q[1] is: Re=0.658069, Img= 0.834685
Q[2] is: Re=0.572306, Img= 0.038711
Q[3] is: Re=0.337364, Img= 0.168789
Q[4] is: Re=0.487403, Img= 0.098716
Q[5] is: Re=0.827063, Img= 0.897428
Q[6] is: Re=0.357013, Img= 0.058893
Q[7] is: Re=0.537403, Img= 0.035785
Q[8] is: Re=0.117078, Img= 0.168708
Q[9] is: Re=0.027110, Img= 0.368633
Q[10] is: Re=0.937386, Img= 0.538780
Q[11] is: Re=0.057516, Img= 0.869387
Q[12] is: Re=0.727168, Img= 0.362878
Q[13] is: Re=0.307698, Img= 0.328632
Q[14] is: Re=0.617894, Img= 0.686541
```

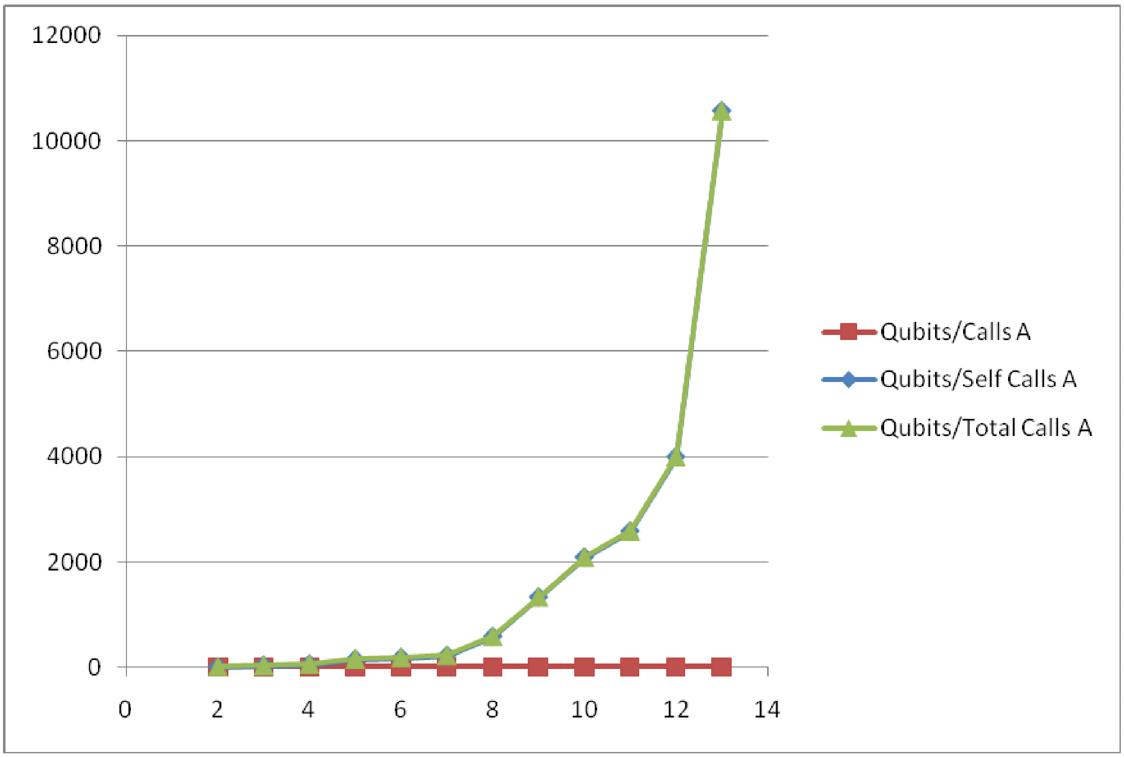
Εικόνα 3.3: *Έξοδος κώδικα μετά το τέλος της εκτέλεσης*

3.5 Αποτελέσματα Συστήματος Μοντελοποίησης

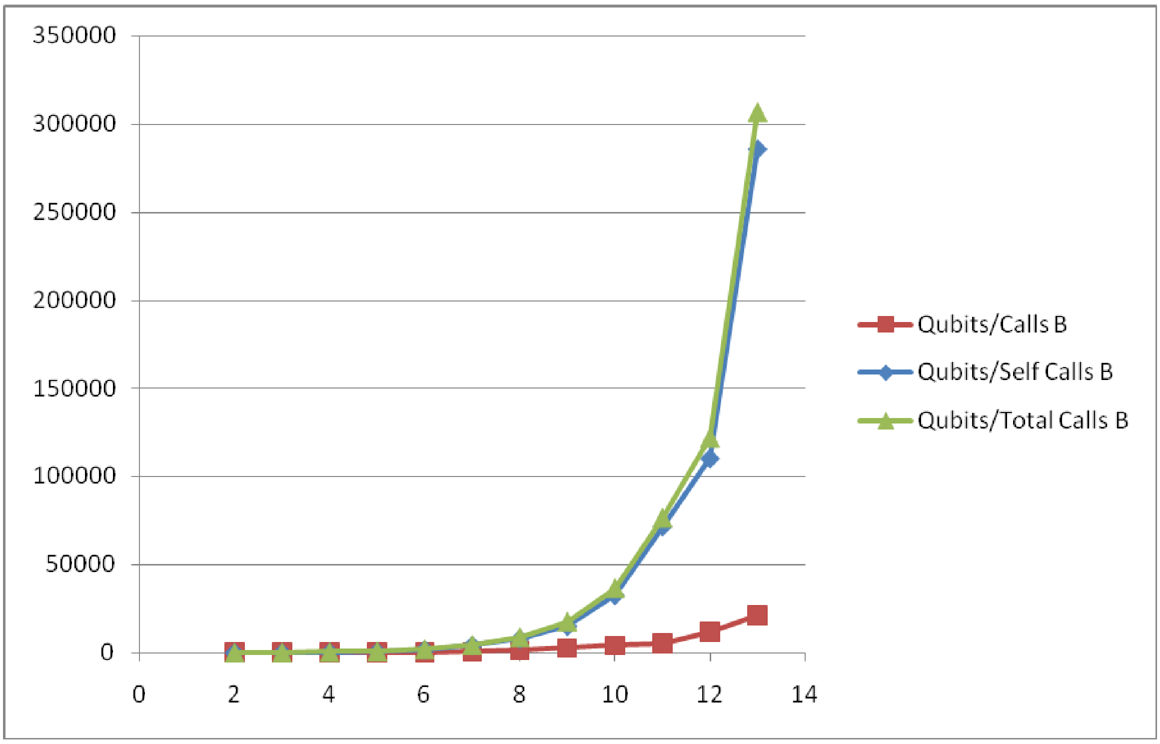
Αφού πιστοποιήσαμε την ορθή και πλήρη λειτουργία του αλγόριθμου μοντελοποίησης του συστήματος εξήγαμε κάποια αποτελέσματα που αφορούν την αποδοτικότητά του και τα όριά του. Έτσι, χρησιμοποιώντας τον αλγόριθμο παραγωγής τυχαίων κβαντικών κυκλωμάτων δοκιμάσαμε τη συμπεριφορά του αλγορίθμου κάτω από διάφορες συνθήκες. Γι' αυτό το σκοπό χρειαστήκαμε ένα εργαλείο το οποίο να κάνει profiling κώδικα σε γλώσσα προγραμματισμού C. Έτσι χρησιμοποιήσαμε το εργαλείο gprof v 2.17.50 των Linux προκειμένου να δούμε τη συμπεριφορά του αλγορίθμου.

3.5.1 Αποτελέσματα για Πύλες CNOT

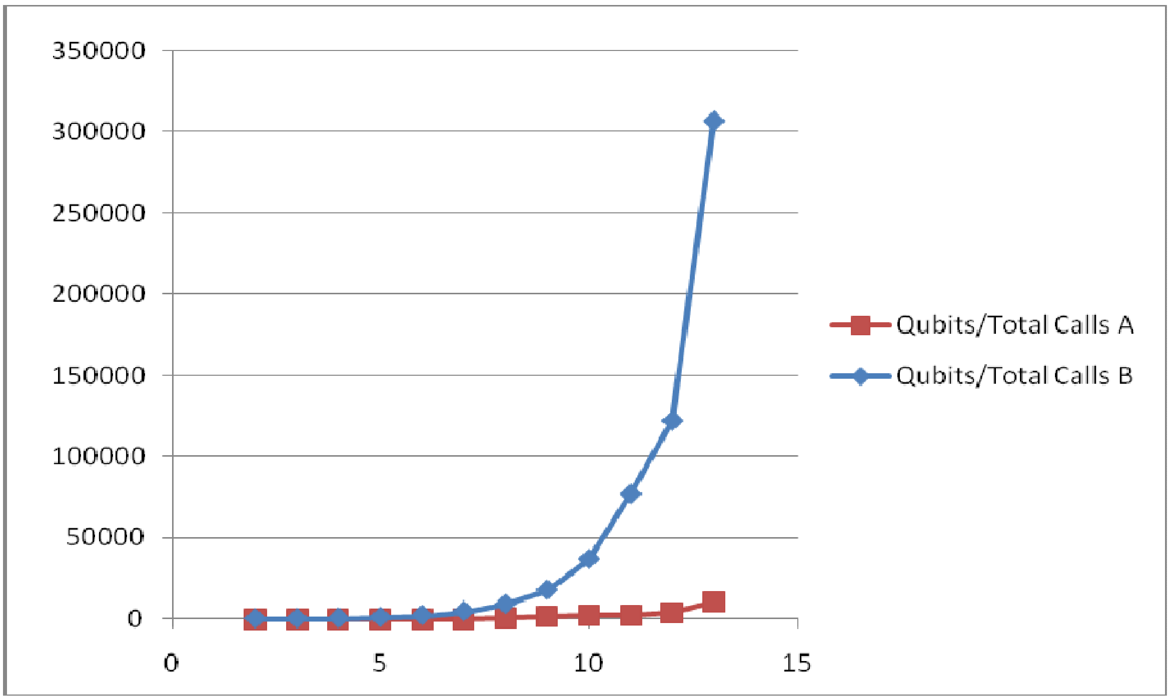
Αρχικά δοκιμάσαμε αποκλειστικά πύλες Controlled Not (CNOT) αλλάζοντας το πλήθος των qubit κάθε φορά. Έτσι δοκιμάζαμε δέκα CNOT πύλες με 2 έως 13 qubit κάθε φορά. Με τη χρήση του εργαλείου gprof παρατηρούσαμε κάθε φορά διάφορα στοιχεία όπως το συνολικό χρόνο εκτέλεσης του αλγόριθμου, το πόσες φορές κλήθηκε η κάθε συνάρτηση και πόσος χρόνος ξοδεύθηκε σε αυτήν κ.α.. Αυτά παρουσιάζονται εδώ:



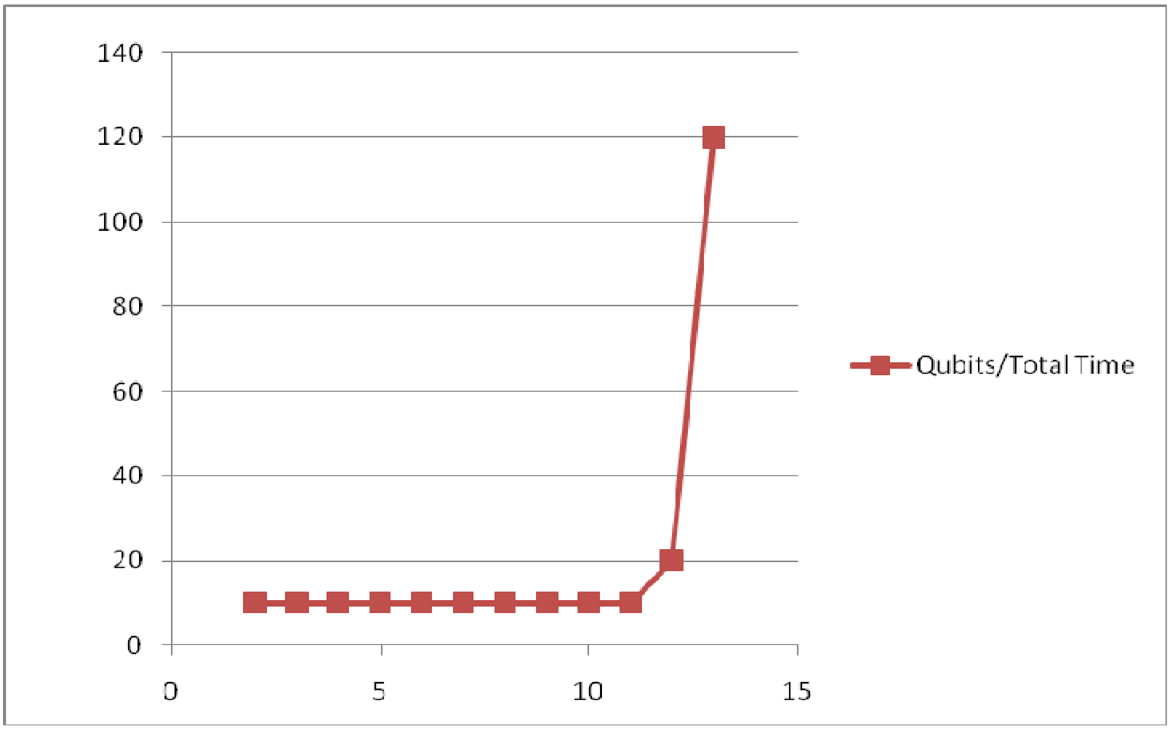
Εικόνα 3.4: *Κλήση 1ης αναδρομής σε σχέση με το πλήθος των qubits*



Εικόνα 3.5: *Κλήση 2ης αναδρομής σε σχέση με το πλήθος των qubits*



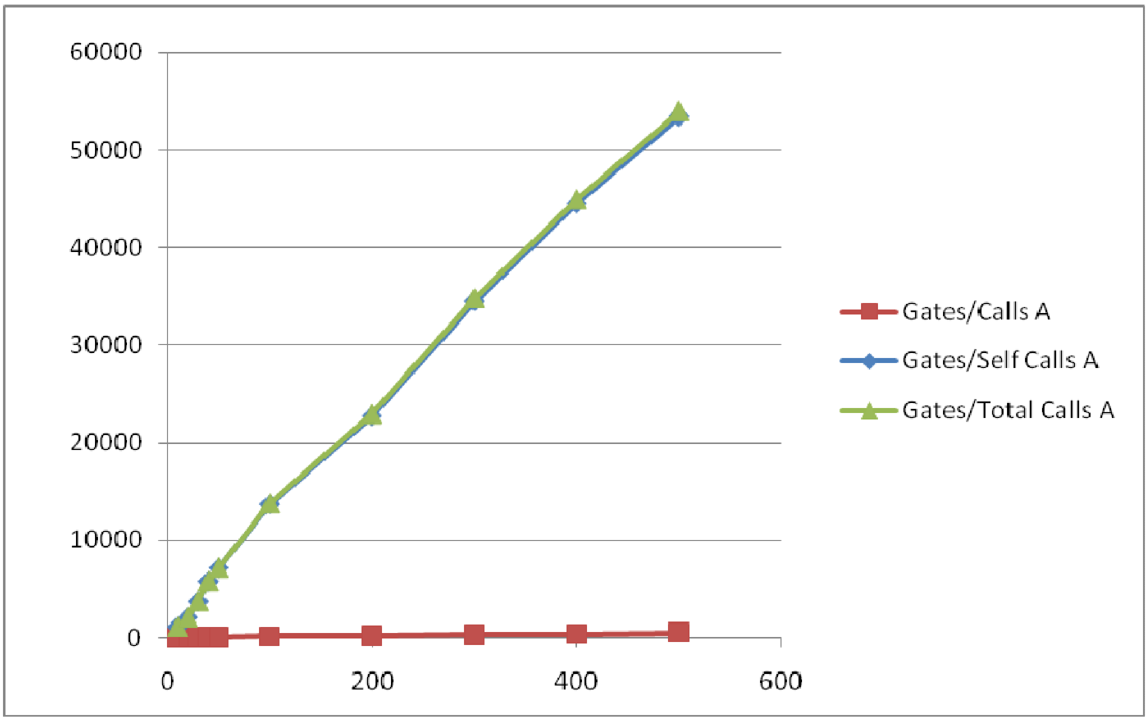
Εικόνα 3.6: Κλήσεις *1ης και 2ης αναδρομής σε σχέση με το πλήθος των qubits*



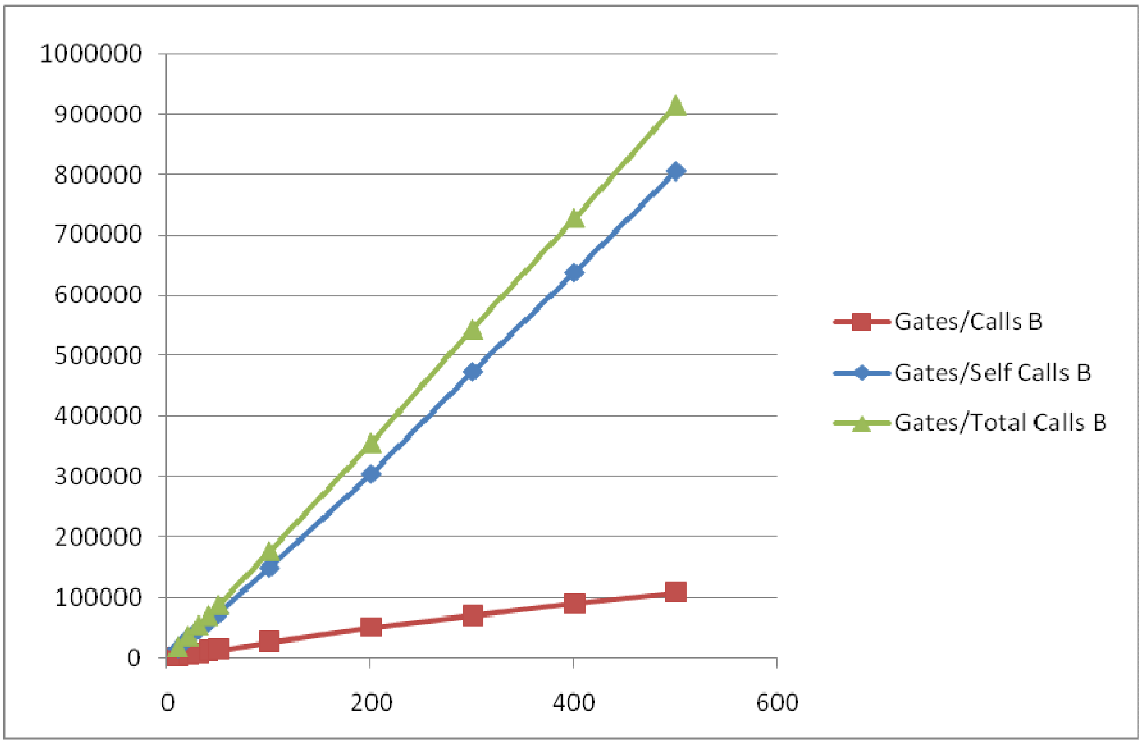
Εικόνα 3.7: *Συνολικός χρόνος σε σχέση με το πλήθος των qubits*

Με βάση τα παραπάνω διαγράμματα παρατηρούμε πως οι κλήσεις και των δυο αναδρομικών συναρτήσεων αυξάνονται εκθετικά με την αύξηση του αριθμού των qubits κάτι το οποίο ήταν αναμενόμενο, αφού αυξάνοντας τα qubits αυξάνουν εκθετικά οι πιθανές καταστάσεις του συστήματος. Αυτό οδηγεί στην εκθετική αύξηση του χρόνου εκτέλεσης του συνολικού προγράμματος, όπως είναι φυσικό, το οποίο παρατηρείται στο τελευταίο διάγραμμα. Παρατηρούμε επίσης πως, εκθετικά αυξάνονται κυρίως οι αναδρομικές κλήσεις των συναρτήσεων και όχι οι αρχικές κλήσεις.

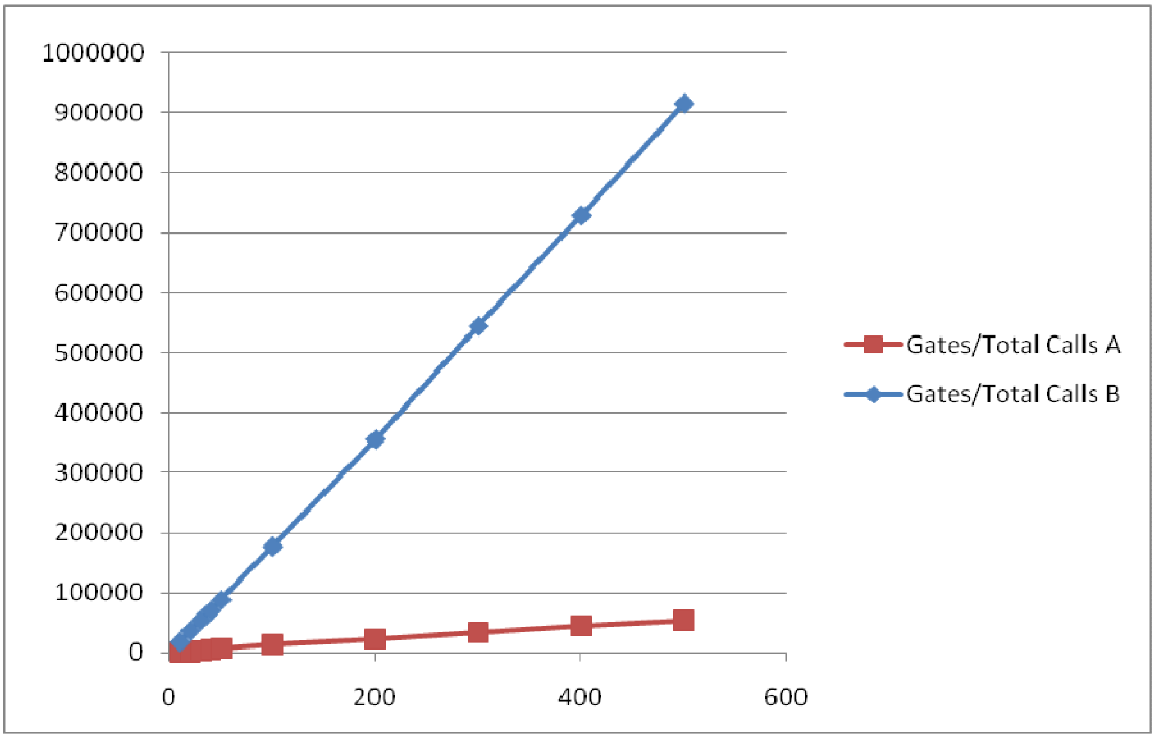
Αυξάνοντας τώρα το πλήθος των πυλών CNOT σταδιακά από 10 έως 500 και διατηρώντας το πλήθος των qubit σταθερό και ίσο με 9 παρατηρήσαμε τη συμπεριφορά του αλγορίθμου. Έτσι δημιουργήσαμε τα εξής διαγράμματα:



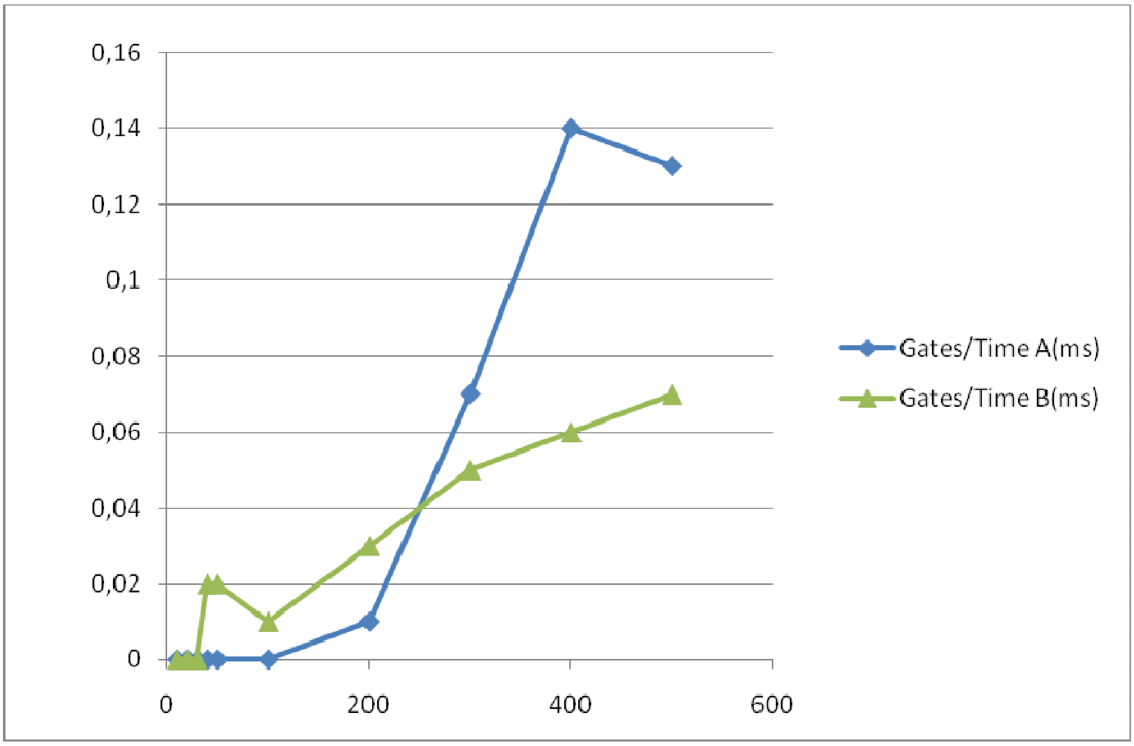
Εικόνα 3.8: *Κλήσεις 1ης αναδρομής σε σχέση με το πλήθος των πυλών*



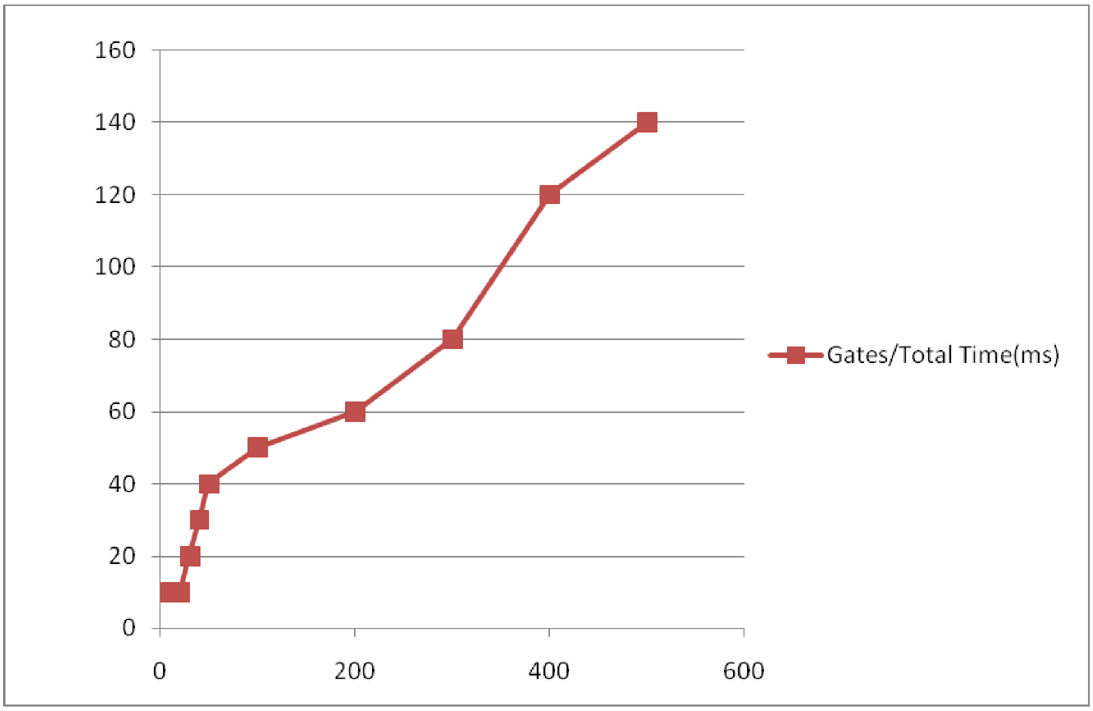
Εικόνα 3.9: *Κλήσεις 2ης αναδρομής σε σχέση με το πλήθος των πυλών*



Εικόνα 3.10: Κλήσεις *1ης και 2ης αναδρομής σε σχέση με το πλήθος των πυλών*



Εικόνα 3.11: *Χρόνος εκτέλεσης 1ης και 2ης αναδρομής σε σχέση με το πλήθος των πυλών*

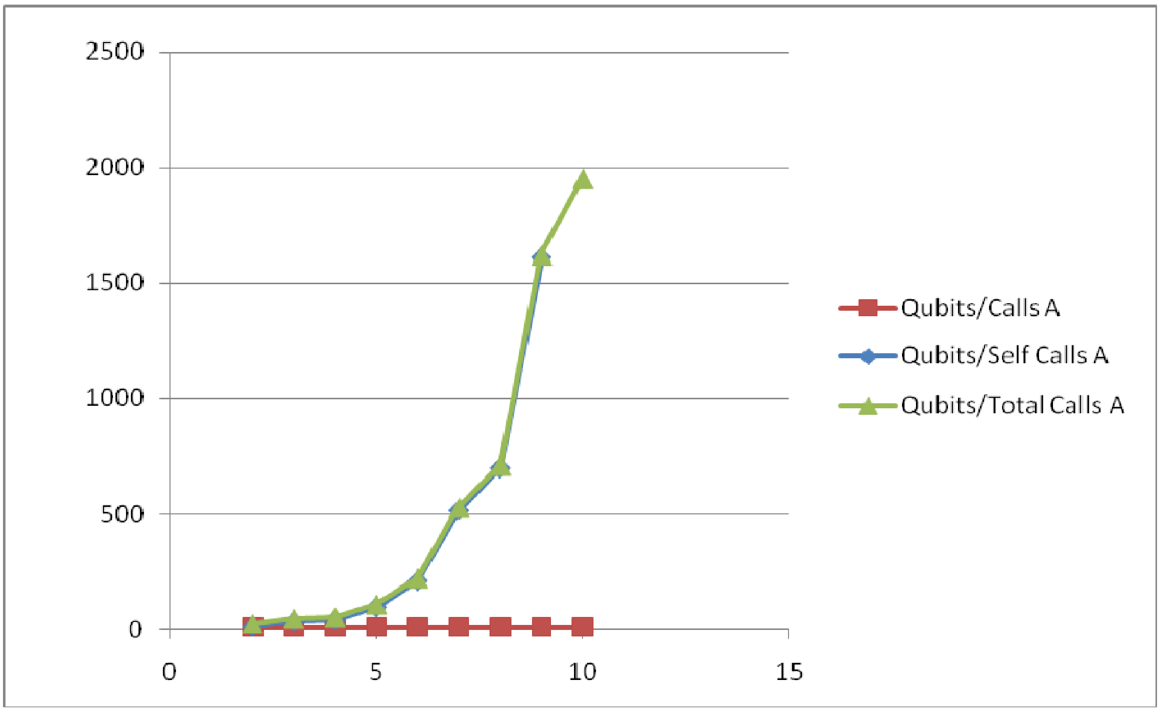


Εικόνα 3.12: *Συνολικός χρόνος εκτέλεσης σε σχέση με το πλήθος των πυλών*

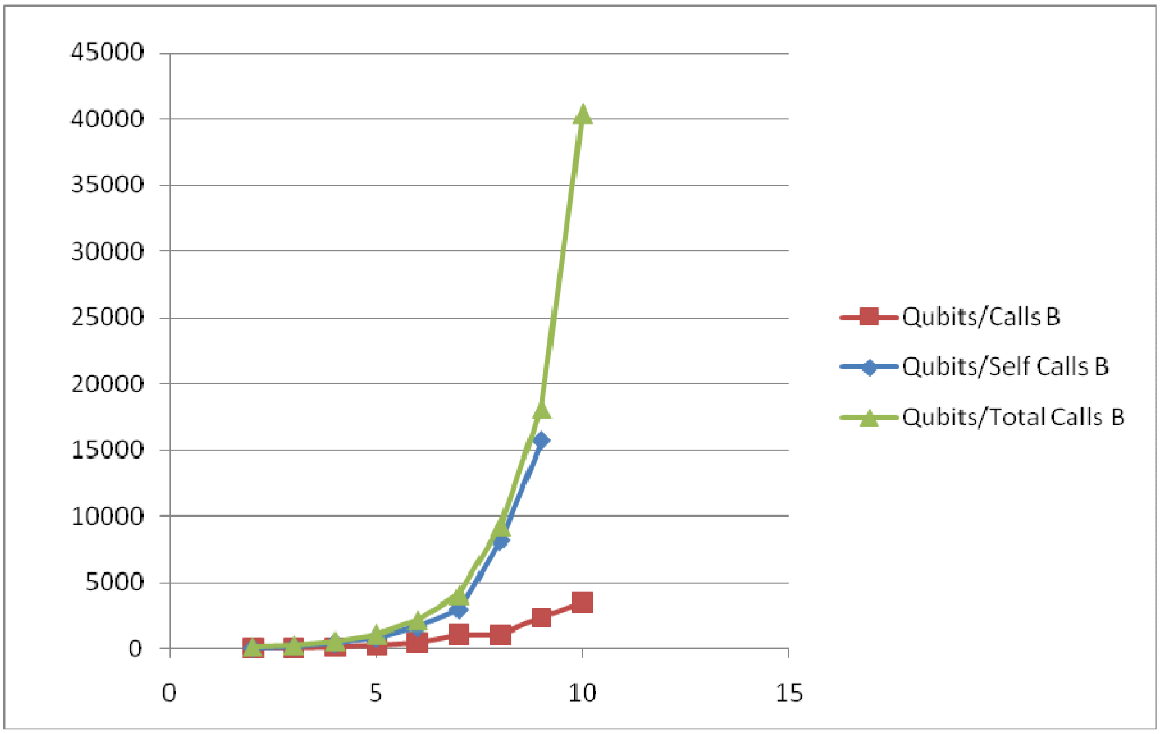
Με βάση τα παραπάνω διαγράμματα παρατηρούμε πως οι κλήσεις και των δυο αναδρομικών συναρτήσεων αυξάνονται γραμμικά με την αύξηση του αριθμού των πυλών κάτι το οποίο ήταν αναμενόμενο, αφού αυξάνοντας τις πύλες αυξάνει απλά η πρόοδος του αλγορίθμου χωρίς να αυξάνει η πολυπλοκότητά του. Παρατηρούμε επίσης πως, γραμμικά αυξάνονται όχι μόνο οι αναδρομικές κλήσεις των συναρτήσεων αλλά και οι αρχικές κλήσεις κάτι το οποί η είναι φυσιολογικό με βάση τον αλγόριθμο. Επίσης ο χρόνος εκτέλεσης αυξάνει εκθετικά (έστω και με μικρό ρυθμό) κάτι το οποίο οφείλεται στην μικρή αύξηση των διαφορετικών τιμών στο άνυσμα καταστάσεων και συνεπώς στην αύξηση το μεγέθους του δένδρου που αποθηκεύονται με αποτέλεσμα να καθυστερούν οι διεργασίες εκείνου.

3.5.2 Αποτελέσματα για Πύλες Hadamard

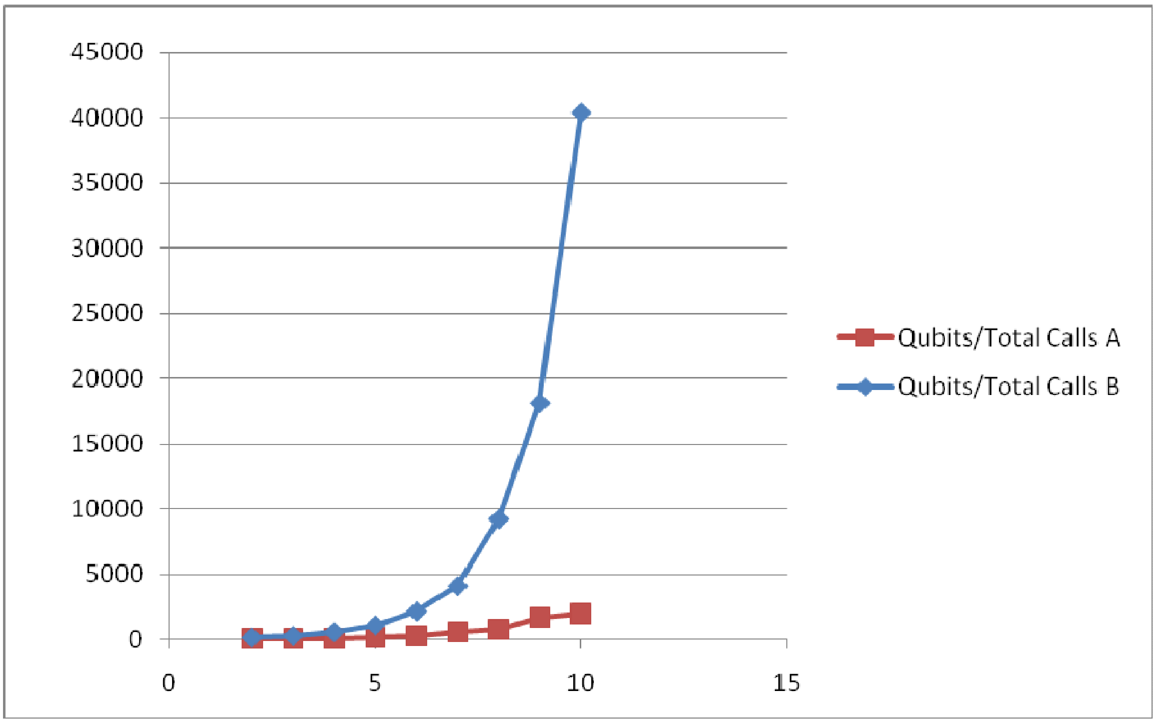
Έπειτα δοκιμάσαμε αποκλειστικά πύλες Hadamard αλλάζοντας το πλήθος των qubit κάθε φορά. Έτσι δοκιμάζαμε δέκα Hadamard πύλες με 2 έως 10 qubit κάθε φορά. Με τη χρήση του εργαλείου gprof παρατηρούσαμε κάθε φορά διάφορα στοιχεία όπως το συνολικό χρόνο εκτέλεσης του αλγόριθμου, το πόσες φορές κλήθηκε η κάθε συνάρτηση και πόσος χρόνος ξοδεύτηκε σε αυτήν κ.α.. τα αποτελέσματα παρουσιάζονται εδώ:



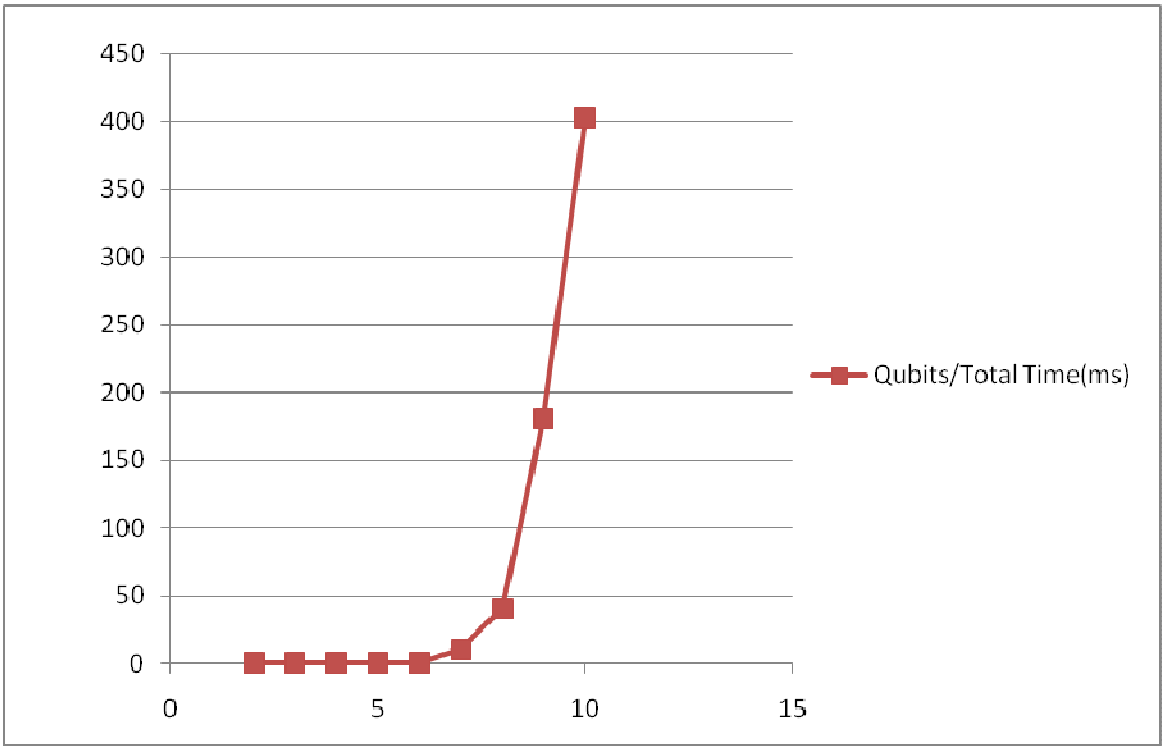
Εικόνα 3.13: *Κλήσεις 1ης αναδρομής σε σχέση με το πλήθος των qubit*



Εικόνα 3.14: *Κλήσεις 2ης αναδρομής σε σχέση με το πλήθος των qubit*



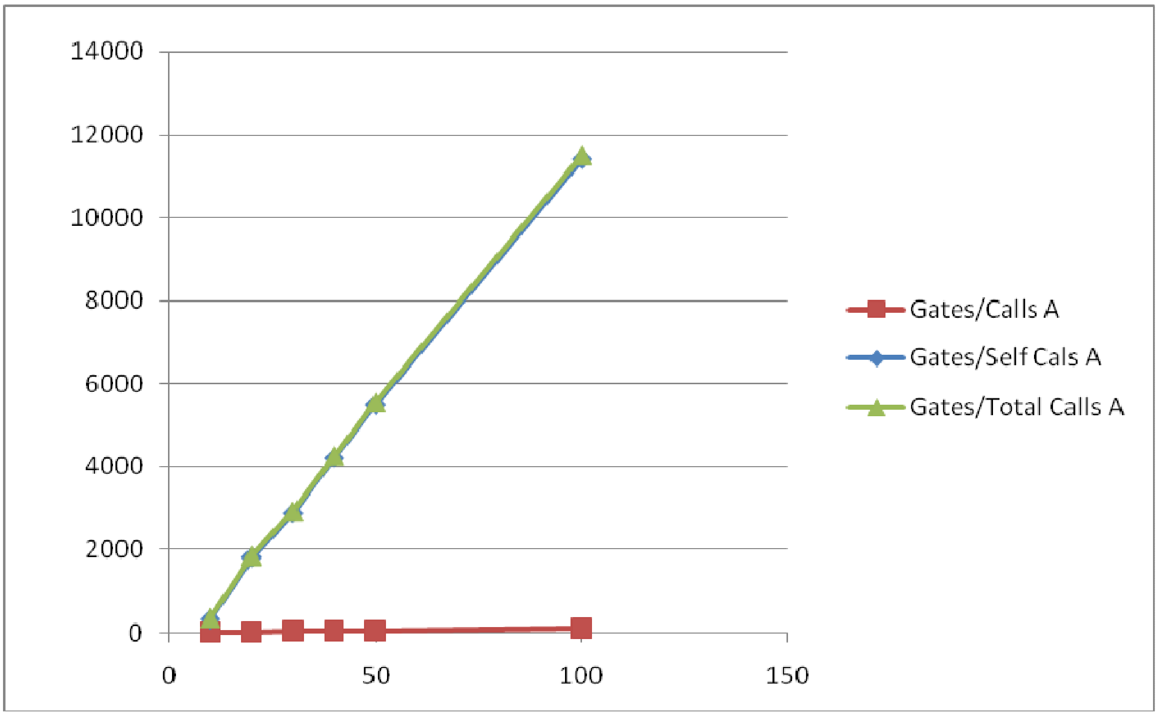
Εικόνα 3.15: *Κλήσεις 1ης και 2ης αναδρομής σε σχέση με το πλήθος των qubit*



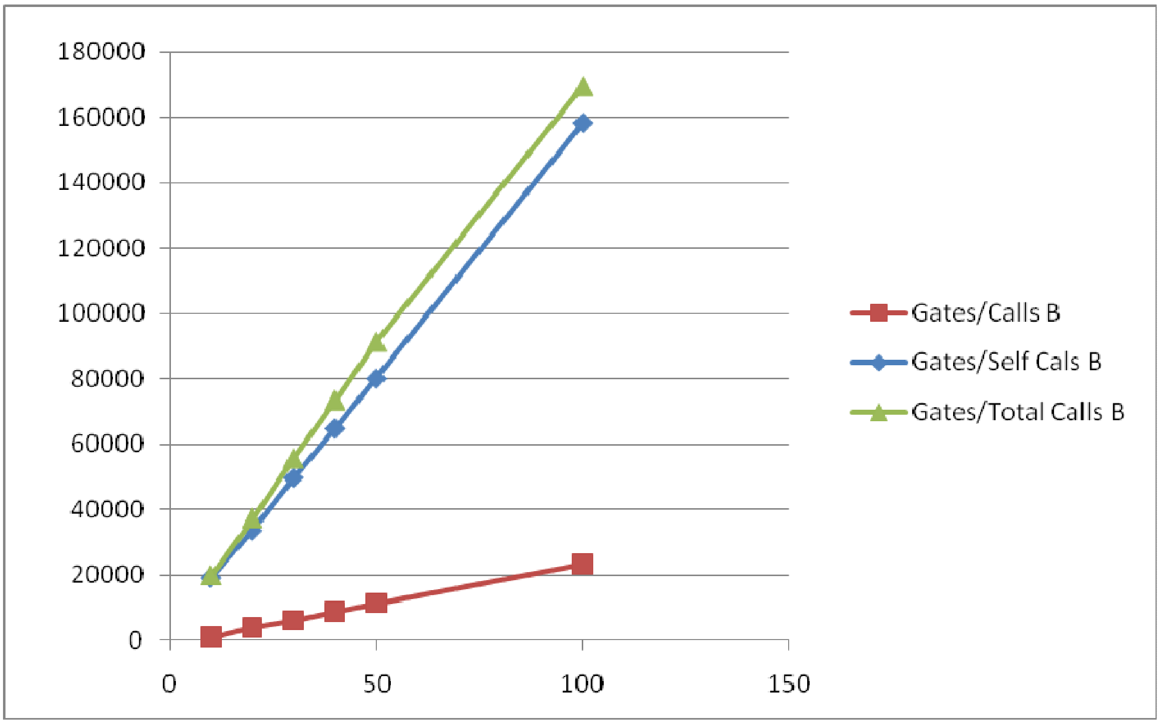
Εικόνα 3.16: *Συνολικός χρόνος εκτέλεσης σε σχέση με το πλήθος των qubit*

Με βάση τα παραπάνω διαγράμματα παρατηρούμε πως οι κλήσεις και των δυο αναδρομικών συναρτήσεων αυξάνονται εκθετικά με την αύξηση του αριθμού των qubits κάτι το οποίο ήταν αναμενόμενο, αφού αυξάνοντας τα qubits αυξάνουν εκθετικά οι πιθανές καταστάσεις του συστήματος. Αυτό οδηγεί στην εκθετική αύξηση του χρόνου εκτέλεσης του συνολικού προγράμματος, όπως είναι φυσικό, το οποίο παρατηρείται στο τελευταίο διάγραμμα. Παρατηρούμε επίσης πως, εκθετικά αυξάνονται κυρίως οι αναδρομικές κλήσεις των συναρτήσεων και όχι οι αρχικές κλήσεις.

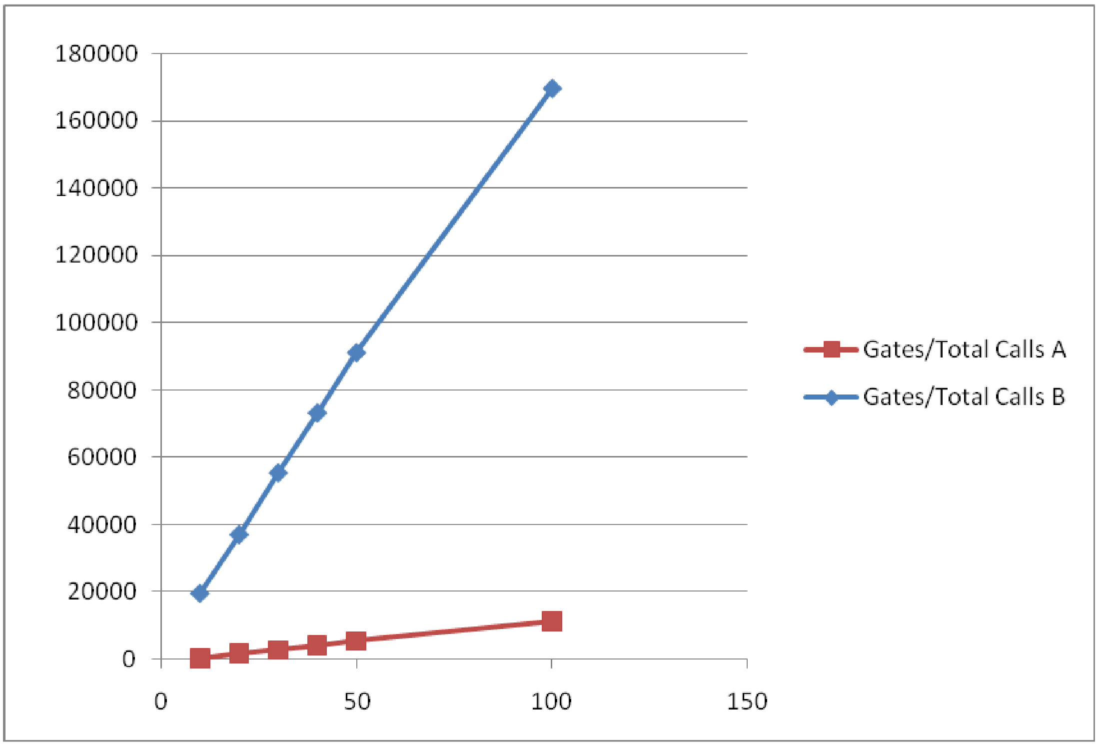
Αυξάνοντας τώρα το πλήθος των πυλών Hadamard σταδιακά από 10 έως 100 και διατηρώντας το πλήθος των qubit σταθερό και ίσο με 9 παρατηρήσαμε τη συμπεριφορά του αλγορίθμου. Έτσι δημιουργήσαμε τα εξής διαγράμματα:



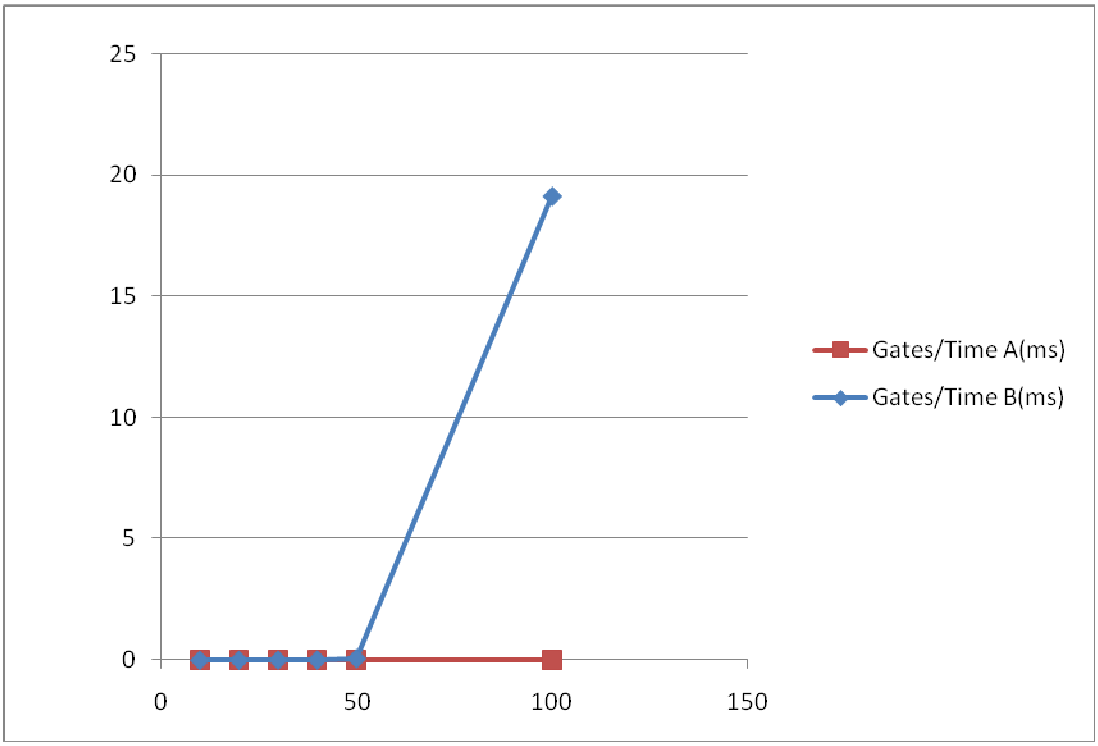
Εικόνα 3.17: *Κλήσεις $1^{ης}$ αναδρομής σε σχέση με το πλήθος των πυλών*



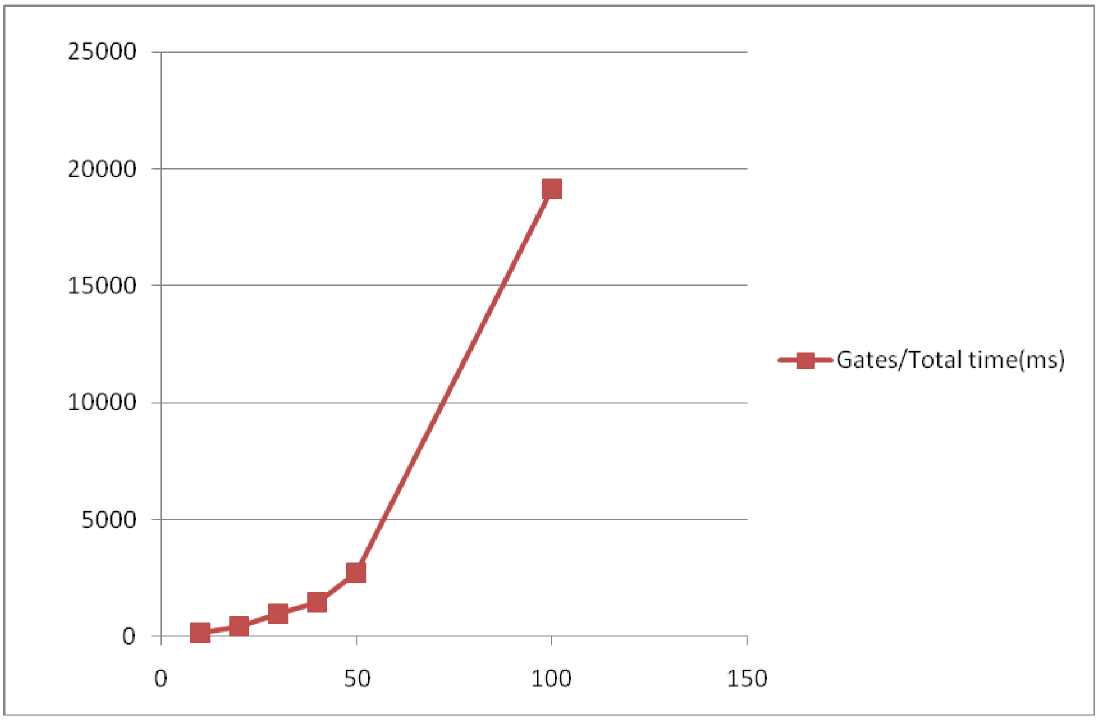
Εικόνα 3.18: *Κλήσεις 2ης αναδρομής σε σχέση με το πλήθος των πυλών*



Εικόνα 3.19: *Κλήσεις 1ης και 2ης αναδρομής σε σχέση με το πλήθος των πυλών*



Εικόνα 3.20: *Χρόνος εκτέλεσης 1ης και 2ης αναδρομής σε σχέση με το πλήθος των πυλών*

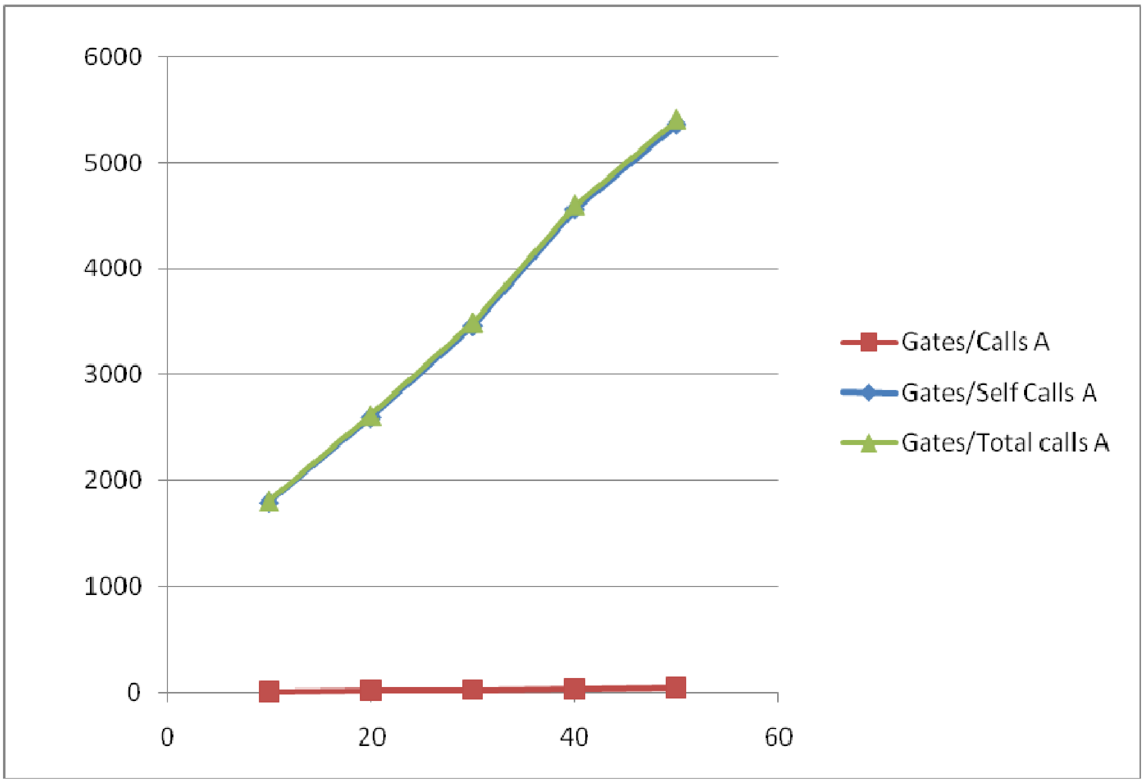


Εικόνα 3.21: *Συνολικός χρόνος εκτέλεσης σε σχέση με το πλήθος των πυλών*

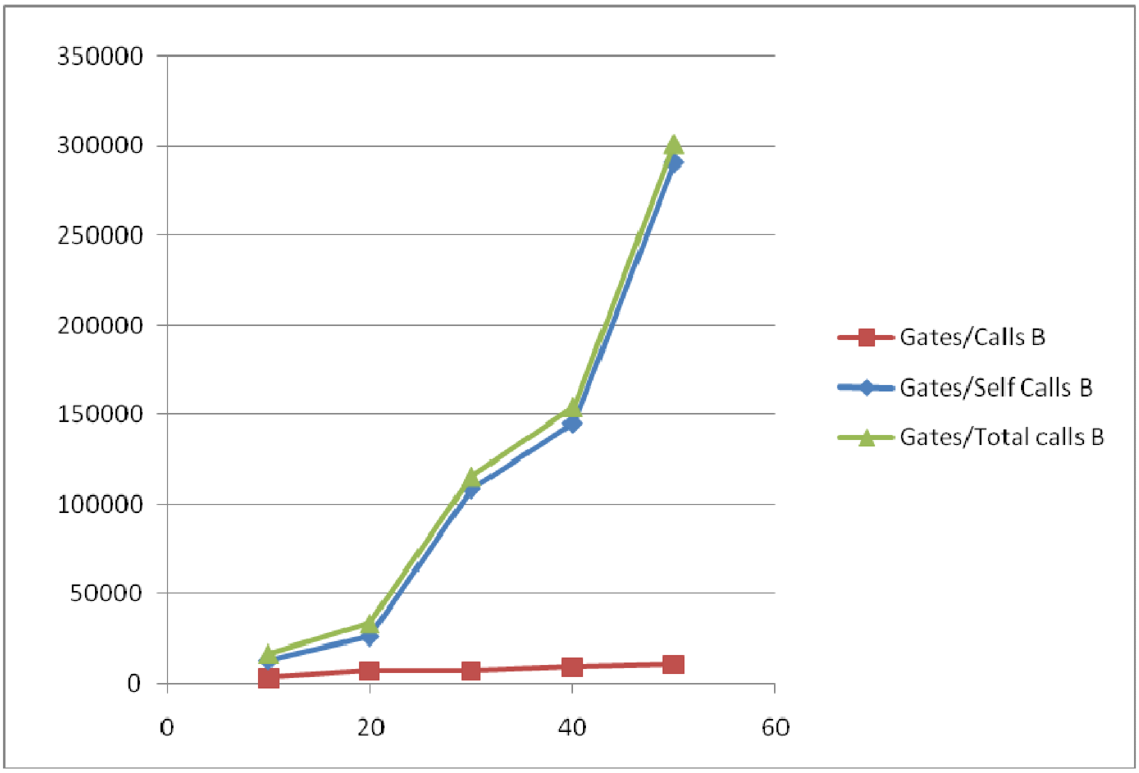
Με βάση τα παραπάνω διαγράμματα παρατηρούμε πως οι κλήσεις και των δυο αναδρομικών συναρτήσεων αυξάνονται γραμμικά με την αύξηση του αριθμού των πυλών κάτι το οποίο ήταν αναμενόμενο, αφού αυξάνοντας τις πύλες αυξάνει απλά η πρόοδος του αλγορίθμου χωρίς να αυξάνει η πολυπλοκότητά του. Παρατηρούμε επίσης πως, γραμμικά αυξάνονται όχι μόνο οι αναδρομικές κλήσεις των συναρτήσεων αλλά και οι αρχικές κλήσεις κάτι το οποί η είναι φυσιολογικό με βάση τον αλγόριθμο. Επίσης ο χρόνος εκτέλεσης αυξάνει εκθετικά (με μεγάλο ρυθμό) κάτι το οποίο οφείλεται στην αύξηση των διαφορετικών τιμών στο άνυσμα καταστάσεων και συνεπώς στην αύξηση το μεγέθους του δένδρου που αποθηκεύονται με αποτέλεσμα να καθυστερούν οι διεργασίες εκείνου.

3.5.3 Αποτελέσματα για Τυχαίες Πύλες

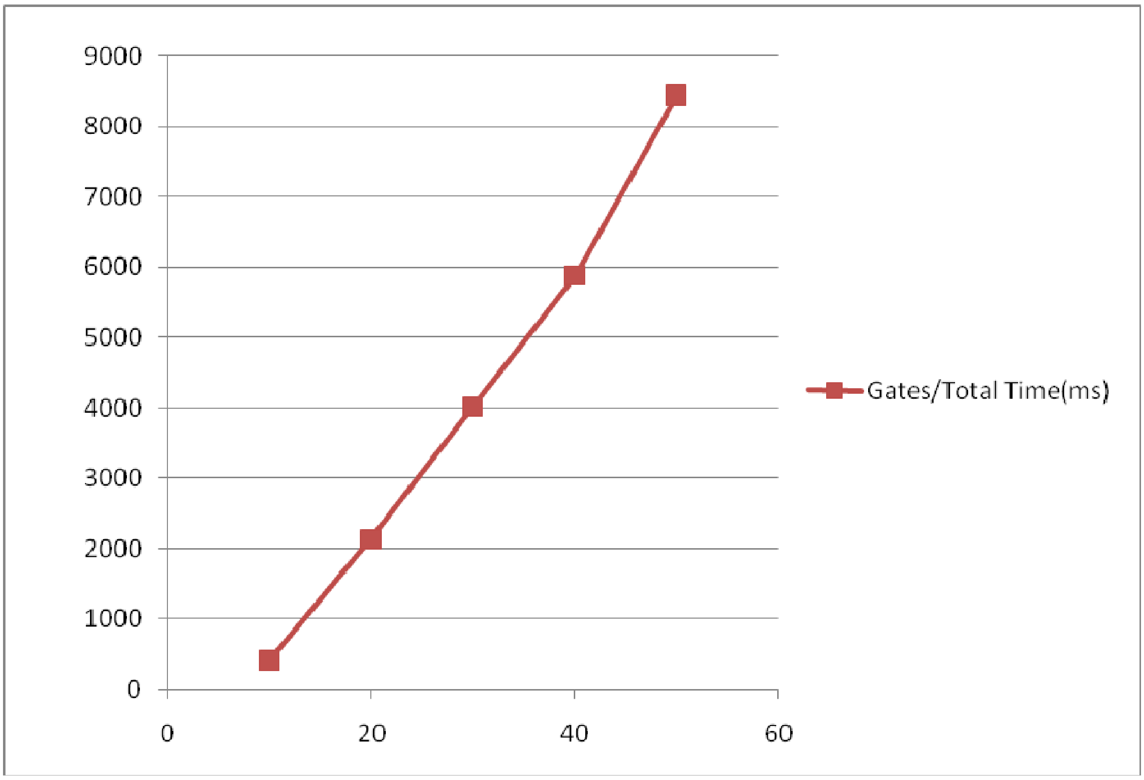
Δοκιμάσαμε ακόμα διάφορες τυχαίες πύλες διατηρώντας το πλήθος των qubit σταθερό και ίσο με 9 και παρατηρήσαμε τη συμπεριφορά του αλγορίθμου και κυρίως του ανύσματος καταστάσεων το οποίο επηρεάζεται από τις πολλές διαφορετικές τιμές. Έτσι δημιουργήσαμε τα εξής διαγράμματα:



Εικόνα 3.22: *Κλήσεις 1ης και 2ης αναδρομής σε σχέση με το πλήθος των πυλών*



Εικόνα 3.23: *Χρόνος εκτέλεσης 1ης και 2ης αναδρομής σε σχέση με το πλήθος των πυλών*

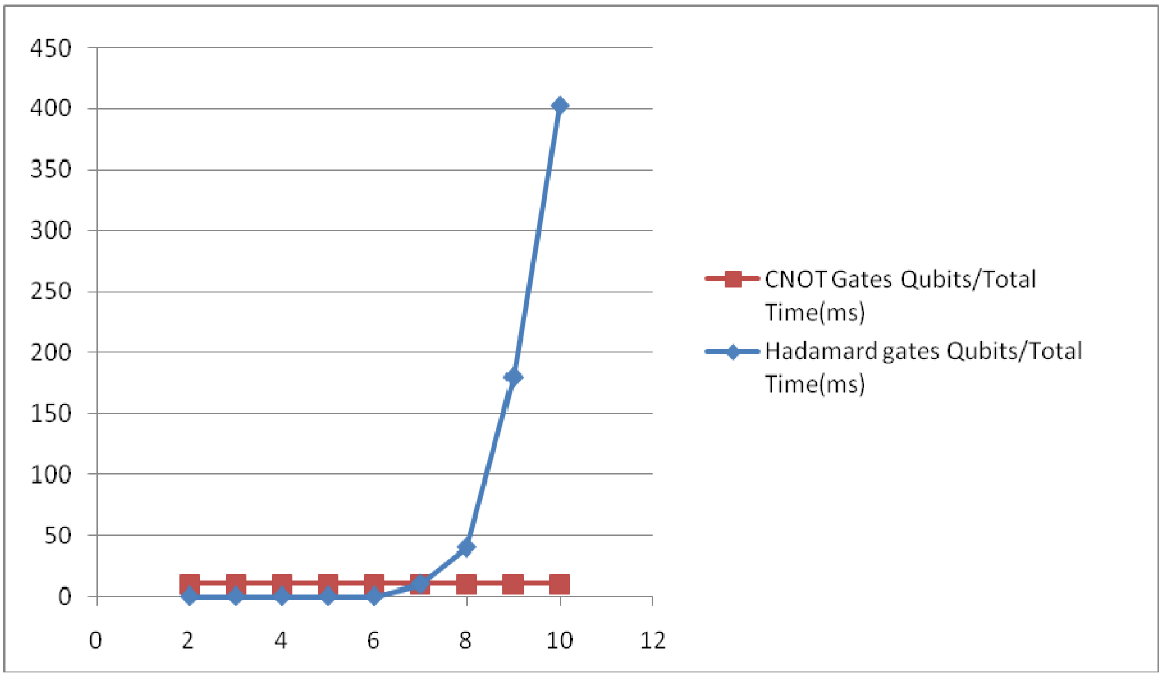


Εικόνα 3.24: *Συνολικός χρόνος εκτέλεσης σε σχέση με το πλήθος των πυλών*

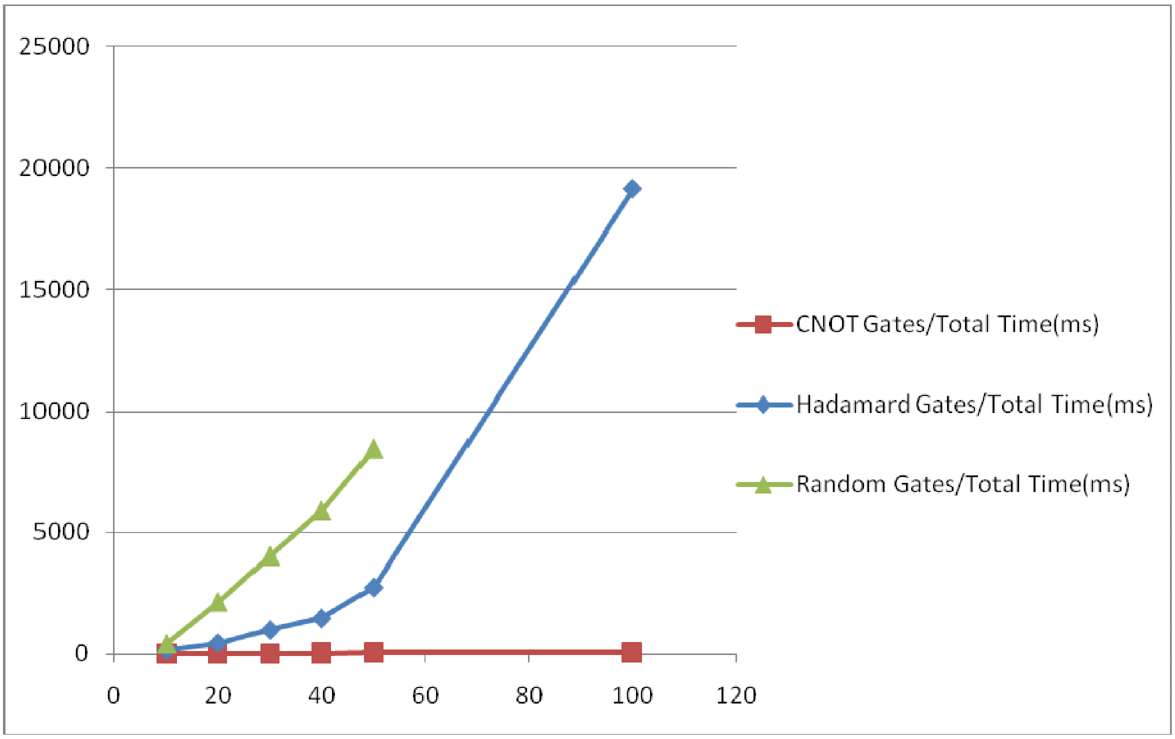
Με βάση τα παραπάνω διαγράμματα παρατηρούμε πως οι κλήσεις και των δυο αναδρομικών συναρτήσεων αυξάνονται γραμμικά με την αύξηση του αριθμού των πυλών κάτι το οποίο ήταν αναμενόμενο, αφού αυξάνοντας τις πύλες αυξάνει απλά η πρόοδος του αλγορίθμου χωρίς να αυξάνει η πολυπλοκότητά του. Παρατηρούμε επίσης πως, γραμμικά αυξάνονται όχι μόνο οι αναδρομικές κλήσεις των συναρτήσεων αλλά και οι αρχικές κλήσεις κάτι το οποί η είναι φυσιολογικό με βάση τον αλγόριθμο. Επίσης ο χρόνος εκτέλεσης αυξάνει εκθετικά (με μεγάλο ρυθμό) κάτι το οποίο οφείλεται στην μεγάλη αύξηση των διαφορετικών τιμών στο άνυσμα καταστάσεων και συνεπώς στην αύξηση το μεγέθους του δένδρου που αποθηκεύονται με αποτέλεσμα να καθυστερούν οι διεργασίες εκείνου.

3.5.4 Συγκριτικά Αποτελέσματα για διάφορες πύλες

Παρατηρώντας τα παραπάνω αποτελέσματα μπορούμε να συγκρίνουμε τη διαφορά στην ταχύτητα εκτέλεσης μεταξύ των διαφόρων τύπων πυλών (CNOT, Hadamard, Τυχαίες) και να εξάγουμε κάποια σημαντικά συμπεράσματα.



Εικόνα 3.25: *Συνολικός χρόνος εκτέλεσης σε σχέση με το πλήθος των qubit για δέκα πύλες CNOT και Hadamard*



Εικόνα 3.26: *Συνολικός χρόνος εκτέλεσης σε σχέση με το πλήθος των πυλών με εννέα qubit για πύλες CNOT, Hadamard και τυχαίες*

Στο πρώτο διάγραμμα παρατηρούμε πως ο χρόνος εκτέλεσης για πύλες Hadamard όσο αυξάνει το πλήθος των qubits γίνεται πολύ μεγαλύτερος σε σχέση με αυτόν για τις πύλες CNOT. Αυτό είναι αναμενόμενο αφού η πύλη Hadamard απαιτεί περισσότερες πράξεις και δημιουργεί νέες τιμές στο άνυσμα καταστάσεων με αποτέλεσμα η δομή που αποθηκεύονται να εισάγει επιπλέον πολυπλοκότητα στο σύστημα.

Στο δεύτερο διάγραμμα παρατηρούμε πως όσο αυξάνουν οι πύλες τόσο μεγαλώνει ο χρόνος εκτέλεσης του προγράμματος για όλους τους τύπους πυλών. Κυρίως όμως μεγαλώνει ο χρόνος εκτέλεσης των πυλών Hadamard και ακόμη περισσότερο των τυχαίων πυλών.

3.5.5 Αποτελέσματα για την Πράξη της Μέτρησης

Η πράξη της μέτρησης δεν έχει ιδιαίτερα μεγάλη πολυπλοκότητα, μιας και είναι μια απλή συνάρτηση υπολογισμού. Ο χρόνος εκτέλεσής της αυξάνει όσο αυξάνει το μέγεθος του ανύσματος καταστάσεων αλλά παραμένει πολύ μικρός σε σχέση με το χρόνο που δαπανάται για κάποια πύλη.

3.5.6 Γενικά Αποτελέσματα

Γενικά παρατηρήσαμε πως όσο αυξάνει το πλήθος των qubit στο σύστημα, ο χρόνος εκτέλεσης αυξάνει εκθετικά. Ενώ όσο αυξάνουν οι πύλες ο χρόνος αυξάνει γραμμικά. Στην περίπτωση όμως οπού οι πύλες είναι τυχαίες, ο χρόνος εκτέλεσης του προγράμματος αυξάνει κατά πολύ σε σχέση με τις άλλες περιπτώσεις. Αυτό οφείλεται στην εισαγωγή νέων καταστάσεων στο άνυσμα κατάστασης με αποτέλεσμα να μεγαλώνει η δενδρική δομή όπου αποθηκεύονται. Όλα τα παραπάνω διαγράμματα δείχνουν τα όρια εκτέλεσης του αλγορίθμου λόγω χρόνου. Δηλαδή, για παράδειγμα, για τις 10 πύλες CNOT καταφέραμε να τις εκτελέσουμε με μέχρι 13 qubit, ενώ αντίστοιχα για τις Hadamard μέχρι 10 qubit. Από εκεί κι έπειτα ο χρόνος εκτέλεσης ήταν ιδιαίτερα μεγάλος, χωρίς να μας επιτρέπει την εκτέλεση του αλγορίθμου.

Κεφάλαιο 4

Αρχιτεκτονική Υλοποίηση #1

Αφού τελειώσαμε με την μοντελοποίηση του συστήματος με χρήση αλγορίθμου και εξήγαμε κάποια συμπεράσματα γι' αυτόν, ξεκινήσαμε την υλοποίηση της πρώτης αρχιτεκτονικής υλοποίησης. Χρησιμοποιώντας αναδιατασσόμενους πόρους και γλώσσα περιγραφής υλικού αναπτύξαμε το σύστημα το οποίο θα κάνει τους κβαντικούς υπολογισμούς. Οι αναδιατασσόμενοι πόροι προσφέρουν τη δυνατότητα δημιουργίας συστημάτων αφιερωμένα σε συγκεκριμένη διεργασία, τον παραλληλισμό των διεργασιών, τη δυνατότητα προσθήκης διαφόρων περιφερειακών συσκευών (μνήμες, επεξεργαστές, διαύλους, θύρες εσόδου-εξόδου…). Παρόλα αυτά έχουν και μειονεκτήματα όπως χαμηλή ταχύτητα παραγωγής συστημάτων, χαμηλός χρονισμός συστήματος σε σχέση με συστήματα ASIC ή VLSI, περιορισμοί χωρητικότητας πόρων κ.α..

Σε αυτό το κεφάλαιο παρουσιάζουμε τη μελέτη που έγινε πριν τη δημιουργία του συστήματος, τον τρόπο υλοποίησής του και τα αποτελέσματα που πήραμε από τη συγκεκριμένη αρχιτεκτονική. Επίσης περιγράφεται αναλυτικά το σύστημα που αναπτύξαμε με τη χρήση των αναδιατασσόμενων πόρων καθώς και οι δυνατότητες και οι αδυναμίες που έχει.

4.1 Μελέτη 1^{ης} Αρχιτεκτονικής Υλοποίησης

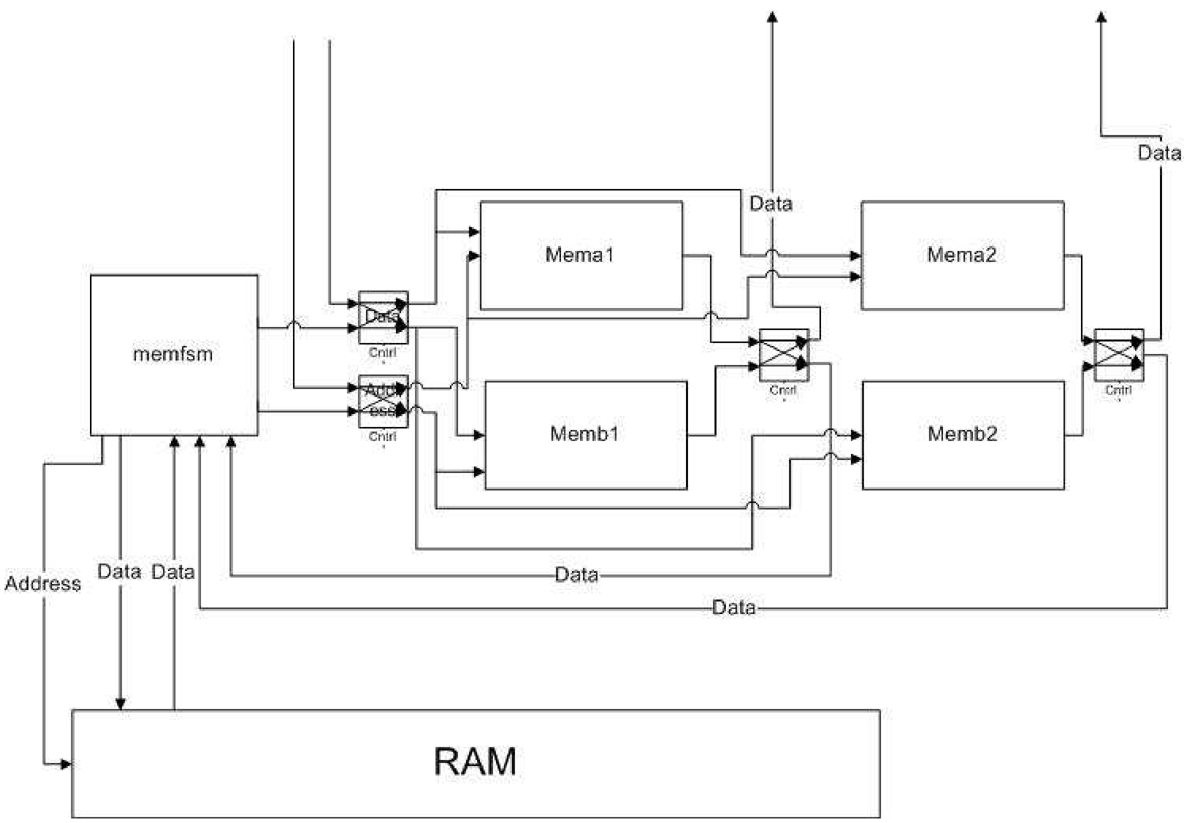
Η συγκεκριμένη αρχιτεκτονική υλοποίηση θα εξομοιώνει ένα κβαντικό επεξεργαστή και θα είναι σε θέση να εξομοιώνει τις ίδιες ενέργειες με αυτόν. Άρα θα πρέπει να έχει κάποιες μνήμες αποθήκευσης των καταστάσεων, των πυλών και των κυκλωμάτων και κάποια στοιχεία εκτέλεσης. Αυτά θα είναι μονάδες οι οποίες θα κάνουν αριθμητικές πράξεις και κάποιες μονάδες ελέγχου του συστήματος. Έτσι συγχρονίζοντας με κατάλληλο τρόπο όλα αυτά, θα μπορεί να εκτελεί τις απαιτούμενες λειτουργίες.

Το σύστημα θα χρειάζεται ένα προσθετή, ένα αφαιρετή και ένα πολλαπλασιαστή αριθμών double 64 bit (11 bit exponent, 53 bit mantissa και 1 bit το πρόσημο). Αυτές οι μονάδες θα συντονίζονται με κατάλληλο τρόπο προκειμένου να εκτελέσουν τη ζητούμενη πύλη πάνω στο άνυσμα καταστάσεων. Επίσης θα χρειαστούν μνήμες στις οποίες θα αποθηκεύονται μέρη του ανύσματος καταστάσεων, διαφορετικά κάθε φορά μιας και το άνυσμα καταστάσεων είναι αρκετά μεγάλο συνήθως για να χωρέσει σε μία εσωτερική μνήμη. Έτσι κάθε φορά θα χρησιμοποιούμε κάποια τμήματα του ανύσματος τα οποία αφού θα τα επεξεργαστούμε θα γράφουμε πίσω στην αρχική μνήμη. Τέλος θα χρειαστούμε κάποιες μνήμες αποθήκευσης των πυλών, των κυκλωμάτων καθώς και κάποιους προσωρινούς χώρους αποθήκευσης για ενδιάμεσους υπολογισμούς, τη λειτουργία του συστήματος κ.α.

Το σύστημα ακόμα θα πρέπει να έχει δυο μονάδες ελέγχου. Η πρώτη θα διαχειρίζεται τα δεδομένα από τις μνήμες. Θα τοποθετεί τα απαιτούμενα δεδομένα από την κυρίως μνήμη στις μικρότερες μνήμες και αντίστροφα όταν χρειάζεται. Η δεύτερη μονάδα θα είναι αυτή η οποία θα υπολογίζει τα αποτελέσματα. Θα διαβάζει από τις μνήμες τις καταστάσεις, την πύλη και το κύκλωμα, θα την εκτελεί και τέλος θα γράφει πίσω το αποτέλεσμα.

Για την αδιάλειπτη εκτέλεση του συστήματος χρησιμοποιήσαμε μια συστοιχία από μνήμες προκειμένου το όσην ώρα το σύστημα υπολογισμού εκτελεί εργασίες κάποιες μνήμες να γεμίζουν με τα νέα δεδομένα, έτσι ώστε μόλις χρειαστούν να βρίσκονται σε κάποια κοντινή μνήμη και απλά να αρχίσει το σύστημα υπολογισμού να χρησιμοποιεί αυτές. Έτσι το σύστημα διαχείρισης των μνημών να γράψει πίσω στην κυρίως μνήμη τα δεδομένα της μικρότερης που μόλις επεξεργάστηκε και να γεμίσει αυτή τώρα τη μνήμη με νέα δεδομένα. Με αυτόν τον τρόπο το σύστημα υπολογισμού δεν μένει ποτέ αδρανές, πέρα από τη στιγμή της αρχικοποίησης των μνημών και έτσι έχουμε καλύτερη εκμετάλλευση του συστήματος.

Μια αναπαράσταση του παραπάνω φαίνεται εδώ:



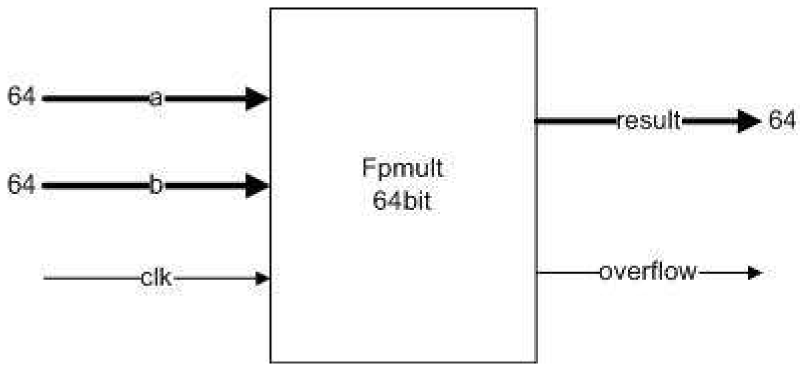
Εικόνα 4.1: *Αρχιτεκτονική υλοποίησης υποσυστήματος μνημών*

4.2 Υλοποίηση 1ης Αρχιτεκτονικής Υλοποίησης

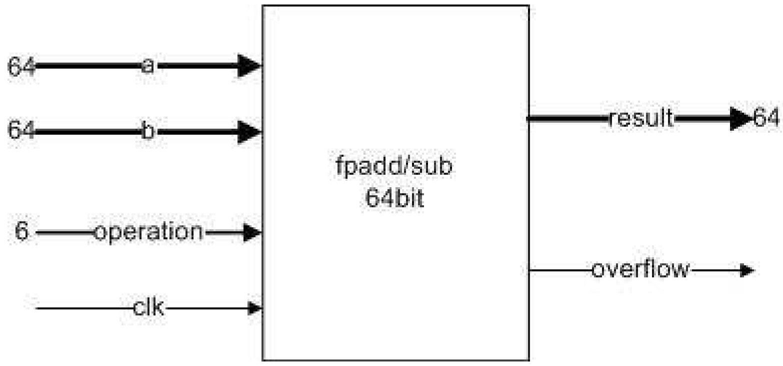
Για την ανάπτυξη του συστήματος χρησιμοποιήσαμε τη γλώσσα περιγραφής υλικού VHDL (VHSIC hardware description language) και το αναπτυξιακό εργαλείο ISE Xilinx 7.1 και Xilinx Core Generator. Το πρώτο εργαλείο χρησιμοποιείται για την ανάπτυξη του κώδικα σε γλώσσα περιγραφής υλικού, τη δημιουργία του σχηματικού του, τη σύνθεσή του, τη δημιουργία του net list του συστήματος κ.α.. Το Xilinx Core Generator δημιουργεί αυτοματοποιημένα βασικές μονάδες λειτουργικότητας, όπως μνήμες καταχωρητές, αριθμητικές μονάδες κ.α.. Για την ανάπτυξη του συστήματος χρησιμοποιήσαμε ως πλατφόρμα ανάπτυξης μια FPGA (Field Programmable Gate Array) της οικογένειας Virtex4 και συγκεκριμένα την xc4vfx12-ff668-12. Το σύστημα που αναπτύξαμε υποστηρίζει 9 qubit, δηλαδή έχει άνυσμα καταστάσεων 512 θέσεων. Ολόκληρο το σύστημα είναι έτσι κατασκευασμένο ώστε να μπορεί εύκολα και γρήγορα να αλλάξει το πλήθος των qubit που θα υποστηρίζει προκειμένου να εξομοιώσουμε οποιοδήποτε επιθυμητό σύστημα. Επίσης οι μικρότερες εσωτερικές μνήμες θα αποθηκεύουν μέχρι 128 καταστάσεις η κάθε μια.

Αρχικά δημιουργήσαμε, με τη χρήση του εργαλείου Xilinx Core Generator, μια μονάδα πολλαπλασιασμού double αριθμών 64 bit (11 bit exponent, 53 bit mantissa και 1 bit το πρόσημο) και μια μονάδα πρόσθεσης και αφαίρεσης πάλι double αριθμών 64 bit (11 bit exponent, 53 bit mantissa και 1 bit το πρόσημο). Η μονάδα πολλαπλασιασμού χρειάζεται 21 κύκλους ρολογιού συστήματος για να εκτελέσει ένα πολλαπλασιασμό 64 bit ενώ ο προσθαφαιρετής χρειάζεται 11 κύκλους ρολογιού. Και οι δυο μονάδες υποστηρίζουν ομοχειρία (pipeline) με αποτέλεσμα να μπορούμε να τις τροφοδοτούμε και να παίρνουμε αποτελέσματα ταχύτερα. Για την ταχύτερη λειτουργία των μονάδων χρησιμοποιήσαμε κάποιους ενδιάμεσους καταχωρητές για την αποθήκευση των αποτελεσμάτων των μονάδων.

Αυτές οι δυο μονάδες έχουν την εξής διεπαφή:



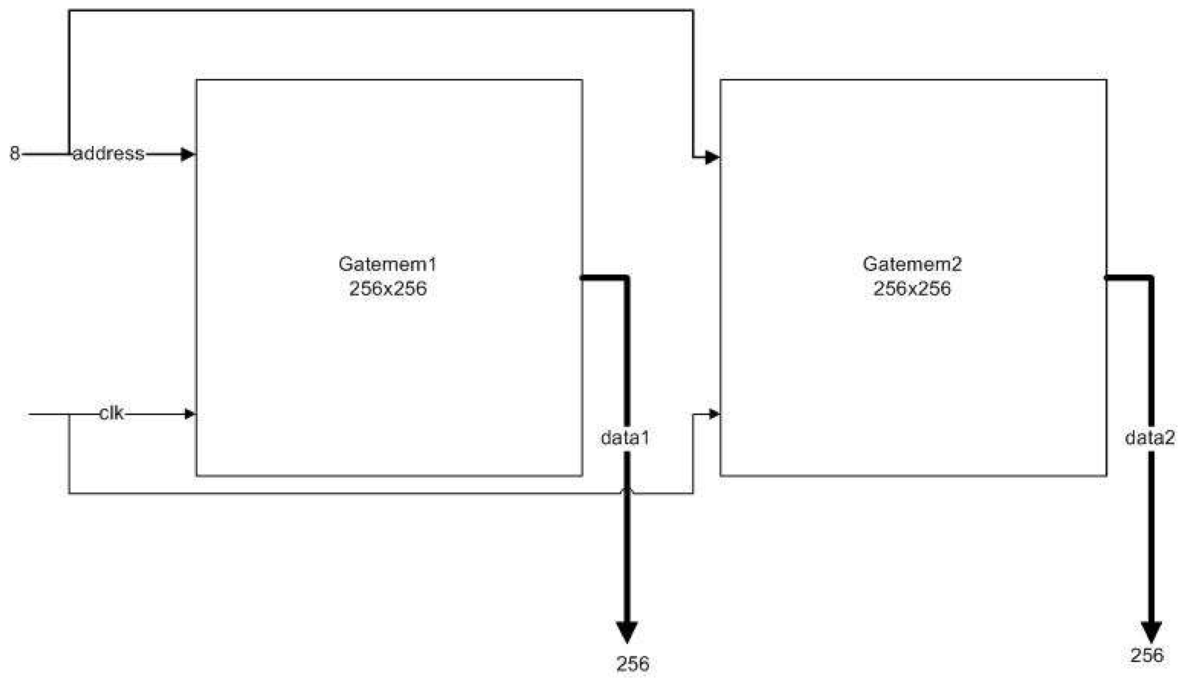
Εικόνα 4.2: *Σχηματικό διάγραμμα πολλαπλασιαστή double αριθμών*



Εικόνα 4.3: *Σχηματικό διάγραμμα προσθαφαιρετή double αριθμών*

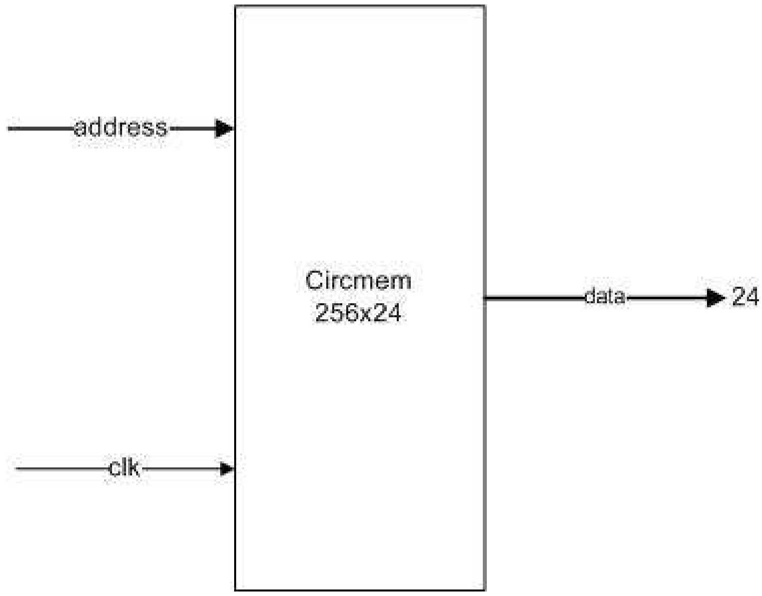
Ακόμα, δημιουργήσαμε δυο μνήμες μόνο προς ανάγνωση για τα κυκλώματα. Στην μνήμη των πυλών θα αποθηκεύονται οι πύλες. Η κάθε πύλη αποτελείται, όπως είδαμε, από τέσσερα μιγαδικά στοιχεία. Άρα συνολικά θα αποτελείται από οκτώ double αριθμούς, δηλαδή 8*64 bit=512 bit. Η συγκεκριμένη μνήμη χωρίστηκε σε δυο κομμάτια μιας και η τεχνολογία που χρησιμοποιήσαμε επιτρέπει μνήμες μέγιστου πλάτους 256 bit. Η μνήμη στην οποία θα αποθηκεύονται τα κυκλώματα θα χρειάζεται να αποθηκεύει αρχικά το κύκλωμα το οποίο αποτελείται από 9 σύμβολα (όσα τα qubit) που χρειαζόμαστε για την αναπαράσταση του κυκλώματος. Αυτά είναι τα “0”, “1”, “X” και “G” και χρειαζόμαστε 2bit για να τα αναπαραστήσουμε. Δηλαδή συνολικά 9*2 bit=18 bit. Ακόμη όμως χρειάζεται να αποθηκεύσουμε κάποιο «κωδικό» για τον τύπο της πύλης, αν δηλαδή είναι τύπου control κ.α.. Γι’ αυτό το λόγω θα χρειαστούμε άλλα 2 bit. Ακόμη θα χρειαστούμε να αποθηκεύσουμε τη θέση που εφαρμόζεται η πύλη στο κύκλωμα. Σε ποιο qubit δηλαδή θα εφαρμοστεί η συγκεκριμένη πύλη. Άρα χρειαζόμαστε ακόμα 4 bit (για να μπορούμε να ξέρουμε τη θέση από το 1 μέχρι το 9). Έτσι συνολικά προκύπτει ότι η μνήμη των κυκλωμάτων θα έχει 18+2+4 bit = 24 bit. Και οι δυο μνήμες θα είναι «παράλληλες», αφού η κάθε πύλη συνδυάζεται με το κύκλωμά της και αντίστροφα. Γι’ αυτό ορίσαμε πως και οι δυο μνήμες θα έχουν κοινό βάθος και ίσο με 256. Άρα το σύστημα θα μπορεί να εξομοιώνει μέχρι 256 πύλες. Για αυτό το λόγω διατηρείται σε ένα καταχωρητή 7bit η διεύθυνση στην οποία βρίσκεται η τελευταία πύλη έτσι ώστε εκεί να τερματίσει η εκτέλεση του συστήματος.

Η διεπαφή τους παρουσιάζεται εδώ:



Εικόνα 4.4: *Σχηματικό διάγραμμα μνήμης πυλών*

Εικόνα 4.5: *Ισοδύναμο σχηματικό διάγραμμα μνήμης πυλών*

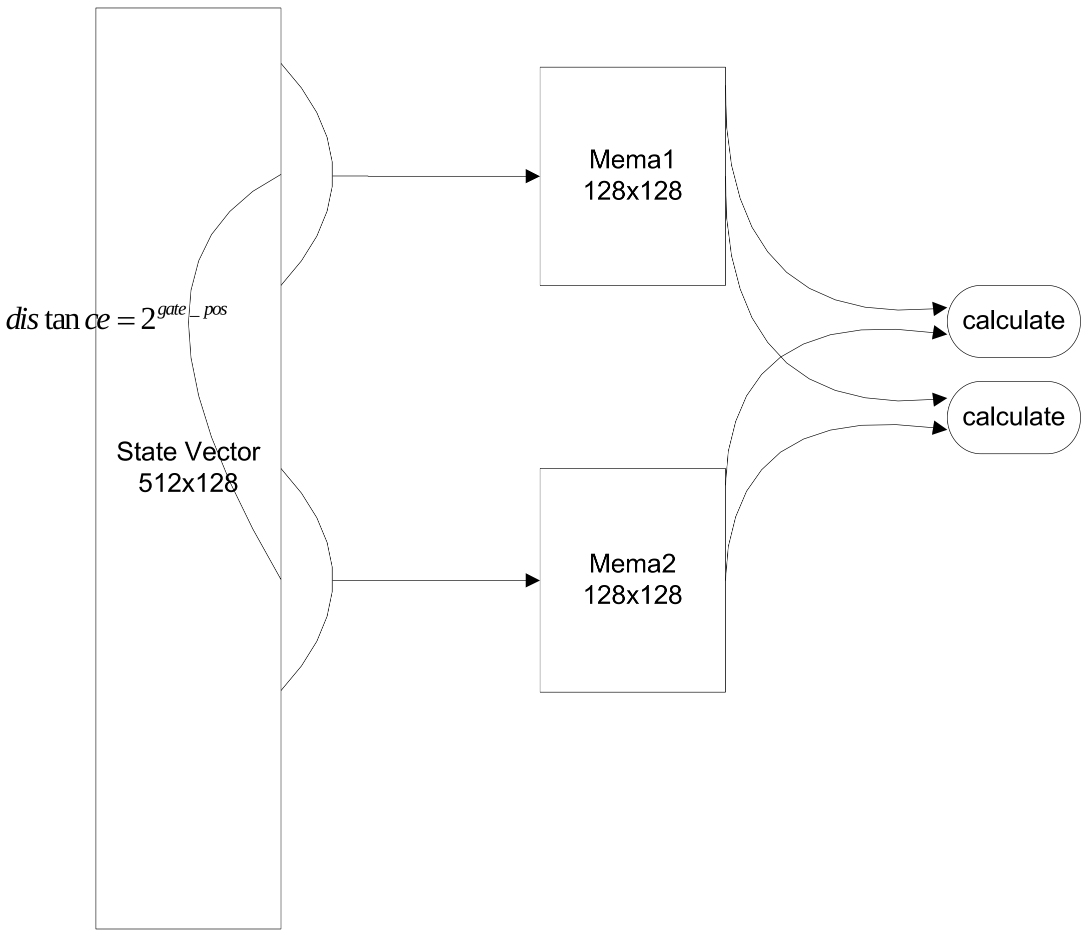


Εικόνα 4.6: *Σχηματικό διάγραμμα μνήμης κυκλωμάτων*

Η κυρίως μνήμη θα θέλαμε να έχει τη δυνατότητα να χωρέσει ολόκληρο το άνυσμα καταστάσεων. Θα θέλαμε δηλαδή να έχει βάθος $2^{qubits} = 2^9 = 512bit$ τουλάχιστον και πλάτος όσα bit χρειαζόμαστε για να περιγράψουμε μια κατάσταση. Αυτό είναι ίσο με το μέγεθος ενός μιγαδικού αριθμού, δηλαδή ίσο με δυο αριθμούς double 64 bit. Άρα τελικά θέλουμε να έχει χωρητικότητα τουλάχιστο 512x128.

Οι μικρότερες εσωτερικές μνήμες θα είναι τέσσερις και θα χωρίζονται σε δυο ζευγάρια. Η κάθε μια από αυτές θα έχει βάθος 128 θέσεων (το ορίσαμε τόσο ώστε να χωράει αρκετές καταστάσεις χωρίς να είναι ιδιαίτερα μεγάλες και να χωράνε σε κάποιο πραγματικό σύστημα). Το ζευγάρι από μνήμες θα δέχεται καταστάσεις από το άνυσμα κατάστασης το οποίο θα βρίσκεται στην κυρίως μνήμη. Γι' αυτό ο λόγω το πλάτος της κάθε μνήμης θα είναι κι αυτό 128 bit. Κάθε μνήμη από το ζευγάρι θα γεμίζει με στοιχεία του ανύσματος κατάστασης ανάλογα με τη θέση της πύλης στο κύκλωμα για κάθε πύλη. Μαζί με κάθε μνήμη θα διατηρούνται καταχωρητές οι οποίοι θα αποθηκεύουν στοιχεία όπως η διεύθυνση μέχρι την οποία υπάρχουν δεδομένα, οι διευθύνσεις του ανύσματος καταστάσεων που περιλαμβάνουν κ.α.. Ο λόγος που χρησιμοποιούμε τις μικρότερες μνήμες είναι γιατί το άνυσμα καταστάσεων αυξάνει εκθετικά με την αύξηση των qubit και συνήθως είναι αρκετά μεγάλο για να χωρέσει μαζί με το κύκλωμα. Έτσι χρειαζόμαστε κάποιες μικρότερες μνήμες ώστε να γράφουμε τμήματα του ανύσματος καταστάσεων και να τα επεξεργαζόμαστε ώσπου τελικά να τα γράφουμε πίσω στο άνυσμα καταστάσεων.

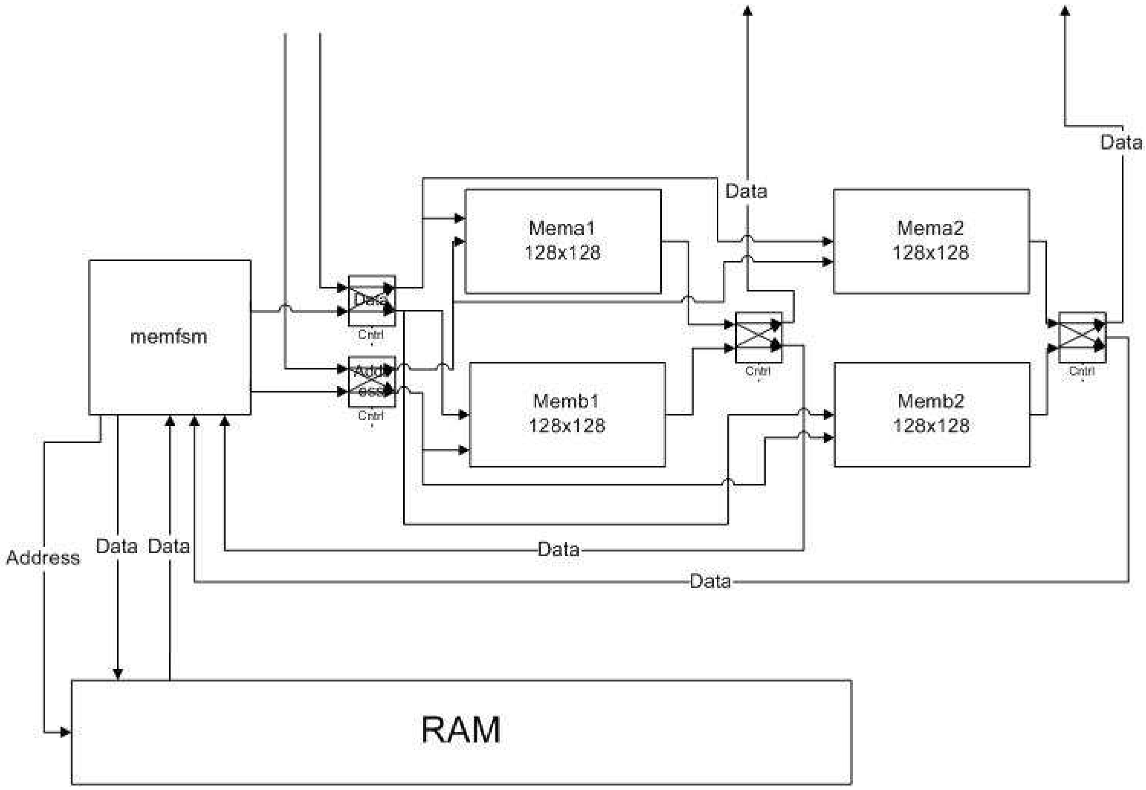
Αυτό το ζευγάρι μνημών θα δούλευε με τρόπο που φαίνεται εδώ:



Εικόνα 4.7: *Σχηματικό διάγραμμα λειτουργίας μνημών*

Αν χρησιμοποιούσαμε μόνο ένα ζευγάρι από μικρές μνήμες, τότε την διάρκεια κατά την οποία θα γέμιζαν οι μνήμες με δεδομένα από την κυρίως μνήμη, το υπόλοιπο σύστημα θα βρισκόταν σε αδράνεια, ενώ όση ώρα γινόταν υπολογισμοί το υποσύστημα το οποίο γεμίζει και αδειάζει τις μικρότερες μνήμες θα ήταν σε αδράνεια. Γι' αυτό το λόγω χρησιμοποιούμε δυο ζευγάρια μνημών έτσι ώστε τα υποσυστήματα που γεμίζουν τις μνήμες και κάνουν τους υπολογισμούς να μένουν όσο το δυνατόν λιγότερο χρόνο σε αδράνεια. Συνεπώς όση ώρα το υποσύστημα υπολογισμού επεξεργάζεται τις καταστάσεις από το ζευγάρι μνημών "a" το υποσύστημα που διαχειρίζεται τις μνήμες θα γράφει στην κυρίως μνήμη τα αποτελέσματα από τις μικρότερες μνήμες "b" και θα τις γεμίζει με νέες καταστάσεις από το άνυσμα καταστάσεων που βρίσκεται στην κυρίως μνήμη.

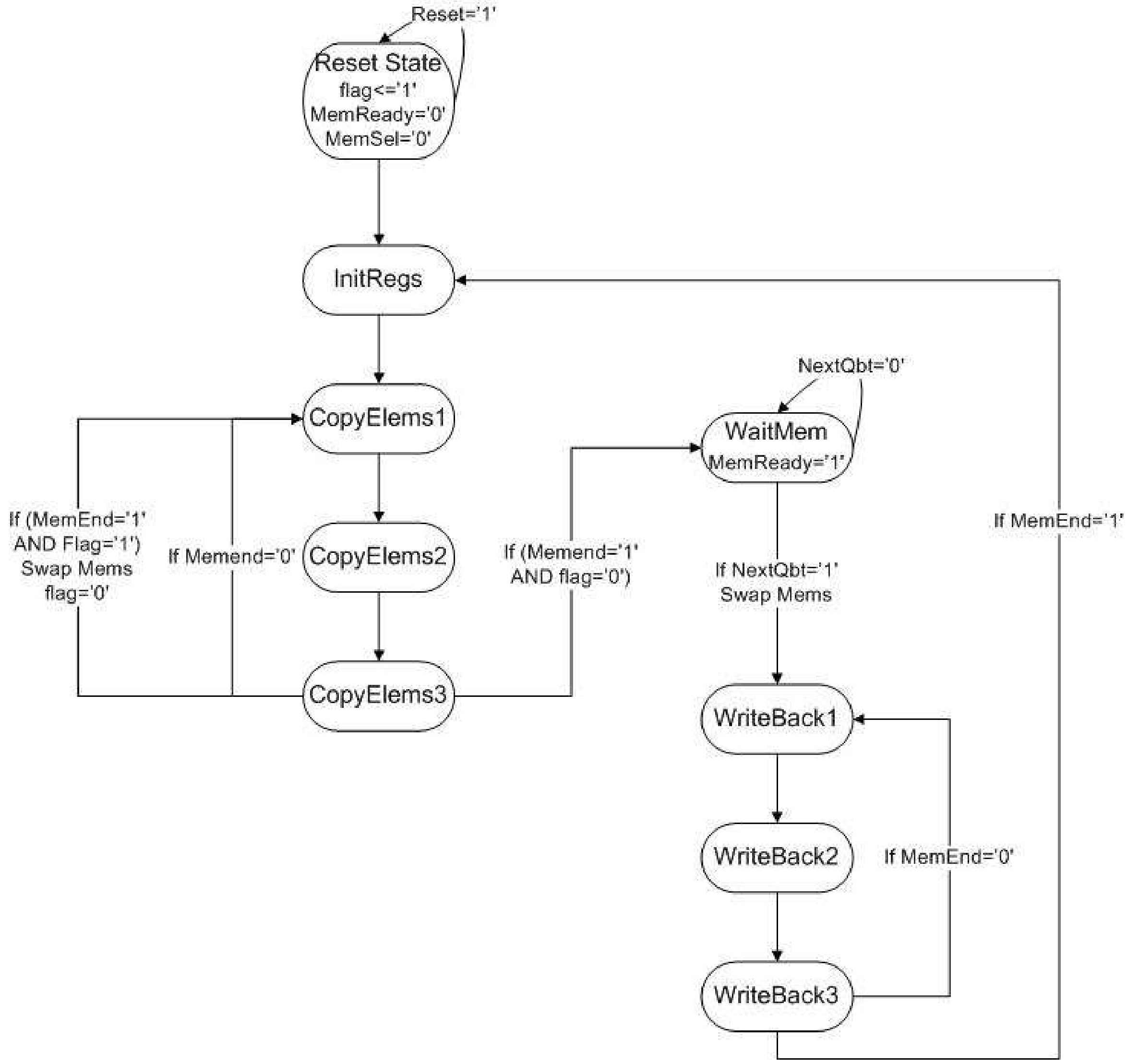
Ο τρόπος διασύνδεσής τους φαίνεται παρακάτω:



Εικόνα 4.8: *Αρχιτεκτονική υλοποίησης υποσυστήματος μνημών*

Ο συντονισμός όλων αυτών απαιτεί την ύπαρξη κάποιας μονάδας ελέγχου. Έτσι αναπτύξαμε μια μονάδα ελέγχου για την λειτουργία και το συντονισμό του συστήματος των μνημών και μια μονάδα για τον έλεγχο της εκτέλεσης των πυλών στο σύστημα.

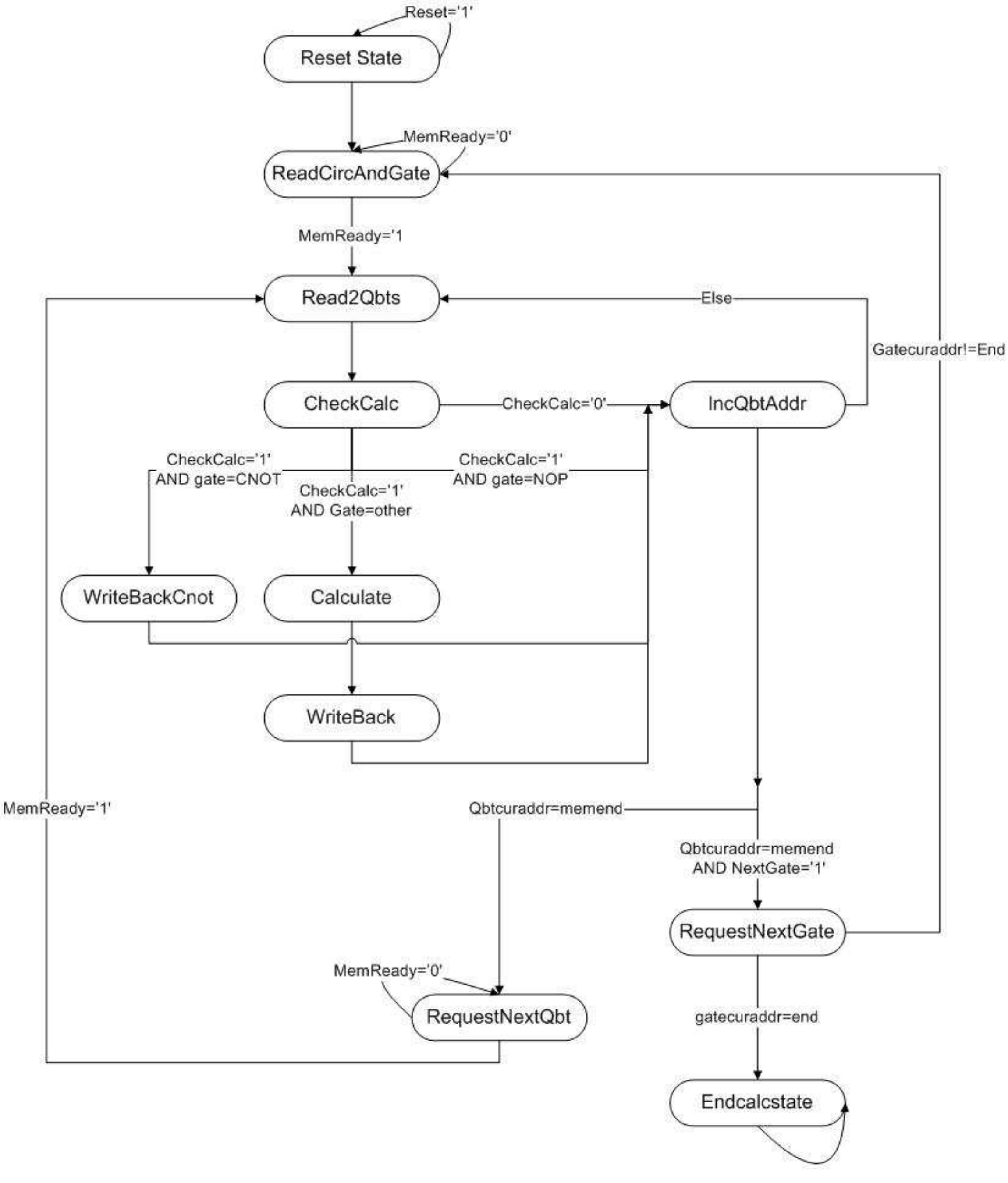
Η μονάδα ελέγχου των μνημών έχει ως σκοπό τη διαχείριση της κυρίως μνήμης που περιέχει το άνυσμα καταστάσεων και των μικρότερων μνημών που περιέχουν τμήματα αυτού του ανύσματος καθώς και την ενημέρωση της μονάδας υπολογισμού για το τι συμβαίνει κάθε στιγμή. Η μονάδα αυτή υλοποιείται με τη χρήση μιας μηχανής πεπερασμένων καταστάσεων (finite state machine-fsm) και διαχειρίζεται όλα τα παραπάνω με τον τρόπο που παρουσιάζεται και εξηγείται παρακάτω.



Εικόνα 4.9: *Σχηματικό διάγραμμα λειτουργίας υποσυστήματος μνημών*

Το υποσύστημα διαχείρισης των μνημών ξεκινά με την αρχικοποίηση τους. Έτσι αρχικά γεμίζει και τα δυο ζευγάρια μνημών με τα κατάλληλα δεδομένα του ανύσματος καταστάσεων. Αφού αρχικοποιήσει τους απαιτούμενους καταχωρητές αρχίζει την αντιγραφή των στοιχείων και με τη χρήση του σήματος flag εξασφαλίζει πως και τα δυο ζευγάρια από μνήμες θα γεμίζουν με τα κατάλληλα δεδομένα. Αφού τελειώσει με αυτό, ενημερώνει το υπόλοιπο σύστημα πως είναι έτοιμα κάνοντας το σήμα memready=1 και μεταβαίνει στην κατάσταση WaitMem οπού περιμένει το υπόλοιπο σύστημα να επεξεργαστεί τα δεδομένα από κάποιο ζευγάρι μνημών. Όταν έχει τελειώσει η επεξεργασία ενός ζευγαριού μνημών ενημερώνεται το υποσύστημα μνημών με το σήμα NextQbt. Τότε εναλλάσονται οι μνήμες (Swap mems) και το υποσύστημα υπολογισμού συνεχίζει τους υπολογισμούς στο άλλο ζευγάρι μνημών. Το υποσύστημα μνημών γράφει πίσω στην αρχική μνήμη και το άνυσμα καταστάσεων τα δεδομένα του ζευγαριού των μνημών στις κατάλληλες θέσεις. Μετά από αυτό ξεκινάει πάλι να γεμίζει με νέα δεδομένα το ζευγάρι αυτό των μνημών κ.ο.κ..

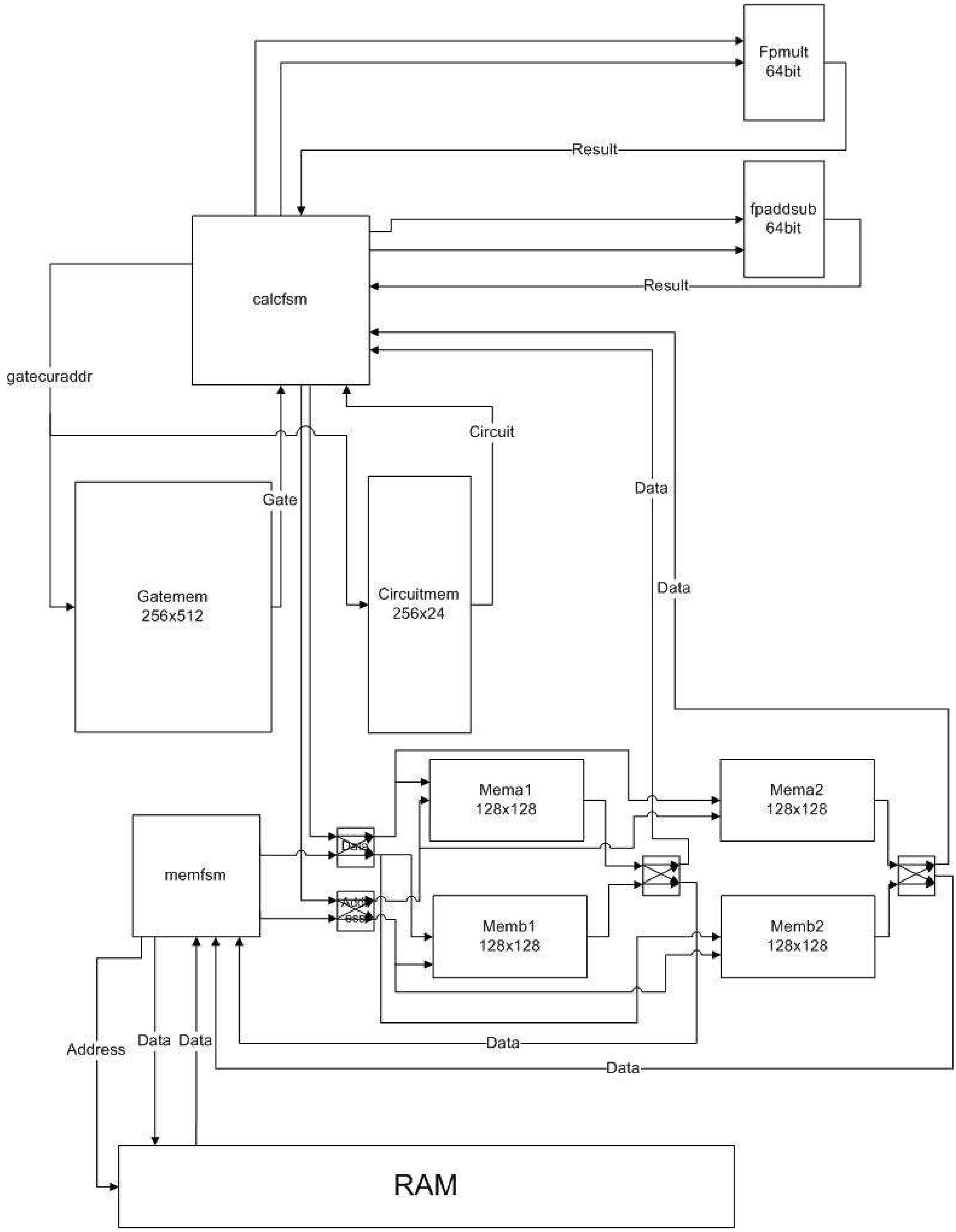
Η μονάδα υπολογισμού έχει ως σκοπό τη διαχείριση των αριθμητικών μονάδων, των μνημών των πυλών και των κυκλωμάτων και ακόμα των δεδομένων του ενός από τα δυο ζευγάρια μνημών κάθε στιγμή. Με τη χρήση αυτών των λειτουργικών μονάδων και μιας μονάδας λογικής εκτελεί τις απαραίτητες πράξεις κάθε στιγμή προκειμένου να υπολογίσει τις απαιτούμενες ποσότητες. Η μονάδα αυτή υλοποιείται με τη χρήση μιας μηχανής πεπερασμένων καταστάσεων (finite state machine-fsm) και διαχειρίζεται όλα τα παραπάνω με τον τρόπο που παρουσιάζεται και εξηγείται παρακάτω.



Εικόνα 4.9: *Σχηματικό διάγραμμα λειτουργίας υποσυστήματος υπολογισμού*

Η μονάδα εκτέλεσης, αφού το σήμα reset πάρει την τιμή «0», διαβάζει από τις αντίστοιχες μνήμες την πύλη και το κύκλωμα (ReadCircAndGate). Έπειτα, μόλις λάβει το σήμα πως οι μνήμες είναι έτοιμες, διαβάζει τη πρώτη κατάσταση από κάθε μνήμη (Read2Qubits) και έπειτα ελέγχει με βάση το κύκλωμα αν πρέπει να γίνει κάποια πράξη (CheckCalc). Αν όχι τότε προχωράει στην επόμενη θέση των μνημών (IncQbtAddr) και κάνει κάποιους ελέγχους. Ελέγχει αν έχουν τελειώσει οι καταστάσεις στη μνήμη οπότε ενεργοποιεί το σήμα RequestNextQbt με το οποίο ενημερώνει τη μονάδα διαχείρισης μνημών προκειμένου να κάνει την αλλαγή του ζευγαριού των μνημών. Μόλις έρθουν (Mem_ready=1) το σύστημα μεταβαίνει πάλι στην κατάσταση ανάγνωσης των καταστάσεων (Read2Qbts). Αν πάλι έχουν τελειώσει οι καταστάσεις στις μνήμες και στο άνυσμα καταστάσεων, τότε θα πρέπει να ξεκινήσουμε να εκτελούμε την επόμενη πύλη (RequestNextGate) μιας και οι υπολογισμοί της τρέχουσας θα έχουν τελειώσει. Αν υπάρχει νέα πύλη τότε συνεχίζουμε με αυτήν μεταβαίνοντας στην κατάσταση ανάγνωσης της πύλης και του κυκλώματος (ReadCircAndGate). Αν πάλι δεν υπάρχει άλλη πύλη τότε μεταβαίνουμε στην κατάσταση (EndCalcState) η οποία σηματοδοτεί το πέρας των υπολογισμών. Σε κάθε άλλη περίπτωση απλά συνεχίζει με το επόμενο ζευγάρι από καταστάσεις από την κατάσταση Read2Qubits. Στη περίπτωση που χρειάζεται κάποιος υπολογισμός αυτός εκτελείται γράφεται το αποτέλεσμα πίσω στη μνήμη και το σύστημα συνεχίζει από την κατάσταση IncQbtAddr. Η εκτέλεση των πράξεων για ένα ζευγάρι από καταστάσεις μπορεί να χρειαστεί μέχρι και 64 κύκλους ρολογιού συστήματος για να περατωθεί.

Το συνολικό σύστημα το οποίο αναπτύχθηκε παρουσιάζεται εδώ:



Εικόνα 4.10: *Σχηματικό διάγραμμα συνολικού συστήματος*

4.3 Αποτελέσματα $1^{ης}$ Αρχιτεκτονικής Υλοποίησης

Το σύστημα το οποίο αναπτύχθηκε είναι σε θέση να εξομοιώνει κβαντικούς επεξεργαστές. Μπορεί να εκτελεί οποιοδήποτε κβαντικό κύκλωμα ανεξαρτήτως μεγέθους, αλλάζοντας κάποιες βασικές παραμέτρους μόνο κάθε φορά (όπως μέγεθος μνημών καταχωρητών κ.α.). Αυτή η δύναμη του συστήματος όμως προκαλεί επιπλέον καθυστέρηση. Η πολυπλοκότητα και το μέγεθος των μονάδων προκαλούν τη μείωση της ταχύτητας της συχνότητας του ρολογιού με αποτέλεσμα το σύστημα να είναι ιδιαίτερα αργό σε σχέση με τη συχνότητα που μπορούν να επιτύχουν αυτές οι αναπτυξιακές μονάδες. Επίσης το μεγάλο πλήθος διακίνησης και επεξεργασίας των δεδομένων επιβαρύνει το σύστημα με επιπλέον κύκλους λειτουργίας το οποίο πάλι επιφέρει μικρότερη ταχύτητα του συστήματος.

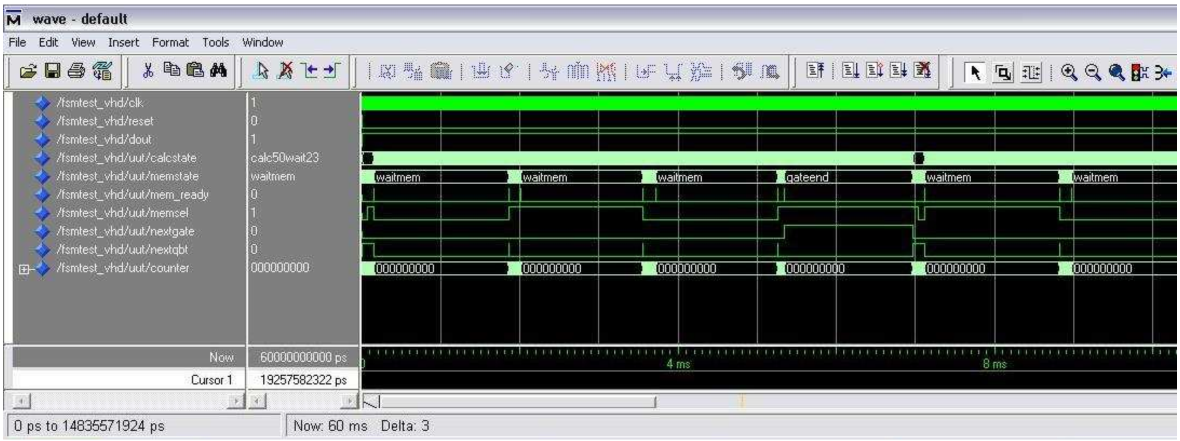
Την ορθή λειτουργία του συστήματος την επιβεβαιώσαμε με τη χρήση ενός εξομοιωτή της γλώσσας VHDL. Έτσι με τη χρήση του εργαλείου ModelSim v6.0a προσομοιώσαμε τη λειτουργία του συστήματος. Αρχικά με τη χρήση του εργαλείου Xilinx Ise v7.1 συνθέσαμε το σύστημα και πήραμε τα εξής αποτελέσματα:

```
=========================================================================
*                            Final Report                              *
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 4vfx12ff668-12

 Number of Slices:                    2198  out of   5472    40%
 Number of Slice Flip Flops:          2986  out of  10944    27%
 Number of 4 input LUTs:              3924  out of  10944    35%
 Number of bonded IOBs:                  3  out of    320     0%
 Number of GCLKs:                        1  out of     32     3%


=========================================================================
TIMING REPORT


Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 2986  |
-----------------------------------+------------------------+-------+

Timing Summary:
---------------
Speed Grade: -12

   Minimum period: 5.031ns (Maximum Frequency: 198.785MHz)
   Minimum input arrival time before clock: 7.482ns
   Maximum output required time after clock: 3.921ns
   Maximum combinational path delay: 2.371ns
=========================================================================
```

Εικόνα 4.11: *Αποτελέσματα σύνθεσης $1^{ης}$ αρχιτεκτονικής υλοποίησης*

Από τα παραπάνω μπορούμε να ξεχωρίσουμε πως έχουμε καταλάβει περίπου το ένα τρίτο του συνολικά διαθέσιμου χώρου και πως το σύστημα θα μπορεί να τρέχει με συχνότητα 198,785 MHz (περίοδος 5,031ns).

Η προσομοίωση με τη χρήση του εργαλείου ModelSim v6.0a έγινε αφού κάναμε post place and route έτσι ώστε να δώσει ορθά αποτελέσματα όπως θα έδινε το σύστημα αν βρισκόταν μέσα σε μια FPGA και έδωσε το παρακάτω αποτέλεσμα:



Εικόνα 4.12: *Αποτελέσματα προσομοίωσης 1ης αρχιτεκτονικής υλοποίησης*

Από εδώ μπορούμε να παρατηρήσουμε τη συμπεριφορά του συστήματος. Οι χρόνοι που φαίνονται στην παραπάνω εικόνα και χρησιμοποιούνται στην περιγραφή της είναι χρόνοι που δεν αντιπροσωπεύουν την πραγματική ταχύτητα του συστήματος. Αρχικά γεμίζουν και τα δυο ζευγάρια μνημών όσο η κατάσταση caclcstate παραμένει στάσιμη. Αφού τελειώσει η αρχικοποίηση η κατάσταση memstate αδρανεί στην κατάσταση WaitMem ενώ αρχίζουν οι υπολογισμοί. Κατά τη διάρκεια της αρχικοποίησης παρατηρούμε πως το σήμα MemSel αλλάζει έτσι ώστε να γεμίσουν και τα δυο ζευγάρια μνημών. Αργότερα, αφού τελειώσει η επεξεργασία των δεδομένων του πρώτου ζευγαριού μνημών (λίγο πριν τα 2ms) το σήμα NextQbt ενεργοποιείται από το υποσύστημα υπολογισμού και τότε το υποσύστημα διαχείρισης μνημών εναλλάσσει τα ζευγάρια μνημών (το σήμα MemSel ενεργοποιείται) και το υποσύστημα υπολογισμού συνεχίζει με το νέο ζευγάρι μνημών. Το υποσύστημα διαχείρισης μνημών τότε από την κατάσταση WaitMem ξεκινά να γράφει πίσω στο άνυσμα καταστάσεων τα επεξεργασμένα δεδομένα και να ξαναγεμίζει με δεδομένα το ένα ζευγάρι μνημών. Αυτό γίνεται μέχρι να τελειώσει η διάσχιση του ανύσματος καταστάσεων όπου τότε έχει εκτελεστεί η τρέχουσα πύλη και το σήμα NextGate ενεργοποιείται (λίγο μετά τα 5ms) από το υποσύστημα διαχείρισης μνημών. Τότε, την επόμενη φορά όπου το υποσύστημα εκτέλεσης δεν θα έχει νέα στοιχεία στο ζευγάρι μνημών που διαχειρίζεται, δεν θα ζητήσει να γεμίσουν οι μνήμες, αλλά θα διαβάσει τη νέα πύλη (7ο ms) και το αντίστοιχο κύκλωμα και θα ξεκινήσει την εκτέλεση από την αρχή.

Όταν θα έχει εκτελεστεί και η τελευταία πύλη τότε το σύστημα μεταβαίνει σε κατάσταση τερματισμού.

Το σύστημα είναι σε θέση να εκτελεί οποιαδήποτε πύλη του δοθεί με το αντίστοιχο κύκλωμα. Από το παραπάνω σχήμα της προσομοίωσης παρατηρήσαμε πως για να εξομοιώσει το σύστημα μια ολόκληρη πύλη χρειάζεται 26317 κύκλους. Αυτοί οι κύκλοι έχουν υπολογιστεί στην χειρότερη περίπτωση όπου όλοι οι υπολογισμοί θέλουν το μέγιστο χρόνο και τα ζευγάρια μνημών γεμίζουν με στοιχεία του ανύσματος όσο το δυνατόν περισσότερες φορές. Έτσι με περίοδο ρολογιού 5,013ns, το σύστημα είναι σε θέση να εκτελεί μια πύλη κάθε 132ms περίπου.

Κεφάλαιο 5

Αρχιτεκτονική Υλοποίηση #2

Χρησιμοποιώντας και αυτή τη φορά αναδιατασσόμενους πόρους και γλώσσα περιγραφής υλικού αναπτύξαμε ένα νέο σύστημα το οποίο θα κάνει κβαντικούς υπολογισμούς. Το σύστημα αυτό θα εκμεταλλεύεται το γεγονός πως οι υπολογισμοί που γίνονται από κβαντικά κυκλώματα συχνά αποτελούνται από λίγες διαφορετικές καταστάσεις. Αυτό είχαμε εκμεταλλευτεί και στη δημιουργία του αλγόριθμου μοντελοποίησης και είχαμε δημιουργήσει τη δενδρική δομή αποθήκευσης καταστάσεων προκειμένου να χρησιμοποιήσουμε λιγότερο χώρο στη μνήμη του υπολογιστή.

Σε αυτό το κεφάλαιο παρουσιάζουμε τη μελέτη που έγινε πριν τη δημιουργία του νέου αυτού συστήματος, τον τρόπο υλοποίησής του και τα αποτελέσματα που πήραμε από τη συγκεκριμένη αρχιτεκτονική. Επίσης περιγράφεται αναλυτικά το σύστημα που αναπτύξαμε με τη χρήση των αναδιατασσόμενων πόρων καθώς και οι δυνατότητες και οι αδυναμίες που έχει.

5.1 Μελέτη 2^{ης} Αρχιτεκτονικής Υλοποίησης

Η 2η αρχιτεκτονική υλοποίηση θα εξομοιώνει και αυτή τις λειτουργίες ενός κβαντικού επεξεργαστή αλλά υπό κάποιες προϋποθέσεις. Ο τρόπος λειτουργίας του θα βασίζεται σε πίνακες αναζήτησης στοιχείων. Εκεί θα αναζητά το κατάλληλο αποτέλεσμα μιας πύλης και ενός κυκλώματος και θα το γράφει. Έτσι θα λειτουργεί με τέτοιο τρόπο ώστε να μην έχει ανάγκη αριθμητικές μονάδες αφού δεν θα χρειάζεται ποτέ να εκτελεί πράξεις.

Αυτή η αρχιτεκτονική σχεδίαση θα εκμεταλλεύεται το γεγονός πως το άνυσμα καταστάσεων αποτελείται από λίγες διαφορετικές καταστάσεις. Έτσι ανάλογα με τον τύπο της πύλης, το κύκλωμα και την κατάσταση θα έχουμε και το αντίστοιχο αποτέλεσμα. Ο λόγος για τον οποίο δεν θα υποστηρίζονται πύλες πολλών διαφορετικών τύπων είναι γιατί έτσι θα υπήρχε η περίπτωση να έχουμε πολλές πιθανές καταστάσεις και δεν θα μπορούσαμε να έχουμε μια τόσο μεγάλη μνήμη η οποία να τα αποθηκεύει. Συνεπώς θα θέλουμε ένα πίνακα αναζήτησης στοιχείων με το κατάλληλο μέγεθος, ο οποίος θα δίνει το επιθυμητό αποτέλεσμα κάθε φορά

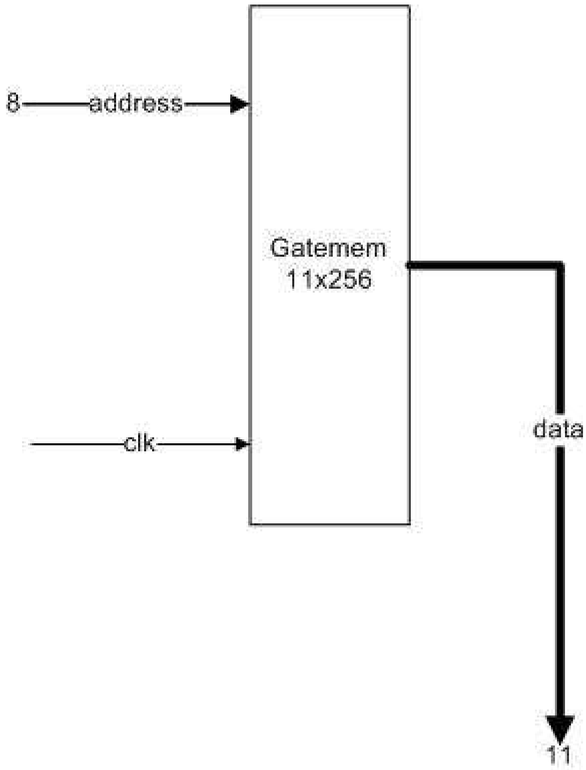
Το υποσύστημα μνημών θα αποτελείται από μια μνήμη πυλών η οποία θα περιέχει τον τύπο της πύλης και το κύκλωμα καθώς και την κυρίως μνήμη. Η κυρίως μνήμη θα αποτελείται από όλες τις καταστάσεις αλλά αντί για τιμές σε κάθε μια θα αποθηκεύει κάποιο αριθμό ο οποίος θα δείχνει ποια τιμή κατάστασης θα έχει. Τα δυο ζευγάρια μνημών (2x2x128x128) δεν θα υπάρχουν σε αυτή την αρχιτεκτονική υλοποίηση ούτε και η μνήμη στην οποία θα αποθηκεύεται η πύλη αφού θα υποστηρίζονται μόνο κάποιες κλασσικές πύλες.

5.2 Υλοποίηση 2^{ης} Αρχιτεκτονικής Υλοποίησης

Για την ανάπτυξη του συστήματος χρησιμοποιήσαμε τη γλώσσα περιγραφής υλικού VHDL (VHSIC hardware description language) και το αναπτυξιακό εργαλείο ISE Xilinx 7.1 και Xilinx Core Generator. Το πρώτο εργαλείο χρησιμοποιείται για την ανάπτυξη του κώδικα σε γλώσσα περιγραφής υλικού, τη δημιουργία του σχηματικού του, τη σύνθεσή του, τη δημιουργία του net list του συστήματος κ.α.. Το Xilinx Core Generator δημιουργεί αυτοματοποιημένα βασικές μονάδες λειτουργικότητας, όπως μνήμες καταχωρητές, αριθμητικές μονάδες κ.α.. Για την ανάπτυξη του συστήματος χρησιμοποιήσαμε ως πλατφόρμα ανάπτυξης μια FPGA (Field Programmable Gate Array) της οικογένειας Virtex4 και συγκεκριμένα την xc4vfx12-ff668-12. Το σύστημα που αναπτύξαμε υποστηρίζει μέχρι 9 qubit, δηλαδή έχει άνυσμα καταστάσεων 512 θέσεων. Ολόκληρο το σύστημα είναι έτσι κατασκευασμένο ώστε να μπορεί εύκολα και γρήγορα να αλλάξει το πλήθος των qubit που θα υποστηρίζει προκειμένου να εξομοιώσουμε το επιθυμητό σύστημα.

Αρχικά δημιουργήσαμε μια μνήμη μόνο προς ανάγνωση για τις πύλες με το εργαλείο Xilinx Core Generator. Σε αυτή τη μνήμη θα αποθηκεύονται ο τύπος της πύλης και το κύκλωμά της. Η κάθε πύλη θα αποτελείται από το κύκλωμα το οποίο αποτελείται από 9 σύμβολα (όσα τα qubit) που χρειαζόμαστε για την αναπαράσταση του κυκλώματος. Ακόμη όμως χρειάζεται να αποθηκεύσουμε κάποιο «κωδικό» για τον τύπο της πύλης, αν δηλαδή είναι τύπου control κ.α.. Γι' αυτό το λόγω θα χρειαστούμε άλλα 2 bit. Έτσι συνολικά προκύπτει ότι η μνήμη των κυκλωμάτων θα έχει 18+2 bit = 20 bit. Η μνήμη ορίσαμε πως θα έχει βάθος ίσο με 256. Άρα το σύστημα θα μπορεί να εξομοιώνει μέχρι 256 πύλες. Για αυτό το λόγω διατηρείται σε ένα καταχωρητή 7bit η διεύθυνση στην οποία βρίσκεται η τελευταία πύλη έτσι ώστε εκεί να τερματίσει η εκτέλεση του συστήματος.

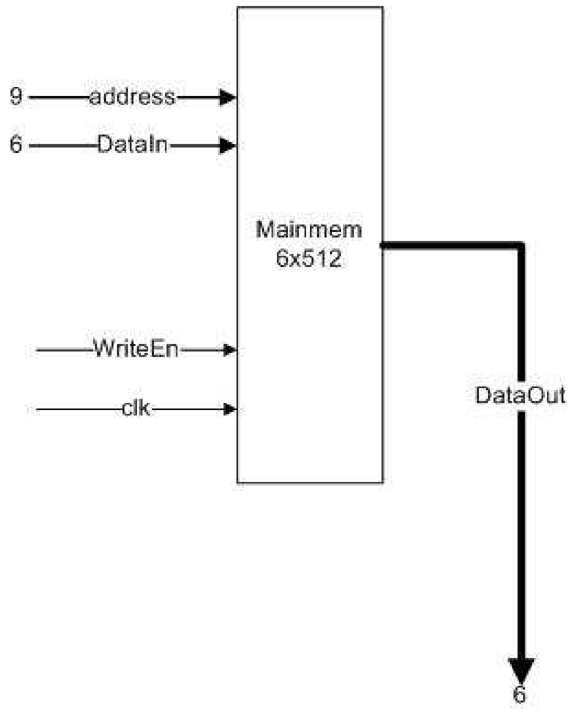
Η διεπαφή της παρουσιάζεται εδώ:



Εικόνα 5.1: *Σχηματικό διάγραμμα μνήμης τύπου πύλης και κυκλώματος*

Η κυρίως μνήμη θα αποτελείται από 512 στοιχεία πάλι, μόνο που δεν θα αποθηκεύει αυτή τη φορά την τιμή της κατάστασης. Θα αποθηκεύει κάποιον αριθμό ο οποίος θα δείχνει μια από τις διαφορετικές καταστάσεις, που ορίσαμε πως θα είναι το πολύ 64 (6bit).

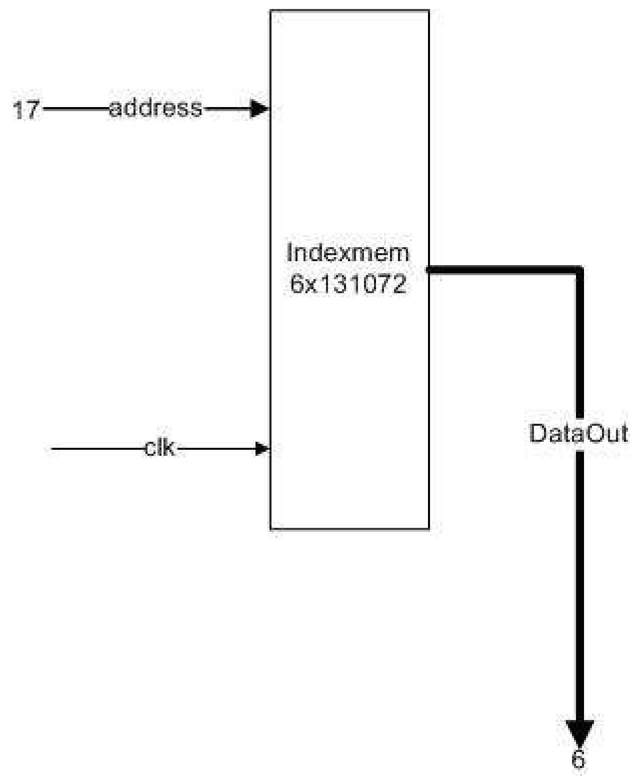
Έτσι η κυρίως μνήμη θα έχει την εξής διεπαφή:



Εικόνα 5.2: *Σχηματικό διάγραμμα κυρίως μνήμης συστήματος*

Η μνήμη στην οποία θα αποθηκεύονται τα αποτελέσματα θα πρέπει να έχει σαν έξοδο 6bit, όσα δηλαδή αποθηκεύονται και στην κυρίως μνήμη, ώστε να μπορούμε να έχουμε 64 διαφορετικές καταστάσεις. Οι καταχωρήσεις οι οποίες θα πρέπει να χωράει θα είναι τόσες όσο το πλήθος των διαφορετικών καταστάσεων (64-6bit), κυκλωμάτων (512-9bit) και των δυνατών τύπων πυλών (4-2bit). Έτσι θα επιτύχουμε σε κάθε περίπτωση να έχουμε το επιθυμητό αποτέλεσμα. Άρα το βάθος της μνήμης αναζήτησης στοιχείων θα είναι $2^{6+9+2}\theta\varepsilon\sigma\varepsilon\iota\varsigma = 131072\theta\acute{\varepsilon}\sigma\varepsilon\iota\varsigma$ και το μήκος της 6bit.

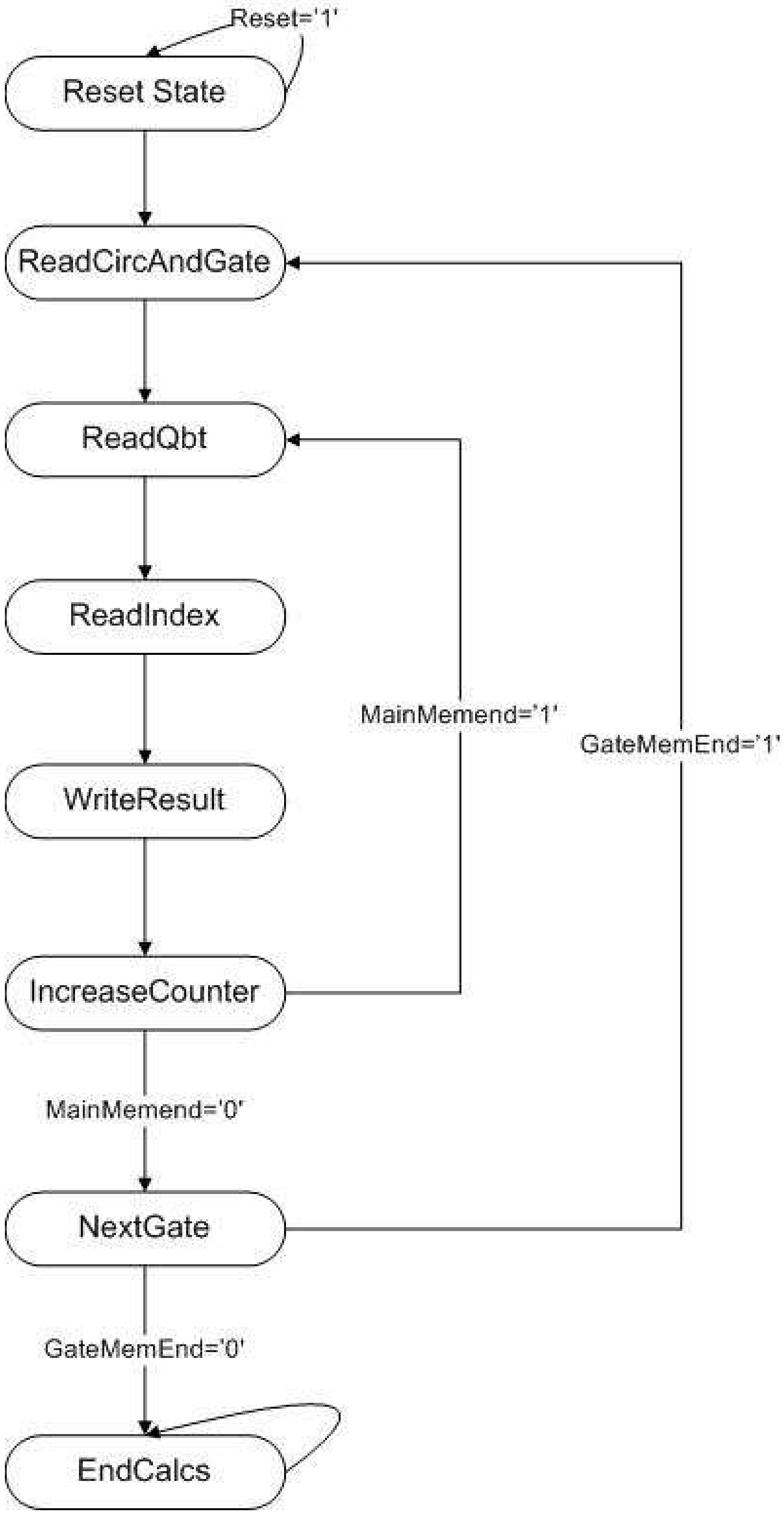
Έτσι η μνήμη αναζήτησης στοιχείων θα έχει την εξής διεπαφή:



Εικόνα 5.3: *Σχηματικό διάγραμμα μνήμης αναζήτησης στοιχείων*

Ο συντονισμός όλων αυτών απαιτεί την ύπαρξη κάποιας μονάδας ελέγχου. Έτσι αναπτύξαμε μια μονάδα ελέγχου για την λειτουργία και το συντονισμό του συστήματος των μνημών και μια μονάδα για τον έλεγχο της εκτέλεσης των πυλών στο σύστημα.

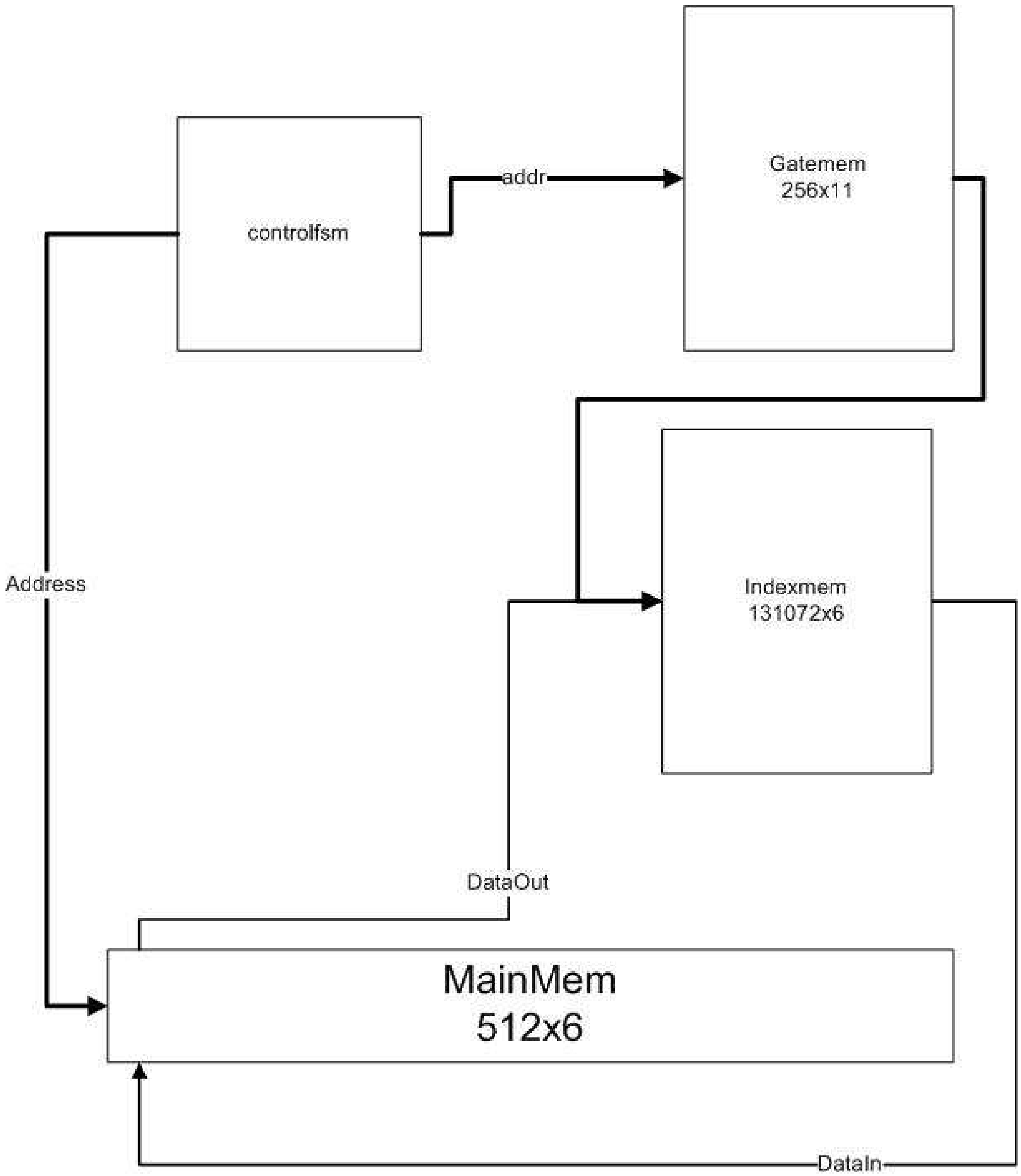
Η μονάδα ελέγχου έχει ως σκοπό τη διαχείριση όλων των μνημών και όλων των υπόλοιπων στοιχείων του συστήματος (καταχωρητές, αθροιστές κ.α.). Με τη χρήση αυτών των λειτουργικών μονάδων και μιας μονάδας λογικής εκτελεί τις απαραίτητες λειτουργίες κάθε στιγμή προκειμένου να υπολογίσει τις απαιτούμενες ποσότητες. Η μονάδα αυτή υλοποιείται με τη χρήση μιας μηχανής πεπερασμένων καταστάσεων (finite state machine-fsm) και διαχειρίζεται όλα τα παραπάνω με τον τρόπο που παρουσιάζεται και εξηγείται παρακάτω.



Εικόνα 5.4: *Σχηματικό διάγραμμα λειτουργίας μονάδας* ελέγχου

Η μονάδα εκτέλεσης, αφού το σήμα reset πάρει την τιμή «0», διαβάζει από τη μνήμη τον τύπο της πύλης και το κύκλωμα (ReadCircAndGate). Έπειτα διαβάζει την πρώτη κατάσταση από τη μνήμη (ReadQubit) και την αντίστοιχη καταχώρηση από τη μνήμη αναζήτησης στοιχείων (Read Index). Μετά προχωράει στην επόμενη θέση κατάστασης αυξάνοντας κάποιο μετρητή (IncreaseCouner) και ελέγχει αν έχει φτάσει στο τέλος της κυρίως μνήμης. Αν όχι τότε προχωράει να διαβάσει την τιμή της επόμενης κατάστασης πάλι (ReadQbt). Αν τώρα, έχει φτάσει μέχρι το τέλος της κυρίως μνήμης τότε προχωράει για να διαβάσει την επόμενη πύλη (ReadNextGate). Αν υπάρχει νέα θα μεταβεί στην κατάσταση στην οποία διαβάζει τον τύπο της νέας πύλης και το κύκλωμα της, αλλιώς θα μεταβεί στην τερματική κατάσταση (EndCalcs) αφού δεν θα υπάρχουν επιπλέον πύλες. Σε αυτή τη φάση θα έχει τελειώσει και η εξομοίωση του επεξεργαστή.

Το συνολικό σύστημα το οποίο αναπτύχθηκε παρουσιάζεται εδώ:



Εικόνα 5.4: *Σχηματικό διάγραμμα λειτουργίας μονάδας* ελέγχου

5.3 Αποτελέσματα 2ης Αρχιτεκτονικής Υλοποίησης

Το σύστημα το οποίο αναπτύχθηκε είναι σε θέση να εξομοιώνει κβαντικούς επεξεργαστές. Μπορεί να εκτελεί κβαντικά κυκλώματα ανεξαρτήτως μεγέθους, αλλάζοντας κάποιες βασικές παραμέτρους μόνο κάθε φορά (όπως μέγεθος μνημών καταχωρητών κ.α.). Όμως το σύστημα έχει όρια που τίθενται από την ίδια την ταχύτητά του. Δεν μπορεί να εξομοιώσει οποιαδήποτε πύλη μιας και δεν έχει την δυνατότητα να αποθηκεύει μεγάλο πλήθος καταστάσεων. Επίσης δεν μπορεί να επεξεργαστεί συστήματα πολλών καταστάσεων μιας και τότε η μνήμη η οποία θα είναι απαραίτητη για την αναζήτηση στοιχείων θα αυξηθεί σε μέγεθος κατά πολύ. Παρόλα αυτά το σύστημα είναι ιδιαίτερα απλό και γρήγορο κάτι που πετυχαίνει πολύ καλή απόδοση για εξομοίωση απλών κβαντικών συστημάτων με λίγες διαφορετικές καταστάσεις.

Την ορθή λειτουργία του συστήματος την επιβεβαιώσαμε με τη χρήση ενός εξομοιωτή της γλώσσας VHDL. Έτσι με τη χρήση του εργαλείου ModelSim v6.0a προσομοιώσαμε τη λειτουργία του συστήματος. Αρχικά με τη χρήση του εργαλείου Xilinx Ise v7.1 συνθέσαμε το σύστημα και πήραμε τα εξής αποτελέσματα:

```
=========================================================================
*                          Final Report                               *
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 4vfx12ff668-12

 Number of Slices:                     32  out of   5472     0%
 Number of Slice Flip Flops:           31  out of  10944     0%
 Number of 4 input LUTs:               58  out of  10944     0%
 Number of bonded IOBs:                 3  out of    320     0%
 Number of GCLKs:                       1  out of     32     3%


=========================================================================
TIMING REPORT


Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 31    |
-----------------------------------+------------------------+-------+

Timing Summary:
---------------
Speed Grade: -12

   Minimum period: 2.632ns (Maximum Frequency: 379.903MHz)
```

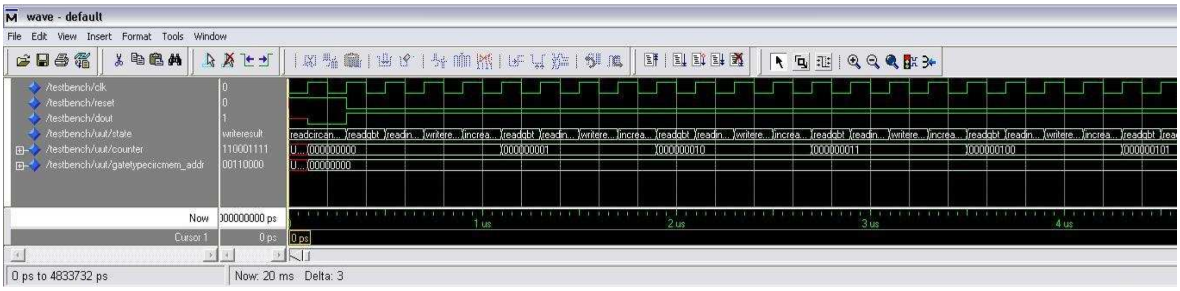
Minimum input arrival time before clock: 2.576ns
Maximum output required time after clock: 3.921ns
Maximum combinational path delay: 0.864ns

===

Εικόνα 5.5: *Αποτελέσματα σύνθεσης $2^{ης}$ αρχιτεκτονικής υλοποίησης*

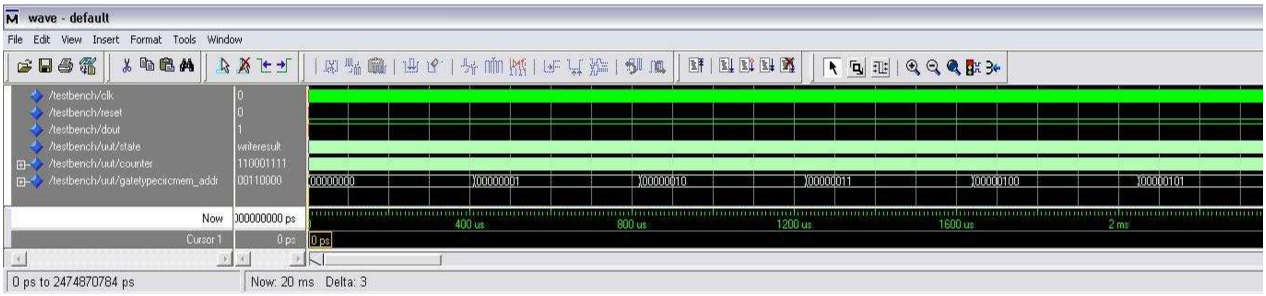
Από τα παραπάνω μπορούμε να ξεχωρίσουμε πως έχουμε καταλάβει ελάχιστο του συνολικά διαθέσιμου χώρου και πως το σύστημα θα μπορεί να τρέχει με συχνότητα 379,903 MHz (περίοδος 2,632ns).

Η προσομοίωση με τη χρήση του εργαλείου ModelSim v6.0a έγινε αφού κάναμε post place and route έτσι ώστε να δώσει ορθά αποτελέσματα όπως θα έδινε το σύστημα αν βρισκόταν μέσα σε μια FPGA και έδωσε το παρακάτω αποτέλεσμα:



Εικόνα 5.6: *Αποτελέσματα προσομοίωσης $2^{ης}$ αρχιτεκτονικής υλοποίησης*

Από εδώ μπορούμε να παρατηρήσουμε τη συμπεριφορά του συστήματος. Οι χρόνοι που φαίνονται στην παραπάνω εικόνα και χρησιμοποιούνται στην περιγραφή της είναι χρόνοι που δεν αντιπροσωπεύουν την πραγματική ταχύτητα του συστήματος. Μπορούμε να δούμε διαδοχικά τις καταστάσεις του συστήματος ελέγχου, που περιγράψαμε παραπάνω, στο σήμα calcstate να εναλλάσωνται διαδοχικά μετά την απενεργοποίηση του σήματος Reset (ReadCircAndGate, ReadQbt, ReadIndexMem, WriteReslt, IncreaseCounter, ReadQbt κ.ο.κ.). Παράλληλα παρατηρούμε τη τιμή του μετρητή counter να αυξάνει κάθε φορά που διαβάζεται μια νέα κατάσταση. Επίσης παρατηρούμε μέσω του σήματος GateTypeCircMem_addr, το οποίο έχει την τιμή 0, πως εκτελείται αυτή τη στιγμή η $1^{η}$ πύλη.



Εικόνα 5.7: *Αποτελέσματα προσομοίωσης 2ης αρχιτεκτονικής υλοποίησης*

Σε αυτό το αποτέλεσμα της προσομοίωσης του συστήματος μπορούμε να παρατηρήσουμε τη συμπεριφορά του συστήματος. Οι χρόνοι που φαίνονται στην παραπάνω εικόνα και χρησιμοποιούνται στην περιγραφή της είναι χρόνοι που δεν αντιπροσωπεύουν την πραγματική ταχύτητα του συστήματος. Παρατηρούμε πως ο αριθμός της εκτελουμένης πύλης (GateTypeCircMem_addr) αυξάνεται κάθε περίπου 400ms από την 1η πύλη μέχρι την 6η. Έτσι σε κάθε μεσοδιάστημα εκτελείται μια πύλη.

Από το παραπάνω σχήμα της προσομοίωσης παρατηρήσαμε πως για να εξομοιώσει το σύστημα μια ολόκληρη πύλη χρειάζεται 2048 κύκλους. Έτσι με περίοδο ρολογιού 2,632ns, το σύστημα είναι σε θέση να εκτελεί μια πύλη κάθε 5,360ms περίπου.

Κεφάλαιο 6

Πιστοποίηση Ορθής Λειτουργίας Συστημάτων

Κάθε σύστημα το οποίο αναπτύσσεται θα πρέπει να πιστοποιείται για την ορθή λειτουργία του. Ο λόγος για τον οποίο κάτι τέτοιο είναι απαραίτητο είναι πως μόνο έτσι μπορούμε να πειστούμε αλλά και να αποδείξουμε πως το σύστημα δουλεύει όπως θα έπρεπε. Η διαδικασία της πιστοποίησης συχνά είναι χρονοβόρα και επίπονη. Αυτό οφείλεται στο ότι κατά τη διάρκεια της πιστοποίησης μπορεί να προκύψει κάποιο ανεπιθύμητο αποτέλεσμα και να χρειαστεί να αναθεωρήσουμε ολόκληρο το σύστημα ακόμα και από την αρχή του. Και αυτό μπορεί να χρειαστεί να γίνει αρκετές φορές. Επίσης η πιστοποίηση ενός συστήματος θα πρέπει να γίνει σε βάθος και να καλύπτει όσο το δυνατόν μεγαλύτερο εύρος περιπτώσεων έτσι ώστε να είμαστε βέβαιοι για την ορθή λειτουργία του.

Σε αυτό το κεφάλαιο παρουσιάζουμε τη μελέτη που έγινε για την πιστοποίηση ορθής λειτουργίας των δυο συστημάτων τα οποία αναπτύχθηκαν και παρουσιάστηκαν στα δυο προηγούμενα κεφάλαια. Επίσης παρουσιάζονται τα αποτελέσματα τα οποία έδωσε η πιστοποίηση για κάθε μια από τις δυο αρχιτεκτονικές χωριστά.

6.1 Πιστοποίηση Ορθής Λειτουργίας 1^{ης} Αρχιτεκτονικής

Το 1^{ο} σύστημα το οποίο αναπτύχθηκε είναι αυτό το οποίο θα παρουσιάσουμε πρώτο για το πώς πιστοποιήθηκε η λειτουργία του. Οι μνήμες που χρησιμοποιηθήκαν στο σύστημα το οποίο αναπτύξαμε έχουν τη δυνατότητα αρχικοποίησης μέσω ενός αρχείου με όνομα το ίδιο της μνήμης και κατάληξη ".mif". Αρχικοποιώντας με τα επιθυμητά δεδομένα κάθε φορά ελέγχαμε τα αποτελέσματα που παίρναμε.

Αρχικά εισήγαμε διάφορα δεδομένα ένα προς ένα και ελέγχαμε με τον προσωμοιωτή αν εκτελείται ότι χρειαζόταν με τον τρόπο που θέλαμε. Αυτή η διαδικασία έγινε αρκετές φορές και ήταν ιδιαίτερα χρονοβόρα, όμως προσέφερε σημαντικές παρατηρήσεις και βελτιώσεις για το σύστημα. Με τη χρήση αυτής της διαδικασίας ελέγξαμε περιπτώσεις οι οποίες αφορούν ιδιάζοντα μέρη του συστήματος τα οποία ίσως να μην μπορούσαν να ελεγχτούν κατά τη διάρκεια της πιστοποίησης και θα μπορούσαν να προκαλέσουν προβλήματα κατά τη λειτουργία του συστήματος.

Μετά από αυτή τη διαδικασία ελέγχτηκαν κάποια κυκλώματα που είχαν χρησιμοποιηθεί κατά τη φάση της πιστοποίησης της ορθής λειτουργίας του αλγορίθμου μοντελοποίησης του συστήματος. Αυτά ήταν κυκλώματα εκτελούν κάποιες ευρέως γνωστές διεργασίες όπως πρόσθεση, κωδικοποίηση, αποκωδικοποίηση κ.α.. Για κάθε ένα από αυτά τα κυκλώματα υπάρχει σύνοψη περιγραφή, η κωδικοποίησή του σε κβαντικό κύκλωμα, ένα κβαντικό κύκλωμα και οι προδιαγραφές του [23]. Με τη χρήση αυτών των εξακριβώσαμε την ορθή λειτουργία του συστήματος κατά ένα μέρος.

Η τελική μέθοδος πιστοποίησης του συστήματος το οποίο αναπτύχθηκε έκανε χρήση του αλγόριθμου μοντελοποίησης. Που αναπτύξαμε και παρουσιάσαμε σε προηγούμενο κεφάλαιο. Έτσι παρήγαμε τυχαία κυκλώματα με η χρήση του αλγορίθμου του Scott Aaronson για "Random Clifford-Group Quantum Circuit Generator". Αυτά τα μετατρέψαμε σε μορφή κατάλληλη για χρήση από τον αλγόριθμο αλλά και για αρχικοποίηση των μνημών του συστήματος και τα εκτελέσαμε πολλές φορές με πολλές διαφορετικές τυχαίες πύλες. Μετά το πέρας κάθε εξομοίωσης του αλγορίθμου και του συστήματος μετατρέπαμε πάλι τα αποτελέσματα σε μια κοινή μορφή ώστε να μπορούν να συγκριθούν εύκολα και γρήγορα. Τέλος με τη χρήση του εργαλείου diff των Linux συγκρίναμε πολύ γρήγορα τα αποτελέσματα προκειμένου να πιστοποιήσουμε την ορθή λειτουργία του συστήματος που αναπτύξαμε.

6.2 Πιστοποίηση Ορθής Λειτουργίας 2ης Αρχιτεκτονικής

Το 2ο σύστημα το οποίο αναπτύχθηκε είναι αυτό το οποίο θα παρουσιάσουμε σε αυτή την παράγραφο για το πώς πιστοποιήθηκε η λειτουργία του. Οι μνήμες που χρησιμοποιηθήκαν και σε αυτό το σύστημα έχουν τη δυνατότητα αρχικοποίησης μέσω ενός αρχείου με όνομα το ίδιο της μνήμης και κατάληξη ".mif". Αρχικοποιώντας με τα επιθυμητά δεδομένα κάθε φορά ελέγχαμε τα αποτελέσματα που παίρναμε.

Αρχικά εισήγαμε διάφορα δεδομένα ένα προς ένα και σε αυτή την περίπτωση και ελέγχαμε με τον προσωμοιωτή αν εκτελείται ότι χρειαζόταν με τον τρόπο που θέλαμε. Αυτή η διαδικασία έγινε αρκετές φορές και ήταν ιδιαίτερα χρονοβόρα, όμως προσέφερε σημαντικές παρατηρήσεις και βελτιώσεις για το σύστημα. Με τη χρήση αυτής της διαδικασίας ελέγξαμε περιπτώσεις οι οποίες αφορούν ιδιάζοντα μέρη του συστήματος τα οποία ίσως να μην μπορούσαν να ελεγχτούν κατά τη διάρκεια της πιστοποίησης ενώ θα μπορούσαν να προκαλέσουν προβλήματα κατά τη λειτουργία του συστήματος.

Μετά από αυτή τη διαδικασία ελέγχτηκαν κάποια πολύ μικρά κυκλώματα τω οποίων τα αποτελέσματα βρήκαμε ένα προς ένα και τα συγκρίναμε με αυτά που μας έδωσε το σύστημα. Αυτά τα συστήματα επιλέχτηκαν με τέτοιο τρόπο ώστε να πληρών τις προδιαγραφές του 2ου συστήματος που αναπτύξαμε (λίγες διαφορετικές καταστάσεις και χωρίς την αναγκαιότητα υπολογισμού νέων καταστάσεων). Με αυτόν κυρίως τον τρόπο έγινε η πιστοποίηση της ορθής λειτουργίας του 2ου συστήματος που αναπτύχθηκε.

Μεγαλύτερα κυκλώματα δεν ήταν δυνατόν να ελεγχθούν μιας και πολλά από αυτά απαιτούν την ύπαρξη πολλών δυνατών καταστάσεων, κάτι που δεν επιτρέπει η συγκεκριμένη αρχιτεκτονική, καθώς και η ιδιαίτερη δουλειά που απαιτείται έξω από το σύστημα προκειμένου να εξομοιωθεί .

Κεφάλαιο 7

Αποτελέσματα Αρχιτεκτονικών Υλοποιήσεων

Η μελέτη και ο σχολιασμός των αποτελεσμάτων είναι ίσως το βασικότερο μέρος μιας έρευνας μιας και αυτό είναι το κυρίως ζητούμεν;ο της. Τα αποτελέσματα είναι αυτά τα οποία θα δείξουν τι πετυχαίνει μια εργασία και τι όχι.

Σε αυτό το κεφάλαιο παρουσιάζουμε και συγκρίνουμε τα αποτελέσματα από τις δύο αρχιτεκτονικές υλοποιήσεις μεταξύ τους αλλά και σε σύγκριση με τον αλγόριθμο μοντελοποίησης που αναπτύχθηκε. Από αυτά θα εξάγουμε κάποια χρήσιμα συμπεράσματα. Αξίζει να σημειώσουμε ακόμα πως όλοι οι χρόνοι οι οποίοι παρουσιάζονται παρακάτων και αφορούν τα συστήματα τα οποία αναπτύχθηκαν έχουν μετρηθει μετά το post place and route των συστημάτων με τη χρήση των εργαλείων.

7.1 Σύγκριση Αρχιτεκτονικών υλοποιήσεων

Αρχικά χρειάζεται να συγκρίνουμε τις δυο αρχιτεκτονικές υλοποιήσεις από τη σκοπιά των δυνατοτήτων της κάθε μιας. Και οι δυο αρχιτεκτονικές μπορούν να προκαλέσουν πρόβλημα από άποψη ακρίβειας. Αν και χρησιμοποιήθηκαν αριθμητικά στοιχεία εκτέλεσης πράξεων για αριθμούς double 64bit, μπορεί να χρειαστούμε επιπλέον ακρίβεια κάτι το οποίο θα κάνει και τα δυο συστήματα ακατάλληλα. Η πρώτη αρχιτεκτονική που αναπτύχθηκε είναι σε θέση να εξομοιώνει τους κβαντικούς επεξεργαστές με όρια που τίθενται από την πλατφόρμα ανάπτυξης. Μπορούμε δηλαδή να έχουμε συστήματα πολλών καταστάσεων (δηλαδή qubit) αυξάνοντας απλά το μέγεθος της κυρίως μνήμης ώστε να μπορεί να χωρέσει ολόκληρο το άνυσμα καταστάσεων σε αυτή. Από τα υπόλοιπα στοιχεία το μόνο που θα χρειαστεί να αλλάξει είναι κάποιοι καταχωρητές προσωρινής αποθήκευσης και κάποια σήματα ελέγχου. Έτσι, δεν υπάρχει σχεδόν κανένα όριο που τίθεται από το ίδιο το σύστημα. Το δεύτερο σύστημα μπορεί να εξομοιώσει επίσης ένα κβαντικό επεξεργαστή με την προϋπόθεση πως οι διαφορετικές καταστάσεις που θα προκύψουν θα είναι λίγες κάτι το οποίο συμβαίνει συχνά σε μικρά κυκλώματα. Επίσης θα πρέπει εκ των προτέρων να γνωρίζουμε ποιες είναι οι πιθανές καταστάσεις και πως μπορούν αυτές να προκύψουν έτσι ώστε να αρχικοποιούμε τις μνήμες με τα απαραίτητα δεδομένα. Αυτά τα δυο θέτουν τα όρια της δεύτερης αρχιτεκτονικής που αναπτύχθηκε.

Επίσης, θα πρέπει να εξετάσουμε και την απόδοση των δυο συστημάτων. Το πρώτο σύστημα έχει συχνότητα χρονισμού ρολογιού συστήματος 198,785MHz (περίοδο 5,031ms) ενώ το δεύτερο έχει 379,903MHz (περίοδο 2,632ms).

Παρατηρούμε πως το δεύτερο σύστημα έχει πολύ καλύτερη απόδοση (υψηλότερη συχνότητα χρονισμού ρολογιού συστήματος) σε σχέση με το πρώτο κάτι που είναι αναμενόμενο μιας και η πολυπλοκότητα της δεύτερης αρχιτεκτονικής υλοποίησης είναι πολύ πιο απλή και μικρή σε σχέση με αυτή της πρώτης. Αυτό επιτρέπει στα εργαλεία σύνθεσης συστήματος να τοποθετήσουν και να συνδέσουν τα επί μέρους τμήματα του συστήματος με τέτοιο τρόπο ώστε να επιτύχει μια υψηλή ταχύτητα χρονισμού. Επίσης το πρώτο σύστημα μπορεί να εκτελεί μια πύλη σε 26317 κύκλους ρολογιού , ενώ το δεύτερο χρειάζεται μόλις 2048 κύκλους ρολογιού. Κάτι το οποίο ήταν επίσης αναμενόμενο αφού το δεύτερο σύστημα δεν κάνει καμία χρήση αριθμητικών μονάδων, οι οποίες απαιτούν αρκετούς κύκλους για να δώσουν ένα αποτέλεσμα, ενώ χρειάζεται να γίνουν αρκετές προσθέσεις, αφαιρέσεις και πολλαπλασιασμοί προκειμένου να πάρουμε μια νέα κατάσταση.

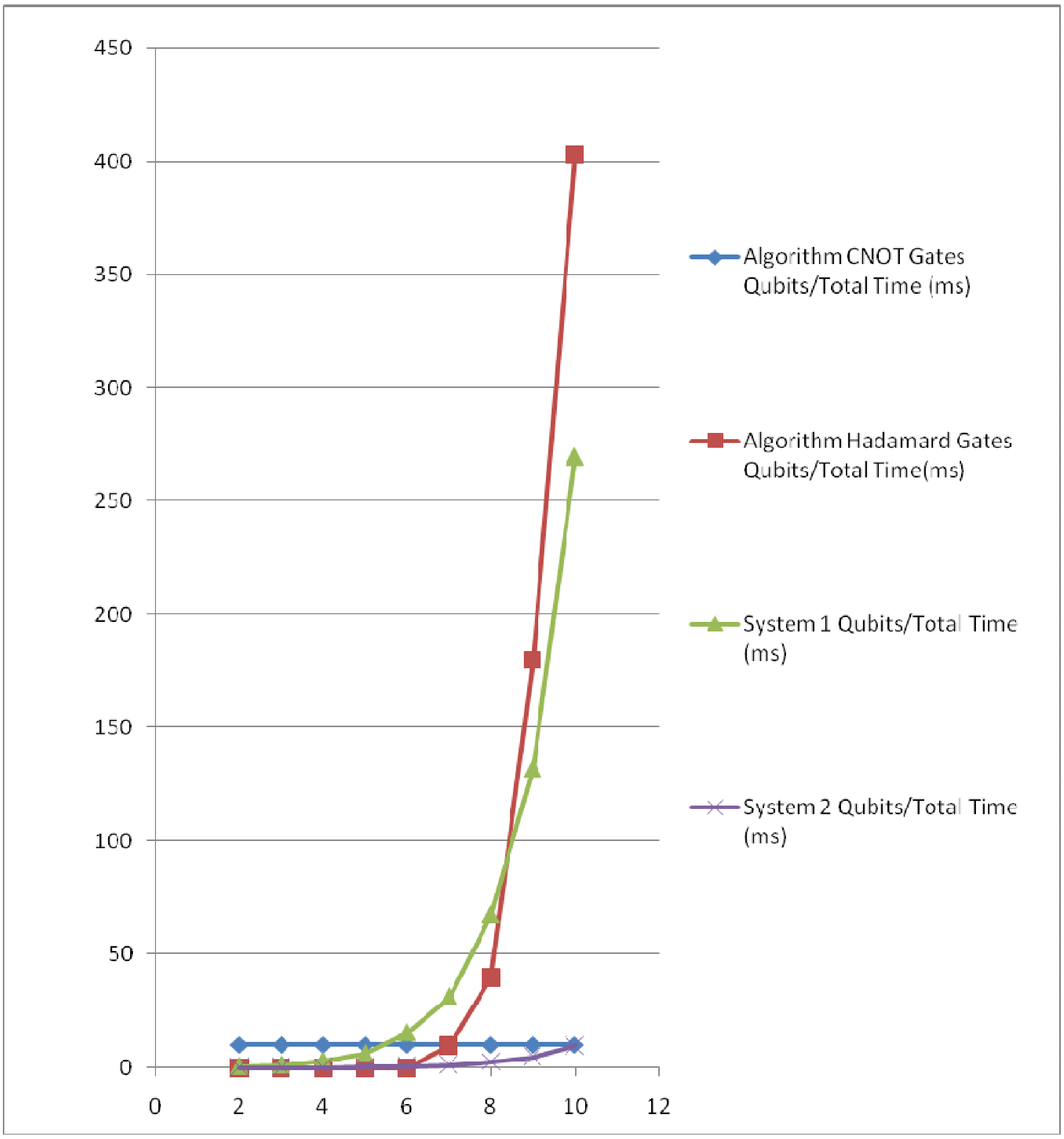
Τα δυο παραπάνω έχουν σαν αποτέλεσμα το πρώτο σύστημα να απαιτεί 132ms περίπου για την εξομοίωση μιας πύλης ενώ το δεύτερο να χρειάζεται 5,360ms περίπου.

7.2 Σύγκριση Αρχιτεκτονικών Υλοποιήσεων με Αλγόριθμο

Αφού συγκρίναμε τα δυο συστήματα που αναπτύχθηκαν μεταξύ τους, τα συγκρίναμε και με τον αλγόριθμο μοντελοποίησης του συστήματος που αναπτύχθηκε. Στο κεφάλαιο 3 παρουσιάσαμε και σχολιάσαμε τα αποτελέσματα του αλγόριθμου μοντελοποίησης αναλυτικά. Γενικά παρατηρήσαμε πως όσο αυξάνει το πλήθος των qubit στο σύστημα, ο χρόνος εκτέλεσης αυξάνει εκθετικά. Ενώ όσο αυξάνουν οι πύλες ο χρόνος αυξάνει γραμμικά. Στην περίπτωση όμως οπού οι πύλες είναι τυχαίες, ο χρόνος εκτέλεσης του προγράμματος αυξάνει κατά πολύ σε σχέση με τις άλλες περιπτώσεις. Αυτό οφείλεται στην εισαγωγή νέων καταστάσεων στο άνυσμα κατάστασης με αποτέλεσμα να μεγαλώνει η δενδρική δομή όπου αποθηκεύονται.

Στα συστήματα ισχύουν πως αυξάνοντας τα qubit αυξάνουν επίσης εκθετικά οι κύκλοι που απαιτούνται για την ολοκλήρωση μιας πύλης όπως και στον αλγόριθμο. Όμως με διαφορετικό ρυθμό αύξησης. Έτσι για ένα σύστημα με δέκα πύλες και αλλάζοντας το πλήθος των qubits κάθε φορά μετρήσαμε το συνολικό χρόνο που χρειάζεται κάθε σύστημα για να ολοκληρώσει την εκτέλεσή του.

Αυτό παρουσιάζεται στο παρακάτω διάγραμμα:

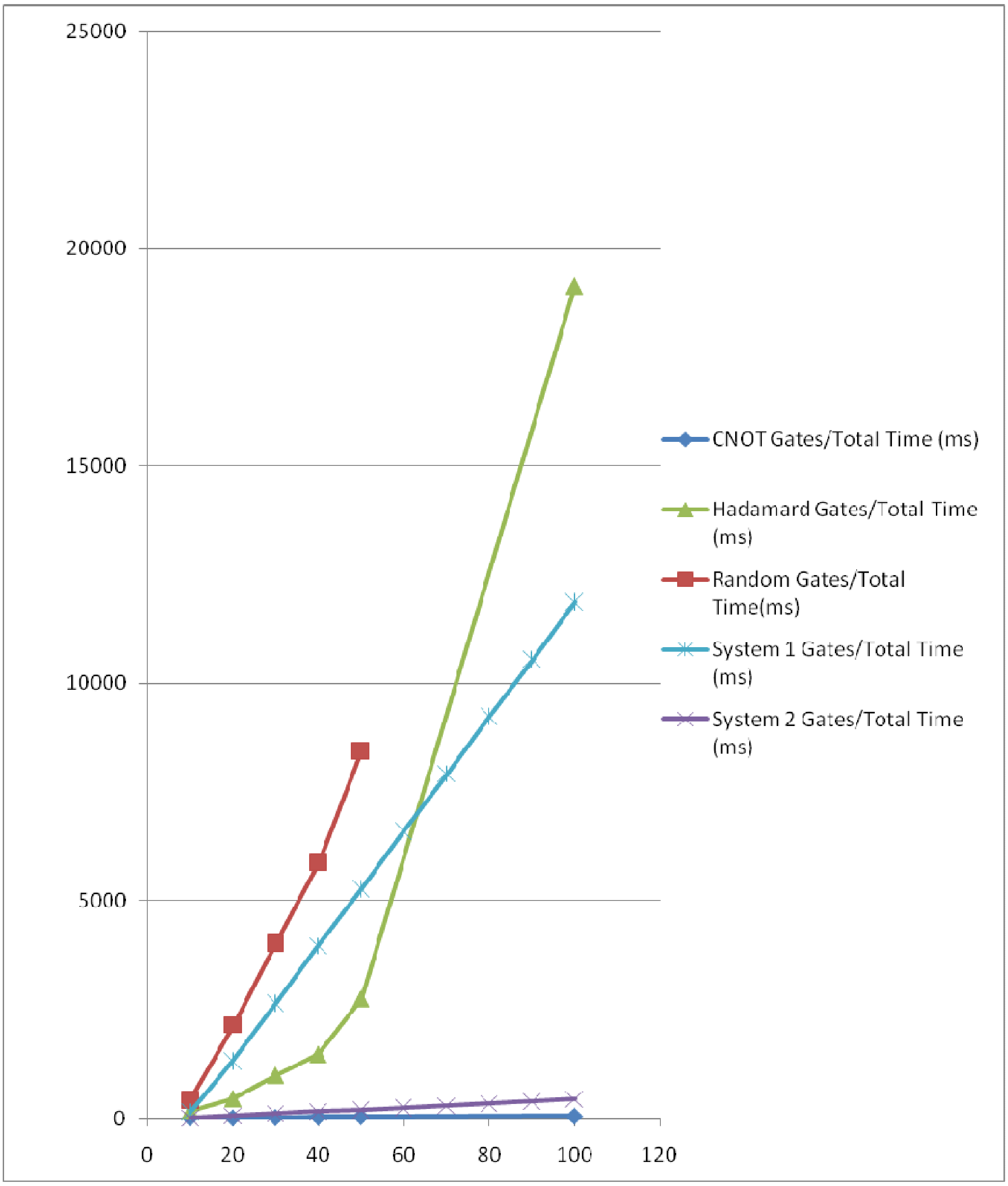


Εικόνα 7.1: Συνολικός χρόνος εκτέλεσης δέκα πυλών ως προς το πληθος των qubits για κάθε σύστημα και τον αλγόριθμο

Στο παραπάνω διάγραμμα μπορούμε να παρατηρήσουμε πως ο αλγόριθμος όταν εκτελεί αποκλειστικά πύλες CNOT θέλει ελάχιστο χρόνο ο οποίος αυξάνεται με πολύ μικρό ρυθμό. Αντίστοιχα το δεύτερο σύστημα που αναπτύχθηκε θέλει ακόμα λιγότερο χρόνο όμως αυξάνεται με ρυθμό μεγαλύτερο από αυτό των πυλών CNOT που εκτελούνται από τον αλγόριθμο. Το πρώτο σύστημα το οποίο αναπτύχθηκε παρουσιάζει πολύ μεγαλύτερο ρυθμό αύξησης του χρόνου εκτέλεσής του σε σχέση με τα δυο προηγούμενα. Αντίστοιχη αύξηση του χρόνου εκτέλεσης παρουσιάζει και ο αλγόριθμος ο οποίος εκτελεί πύλες Hadamard. Μέχρι 8 qubit ο χρόνος εκτέλεσης του αλγόριθμου για δέκα πύλες Hadamard είναι μικρότερος από αυτόν του πρώτου συστήματος. Όμως από εκεί κι έπειτα ο αλγόριθμος χρειάζεται χρόνο περισσότερο σε σχέση με το σύστημα ο οποίος αυξάνεται με πολύ μεγαλύτερο ρυθμό.

Ακόμα θελήσαμε να δούμε και να συγκρίνουμε τη συμπεριφορά του αλγόριθμου και των συστημάτων που αναπτύξαμε διατηρώντας σταθερό το πλήθος των qubit και ίσο με 9 και εκτελώντας διαφορετικό πλήθος πυλών.

Τα αποτελέσματα που πήραμε παρουσιάζονται στο παρακάτω διάγραμμα:



7.2: Συνολικός χρόνος εκτέλεσης ως προς το πληθος των πυλών για κάθε σύστημα και τον αλγόριθμο για εννέα qubit

Στο παραπάνω διάγραμμα μπορούμε να παρατηρήσουμε το δεύτερο σύστημα και ο αλγόριθμος που εκτελεί CNOT πύλες έχουν πολύ κοντινούς χρόνους εκτέλεσης με το σύστημα να χρειάζεται λίγο παραπάνω χρόνο. Οσον αφορά το δεύτερο σύστημα αυτό θέλει περισσότερο χρόνο από τον αλγόριθμο που εκτελεί Hadamard πύλες μέχρι τις 60 πύλες περίπου. Από εκεί κι έπειτα ο αλγόριθμος δαπανά πολύ περισσότερο χρόνο για την εκτέλεση των πυλών σε σχέση με τι πρώτο σύστημα. Όταν ο αλγόριθμος εκτελεί πλήρως τυχαίες πύλες χρειάζεται ακόμα και στις δέκα πύλες περισσότερο χρόνο απ' ότι το πρώτο αρχιτεκοκτονικό σύστημα.

Παρατηρώντας τα δυο παραπάνω διαγράμματα μπορούμε να συμπεράνουμε πως πως είναι προτιμότερο να χρησιμοποιούμε το πρώτο αρχιτεκτονικό σύστημα που αναπτύξαμε όταν θέλουμε να εξομοιώσουμε συστήματα με τυχαίες πύλες ή με πάνω από 60 Hadamard πύλες. Οσον αφορά το δεύτερο αρχιτεκτονικό σύστημα αυτό είναι καλό να το χρησιμοποιούμε σε συστήματα με λίγες πύλες, λίγα qubit αλλά με οποιεσδήποτε πύλες αρκεί οι διαφορετικές καταστάσεις να μπορούν να αποθηκευθούν στο σύστημα.

Κεφάλαιο 8

Συμπεράσματα και Μελλοντικές Προτάσεις

Τελειώνοντας την εργασία, είναι επιθυμητό να παρουσιαστούν κάποια γενικά συμπεράσματα και κάποιες προτάσεις για μελλοντικές επεκτάσεις ανάπτυξης των συστημάτων που υλοποιήθηκαν.

Σε αυτό το κεφάλαιο παρουσιάζουμε ξανά το τι έγινε σε αυτή την εργασία, αναφέρουμε τα αποτελέσματα που παρατηρήσαμε καθώς και τις δυσκολίες που συναντήσαμε. Τέλος αναφέρουμε κάποιες πιθανές εξελίξεις των συστημάτων που έχουν μελετηθεί και αναπτυχθεί.

8.1 Συμπεράσματα Εργασίας

Σε αυτή την εργασία μελετήθηκε και υλοποιήθηκε η δημιουργία ενός εξομοιωτή κβαντικών υπολογιστών με χρήση της σημερινής τεχνολογίας των κλασικών υπολογιστών. Αρχικά μελετήθηκε η λειτουργία των κβαντικών υπολογιστών και μοντελοποιήθηκε η λειτουργία τους με τη χρήση ενός αλγορίθμου στη γλώσσα προγραμματισμού C προκειμένου να γίνει περισσότερο αντιληπτή η λειτουργία τους. Έπειτα μελετήθηκαν και σχεδιάστηκαν δυο αρχιτεκτονικές προσεγγίσεις, με διαφορετικά χαρακτηριστικά, οι οποίες επιτρέπουν την εξομοίωση κβαντικών επεξεργαστών. Τέλος μελετήθηκαν και συγκρίθηκαν τα αποτελέσματα τα οποία προέκυψαν από κάθε μια σχεδίαση και συγκρίθηκαν για ορθότητα και αποτελεσματικότητα με βάση τον αλγόριθμο μοντελοποίησης.

Παρατηρήσαμε πως ο αλγόριθμος τρέχει ικανοποιητικά αλλά έχει όρια όπως το μέγιστο πλήθος qubit που μπορεί να έχει το κύκλωμα και το μέγιστο πλήθος πυλών που μπορεί να εξομοιώσει. Το πρώτο σύστημα που αναπτύχθηκε έχει πολύ καλή απόδοση σε σχέση με τον αλγόριθμο σε συστήματα με πολλές και τυχαίες πύλες αλλά και qubits. Είναι εύκολα μεταβαλλόμενο ώστε να μπορεί να εξομοιώσει διαφόρων τύπων κυκλώματα και έχει ελάχιστους περιορισμούς. Το δεύτερο σύστημα που αναπτύχθηκε έχει περιορισμούς στο πλήθος των διαφορετικών καταστάσεων και των qubit του συστήματος όμως έχει πάρα πολλή καλή απόδοση και καταλαμβάνει ελάχιστο χώρο.

8.2 Δυσκολίες που Συναντήθηκαν

Η κυριότερη δυσκολία που συναντήσαμε κατά τη διάρκεια αυτής της εργασίας ήταν η εξοικείωση μας με τον τρόπο λειτουργίας των κβαντικών επεξεργαστών. Η συγκεκριμένη τεχνολογία βρίσκεται ακόμα σε πρώιμο στάδιο, κι έτσι ήταν δύσκολο να βρεθεί βιβλιογραφία κατάλληλη και πηγές στον παγκόσμιο ιστό προκειμένου να μπορέσουμε να εξοικειωθούμε καλύτερα.

Ένα ακόμα μεγάλο πρόβλημα ήταν η εύρεση κατάλληλων συστημάτων για την αποσφαλμάτωση και την πιστοποίηση του αλγορίθμου και των συστημάτων που αναπτύχθηκαν. Τέτοια συστήματα υπάρχουν, όμως λίγα είναι κατάλληλα για χρήση και από τα δυο συστήματα και τον αλγόριθμο μιας και τα τρία έχουν κάποιους περιορισμούς όσον αφορά τις δυνατότητές τους.

8.3 Μελλοντικές Επεκτάσεις

Τελειώνοντας με αυτή την εργασία θα μπορούσαμε να επισημάνουμε κάποιες βελτιώσεις και επεκτάσεις των συστημάτων που θα μπορούσαν να γίνουν μελλοντικά.

Αρχικά θα μπορούσαμε να χρησιμοποιήσουμε μεγαλύτερες μνήμες στη δεύτερη αρχιτεκτονική υλοποίηση προκειμένου να μπορούμε να εξομοιώσουμε συστήματα με περισσότερες διαφορετικές καταστάσεις και περισσότερα qubits. Επίσης στην πρώτη αρχιτεκτονική υλοποίηση θα μπορούσαμε να χρησιμοποιούμε εσωτερικές μνήμες τόσο μεγάλες ώστε να χωράνε ολόκληρο το άνυσμα καταστάσεων και να μην χρειάζεται να γίνεται η ανταλλαγή δεδομένων μεταξύ των μνημών. Επίσης θα μπορούσαμε να βάλουμε μνήμες τόσες ώστε να είναι αρκετά γρήγορες και αρκετά μεγάλες για τις ανάγκες του συστήματος.

Μια βασική βελτίωση του συστήματος θα ήταν η συνένωση και των δυο συστημάτων που αναπτύχθηκαν σε ένα νέο σύστημα. Αυτό το νέο σύστημα θα έχει τα οφέλη μόνο των δύο συστημάτων αφού θα μπορεί να εκτελεί γρήγορα πύλες με γνωστό αποτέλεσμα όπως το δεύτερο σύστημα, και σε περίπτωση άγνωστης πύλης θα έχει την δυνατότητα να την υπολογίζει με τη χρήση του πρώτου συστήματος. Κάτι τέτοιο μπορεί να γίνει είτε με τη πλήρη συνύπαρξη των δυο συστημάτων στην FPGA είτε με τη χρήση δυναμικής αναδιάταξης σε περίπτωση όπου η αρχιτεκτονική είναι αρκετά μεγάλη ώστε να χωρέσει στην FPGA.

Θα μπορούσε ακόμη να υλοποιηθεί ταχύτερη αρχιτεκτονική επεξεργασίας δεδομένων. Ταχύτερες αριθμητικές μονάδες, παράλληλη επεξεργασία, διασωλήνωση επεξεργασίας και υπολογισμών είναι μερικές από τις τεχνικές που μπορούν να εφαρμοστούν και στα δυο συστήματα που αναπτύχθηκαν προκειμένου να επιτευχθεί ταχύτερη επεξεργασία δεδομένων και παραγωγή αποτελεσμάτων.

Τέλος, θα μπορούσε να χρησιμοποιηθεί κάποια μεγαλύτερη ή και ταχύτερη FPGA προκειμένου να επιτύχουμε υψηλότερη συχνότητα χρονισμού ρολογιού συστήματος και έτσι να μπορούμε να μειώσουμε το χρόνο εκτέλεσης ενός κβαντικού συστήματος.

Βιβλιογραφία

[1] Jumpei Niwa, Keiji Matsumoto and Hiroshi Imai. General-Purpose parallel simulator for quantum computing. The American Physical Society A 66 062317. Decemeber 2002.

[2] G. Viamontes, M. Rajagopalan, I. Markov and J. Hayes. Gate-Level Simulation of Quantum Circuits. Proceedings of the 2003 conference on Asia South Pacific design automation. Kitakyushu, Japan. March 2007.

[3] G. Viamontes, I. Markov and J. Hayes. Improving Gate-Level Simulation of Quantum Circuits. Quantum Information Processing, vol. 2(5), pp. 347-380. October 2003.

[4] N. Fujishima, K. Saito, M. Onouchi and H. Hoh. High-speed processor for quantum-computing emulation and its applications. Proceedings of the 2003 International Symposium on Circuits and Systems. 2003.

[5] A. Gothandaraman, G. Lee Warren, Gregory D. Peterson and Robert J. Harrison. Hardware Acceleration of a Quantum Monte Carlo Application. Proceedings of the Third Annual Reconfigurable Systems Summer Institute. 2007.

[6] Michael T. Niemier, Arun F. Rodrigues and Peter M. Kogge. A Potentially Implementable FPGA for Quantum Dot Cellular Automata. 1st Workshop on Non-Silicon Computation. Boston, MS. 2002.

[7] Himanshu Thapliyal, Hamid R. Arabnia and A.P Vinod. Combined Integer and Floating Point Multiplication Architecture(CIFM) for FPGAs and Its Reversible Logic Implementation. Symposium on Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest. August 2006

[8] Ahmed Usman Khalid, Zeljko Zilic and Katarzyna Radecka. FPGA Emulation of Quantum Circuits. Proceedings of the IEEE International Conference on Computer Design (ICCD'04) - Volume 00. 2004.

[9] ”Libquantum”, Online Quantum Library Documentation, http://www.enyo.de/libquantum/

[10] Owen Callanan, Andy Nisbet, Emre Özer, James Sexton and David Gregg. FPGA Implementation of a Lattice Quantum Chromodynamics Algorithm Using Logarithmic Arithmetic. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3 – Volume 04 Page: 146.2. 2005.

[11] Carl J. Williams, Xiao Tang, Mikko Hiekkero, Julie Rouzaud, Richang Lu, Andreas Goedecke, Alan Migdall, Alan Mink, Anastase Nakassis, Leticia Pibida, Jesse Wen, Edward Hagley, and Charles W. Clark. High Speed Quantum Communication Testbed. Proceedings- Spie The International society for optical engineering, ISSU 4821, pages 421-426. 2002.

[12] Neil Gershenfeld and Isaac L. Chuang Quantum Computing with Molecules through ther WWW site http://www.media.mit.edu/physics/publications/papers /98.06.sciam/0698gershenfeld.html. Article in Scientific America.

[13] DiVincenzo, David. Quantum Computation. Through ther WWW site http://www.sciencemag.org/cgi/content/abstract/270/5234/255. Article in Science Magazine. October 1995.

[14] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms. Society of Industrial and Applied Mathematics. SICOMP Volume 26 Issue 5Pages 1484-1509. 2005.

[15] Jozef Gruska. Quantum Computing. McGraw-Hill Publishing Company, London. 1999.

[16] Lov K. Grover. A fast quantum mechanical algorithm for database search. Proceedings of the twenty-eighth annual ACM symposium on Theory of computing Philadelphia, Pennsylvania, United States Pages: 212 – 219. 1996.

[17] Dirac P. The Principles of Quantum Mechanics. Oxford University Press. 1948.

[18] Giulano Beneti, Giulio Casati, Giuliano Strini. Principles of Quantum Computation and Information Volume I. World Scientific. 2004.

[19] Reversible Logic Synthesis Benchmarks Page. http://webhome.cs.uvic.ca/~dmaslov/. 2007.

[20] Vasilis Samoladas. Algebraic Structures for the Manipulation of Quantum State Representations. October 2006.

[21] CHP: CNOT-Hadamard-Phase. http://www.scottaaronson.com/chp/.

[22] University of Victoria. http://www.uvic.ca/.

[23] Reversible Logic Synthesis Benchmarks Page. http://webhome.cs.uvic.ca/~dmaslov/rd32.html. October 2007.