DOUBLE DATA RATE (DDR) DRAM CONTROLLER

Ευστάθιος Μπούρας

Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών

Επιτροπή Καθηγητών :

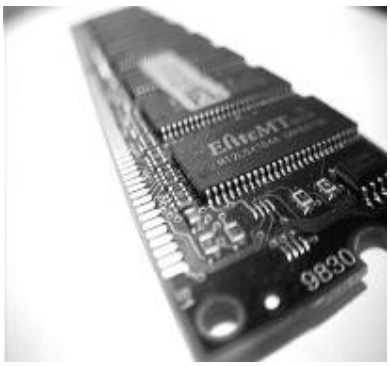
Δ. Πνευματικάτος (Επιβλέπων)
Ι. Παπαευσταθίου
Κ. Καλαϊτζάκης

Χανιά 2006

Περιεχόμενα

Περίληψη

Οι DDR DRAM μνήμες αποτελούν μία από τις πιο σύγχρονες κατηγορίες μνημών. Έχουν τη δυνατότητα να συνδυάζουν ταχύτητα, μέγεθος και αξιοπιστία για αυτό και είναι και οι πλέον διαδεδομένες στα υπολογιστικά συστήματα. Η σύνθετη λειτουργία και η πολύπλοκη λογική πάνω στην οποία στηρίχθηκαν αυτές οι μνήμες ήταν αναπόφευκτη προκειμένου να επιτευχθεί ο συνδυασμός ταχύτητας και μεγέθους (μεταφορά δεδομένων με διπλάσιο ρυθμό από τις προηγούμενες μνήμες) . Στόχος αυτής της διπλωματικής εργασίας ήταν η υλοποίηση ενός Ελεγκτή για τη διαχείριση μίας τέτοιας μνήμης. Ο Ελεγκτής θα πρέπει να αξιοποιεί τις δυνατότητές της μνήμης προκειμένου να επιτύχει μεγάλη απόδοση, χρησιμοποιώντας ένα απλό Interface για απλοποίηση του συστήματος και ευκολότερη ενσωμάτωσή του σε τυχόν μελλοντικά συστήματα. Για τον έλεγχο της σωστής λειτουργίας του συστήματος Ελεγκτή – Μνήμης υλοποιήθηκε μία μονάδα Testing η οποία με τον απλούστερο δυνατό τρόπο παρέχει τη δυνατότητα σε ένα χρήστη (μέσα από βασικές εντολές) να διαχειριστεί και να ελέγξει το σύστημα και την ορθή λειτουργία του.



1. Εισαγωγή

Καθώς, στις μέρες μας, η τεχνολογία εξελίσσεται με ταχύτατους ρυθμούς σε όλους τους τομείς είναι λογικό πρώτα απ' όλα να εξελίσσεται και στον τομέα της πληροφορικής και των υπολογιστών. Αυξάνοντας τις δυνατότητες των υπολογιστικών συστημάτων αυξάνεται και η δυνατότητα εξέλιξης και σε όλους τους υπόλοιπους τομείς. Αυτό γίνεται διότι το βασικότερο ίσως εργαλείο στα χέρια του κάθε μηχανικού είναι ένας Ηλεκτρονικός Υπολογιστής. Εκτός όμως από εργαλείο, ο Ηλεκτρονικός Υπολογιστής αποτελεί για πολλούς και ένα μέσω διασκέδασης, έρευνας, μόρφωσης, ακόμα και επικοινωνίας!

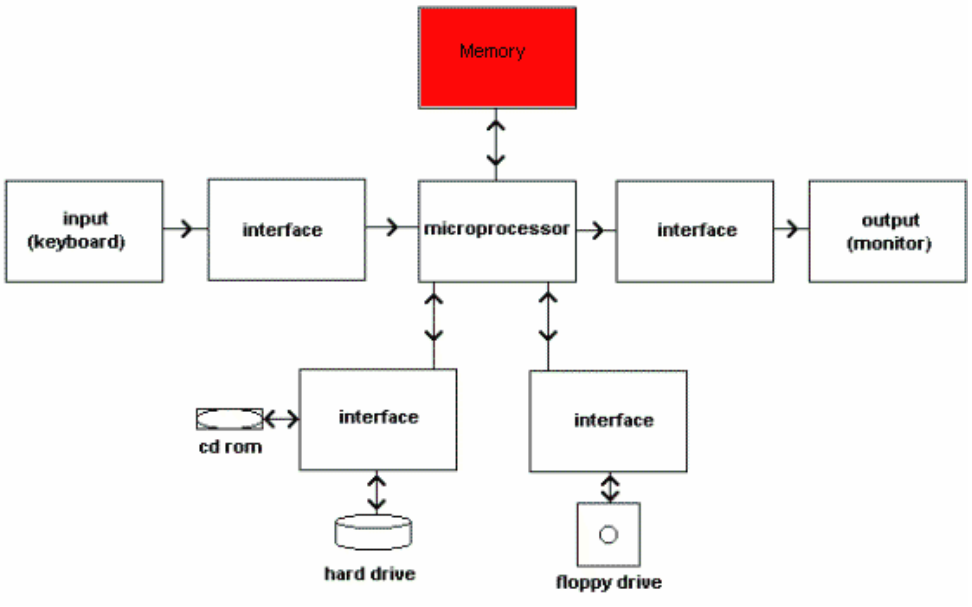
Πλέον, σχεδόν σε κάθε σπίτι, θεωρείται απαραίτητη και αναγκαία η ύπαρξη ενός τουλάχιστον Ηλεκτρονικού Υπολογιστή. Η παράλληλη εξέλιξη της τεχνολογίας σε συνδυασμό με τις ανάγκες του κόσμου οδηγεί σε περισσότερες και μεγαλύτερες απαιτήσεις από τα υπολογιστικά συστήματα (είτε για λόγους, όπως προαναφέρθηκε, εργασίας είτε για λόγους διασκέδασης). Οι απαιτήσεις των χρηστών εστιάζουν βασικά σε τρεις σημαντικούς τομείς. Τον τομέα της ταχύτητας (ο κόσμος απαιτεί συνεχώς ταχύτερους Ηλεκτρονικούς Υπολογιστές), τον τομέα του μεγέθους (αφού καθώς εξελίσσεται η τεχνολογία όλα μικραίνουν) και τέλος τον πάντοτε επίκαιρο οικονομικό τομέα!

Σε αυτή τη διπλωματική εργασία θα ασχοληθούμε με τον πρώτο τομέα που αφορά την ταχύτητα. Οι δυνατότητες υπολογιστικής ταχύτητας ενός Ηλεκτρονικού Υπολογιστή αυξάνονται με πολύ γρήγορους ρυθμούς κυρίως παρεμβαίνοντας στα επιμέρους εξαρτήματα ενός Υπολογιστή όπως είναι ο Επεξεργαστής, οι Motherboards, ο Σκληρός Δίσκος, οι Μνήμες κ.τ.λ. (**Σχήμα** 1.1). Οι μνήμες (RAM) είναι το αντικείμενο με το οποίο θα ασχοληθεί αυτή η εργασία και πιο συγκεκριμένα με την υλοποίηση ενός Ελεγκτή που θα διαχειρίζεται ικανοποιητικά μία DDR DRAM.

Σκοπός του Ελεγκτή αυτού θα είναι η σωστή διαχείριση και λειτουργία της μνήμης με ικανοποιητική απόδοση. Ο Ελεγκτής θα διαχειρίζεται τη μνήμη καλύπτοντας τους περιορισμούς της και αξιοποιώντας κάποιες από τις δυνατότητες της. Αποτέλεσμα των παραπάνω είναι η υλοποίηση ενός συστήματος το οποίο λειτουργεί με αρκετά καλή απόδοση και πάνω από όλα απλοποιεί σημαντικά την αρκετά σύνθετη λειτουργία της μνήμης για την πιο εύκολη και κατανοητή χρήση του συστήματος από ένα πιθανό χρήστη.

Για να γίνει εύκολα κατανοητή η λειτουργία ενός τέτοιου Ελεγκτή κρίνεται απαραίτητη αρχικά η, στοιχειώδης έστω, κατανόηση της λειτουργίας μίας μνήμης γενικότερα αλλά και πιο συγκεκριμένα μίας DDR DRAM. Για αυτό το λόγο πριν ξεκινήσει η περιγραφή της λειτουργίας και υλοποίησης του Ελεγκτή παρουσιάζονται κάποια βασικά στοιχεία λειτουργίας των μνημών και πάνω απ' όλα, το τι είναι μία μνήμη RAM.

Σχήμα 1.1 Basic PC Block Diagram



1.1 **Τι Είναι Μία Μνήμη** RAM

Η λέξη RAM αποτελεί συντομογραφία του Random Access Memory, δηλαδή Μνήμη Τυχαίας Προσπέλασης. Ένας απλοϊκός τρόπος για να κατανοήσουμε την λειτουργία της μνήμης RAM είναι να παρομοιάσουμε τον ανθρώπινο εγκέφαλο με έναν Ηλεκτρονικό Υπολογιστή. Ο άνθρωπος έχει μία προσωρινή μνήμη και την κανονική μνήμη. Π.χ. Αν ο άνθρωπος προσπαθήσει να λύσει ένα μαθηματικό πρόβλημα, τότε η προσωρινή μνήμη είναι αυτή που κάνει τους υπολογισμούς ανακαλώντας όμως στοιχεία που την βοηθάνε (π.χ. την προπέδια) από την μόνιμη μνήμη στην οποία αυτά έχουν κάποια στιγμή αποθηκευτεί / καταχωρηθεί. Αφού το πρόβλημα επιλυθεί, η προσωρινή μνήμη ''αδειάζει / ξεχνάει'' όσα έκανε (έτσι ώστε να είναι έτοιμη για χρήση σε κάποια άλλη στιγμή που θα χρειασθεί) αλλά ο ανθρώπινος εγκέφαλος εξακολουθεί να κατέχει την πληροφορία για το πώς επιλύθηκε το πρόβλημα αφού η διαδικασία έχει αποθηκευτεί στην μόνιμη μνήμη του! Σε αυτή την παρομοίωση, το μαθηματικό πρόβλημα αποτελεί ένα πρόγραμμα / διεργασία που τρέχει σε έναν υπολογιστή, η μόνιμη μνήμη το σκληρό δίσκο ενός υπολογιστή ενώ η προσωρινή μνήμη τη μνήμη RAM ενός υπολογιστή.

Από τεχνολογικής άποψης μία μνήμη RAM αποτελεί μία ηλεκτρομαγνητική συσκευή προσωρινής αποθήκευσης δεδομένων που όμως χάνονται όταν σταματήσει η τροφοδοσία του Ηλεκτρονικού Υπολογιστή.

Η μνήμη χρησιμοποιείται είτε από το λειτουργικό σύστημα του υπολογιστή, είτε κατά την εκτέλεση προγραμμάτων / διεργασιών, είτε ακόμα και κατά τη διάρκεια ενός παιχνιδιού για την αποθήκευση δεδομένων τα οποία όταν κληθούν ξανά από την αντίστοιχη διεργασία θα μπορούν να επιστραφούν το ταχύτερο δυνατό για την καλύτερη, από άποψη ταχύτητας, απόδοση του συστήματος.

1.2 Γιατί Random Access Memory

Σε μία μνήμη RAM η πρόσβαση είτε για εγγραφή είτε για ανάγνωση μπορεί να πραγματοποιηθεί σε οποιαδήποτε στήλη οποιασδήποτε σειράς και οποιουδήποτε Bank χωρίς να χρειασθεί η πρόσβαση να είναι σειριακή. Το αντίθετο μίας RAM θα ήταν η SAM (Serial Access Memory). Σε μία τέτοια μνήμη η πρόσβαση πρέπει να αρχίσει από την αρχή της μνήμης και να συνεχίσει σειριακά μέχρι να φθάσει στη διεύθυνση που είναι προσωρινά αποθηκευμένα τα επιθυμητά δεδομένα (παράδειγμα αποθηκευτικών μονάδων που λειτουργούν με αυτό τον τρόπο είναι οι ταινίες). Όπως είναι φανερό ο χρόνος που κερδίζει ένα υπολογιστικό σύστημα χρησιμοποιώντας μία RAM είναι αρκετά σημαντικός.

1.3 Πώς Λειτουργεί Μία RAM

Όταν ένας χρήστης π.χ. θέλει να τρέξει στον υπολογιστή του ένα πρόγραμμα, τότε το λειτουργικό σύστημα αντιγράφει το πρόγραμμα από το σκληρό δίσκο στη μνήμη όπου και αυτό τρέχει. Ο υπολογιστής χρησιμοποιεί την προσωρινή μνήμη για την εκτέλεση μίας τέτοιας διεργασίας ακριβώς επειδή η RAM είναι πολύ πιο γρήγορη από το σκληρό δίσκο. Αυτός είναι και ο λόγος που συνήθως όσο μεγαλύτερη η μνήμη RAM σε έναν υπολογιστή, τόσο καλύτερη και η απόδοσή του (πάντοτε από άποψη ταχύτητας). Διότι όσο μεγαλύτερη η συνολική RAM του υπολογιστή τόσο μικρότερος ο αριθμός των προσβάσεων του επεξεργαστή στο σκληρό δίσκο για την ανάκτηση δεδομένων. Αυτό οδηγεί σε ένα σαφώς πιο γρήγορο σύστημα αφού η ''μικρή'' RAM είναι πολλές φορές πιο γρήγορη από το ''μεγάλο'' σκληρό δίσκο!

Τα περισσότερα προγράμματα βέβαια είναι αρκετά μεγάλα ώστε να χωρέσουν με τη μία σε μία RAM. Έτσι συνήθως φορτώνονται στη μνήμη ''κομμάτια'' του προγράμματος με τη σειρά που χρειάζονται για την εκτέλεσή του και αδειάζουν από τη μνήμη όταν πλέον δεν χρειάζονται κρατώντας μόνο κάποια δεδομένα που χρειάζονται για τη σωστή ολοκλήρωση της εκτέλεσης του προγράμματος. Για αυτό το λόγο και τα μεγαλύτερα προγράμματα αργούν περισσότερο στην εκτέλεσή τους, διότι συνεχίζουν να απαιτούν πρόσβαση στον ''πιο αργό'' σκληρό δίσκο! Από την άλλη, προγράμματα που μπορούν να ''χωρέσουν'' με μία πρόσβαση στη μνήμη δεν χρειάζονται για την εκτέλεσή τους προσβάσεις στο σκληρό δίσκο οπότε και εκτελούνται αρκετά πιο γρήγορα.

Ακόμη, μία προσωρινή μνήμη κρατάει δεδομένα από διεργασίες οι οποίες εκτελούνται τη συγκεκριμένη στιγμή έως ότου ‘’σωθούν’’ στο σκληρό δίσκο από το χρήστη με μία εντολή ή απενεργοποιηθεί ο υπολογιστής οπότε και διαγράφονται (Ένα καλό παράδειγμα αυτής της περίπτωσης είναι και οι κειμενογράφοι).

1.4 **Μετρώντας Την Ταχύτητα Μίας** RAM

Όπως στην περίπτωση των επεξεργαστών, έτσι και στις μνήμες η ταχύτητα μετράται σε μονάδες συχνότητας (MHz). Όσο μεγαλύτερη η συχνότητα στην οποία λειτουργεί μία μνήμη, τόσο γρηγορότερη αυτή είναι. (Στην περίπτωση DDR μνημών υπολογίζεται η ‘’ενεργή’’ συχνότητα, δηλαδή αυτή που παράγει αποτέλεσμα. Έτσι αν μία DDR λειτουργεί με συχνότητα 133 MHz διαφημίζεται / δηλώνεται ως 266 MHz).

Βέβαια στις μνήμες παρουσιάζεται ακόμη ένας σημαντικός παράγοντας που μετράει την ταχύτητα και την απόδοση της μνήμης και αυτός είναι το CAS Latency (Column Address Strobe) η οποία θα αναλυθεί αργότερα στην **Παράγραφο** 2.4.3.1.

1.5 **Σκοπός Της Εργασίας**

Όπως έχει προαναφερθεί, σκοπός αυτής της διπλωματικής εργασίας είναι η κατασκευή ενός Ελεγκτή (Controller) μίας DDR (Double Data Rate) DRAM. Σκοπός αυτού του Ελεγκτή θα είναι φυσικά η σωστή διαχείριση της μνήμης έτσι ώστε να τηρούνται οι περιορισμοί της μνήμης και να βελτιστοποιείται στο μέγιστο δυνατό η απόδοσή της. Όπως γίνεται φανερό και από την ονομασία αυτής της μνήμης η βασική διαφορά της από τις απλές SRAM είναι η δυνατότητα μεταφοράς δύο δεδομένων στη διάρκεια ενός κύκλου ρολογιού (Double Data Rate).

Για την κατασκευή του Ελεγκτή έπρεπε φυσικά να ληφθούν υπόψη όλοι οι περιορισμοί της μνήμης και τα τεχνικά χαρακτηριστικά της. Έτσι λοιπόν αρχικά πραγματοποιήθηκε μελέτη ενός μοντέλου μνήμης της εταιρείας ***Micron*** (Πρόκειται για μία DDR SDRAM 256 Mb (MT46V64M4))και πάνω σε αυτό το μοντέλο κατασκευάστηκε και ο Ελεγκτής.

Μία μνήμη σαν αυτή που χρησιμοποιήθηκε έχει αρκετά πολύπλοκο και δυσνόητο τρόπο λειτουργίας (ειδικά για έναν χρήστη που μπορεί να μην ενδιαφέρεται καν για το πώς λειτουργεί αυτό που ονομάζεται μνήμη). Ένα από τα πιο πολύπλοκα και δύσκολα από άποψη υλοποίησης σημεία είναι ο χρονισμός της μνήμης.

Ο συγχρονισμός όλων των σημάτων της για τη σωστή της λειτουργία και συνεπώς για την ορθή εκτέλεση των εντολών. Με βάση τους περιορισμούς της μνήμης και του χρονισμού της και αξιοποιώντας κάποιες από τις δυνατότητες αυτής, υλοποιήθηκε ο Ελεγκτής.

Βασικός αρχικά σκοπός του Ελεγκτή ήταν η σωστή επικοινωνία του με τη μνήμη. Για να επιτευχθεί αυτό χρειάσθηκε να κατανοηθεί καλά ο χρονισμός της μνήμης που είναι ίσως και το πιο σημαντικό σημείο που πρέπει να ληφθεί υπόψιν. Εφόσον αυτό πραγματοποιήθηκε επόμενος στόχος ήταν η βελτίωση της απόδοσης του συστήματος. Για το λόγο αυτό μελετήθηκαν οι δυνατότητες της μνήμης και αξιοποιήθηκαν οι σημαντικότερες από αυτές έτσι ώστε η απόδοση / ταχύτητα του Ελεγκτή να είναι αρκετά καλή. Με βάση τα παραπάνω σε συνδυασμό με την απλότητα του συστήματος, για την ευκολότερη κατανόηση από το χρήστη, προέκυψε ένα σύστημα που ικανοποιούσε αρκετά καλά το συνδυασμό ταχύτητας - απλότητας που ήταν στόχος της εργασίας.

Ακόμη, στην εργασία αυτή υλοποιήθηκε μία μονάδα μέσα από την οποία ο χρήστης διαχειρίζεται (αποστέλλοντας τις εντολές που επιθυμεί να εκτελεστούν) τον Ελεγκτή και συνεπώς και τις εντολές που θα αποσταλούν στη μνήμη. Αποτέλεσμα αυτού είναι η ύπαρξη μίας διεπαφής ανάμεσα στον χρήστη (την μονάδα που αυτός εισάγει τις προς εκτέλεση εντολές) και το υπόλοιπο σύστημα που αποτελείται από τον Ελεγκτή και τη Μνήμη.

Για τους λόγους που έχουν ήδη αναφερθεί η διεπαφή αυτή πρέπει να είναι όσο πιο απλή γίνεται έτσι ώστε να μπορεί ο χρήστης να καταλάβει εύκολα την επικοινωνία ανάμεσα στις δύο μονάδες εφόσον δεν τον απασχολούν τα σύνθετα μέρη του συστήματος. Συνεπώς, σκοπός και στόχος της μονάδας αυτής είναι να μην χρειασθεί καθόλου ο χρήστης του συστήματος να εισέλθει στη λογική της υλοποίησης του Ελεγκτή και της μνήμης εφόσον δεν το επιθυμεί καθώς επίσης και να μπορεί να εισάγει με ευκολία τις εντολές που επιθυμεί στο σύστημα. Απλοποιείται με αυτό τον τρόπο η διαχείριση του συστήματος καθώς επίσης καθίσταται και ευκολότερος ο Έλεγχος (σε επίπεδο Simulation) της σωστής λειτουργίας του συστήματος στο σύνολό του.

2. Περί Μνημών

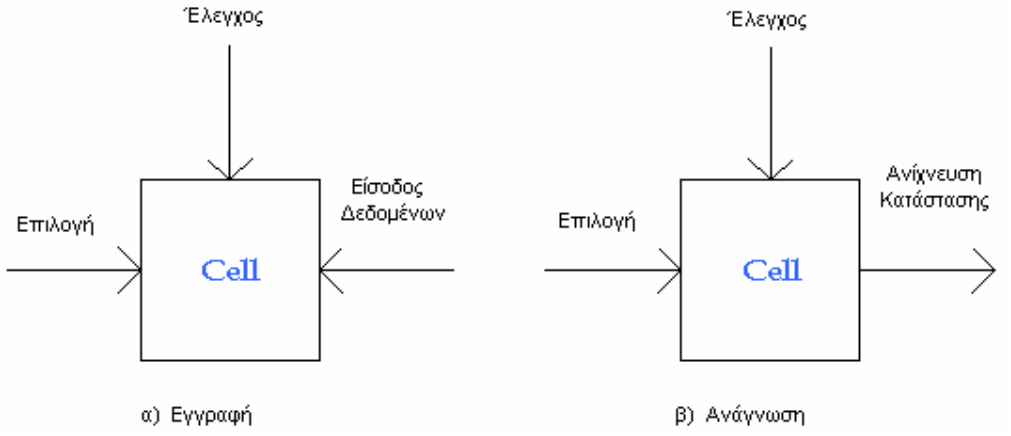
2.1 Εισαγωγικά

Το βασικό στοιχείο μίας μνήμης είναι το ***κύτταρο μνήμης***. Παρόλο που χρησιμοποιούν αρκετές τεχνολογίες ηλεκτρονικής, όλα τα κύτταρα μνήμης με ημιαγωγούς έχουν κάποιες κοινές ιδιότητες :

- Εμφανίζουν δύο σταθερές (ή ημισταθερές) καταστάσεις, οι οποίες μπορούν να χρησιμοποιηθούν για την αναπαράσταση των δύο δυαδικών ψηφίων 1 και 0.
- Υπάρχει η δυνατότητα να υποστούν εγγραφή (έστω για μία μόνο φορά), ώστε να καθοριστεί η λογική τους κατάσταση.
- Υπάρχει η δυνατότητα να ‘’διαβαστούν’’ για να δούμε σε ποια λογική κατάσταση βρίσκονται

Στο **Σχήμα** 2.1 απεικονίζεται η λειτουργία ενός κυττάρου μνήμης. Συνηθέστερα, το κύτταρο έχει τρεις λειτουργικούς ακροδέκτες που έχουν την ικανότητα να φέρουν ένα ηλεκτρικό σήμα. Ο ακροδέκτης επιλογής επιλέγει το κύτταρο το οποίο θα προσπελασθεί για ανάγνωση ή εγγραφή. Ο ακροδέκτης ελέγχου υποδεικνύει αν πρόκειται για εντολή ανάγνωσης ή εγγραφής. Στην περίπτωση εγγραφής, ο τρίτος ακροδέκτης παρέχει ένα ηλεκτρικό σήμα το οποίο θέτει την τιμή του κυττάρου σε 0 ή 1. Σε περίπτωση ανάγνωσης, ο ακροδέκτης αυτός χρησιμοποιείται για την εξαγωγή αυτής της κατάστασης του κυττάρου.

Σχήμα 2.1



Ένα άλλο βασικό χαρακτηριστικό μίας μνήμης RAM είναι ότι είναι ''***πτητική***''. Αυτό σημαίνει ότι μία τέτοια μνήμη πρέπει να τροφοδοτείται συνεχώς με ηλεκτρική ισχύ. Αν αυτή διακοπεί χάνονται και τα δεδομένα που είναι αποθηκευμένα στη μνήμη.

2.2 SRAM Vs DRAM

2.2.1 DRAM

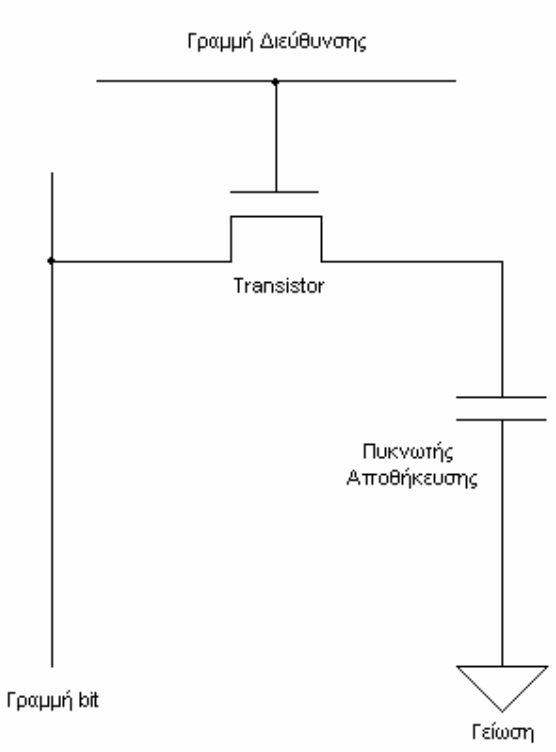
Η τεχνολογία RAM υποδιαιρείται σε δύο τεχνολογίες : τη Δυναμική και τη Στατική. Μία δυναμική RAM (DRAM) κατασκευάζεται με κύτταρα μνήμης τα οποία αποθηκεύουν τα δεδομένα ως φορτίο σε πυκνωτές. Η παρουσία ή η απουσία φορτίου σε ένα πυκνωτή ερμηνεύεται ως ένα δυαδικό ψηφίο 0 ή 1. Επειδή οι πυκνωτές έχουν μία φυσική τάση να εκφορτίζονται, οι δυναμικές RAM απαιτούν την περιοδική ανανέωση του φορτίου για να διατηρηθούν άθικτα τα δεδομένα (Το κύτταρο μίας DRAM κατασκευάζεται από μικρούς σε χωρητικότητα πυκνωτές οι οποίοι με πολύ αργό ρυθμό χάνουν ενέργεια. Αν δεν πραγματοποιηθεί την κατάλληλη στιγμή ένα Refresh τότε ένας ή και περισσότεροι πυκνωτές είναι δυνατόν να χάσουνε αρκετή ενέργεια έτσι ώστε να μην είναι δυνατόν να διακριθεί αν πρόκειται για ένα λογικό 1 ή 0 οπότε και μπορεί να παρουσιαστεί αλλοίωση των αποθηκευμένων τιμών). Ο όρος δυναμική αναφέρεται σε αυτή ακριβώς την τάση του αποθηκευμένου φορτίου να διαρρέει, ακόμη και με μόνιμη παροχή ισχύος. Το **Σχήμα** 2.2 παρουσιάζει την τυπική δομή ενός Κυττάρου μίας DRAM.

Η γραμμή διεύθυνσης ενεργοποιείται όταν πρόκειται να αναγνωσθεί ή να εγγραφεί η τιμή του bit από το κύτταρο αυτό. Το τρανζίστορ ενεργεί ως ένας διακόπτης ο οποίος είναι κλειστός (επιτρέποντας τη ροή του ρεύματος) αν εφαρμοσθεί μία τάση στη γραμμή διεύθυνσης και ανοικτός (δεν ρέει ρεύμα) αν δεν υπάρχει τάση στη γραμμή μεταφοράς.

Για τη λειτουργία εγγραφής, εφαρμόζεται μία τάση σήματος στη Γραμμή bit. Μία υψηλή τάση αναπαριστά το δυαδικό ψηφίο 1, ενώ μία χαμηλή τάση το δυαδικό ψηφίο 0. Κατόπιν εφαρμόζεται ένα σήμα στη γραμμή διεύθυνσης, επιτρέποντας τη μεταφορά ενός φορτίου στον πυκνωτή.

Για τη λειτουργία ανάγνωσης, όταν επιλεγεί η γραμμή διεύθυνσης, το τρανζίστορ γίνεται αγώγιμο και το φορτίο που ήταν αποθηκευμένο στον πυκνωτή τροφοδοτείται σε μία Γραμμή bit και σε έναν ενισχυτή. Ο ενισχυτής συγκρίνει την τάση του πυκνωτή με μία τιμή τάσης αναφοράς και αποφασίζει αν το κύτταρο περιέχει ένα λογικό 1 ή ένα λογικό 0. Η ανάγνωση αυτή εκφορτίζει τον πυκνωτή ο οποίος πρέπει να επαναφορτισθεί για να ολοκληρωθεί η διαδικασία.

Σχήμα 2.2



Παρόλο που το κύτταρο μίας DRAM χρησιμοποιείται για την αποθήκευση ενός μόνο bit (0 ή 1) είναι στην ουσία μία αναλογική συσκευή. Ο πυκνωτής μπορεί να αποθηκεύσει οποιαδήποτε τιμή φορτίου εντός μίας περιοχής τιμών. Έτσι αν το φορτίο θα ερμηνευθεί ως λογικό 1 ή 0 καθορίζεται από μία τιμή κατωφλίου.

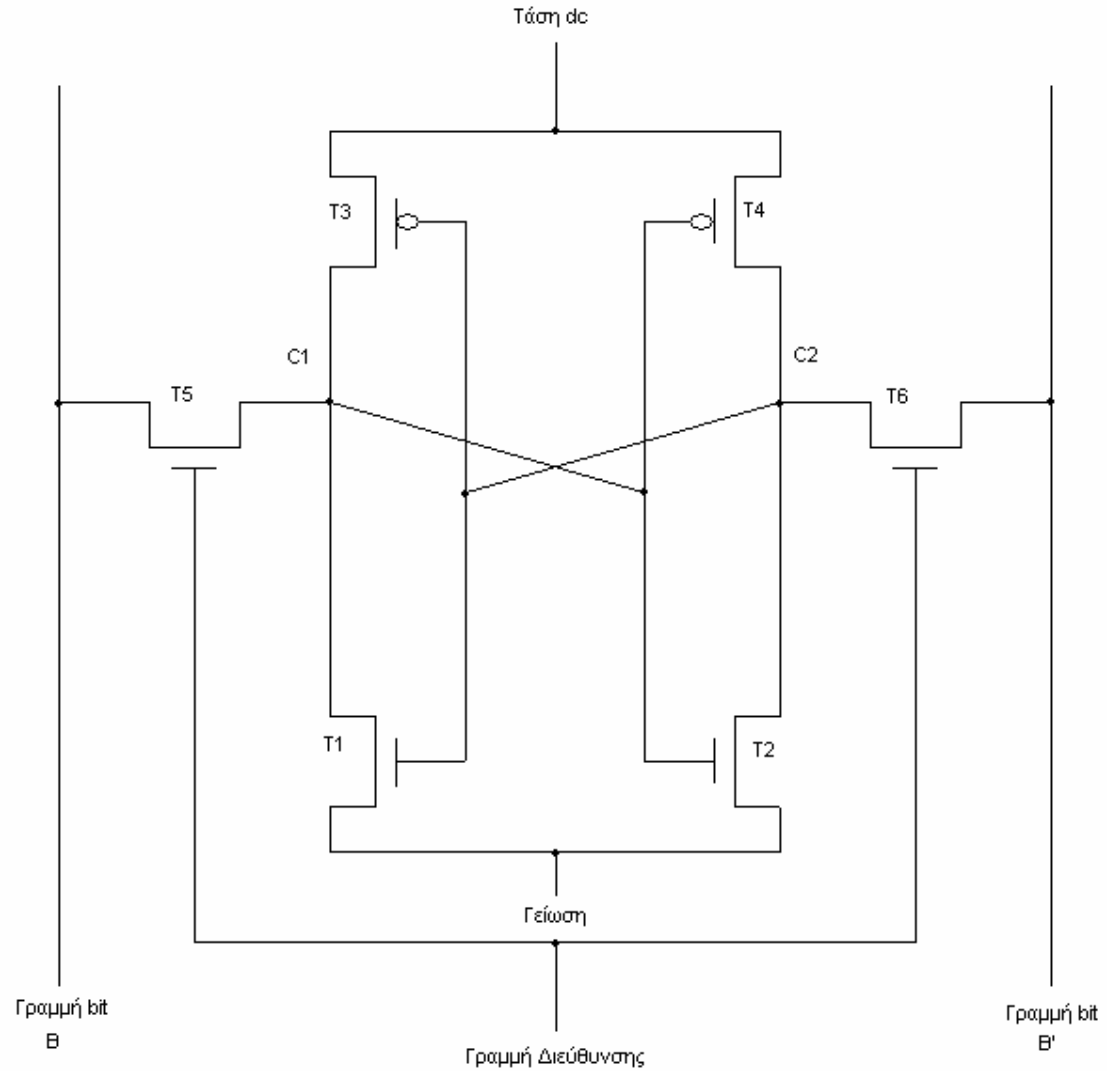
2.2.2 SRAM

Αντίθετα με την DRAM, μία στατική RAM (SRAM) είναι μία ψηφιακή συσκευή η οποία χρησιμοποιεί τα ίδια λογικά στοιχεία με εκείνα που χρησιμοποιούνται στον επεξεργαστή. Σε μία SRAM οι δυαδικές τιμές αποθηκεύονται χρησιμοποιώντας παραδοσιακές διατάξεις λογικής πύλης και Flip-Flop. Μία στατική RAM θα διατηρεί τα δεδομένα της όσο τροφοδοτείται με ηλεκτρική ισχύ. Το **Σχήμα** 2.3 παρουσιάζει την τυπική δομή ενός Κυττάρου μίας SRAM.

Σε ένα τέτοιο κύτταρο όπως φαίνεται και από το σχήμα, τέσσερα τρανζίστορ (T1, T2, T3 και T4) συνδέονται σταυρωτά σε μία διάταξη η οποία παράγει μία σταθερή λογική κατάσταση.

Στη λογική κατάσταση 1, το σημείο C1 έχει υψηλή τάση ενώ το C2 χαμηλή. Στην κατάσταση αυτή τα T1 και T4 είναι μη αγώγιμα ενώ τα T2 και T3 είναι αγώγιμα (οι κύκλοι στα T3 και T4 υποδεικνύον αναστροφή του σήματος). Αντίθετα στη λογική κατάσταση 0, το σημείο C1 έχει χαμηλή τάση ενώ το C2 υψηλή. Στην κατάσταση αυτή τα T1 και T4 είναι αγώγιμα ενώ τα T2 και T3 είναι μη αγώγιμα. Και οι δύο καταστάσεις είναι σταθερές όσο εφαρμόζεται η τάση τροφοδοσίας. Αντίθετα με τη DRAM δεν απαιτείται ανανέωση για τη διατήρηση των δεδομένων.

Σχήμα 2.3



Όπως και στην περίπτωση των DRAM, η γραμμή διεύθυνσης χρησιμοποιείται για να ανοίξει ή να κλείσει ένα διακόπτη. Η γραμμή διεύθυνσης ελέγχει δύο τρανζίστορ (T5 και T6).

Όταν εφαρμόζεται ένα σήμα στη γραμμή αυτή, τα δύο τρανζίστορ γίνονται αγώγιμα και έτσι επιτρέπεται η εκτέλεση μίας ανάγνωσης ή εγγραφής. Για την εκτέλεση εγγραφής η επιθυμητή τιμή του bit εφαρμόζεται στη γραμμή B, ενώ το συμπλήρωμα της εφαρμόζεται στην γραμμή B'. Αυτό αναγκάζει τα τέσσερα τρανζίστορ (T1, T2, T3 και T4) να πάρουν την κατάλληλη λογική κατάσταση. Σε περίπτωση λειτουργίας για μία εντολή ανάγνωσης, η τιμή του bit διαβάζεται από τη γραμμή B.

2.2.3 **Σύγκριση**

Τόσο οι στατικές όσο και οι δυναμικές RAM είναι πτητικές. Αυτό σημαίνει ότι πρέπει να τροφοδοτούνται συνεχώς με ηλεκτρική ισχύ για να διατηρούν άθικτες τις τιμές των bit.

Από τεχνολογικής άποψης, όπως φαίνεται και από τα παραπάνω σχήματα, ένα κύτταρο δυναμικής μνήμης είναι απλούστερο και μικρότερο σε μέγεθος από ένα κύτταρο στατικής μνήμης. Έτσι η DRAM έχει μεγαλύτερη πυκνότητα αποθηκευμένης πληροφορίας (αφού : Μικρότερα κύτταρα = περισσότερα κύτταρα ανά μονάδα επιφάνειας) και είναι πιο σημαντικά πιο φθηνή από μία αντίστοιχη SRAM. Από την άλλη πλευρά, μία DRAM απαιτεί την ύπαρξη των κυκλωμάτων υποστήριξης για την ανανέωση (Refresh) των δεδομένων (Αν ένα Refresh δεν πραγματοποιηθεί τη στιγμή που απαιτείται τότε τα δεδομένα που υποτίθεται είναι γραμμένα σε μία τέτοια μνήμη χάνονται. Αυτό διότι σε κάθε Refresh της μνήμης, γίνεται ανάγνωση των αποθηκευμένων σε αυτή δεδομένων και επανεγγραφή τους).

Για τις μεγαλύτερες μνήμες, το σταθερό κόστος των κυκλωμάτων ανανέωσης αντισταθμίζεται με το παραπάνω από το μικρότερο μεταβλητό κόστος των κυττάρων DRAM. Για τους παραπάνω λόγους μία SRAM θεωρείται αρκετά πιο γρήγορη και αξιόπιστη από μία DRAM. Έτσι η DRAM τείνει να προτιμάται για απαιτήσεις μεγάλου μεγέθους μνήμης.

Ένα τελευταίο θέμα είναι ότι οι SRAM είναι γενικά λίγο πιο γρήγορες από τις DRAM. Λόγω αυτών των σχετικών χαρακτηριστικών, η SRAM χρησιμοποιείται για μνήμες Cache (τόσο επί του Chip όσο και εκτός αυτού), ενώ η DRAM χρησιμοποιείται για την κύρια μνήμη.

2.3 **Βασικές Κατηγορίες Μνημών** DRAM

Με την πάροδο του χρόνου και την εξέλιξη της τεχνολογίας ήταν ταχύτατη και η εξέλιξη των μνημών. Τα πρώτα υπολογιστικά συστήματα είχανε SIMM (Single Inline Memory) μνήμες ενώ στη συνέχεια ακολούθησαν όπως ήταν το επόμενο λογικό βήμα οι μνήμες DIMM (Double Inline Memory). Με την εμφάνιση αυτών των μνημών προσδόθηκαν καινούριες δυνατότητες από άποψη ταχύτητας και μεγέθους στις μνήμες προσδίδοντας έτσι μεγαλύτερες δυνατότητες και στους ίδιους τους υπολογιστές. Βέβαια η ακόμη μεγαλύτερη εξέλιξη οδήγησε στις DDR RAM (Double Data Rate) και πλέον τις QDR RAM (Quad Data Rate). Ας γνωρίσουμε όμως μερικές από αυτές τις κατηγορίες μνημών.

2.3.1 Fast Page DRAM (FPM)

Σε αυτή την περίπτωση μνημών μία σειρά της μνήμης μπορεί να παραμείνει ‘’ανοιχτή’’ έτσι ώστε αναγνώσεις και εγγραφές που αφορούν τη συγκεκριμένη γραμμή να μην χρειασθεί να περιμένουν για κάποιο Precharge και εκ νέου πρόσβαση στη συγκεκριμένη γραμμή. Αυτό όπως είναι λογικό αυξάνει την απόδοση του συστήματος σε περίπτωση εγγραφών ή αναγνώσεων με μεγάλο Burst.

2.3.2 Extended Data Out DRAM (EDO)

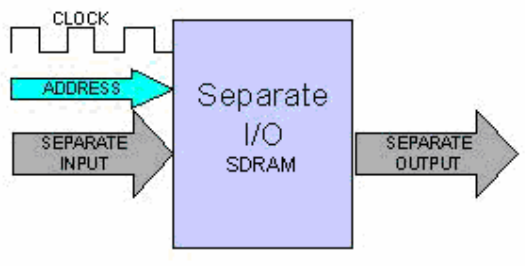
Αυτή η κατηγορία μνημών είναι παρόμοια με την FPM με ένα πρόσθετο χαρακτηριστικό. Μπορεί να εκκινήσει η διαδικασία μίας καινούριας πρόσβαση στη μνήμη μπορεί ενώ διατηρούνται ενεργά τα δεδομένα εξόδου από την προηγούμενη πρόσβαση στη μνήμη. Αυτό πραγματοποιείται μέσω pipelining και επιτρέπει την επικάλυψη ορισμένων διεργασιών στη μνήμη με αποτέλεσμα το κέρδος χρόνου και επομένως και απόδοσης (5% κέρδος σε σχέση με μία FPM).

2.3.3 Synchronous DRAM (SDRAM)

2.3.3.1 Single Data Rate SDRAM (SDR)

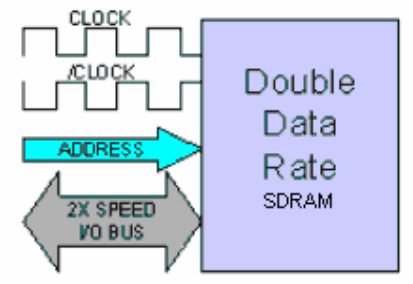
Συνήθως αποκαλείται απλά SDRAM αφού όταν εισήχθηκε ο όρος Synchronous DRAM ήταν η μοναδική περίπτωση τέτοιας μνήμης (αργότερα η εξέλιξη αυτής οδήγησε στην DDR). Τη στιγμή που όλες οι υπόλοιπες μνήμες είχανε ασύγχρονο Interface, πράγμα που σημαίνει ότι αντιδρούσανε απ’ ευθείας σε αλλαγές / μεταβολές των σημάτων εισόδου, η ανακάλυψη της SDRAM οδήγησε σε σύγχρονο Interface. Σε αυτή την περίπτωση η μνήμη περιμένει για ένα παλμό ρολογιού πριν αντιδράσει / αποκριθεί σε μία μεταβολή των σημάτων εισόδου. Με αυτό τον τρόπο συγχρονίζεται με το bus (δίαυλο) του συστήματος και συνεπώς και με τον επεξεργαστή. Το ρολόι χρησιμοποιείται για να οδηγήσει μία εσωτερική FSM (Finite State Machine) η οποία με τη σειρά της οδηγεί σε μία

Pipelined αρχιτεκτονική τις εισερχόμενες εντολές. Αυτό σημαίνει ότι μία τέτοια μνήμη είναι δυνατόν να αποδεχθεί μία καινούρια εντολή πριν ακόμη ολοκληρωθεί η εκτέλεση της προηγούμενης. Έτσι π.χ. μία εντολή εγγραφής μπορεί να ακολουθηθεί από μία άλλη εντολή χωρίς να χρειάζεται η μνήμη να περιμένει την ολοκλήρωση της εγγραφής των δεδομένων, της πρώτης εντολής, στη μνήμη. Από την άλλη σε μία εντολή ανάγνωσης, τα δεδομένα επιστρέφονται από τη μνήμη έπειτα από έναν αριθμό κύκλων του ρολογιού. Σε μία τέτοια μνήμη όμως, η επόμενη εντολή είναι δυνατόν να εισαχθεί στη μνήμη πριν επιστραφούν τα δεδομένα της πρώτης αίτησης ανάγνωσης. Οι SDRAM του εμπορίου κατατάσσονται / αξιολογούνται με βάση τη συχνότητα σε MHz και οι τρεις επίσημες κατηγορίες είναι οι PC66, PC100 και PC133 όπου οι αριθμοί αντιστοιχούν σε MHz.



2.3.3.2 Double Data Rate SDRAM (DDR)

Η λειτουργία μίας τέτοιας μνήμης είναι παρεμφερής με αυτή της απλής SDRAM. Στην περίπτωση όμως μίας DDR το Bandwidth είναι σημαντικά μεγαλύτερο από αυτό της SDRAM αφού μπορούν να μεταφερθούν δεδομένα τόσο στη θετική όσο και στην αρνητική ακμή του ρολογιού. Αυτό ουσιαστικά σχεδόν διπλασιάζει το ρυθμό μεταφοράς χωρίς όμως να χρειασθεί να αυξηθεί η συχνότητα λειτουργίας. Έτσι μία DDR μνήμη που λειτουργεί στα 133 MHz ''ισοδυναμεί'' (αν μας επιτραπεί να γίνει μία τέτοια σύγκριση) με μία SDRAM η οποία λειτουργεί στα 266 MHz. (**Βλέπε** 2.2.6)

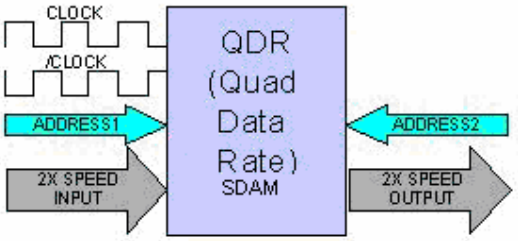


2.3.3.3 Double Data Rate 2 SDRAM(DDR II)

Η DDR II αποτελεί την εξέλιξη της μνήμης DDR. Το πλεονέκτημα αυτής της κατηγορίας μνημών έγκειται στην ικανότητα υποστήριξης πολύ πιο γρήγορου ρολογιού λόγω σχεδιαστικών βελτιστοποιήσεων. Για παράδειγμα, μία SDRAM μεταφέρει δεδομένα σε κάθε θετική ακμή του ρολογιού επιτυγχάνοντας έτσι ένα ρυθμό μεταφοράς δεδομένων στα 100 MHz. Από την άλλη τόσο η DDR όσο και η DDR II είναι ικανές να μεταφέρουν δεδομένα, τόσο στην αρνητική όσο και στη θετική ακμή του ρολογιού (0.0 V και 2.5 V για την DDR και 0.0 V και 1.8 V για την DDR II) επιτυγχάνοντας λειτουργία που φαινομενικά παρουσιάζει τη μνήμη σαν να λειτουργεί στη διπλάσια συχνότητα (200 MHz) χωρίς όμως αυτό να ισχύει. Στην περίπτωση όμως μίας DDR II μνήμης η συχνότητα του ρολογιού ενισχύεται επιπλέον μέσω, όπως προαναφέρθηκαν, σχεδιαστικών / κατασκευαστικών βελτιστοποιήσεων, με χρήση Buffer για την αποθήκευση των εισερχόμενων εντολών και εξωτερικών Drivers. Παρ' όλα αυτά υπεισέρχεται σημαντική καθυστέρηση στη μονάδα αποθήκευσης των εντολών (Buffer). Οι κατασκευαστικές βελτιστοποιήσεις της μνήμης οδηγούν σε εξοικονόμηση ενέργειας (η απαιτούμενη τάση τροφοδοσίας πέφτει από τα 2.5 στα 1.8 V). Επίσης η μικρότερη συχνότητα του ρολογιού μπορεί να οδηγήσει στο ίδιο Bandwidth με μία SDRAM ακόμη και αν η DDR II λειτουργεί στη μισή συχνότητα.

2.3.3.4 Quad Data Rate SDRAM(QDR)

Μία QDR μνήμη αποτελεί την εξέλιξη της DDR. Τη στιγμή που μία DDR μπορεί να πραγματοποιήσει τη μεταφορά 2 δεδομένων στην διάρκεια ενός κύκλου του ρολογιού, η QDR μπορεί να πραγματοποιήσει τη μεταφορά τεσσάρων. Όπως και στην περίπτωση των DDR έτσι και σε αυτές τις μνήμες η μεταφορά των δεδομένων μπορεί να πραγματοποιηθεί τόσο στη θετική όσο και στην αρνητική ακμή του ρολογιού. Αντίθετα όμως από την DDR η οποία έχει μία αμφίδρομη γραμμή μεταφοράς δεδομένων και συνεπώς θύρα για ανάγνωση και εγγραφή, η QDR έχει δύο ξεχωριστές γραμμές / θύρες για την ανάγνωση και την εγγραφή οι οποίες όμως μπορούν να λειτουργήσουν ταυτόχρονα. Αυτό βέβαια απαιτεί μεγαλύτερο αριθμό καλωδίων ανάμεσα στη μνήμη και τον Controller της, θεωρητικά όμως διπλασιάζει το μέγιστο δυνατό ρυθμό μεταφοράς των δεδομένων. Η QDR χρησιμοποιεί δύο διαφορετικά ρολόγια για εγγραφή και ανάγνωση.



2.3.4 Rambus DRAM (RDRAM)

Τα βασικά χαρακτηριστικά που διαφοροποιούν μία RDRAM από την απλή DRAM είναι τρία :

1. Η αρχιτεκτονική (από κατασκευαστικής άποψης) μίας τέτοιας μνήμης είναι τέτοια ώστε να λειτουργεί με όσο το δυνατόν λιγότερους διαύλους / γραμμές μεταφοράς γίνεται σε σχέση με μία κοινή DRAM

2. Στην αρχιτεκτονική μίας RDRAM κάθε δίαυλος μεταφοράς (bus) είναι σχεδιασμένος έτσι ώστε να είναι ικανός να μεταφέρει τρεις διαφορετικούς τύπους πληροφορίας ο κάθε ένας από τους οποίους είναι απαραίτητος για τη σωστή λειτουργία της μνήμης. Έτσι κάθε δίαυλος είναι ικανός να μεταφέρει είτε **α)** Δεδομένα, **β)** Πληροφορίες διευθυνσιοδότησης (η οποία καθορίζει την τοποθεσία / διεύθυνση (Bank, σειρά και στήλη) μέσα στην μνήμη στην οποία βρίσκονται (Ανάγνωση), ή πρέπει να τοποθετηθούν (Εγγραφή) τα επιθυμητά δεδομένα **γ)** Πληροφορίες από τον Controller της μνήμης καθορίζουν κάθε φορά αν πρόκειται για ανάγνωση από τη μνήμη ή εγγραφή σε αυτή. Αντίθετα στην περίπτωση των κοινών DRAM κάθε δίαυλος μεταφοράς είναι ικανός να μεταφέρει ένα μόνο από τους παραπάνω τύπους πληροφορίας από και προς τη μνήμη.

3. Στην περίπτωση των DRAM τα δεδομένα, οι διευθύνσεις και οι υπόλοιπες πληροφορίες αποστέλλονται ξεχωριστά από τον Controller προς τη μνήμη ενώ στην περίπτωση της RDRAM αποστέλλονται όλα μαζί ομαδοποιημένα με τη μορφή ‘’πακέτου’’.

2.3.5 Buffered DRAM

Αυτή η κατηγορία μνημών περιλαμβάνει τις μνήμες εκείνες που έχουν ενσωματωμένους buffer πάνω στα module τους. Σε αυτή την περίπτωση οι buffer που βρίσκονται πάνω στο module της μνήμης καθορίζουν τον τρόπο / σειρά με την οποία θα πραγματοποιηθεί η πρόσβαση στη μνήμη. Τα buffer chips που βρίσκονται πάνω στο module της μνήμης είναι συνήθως πιο μικρά από τα βασικά DRAM chip της μνήμης. Ακόμη, τα buffer chips επαναδρομολογούνε τα καλώδια (signals) ανάμεσα στα chip της μνήμης με τέτοια αρχιτεκτονική έτσι ώστε να επιτρέπει στο module της μνήμης να κατασκευαστεί με περισσότερα chip οπότε και μεγαλύτερη σε χωρητικότητα. Οι Unbuffered μνήμες χρησιμοποιούν συνήθως (για τον ίδιο λόγο) εξωτερικούς buffer που βρίσκονται πάνω στη motherboard.

2.4 Λεπτομερή Χαρακτηριστικά Μίας DDR SDRAM

2.4.1 Βασικά Χαρακτηριστικά

- V_{DD} = +2.5 V ± 0.2 V, V_{DDQ} = +2.5 V ± 0.2 V
- Αμφίδρομο σήμα DQS (Data Strobe) το οποίο αποστέλλεται από και προς τη μνήμη μαζί με τα δεδομένα.
- Εσωτερική (της μνήμης) Pipelined αρχιτεκτονική έτσι ώστε να παρέχεται η δυνατότητα πρόσβασης δύο δεδομένων σε ένα κύκλο ρολογιού.
- Δύο ρολόγια CK και CK# ή CK_n.
- Οι εντολές εισάγονται σε κάθε θετική ακμή του CK.
- Το σήμα DQS σε περίπτωση εγγραφής συγχρονίζεται απόλυτα με τα δεδομένα όπως αυτά αποστέλλονται από τη μνήμη, ενώ στην περίπτωση εγγραφής πρέπει η αλλαγή της τιμής του σήματος να γίνει στο κέντρο των δεδομένων (Περεταίρω ανάλυση παρακάτω).
- DLL για την ευθυγράμμιση των DQ και DQS με το CK.
- 4 εσωτερικά Banks στα οποία χωρίζεται η μνήμη.
- Δυνατότητα προγραμματισμού τριών διαφορετικών Burst Lengths (2, 4 και 8).
- Παρέχεται η δυνατότητα Auto Refresh και Self Refresh της μνήμης.
- Υποστηρίζεται η δυνατότητα Auto Precharge.
- Δυνατότητα προγραμματισμού τριών διαφορετικών CL (Cas Latency 2, 2.5 και 3).

2.4.2 Γενική Περιγραφή

Μία 256 Mb DDR SDRAM χρησιμοποιεί την αρχιτεκτονική Double-Data-Rate για την επίτευξη ταχύτερης λειτουργίας. Πρόκειται για μια αρχιτεκτονική σχεδιασμένη έτσι ώστε να επιτυγχάνει τη μεταφορά δύο δεδομένων (στην περίπτωσή μας των 16 bit) σε ένα κύκλο ρολογιού. Αυτό διότι υποστηρίζει τη μεταφορά δεδομένων είτε στη θετική είτε στην αρνητική ακμή του ρολογιού (καθίσταται δυνατό από την εσωτερική Pipelined **αρχιτεκτονική** της μνήμης). Εκτός όμως από αυτή την ειδοποιό διαφορά της DDR DRAM από τις απλές DRAM παρέχονται και αρκετές άλλες δυνατότητες και είναι στην ευχέρεια του Ελεγκτή αν θα τις αξιοποιήσει όλες ή κάποιες από αυτές. Στην περίπτωση αυτής της διπλωματικής εργασίας σκοπός όπως έχει προαναφερθεί δεν ήταν η εξάντληση όλων των δυνατοτήτων που αυτή παρέχει αλλά κάποιων βασικών χαρακτηριστικών έτσι ώστε να δημιουργηθεί ένα σύστημα που να λειτουργεί σωστά και να αξιοποιεί κάποιες βασικές δυνατότητες της μνήμης έτσι ώστε να επιτυγχάνει αρκετά καλή απόδοση.

Παρέχεται ακόμη ένα αμφίδρομο σήμα DQS το οποίο στην περίπτωση της 16-bit μνήμης είναι 2 bit (ένα για το lower byte και ένα για το upper byte των δεδομένων). Στην περίπτωση ανάγνωσης το σήμα αυτό όπως και τα δεδομένα αποστέλλονται από τη μνήμη απόλυτα συγχρονισμένα ενώ στην περίπτωση εγγραφής δεδομένα και DQS αποστέλλονται προς τη μνήμη μόνο που το σήμα πρέπει να αλλάζει την τιμή του στα μισά της αποστολής των δεδομένων προκειμένου να φθάσουν αυτά valid στη μνήμη (Ο τρόπος λειτουργίας του DQS θα γίνει περισσότερο κατανοητός στην παράθεση των διαγραμμάτων χρονισμού παρακάτω).

Ακόμη υποστηρίζονται δύο ρολόγια CK και CK#. Ουσιαστικά το CK# είναι το ανάστροφο του κανονικού ρολογιού CK. Δηλαδή όταν έρχεται μία θετική ακμή του CK έρχεται μία αρνητική του CK# και αντίστροφα. Οι εντολές που δίνονται από το χρήστη εισάγονται στη μνήμη σε θετική ακμή του ρολογιού ενώ τα δεδομένα (εσόδου ή εξόδου) μπορούν να εισαχθούν είτε στη θετική είτε στην αρνητική ακμή του ρολογιού (Στην περίπτωσή μας έγινε χρήση και ενός τρίτου ρολογιού CK90 το οποίο είναι μετατοπισμένο κατά 90^{o} ή 1/4 του ρολογιού). Από εδώ και στο εξής όταν αναφερόμαστε σε ρολόι χωρίς διευκρινίσεις θα μιλάμε για το κανονικό CK.

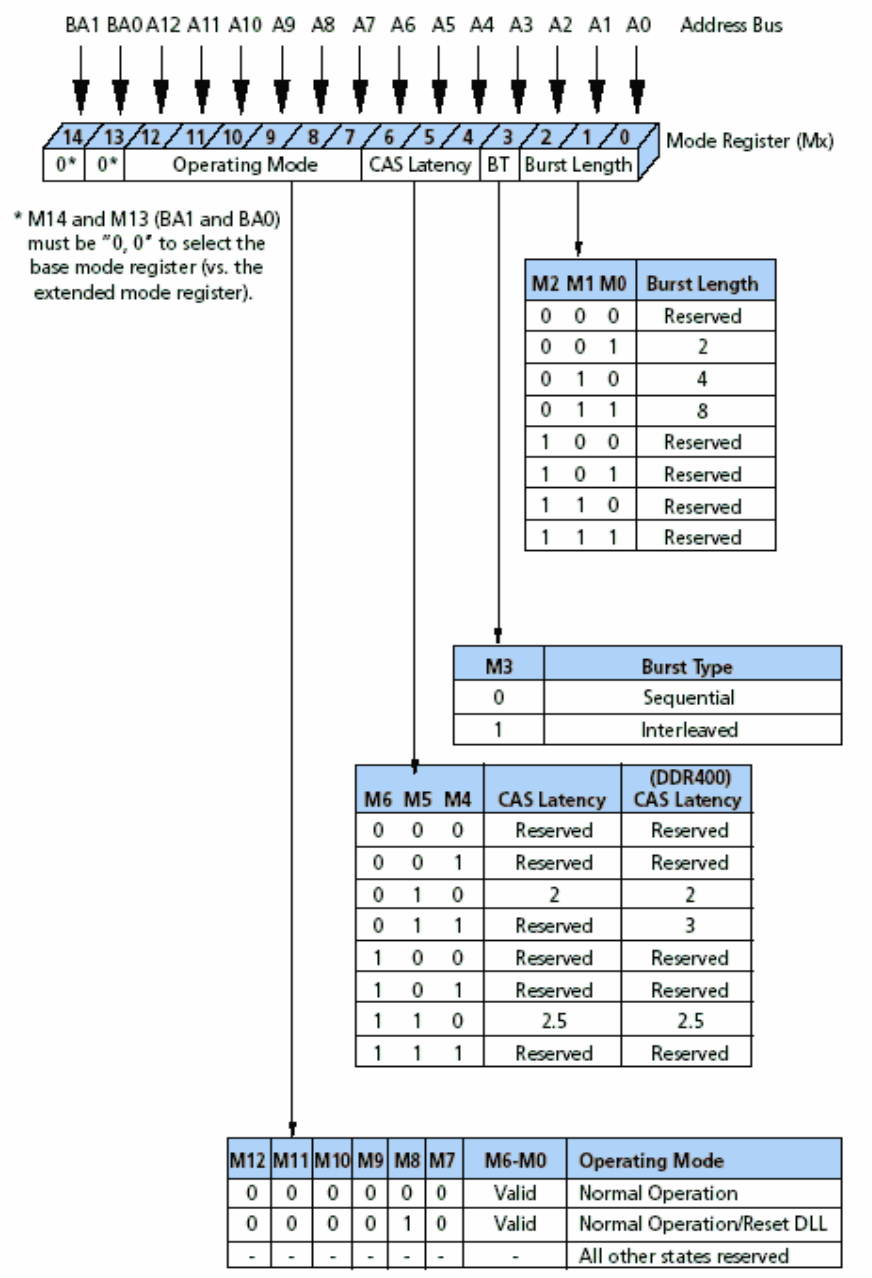
Μία αίτηση για ανάγνωση ή εγγραφή ορίζει μια συγκεκριμένη διεύθυνση από την οποία θα αρχίσει η όλη διαδικασία και συνεχίζει για ένα δεδομένο αριθμό διευθύνσεων σειριακά (Burst). Μία πρόσβαση είτε για ανάγνωση είτε για εγγραφή αρχίζει με την εισαγωγή μίας εντολής Active η οποία στη συνέχεια ακολουθείται από την αντίστοιχη εντολή ανάγνωσης (Read) ή εγγραφής (Write). Τα bit της διεύθυνσης που εισάγονται με την εντολή Active καθορίζουν σε ποιο Bank πρόκειται να γίνει η αντίστοιχη διεργασία καθώς επίσης και σε ποια σειρά, ενώ αφού παρέλθει ο χρόνος που χρειάζεται για να ολοκληρωθεί το Activation (3 κύκλοι ρολογιού) εισάγεται η επιθυμητή εντολή και τα bit της διεύθυνσης που εισάγονται με αυτήν, καθορίζουν το Bank και τη στήλη στην οποία θα γίνει / αρχίσει η απαιτούμενη διεργασία.

Η μνήμη παρέχει τη δυνατότητα προγραμματισμού του Burst Length με βάση το οποίο θα λειτουργεί η μνήμη (2, 4 ή 8). (Στην περίπτωσή μας η μνήμη είναι προγραμματισμένη με Burst Length 2). Επίσης παρέχεται η δυνατότητα Auto Precharge το οποίο κάνει Precharge της σειράς στην οποία πραγματοποιήθηκε η διεργασία αμέσως μόλις αυτή τελειώσει. (Στην περίπτωση μας το Auto Precharge χρησιμοποιείται σε κάθε εντολή).

2.4.3 **Προγραμματιζόμενοι Παράμετροι Της Μνήμης**

Αρχικά πρέπει να αναφερθεί ότι κατά τη διαδικασία αρχικοποίησης της μνήμης (Βλέπε **Παράγραφο** 3.2) παρέχεται στο χρήστη η δυνατότητα μέσα από μία Load Mode Register εντολή να προγραμματιστούν κάποιες παράμετροι της μνήμης που έχουν αυτή τη δυνατότητα. Όπως φαίνεται και από το **Σχήμα** 2.4, οι παράμετροι αυτοί είναι το Cas Latency, το Burst Type και το Burst Length.

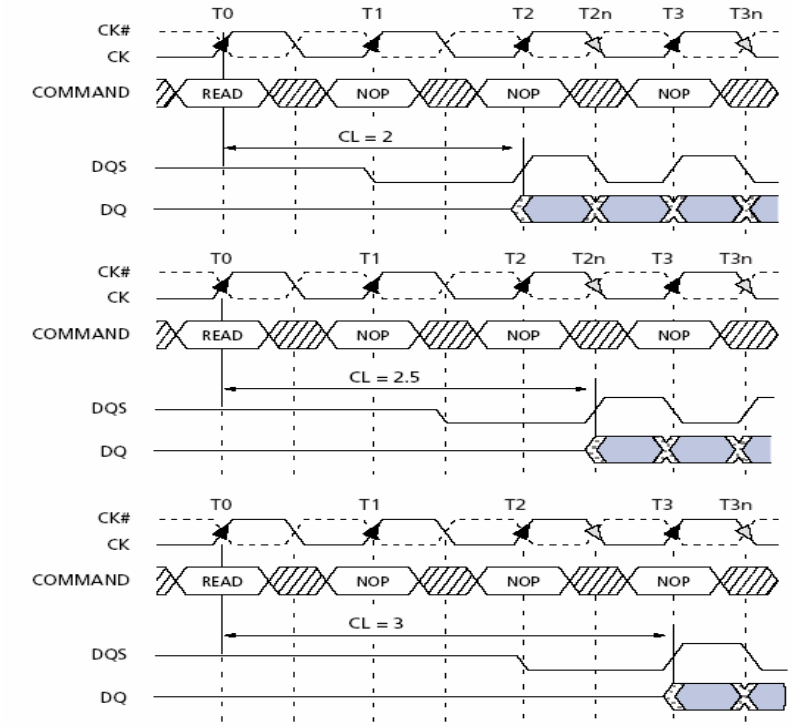
Σχήμα 2.4



2.4.3.1 CAS Latency

M6	M5	M4	CAS Latency	(DDR400) CAS Latency
0	0	0	Reserved	Reserved
0	0	1	Reserved	Reserved
0	1	0	2	2
0	1	1	Reserved	3
1	0	0	Reserved	Reserved
1	0	1	Reserved	Reserved
1	1	0	2.5	2.5
1	1	1	Reserved	Reserved

Το CAS Latency ή Read Latency αποτελεί έναν πολύ σημαντικό παράγοντα της μνήμης και ορίζει την καθυστέρηση (σε κύκλους ρολογιού) από τη στιγμή που εισάχθηκε στη μνήμη μία εντολή ανάγνωσης μέχρι την επιστροφή των πρώτων δεδομένων. Η καθυστέρηση αυτή ορίζεται ως CAS Latency (CL) και μπορεί να λάβει τρεις διαφορετικές τιμές 2, 2.5 και 3 όπως φαίνεται και από τον παρακάτω πίνακα. Όπως είναι φανερό από άποψη ταχύτητας είναι προτιμότερη η τιμή του Cas Latency = 2 για αυτό και οι προσομοιώσεις και δοκιμές του συστήματος που πραγματοποιήθηκαν (και κάποιες από αυτές θα ακολουθήσουν ως παραδείγματα) χρησιμοποιούν αυτή την τιμή. Παρόλα αυτά παρέχεται στον χρήστη του συστήματος η δυνατότητα να προγραμματίσει αυτός το Cas Latency που προτιμάει κατά την αρχικοποίηση της μνήμης.



2.4.3.2 Burst Length

M2	M1	M0	Burst Length
0	0	0	Reserved
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

Όπως διακρίνεται και από τον παραπάνω πίνακα, η μνήμη έχει τη δυνατότητα να λειτουργήσει με τρία διαφορετικά μεγέθη Burst. Αυτά έχουν όπως φαίνεται τις τιμές 2, 4 και 8. Το μέγεθος του Burst ορίζει το πόσα δεδομένα θα αποσταλούν από η προς τη μνήμη με μία και μόνο εντολή. Με αυτό τον τρόπο, στην περίπτωση όπου Burst Length = 2 αποστέλλονται 2 δεδομένα των 16 bit, και αντίστοιχα στις περιπτώσεις όπου Burst Length = 4 ή 8 αποστέλλονται 4 ή 8 δεδομένα των 16 bit.

Τα Burst Length με τιμή μεγαλύτερη του 2 συμφέρουν από άποψη ταχύτητας σε περίπτωση που είναι επιθυμητή η μεταφορά περισσότερων των 2 δεδομένων οπότε και αποστέλλονται χωρίς να παρεμβληθεί καθόλου κενός χρόνος σε ‘’πακέτα’’ των 4 ή των 8. Εφόσον όμως η παράμετρος αυτή προγραμματίζεται κατά την αρχικοποίηση της μνήμης, σε περίπτωση που είναι επιθυμητή η αποστολή λιγότερων δεδομένων και στη συνέχεια ακολουθεί μία νέα αίτηση, τότε η μεταφορά περισσότερων δεδομένων είναι περιττή οπότε πρέπει να διακοπεί με μία Burst Terminate εντολή και μετά να ξεκινήσει η εκτέλεση της επόμενης. Όλο αυτό προσθέτει κάποια καθυστέρηση στο σύστημα.

Για αυτό το λόγο, στην περίπτωση του Ελεγκτή που υλοποιήθηκε σε αυτή τη διπλωματική εργασία, το μοναδικό Burst Length που υποστηρίχθηκε είναι αυτό με τιμή 2 και ο Ελεγκτής είναι κατασκευασμένος έτσι ώστε να λειτουργεί μόνο με βάση αυτή την τιμή. Αυτό διότι, όπως θα αναφερθεί και στη συνέχεια τις υλοποίησης, ο Ελεγκτής δημιουργήθηκε έτσι ώστε να καλύπτει τη δυνατότητα μεταφοράς δεδομένων με μία και μόνο εντολή με Burst μέχρι 256, τα οποία χρησιμοποιούν το Burst Length = 2. Με αυτό τον τρόπο παρέχεται στο χρήστη η δυνατότητα να μην χάνει χρόνο στις μεταφορές των ελάχιστων δυνατών δεδομένων (2) και να υποστηρίζονται αρκετά μεγάλα Burst (αρκετά μεγαλύτερα της μέγιστης τιμής του Burst Length) χωρίς να χάνεται καθόλου χρόνος.

Βέβαια είναι δυνατή η υλοποίηση και των υπολοίπων δύο περιπτώσεων του Burst Length, με απλή από άποψη λογικής και λίγο πιο σύνθετης από άποψη υλοποίησης, επέκτασης της FSM (**Παράγραφος** 3.3) του Ελεγκτή.

2.4.3.3 Burst Type

M3	Burst Type
0	Sequential
1	Interleaved

Όπως διακρίνεται και από το **Σχήμα** 2.4 ο τύπος του Burst μπορεί να λάβει μόνο δύο τιμές. Ανάλογα με το ποιος τύπος επιλέγεται, λειτουργεί αντίστοιχα η μνήμη στον καθορισμό της ακολουθίας των διευθύνσεων των στηλών στις οποίες πρόκειται να πραγματοποιηθεί προσπέλαση από κάποια εντολή με δεδομένο Burst Length. Για καλύτερη κατανόηση των παραπάνω παρατίθεται ο πίνακας με βάση τον οποίο ορίζεται ο τρόπος λειτουργίας των διαφορετικών τύπων του Burst.

Burst Length	Starting Column Address			Order of Accesses Within a Burst	
				Type = Sequential	Type = Interleaved
2			A0		
			0	0-1	0-1
			1	1-0	1-0
4		A1	A0		
		0	0	0-1-2-3	0-1-2-3
		0	1	1-2-3-0	1-0-3-2
		1	0	2-3-0-1	2-3-0-1
		1	1	3-0-1-2	3-2-1-0
8	A2	A1	A0		
	0	0	0	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
	0	0	1	1-2-3-4-5-6-7-0	1-0-3-2-5-4-7-6
	0	1	0	2-3-4-5-6-7-0-1	2-3-0-1-6-7-4-5
	0	1	1	3-4-5-6-7-0-1-2	3-2-1-0-7-6-5-4
	1	0	0	4-5-6-7-0-1-2-3	4-5-6-7-0-1-2-3
	1	0	1	5-6-7-0-1-2-3-4	5-4-7-6-1-0-3-2
	1	1	0	6-7-0-1-2-3-4-5	6-7-4-5-2-3-0-1
	1	1	1	7-0-1-2-3-4-5-6	7-6-5-4-3-2-1-0

Εφόσον όπως έχει αναφερθεί και στην **Παράγραφο** 2.4.3.2 το Burst Length με βάση το οποίο λειτουργεί ο συγκεκριμένος Ελεγκτής (και επομένως και η μνήμη) είναι το 2, γίνεται φανερό ότι η διευθυνσιοδότηση δεν παρουσιάζει αλλαγές. Παρόλα αυτά χάριν δυνατότητας εξέλιξης παρέχεται στο χρήστη η δυνατότητα προγραμματισμού του Burst Type που προτιμά κατά την αρχικοποίηση της μνήμης.

2.4.4 **Λοιπές Δυνατότητες**

Εκτός από τις παραπάνω παραμέτρους τις μνήμης οι οποίες έχουν τη δυνατότητα να λαμβάνουν διαφορετικές τιμές και να οδηγούν σε διαφορετική λειτουργία της μνήμης, υπάρχουν και αρκετά σήματα τα οποία απαιτούν τον κατάλληλο συγχρονισμό έτσι ώστε να λειτουργεί ορθά κάθε εντολή αλλά και η μνήμη στο σύνολό της.

Θεωρητικά, η μνήμη έχει τη δυνατότητα να εκτελεί διάφορες εντολές που θα τις εισάγονται όπως είναι τα Refresh, Precharge, Activate κ.α. Στην περίπτωση αυτού του Ελεγκτή δεν παρέχεται αυτή η δυνατότητα γιατί απλούστατα το σύστημα λειτουργεί από την πλευρά του χρήστη. Οι βασικές ανάγκες λοιπόν ενός χρήστη είναι να μπορεί να γράψει σωστά στη μνήμη κάποια πληροφορία που επιθυμεί και όταν επιθυμεί να μπορεί να αναγνώσει αυτή την πληροφορία.

Για αυτό το λόγο και οι μόνες εντολές που μπορεί να εισάγει είναι η Initialize για την σωστή αρχικοποίηση της μνήμης, η εντολή Write που υποδηλώνει αίτηση για εγγραφή στη μνήμη και η εντολή Read που δηλώνει αίτηση για ανάγνωση από τη μνήμη (Βλέπε **Παράγραφο** 3.4.4).

Η μνήμη όπως έχει ήδη αναφερθεί χωρίζεται σε 4 Bank κάθε ένα από τα οποία αποτελείται από 8 K σειρές και 512 στήλες των 16 bit. Προτού πραγματοποιηθεί πρόσβαση (είτε για ανάγνωση, είτε για εγγραφή) σε κάποιο Bank πρέπει αυτό να αρχικοποιηθεί. Αυτό σημαίνει ότι πρέπει να δοθεί στη μνήμη η κατάλληλη ακολουθία σημάτων που να υποδηλώνει την εισαγωγή μίας εντολής Active. Για την περάτωση της σωστής ενεργοποίησης ενός Bank πρέπει να παρέλθουν τρεις κύκλοι ρολογιού και στη συνέχεια, εφόσον το Bank έχει ενεργοποιηθεί, μπορεί να εκκινήσει την εκτέλεσή της η αντίστοιχη εντολή. Αυτό το Bank παραμένει ενεργό εωσότου δοθεί ξανά στη μνήμη η κατάλληλη ακολουθία σημάτων που να υποδηλώνει της εισαγωγή μίας εντολής Precharge. Με αυτόν τον τρόπο, θεωρητικά θα μπορούσαν να πραγματοποιηθούν στη μνήμη Activate για όλα τα Bank, στη συνέχεια να εκτελούνταν κάποιες εντολές μέσα στο κάθε ένα από αυτά και εφόσον τελείωναν όλες οι εντολές στο κάθε Bank την εκτέλεσή τους να πραγματοποιούνταν ένα Precharge για όλα τα Bank. Στην περίπτωση αυτού του Ελεγκτή κάτι τέτοιο δεν υποστηρίζεται. Αυτό διότι βλέποντας το σύστημα από τη μεριά του χρήστη, αυτό που αυτός ξέρει και επιθυμεί είναι απλά να εκτελέσει μία εντολή. Για αυτό το λόγο, σε κάθε εισαγωγή εντολής από το χρήστη πρέπει να πραγματοποιηθεί ενεργοποίηση του Bank στο οποίο πρόκειται να γίνει η πρόσβαση, στη συνέχεια εκτελείται η εντολή και με την ολοκλήρωσή της αυτόματα εκκινεί μία διαδικασία Auto Precharge για το συγκεκριμένο Bank. Σε περίπτωση βέβαια που πρόκειται για εντολή με Burst Size μεγαλύτερο του 1, το Bank διατηρείται ενεργοποιημένο καθ' όλη τη διάρκεια της εκτέλεσης της εντολής και το Auto Precharge εισάγεται με την διεύθυνση που απευθύνεται στα τελευταία δεδομένα.

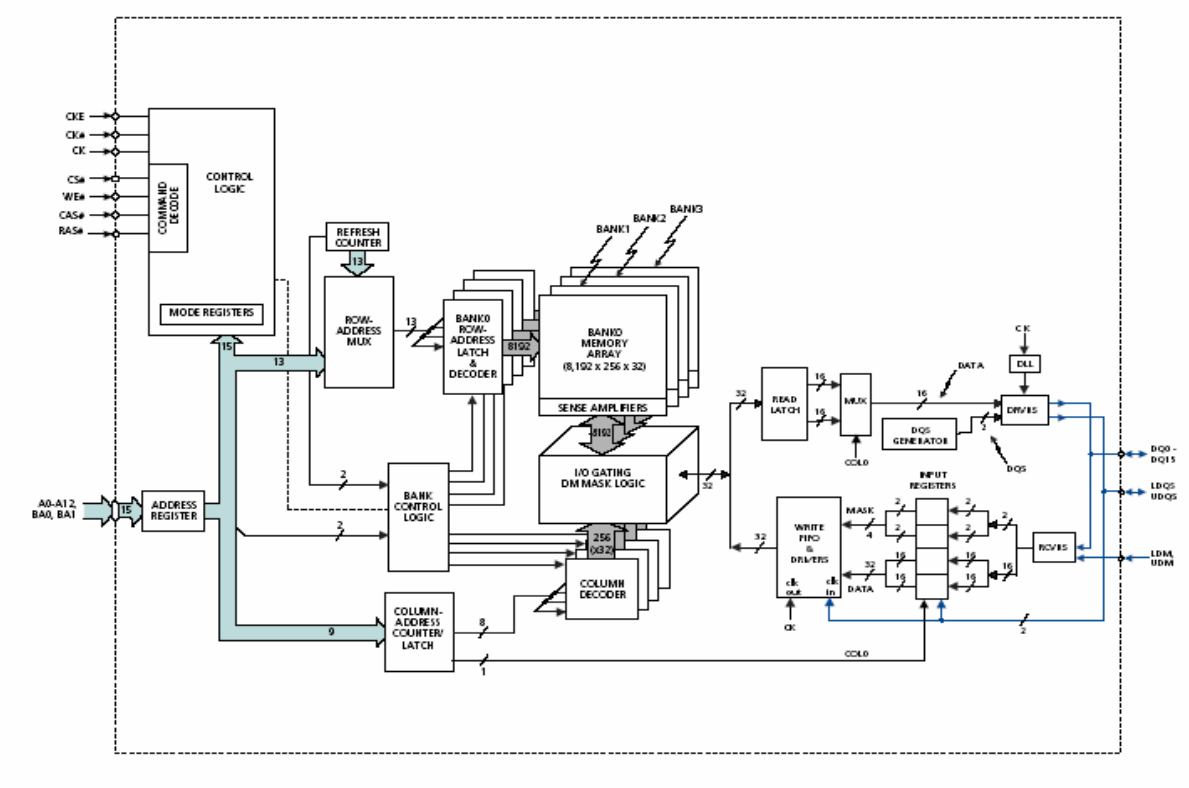
Αξίζει να αναφερθεί ότι έγινε σκέψη για τυχόν δυνατή παράλειψη του Precharge σε περίπτωση που δύο διαδοχικές εντολές απευθύνονται στο ίδιο Bank έτσι ώστε να μην χρειαστεί στην εκτέλεση της δεύτερης εντολής να πραγματοποιηθεί Activation. Προκειμένου όμως να γίνει ο έλεγχος θα χανόταν κάποιος χρόνος και σε περίπτωση που δεν ήταν δυνατή η παράλειψη του Activate (δηλαδή οι δύο συνεχόμενες εντολές δεν αφορούν το ίδιο Bank) θα χρειαζόταν να εισαχθεί μία εντολή Precharge. Με αυτό τον τρόπο θα εξοικονομούνταν κάποιος χρόνος σε περίπτωση που ήταν δυνατή η παράλειψη του Activate, θα χανόταν όμως χρόνος σε περίπτωση που αυτό δεν ήταν δυνατό. Για αυτό το λόγο και προτιμήθηκε η παραπάνω λύση η οποία υποστηρίζει την παράλειψη των Precharge και Activate μόνο σε περίπτωση εντολών με μεγάλο Burst Size.

2.4.5 **Δυνατότητες Διευθυνσιοδότησης**

	64 Meg x 4	32 Meg x 8	16 Meg x 16
Configuration	16 Meg x 4 x 4 banks	8 Meg x 8 x 4 banks	4 Meg x 16 x 4 banks
Row Addressing	8K (A0 - A12)	8K (A0 - A12)	8K (A0 - A12)
Bank Addressing	4 (BA0, BA1)	4 (BA0, BA1)	4 (BA0, BA1)
Column Addressing	2K (A0 - A9, A11)	1K (A0 - A9)	512 (A0 - A8)

Στον παραπάνω πίνακα παρατίθενται οι τρεις δυνατοί τρόποι διευθυνσιοδότησης της 256 Mb μνήμης. Από εδώ και στο εξής θα αναφερόμαστε στην τρίτη περίπτωση (με την οποία και ασχοληθήκαμε). Όπως καθίσταται φανερό και από τον πίνακα η μοναδική διαφορά των τριών δυνατών τρόπων διευθυνσιοδότησης παρουσιάζεται στη διευθυνσιοδότηση των στηλών της μνήμης.Έτσι η μνήμη MT46V64M4 η οποία είναι και αυτή πάνω στην οποία δουλέψαμε χωρίζεται σε 4 Bank εκ των οποίων το κάθε ένα έχει 8 K σειρές και κάθε σειρά έχει 512 στήλες των 16 bit η κάθε μία. Όπως είναι λογικό λοιπόν επαληθεύοντας : (4 Banks) x (8K **σειρές**) x (512 **στήλες**) x 16 bit = 256 Kb.

2.4.6 Block Diagram **Της** 16 Meg x 16 **Μνήμης**



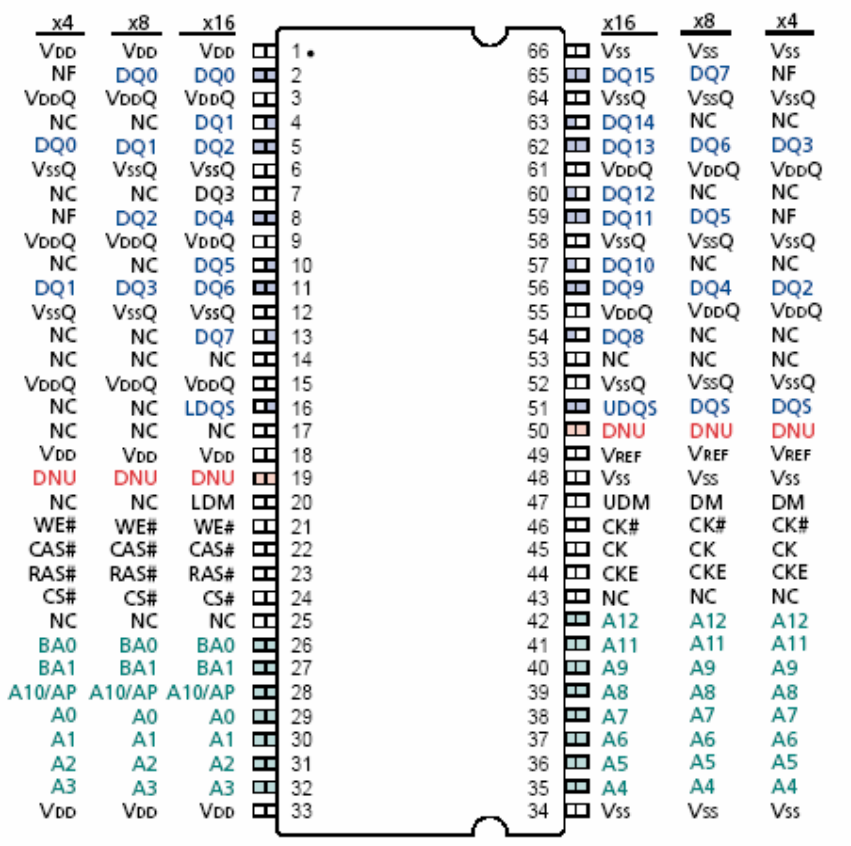
Όπως έχει ήδη αναφερθεί και στο προηγούμενο κεφάλαιο, η λογική της μνήμης είναι αρκετά σύνθετη / πολύπλοκη και δύσκολη στην κατανόησή της. Αυτό φαίνεται και από το Block Diagram της μνήμης που δίνεται στην ιστοσελίδα της ***Micron***. Για αυτό το λόγο αν και δεν είναι εύκολη η κατανόηση του όλου συστήματος και του τρόπου λειτουργίας της κάθε μονάδας του Block Diagram, μπορεί κάποιος να διακρίνει κάποια βασικά στοιχεία για τα οποία θα γίνει λόγος στη συνέχεια. Κάποια από αυτά είναι τα σήματα που φθάνουν στη μνήμη και εισέρχονται σε μία μονάδα ελέγχου από την οποία εξάγεται και το συμπέρασμα για τη λειτουργία που ζητείται από το χρήστη ανάλογα με τις τιμές και τον χρονισμό των σημάτων.

Επίσης παρατηρείται η ύπαρξη ενός καταχωρητή που κρατάει την τιμή της διεύθυνσης και του Bank που έρχονται κάθε στιγμή στη μνήμη. Στη συνέχεια και ανάλογα με το είδος της εντολής που έχει εισαχθεί στο σύστημα αποφασίζεται αν η διεύθυνση απευθύνεται σε γραμμές ή στήλες του Bank που αναφέρεται και ανάλογα αποστέλλεται στην αντίστοιχη μονάδα. Διακρίνονται ακόμη τα τέσσερα διαφορετικά Bank στα οποία χωρίζεται η μνήμη.

Από την άλλη μεριά διακρίνεται η αμφίδρομη λειτουργία των καναλιών τόσο για τα δεδομένα (Dq) όσο και για το σήμα Dqs καθώς επίσης και μία μονάδα Dqs Generator η οποία παράγει και αποστέλλει στον χρήστη κατάλληλα συγχρονισμένα με τα δεδομένα το σήμα Dqs σε περίπτωση εντολής ανάγνωσης.

Η υπόλοιπη λογική είναι αρκετά δύσκολη στην κατανόησή της χωρίς όμως να επηρεάζει την υλοποίηση του Ελεγκτή, αφού τα βασικά χαρακτηριστικά που θα χρησιμοποιηθούν από τον Ελεγκτή είναι και αυτά που αναφέρθηκαν παραπάνω.

2.4.7 Pin Assignment (Top View) 66 bit TSOP



TSOP Numbers	Σύμβολο	Τύπος	Περιγραφή
45,46	CK, CK#	Είσοδος	Τα δύο ρολόγια της Μνήμης. Όλες οι εντολές εισέρχονται
			στη θετική ακμή του CK ενώ τα δεδομένα (εισόδου ή εξόδου)
			μπορούν να εισαχθούν είτε στη θετική είτε στην αρνητική ακμή.
44	CKE	Είσοδος	Ενεργοποιεί ή απενεργοποιεί το εσωτερικό ρολόι. Πρέπει να
			διατηρείται HIGH τη στιγμή ανάγνωσης ή εγγραφής.
24	CS#	Είσοδος	Chip Select : Ενεργοποιεί (στο LOW) και απενεργοποιεί (στο
			HIGH) την αποκωδικοποίηση των εντολών
23, 22, 21	RAS#, CAS#, WE#	Είσοδος	Μαζί με το CS# καθορίζουν την εντολή που εισάγεται.
47	DM	Είσοδος	InputDataMask.
26, 27	BA0, BA1	Είσοδος	Bank Address : Καθορίζουν σε ποιο Bank απευθύνεται η
			εκάστοτε εντολή.
29, 30, 31, 32, 35,	A0, A1, A2, A3, A4,	Είσοδος	Address Inputs : Παρέχουν τα Row Address (με την εντολή
36, 37, 38, 39, 40,	A5, A6, A7, A8, A9,		Active) και Column Address (με Read, Write). Το A10 σε
28, 41, 42	A10, A11, A12		περίπτωση Precharge καθορίζει αν η εντολή απευθύνεται στο
			συγκεκριμένο Bank (A10 LOW) ή σε όλα τα Bank (A10 HIGH).
2, 4, 5, 7, 8, 10,	DQ0 - DQ5	Είσοδος / Έξοδος	Data Input / Output : Η γραμμή μεταφοράς δεδομένων στην
11,13, 54, 56, 57, 59,	DQ6 - DQ11		περίπτωση x16.
60, 62, 63, 65	DQ12 - DQ15		
51	DQS	Είσοδος / Έξοδος	Data Strobe : Έξοδος σε ανάγνωση, είσοδος σε εγγραφή.
1, 18, 33	VDD	Τροφοδοσία	Power Supply

3. **Υλοποίηση**

3.1 **Εισαγωγή**

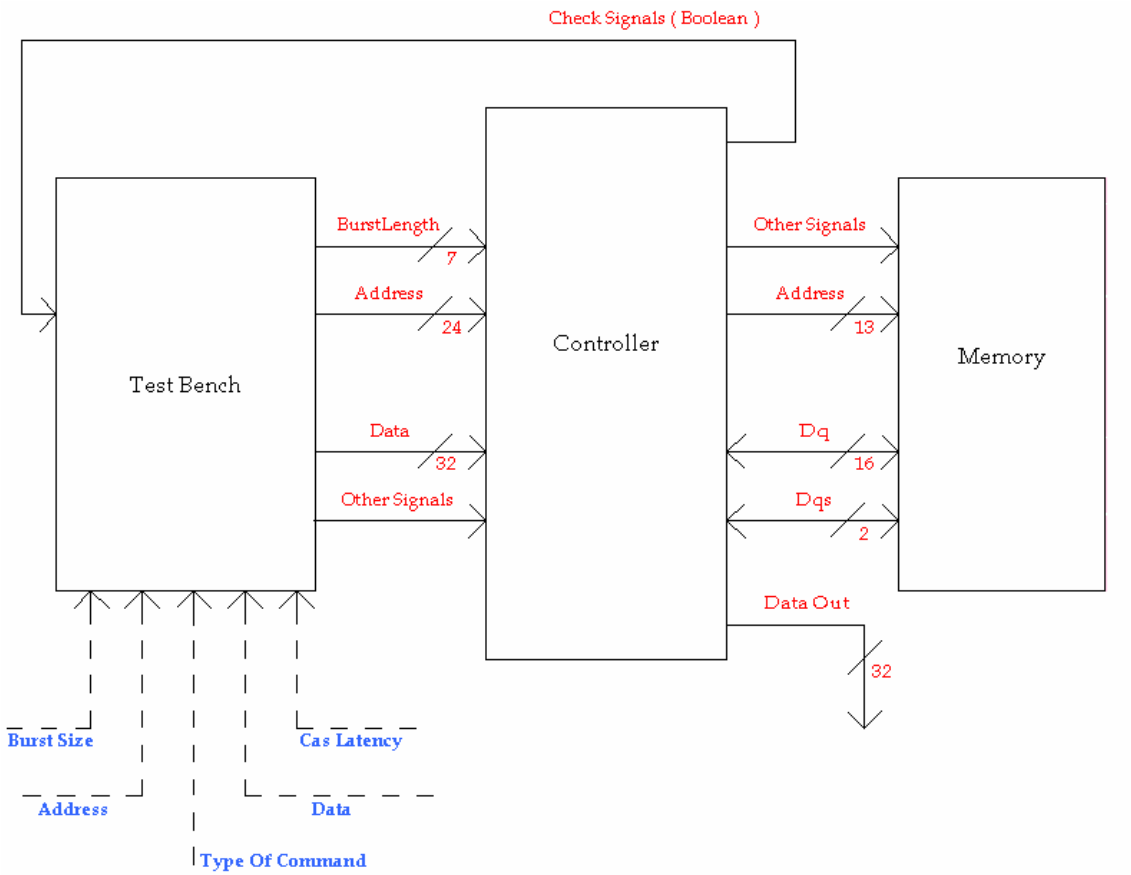
Η υλοποίηση αυτής της διπλωματικής εργασίας απαιτούσε, όπως έχει προαναφερθεί, την δημιουργία ενός Ελεγκτή (Controller) μίας DDR DRAM. Αυτό πραγματοποιήθηκε στη γλώσσα VHDL και με το εργαλείο ModelSim SE 6.0a. Για την δημιουργία αλλά και τη δυνατότητα προσομοίωσης του ελεγκτή χρειάστηκε ένα μοντέλο μνήμης (σε VHDL) πάνω στο οποίο θα κατασκευαζόταν ο ελεγκτής έτσι ώστε πληρώντας τις προδιαγραφές της συγκεκριμένης μνήμης να επιδιώξει όσο μεγαλύτερη απόδοση και ταχύτητα γίνεται. Η μνήμη που χρησιμοποιήθηκε λήφθηκε από την εταιρία Micron και είναι μία DDR SDRAM 256 Mb (MT46V64M4).

Όπως έχει αναφερθεί και στην **Παράγραφο** 1.5 ο Ελεγκτής που υλοποιήθηκε είχε σαν σκοπό την επίτευξη του συνδυασμού ταχύτητας και απλότητας. Για το λόγο αυτό αξιοποιήθηκαν όσο το δυνατόν περισσότερες δυνατότητες παρείχε η μνήμη, υπακούοντας πάντα στους περιορισμούς της, χωρίς όμως το σύστημα να καταστεί ιδιαίτερα πολύπλοκο και δυσνόητο. Στόχος της υλοποίησης ήταν η δημιουργία ενός απλού σχετικά Interface του όλου συστήματος έτσι ώστε με μία πρώτη ματιά να είναι εύκολα κατανοητός ο βασικός τρόπος λειτουργίας του, κρύβοντας με αυτό τον τρόπο την πολύ σύνθετη λειτουργία των επιμέρους μονάδων του συστήματος.

Σε αυτή την περίπτωση είναι αρκετά εύκολη η χρήση του συστήματος διαχείρισης της DDR από έναν χρήστη ο οποίος δεν επιθυμεί να εμβαθύνει στις λεπτομέρειες της υλοποίησης ούτε να κατανοήσει τις λεπτομέρειες λειτουργίας των επιμέρους συστημάτων. Η λογική τόσο της μνήμης όσο και του Ελεγκτή αλλά και της μονάδας Χειρισμού του συστήματος είναι αρκετά σύνθετη. Για αυτό το λόγο, όπως θα φανεί και αργότερα στο **Κεφάλαιο** 4, ο χρήστης προκειμένου να μην εισέλθει καθόλου, εφόσον δεν το επιθυμεί, στην λογική των επιμέρους μονάδων έχει να ασχοληθεί με ένα απλό σύστημα (Interface) της μορφής του **Σχήματος** 3.1 και απλά να εισάγει τις εντολές που επιθυμεί με τον απλό τρόπο που παρατίθεται στο **Κεφάλαιο** 4.

Για την περίπτωση αναγνωστών και χρηστών που ενδιαφέρονται για την περαιτέρω κατανόηση της λειτουργίας της κάθε επιμέρους μονάδας του συστήματος και την ανάλυση του τρόπου λειτουργίας της, ακολουθεί εκτεταμένη περιγραφή της λειτουργίας τόσο της κάθε υπομονάδας χωριστά όσο και του όλου συστήματος ιεραρχικά.

Σχήμα 3.1



Στο παραπάνω σχήμα διακρίνεται η απλότητα του συστήματος, όπως μπορεί ο χρήστης να το αντιληφθεί σε περίπτωση που δεν ενδιαφέρεται για περισσότερες λεπτομέρειες. Παρέχεται λοιπόν η δυνατότητα στο χρήστη να κατανοήσει με τον απλούστερο δυνατό τρόπο την ‘’επικοινωνία’’ ανάμεσα στις τρεις βασικές μονάδες του συστήματος, ‘’καλύπτοντας’’ τις δύσκολες, δυσνόητες και πολύπλοκες λειτουργίες και λογικές των μονάδων αυτών.

Διακρίνεται από το παραπάνω Διάγραμμα η παρέμβαση του χρήστη, ο οποίος εισάγει στο Σύστημα τα δεδομένα που διακρίνονται με **μπλε χρώμα** όπως περιγράφονται αναλυτικότατα στο **Κεφάλαιο** 4. Αυτά είναι το είδος της προς εκτέλεση εντολής (Read, Write και Initialize), το Cas Latency σε περίπτωση εντολής αρχικοποίησης και τα Burst Size, Address και Data σε περίπτωση εντολής Εγγραφής ή Ανάγνωσης.

Μία ακόμη βασική ιδιότητα του συστήματος, και ειδικότερα της μνήμης, που διακρίνεται από το παραπάνω Διάγραμμα είναι η αμφίδρομη ιδιότητα των Buses τόσο για τα δεδομένα όσο και για το σήμα Dqs που τα ‘’συνοδεύει’’.

Τέλος εύκολα εξάγεται το συμπέρασμα επεξεργασίας των εισόδων και των εξόδων του Ελεγκτή έτσι ώστε να μεταφέρει δεδομένα από και προς τη μνήμη, όπως και από και προς τον χρήστη. Αυτό διότι ο χρήστης, όπως διακρίνεται, εισάγει 32 bit πληροφορίας και 24 bit διεύθυνσης ενώ η μνήμη αναγνωρίζει 16 και 13 bit αντίστοιχα. Επίσης σε περίπτωση εντολής ανάγνωσης η μνήμη επιστρέφει 16 bit δεδομένων ενώ στον χρήστη φθάνουν 32 bit. Για την περαιτέρω κατανόηση της λειτουργίας του συστήματος αλλά και της κάθε υπομονάδας αυτού, ακολουθεί λεπτομερής ανάλυση και περιγραφή.

3.2 **Λειτουργία Του Ελεγκτή**

Ο Ελεγκτής που κατασκευάστηκε σε αυτή τη διπλωματική εργασία είχε σαν σκοπό τη σωστή διαχείριση της παραπάνω μνήμης καλύπτοντας τις απαιτήσεις της και προσπαθώντας να συμβάλει στη σωστή διαχείρισή της, βελτιστοποιώντας όσο αυτό ήταν δυνατό την απόδοσή της.

Μια DDR μνήμη πρέπει, για τη σωστή λειτουργία της, να αρχικοποιηθεί πριν εισαχθεί οποιαδήποτε άλλη εντολή σε αυτή. Έτσι αρχικά πρέπει να τροφοδοτηθεί με την κατάλληλη τάση και στη συνέχεια να παρέλθει ένα καθορισμένο χρονικό διάστημα πριν η μνήμη καταστεί έτοιμη για λειτουργία (Αν δεν γίνει η απαραίτητη προετοιμασία / αρχικοποίηση της μνήμης, αυτό μπορεί να οδηγήσει σε εσφαλμένη / μη προσδιορίσιμη λειτουργία της). Για αυτό το λόγο και η πρώτη βασική λειτουργία του Controller που υλοποιήθηκε είναι η σωστή και ολοκληρωμένη αρχικοποίηση της μνήμης. Σε διαφορετική περίπτωση δεν επιτρέπει σε καμία εντολή να προχωρήσει στη μνήμη.

Κατά την διαδικασία αρχικοποίησης μίας τέτοιας μνήμης, καθίσταται δυνατός ο προγραμματισμός κάποιων παραμέτρων της μνήμης οι οποίοι έχουν αυτή τη δυνατότητα (Βλέπε **Παράγραφο** 2.4.3). Τέτοιες παράμετροι είναι το Burst Length της μνήμης και το Burst Type της. Οι δυνατοί συνδυασμοί των δύο αυτών παραμέτρων διακρίνονται στον πίνακα που παρατίθεται στην **Παράγραφο** 2.4.3. Στον Ελεγκτή που υλοποιήθηκε, υποστηρίζεται το Burst Length με τιμή 2 με Burst Type σειριακό (Sequential) χάριν ταχύτητας και απλότητας όπως αναφέρθηκε και προηγουμένως.

Μία ακόμη παράμετρος που λαμβάνει την τιμή της κατά τη διάρκεια της αρχικοποίησης είναι και το CL (Cas Latency – βλέπε **Παράγραφο** 2.4.3.1). Στην περίπτωση αυτού του Ελεγκτή παρέχεται στον χρήστη η δυνατότητα επιλογής της τιμής του CL που επιθυμεί (2, 2.5 ή 3) την οποία και εισάγει όταν εισάγει την εντολή αρχικοποίησης της μνήμης.

Βασικότερη ίσως λειτουργία ενός Ελεγκτή είναι η σωστή διαχείριση των σημάτων από και προς τη μνήμη και ο κατάλληλος συγχρονισμός τους. Αυτό είναι απαραίτητο τόσο κατά την αρχικοποίηση της μνήμης (έτσι ώστε με την κατάλληλη ακολουθία των εντολών και των σημάτων που αυτές συνεπάγονται να ολοκληρωθεί με επιτυχία η αρχικοποίηση) όσο και εφόσον αυτή ολοκληρωθεί σε οποιαδήποτε άλλη εντολή. Η λειτουργία της μνήμης πραγματοποιείται σε χρόνο της τάξης των ns με αποτέλεσμα μία καθυστέρηση ενός εκ των σημάτων της κατά λίγα ns να οδηγεί ενδεχομένως σε λανθασμένη λειτουργία και εκτέλεση της εντολής. Ακόμη ο Ελεγκτής είναι αυτός που διακρίνει κάθε φορά το είδος της εντολής που εισάγεται (π.χ. ανάγνωση ή εγγραφή) και μπαίνει στην αντίστοιχη ακολουθία εντολών ώστε να γίνει κατανοητό και από τη μνήμη για τι εντολή πρόκειται έτσι ώστε να εκτελεστεί και η αντίστοιχη διεργασία. Και σε αυτή την περίπτωση είναι απαραίτητος ο κατάλληλος συγχρονισμός των σημάτων.

Σε μνήμες όπως αυτή που χρησιμοποιείται στην διπλωματική αυτή εργασία είναι δυνατή η χρήση Auto Precharge. Αυτή τη δυνατότητα την αξιοποιεί ο συγκεκριμένος ελεγκτής στο μέγιστο χρησιμοποιώντας την σε κάθε εντολή έτσι ώστε να μην είναι απαραίτητη η ξεχωριστή εισαγωγή μίας εντολής Precharge. Η εντολή Auto Precharge καθιστά δυνατή την εκκίνηση εκτέλεσης του Precharge αμέσως μόλις αυτό είναι δυνατό (δηλαδή με το που τελειώσει η εκτέλεση της αντίστοιχης εντολής που έκανε χρήση του Auto Precharge) σε αντίθεση με μία Precharge εντολή που πρέπει να εισαχθεί έπειτα από ένα κύκλο του ρολογιού εφόσον ολοκληρωθεί η εκτέλεση της εντολής.

Σκοπός της εργασίας δεν ήταν απλώς η διαχείριση των εντολών και η αποστολή τους στη μνήμη από τον Ελεγκτή αλλά και όσο αυτό ήταν δυνατό η βελτιστοποίηση της απόδοσης του συστήματος. Για αυτό το λόγο και υποστηρίχθηκε η δυνατότητα εισαγωγής εντολών με μεγάλο Burst. Με αυτό τον τρόπο, όταν είναι επιθυμητή η πρόσβαση πολλών ‘’γειτονικών’’ δεδομένων (συνεχόμενες στήλες που βρίσκονται στην ίδια σειρά ενός Bank) επιτυγχάνεται πολύ μεγαλύτερη ταχύτητα απ’ ότι αν η κάθε πρόσβαση πραγματοποιούνταν χωριστά (για κάθε επιπλέον (δηλαδή εκτός της πρώτης) πρόσβαση για εγγραφή γίνεται εξοικονόμηση 6 κύκλων του ρολογιού ενώ στην περίπτωση ανάγνωσης 8 κύκλων). Αν κάποιος αναλογιστεί ότι το μέγιστο δυνατό Burst που υποστηρίζεται από τον ελεγκτή είναι 255, το κέρδος σε ταχύτητα και απόδοση είναι πολύ μεγάλο.

Μία ακόμη περίπτωση στην οποία ο Ελεγκτής διακρίνει αν μπορεί να βελτιστοποιήσει την απόδοση του συστήματος είναι η περίπτωση κατά την οποία δύο συνεχόμενες εντολές αφορούν πρόσβαση σε διαφορετικό Bank. Κάθε εντολή ανάγνωσης ή εγγραφής, όπως έχει προαναφερθεί, κάνει χρήση του Auto Precharge. Αυτό σημαίνει ότι με την περάτωση της πρόσβασης σε ένα Bank που είναι Activated πρέπει να παρέλθει κάποιος χρόνος (4 κύκλοι ρολογιού) έτσι ώστε να ολοκληρωθεί το Precharge με επιτυχία. Αυτό σημαίνει ότι σε περίπτωση που δύο εντολές απευθύνονται στη μνήμη με χρονική καθυστέρηση (η μία από την άλλη) μεγαλύτερη από τέσσερις κύκλους ρολογιού όλα συνεχίζουν χωρίς πρόβλημα την εκτέλεσή τους.

Σε περίπτωση όμως που οι δύο εντολές εισέλθουν ''κολλητά'' υπάρχει πρόβλημα επικάλυψης των χρόνων που απαιτούνται για το Precharge του Bank της εντολής που ολοκληρώθηκε με το Activation του Bank της εντολής που ακολουθεί. Σε περίπτωση που οι δύο εντολές απευθύνονται σε δύο διαφορετικά Bank η μνήμη παρέχει τη δυνατότητα, παράλληλα με την ολοκλήρωση του Precharge του πρώτου Bank να εκτελεί τη διαδικασία του Activate του άλλου. Σε περίπτωση όμως που οι δύο εντολές απευθύνονται στο ίδιο Bank οι δύο διαδικασίες δεν είναι δυνατόν να επικαλυφθούν. Χάριν βελτιστοποίησης ο Ελεγκτής εκμεταλλεύεται αυτή τη δυνατότητα της μνήμης και διαχωρίζει τις περιπτώσεις όπου δύο διαδοχικές εντολές απευθύνονται σε διαφορετικά Bank με αποτέλεσμα σε κάθε τέτοια περίπτωση να εξοικονομούνται το λιγότερο από 1 έως 3 κύκλοι του ρολογιού όπως φαίνεται και από τις **Παραγράφους** 3.4.2 και 3.4.2.4.

Τέλος, στον Ελεγκτή που υλοποιήθηκε, αξιοποιείται και η δυνατότητα Auto Refresh που παρέχεται σε αυτό το είδος των μνημών. Έτσι γίνεται έλεγχος κάθε στιγμή για το αν κρίνεται απαραίτητο το Refresh και το σύστημα περνάει από μόνο του στην εκτέλεση του Refresh όταν αυτό χρειάζεται. (Για καλύτερη κατανόηση και μεγαλύτερη ανάλυση όλων των παραπάνω βλέπε **Παράγραφο** 3.3)

3.2.1 **Διαχωρισμός Μονάδων Υλοποίησης**

Σκοπός αυτής της διπλωματικής εργασίας όπως έχει προαναφερθεί ήταν η δημιουργία ενός Ελεγκτή που θα διαχειρίζεται σωστά και ''γρήγορα'' τη συγκεκριμένη μνήμη. Για αυτό το λόγο ήταν απαραίτητη η δημιουργία / υλοποίηση δύο υπομονάδων που αποτελούν το σύστημα. Μία είναι η κύρια και σημαντικότερη μονάδα του Ελεγκτή και η άλλη είναι η λιγότερο σημαντική από άποψη υλοποίησης αλλά διόλου ευκαταφρόνητη από άποψη λειτουργίας και χρησιμότητας μονάδα Testing του Ελεγκτή. Παρατίθεται λοιπόν παρακάτω το Interface του Ελεγκτή καθώς και αυτό της μνήμης που λήφθηκε από τη σελίδα της Micron ενώ η διαχείριση του TestBench αναφέρεται στο **Κεφάλαιο** 4.

3.2.1.1 Interface **Του Ελεγκτή**

Σε αυτή την παράγραφο παρατίθεται το Interface του Ελεγκτή που υλοποιήθηκε και είναι αυτό το οποίο λαμβάνει τα σήματα που αποστέλλονται από το TestBench. Στη συνέχεια μέσα στον Ελεγκτή γίνεται διαχείριση των σημάτων αυτών και μέσα από τις κατάλληλες διεργασίες αποστέλλονται προς τη μνήμη τα σήματα που πρέπει ανάλογα με τις εισαχθείσες εντολές :

```
ENTITY Interface IS
    PORT
    (
            CLK         : IN STD_LOGIC;
            CLK90       : IN STD_LOGIC;
            CLK_N       : IN STD_LOGIC;

            Start       : IN STD_LOGIC;
            RST         : IN STD_LOGIC;
            CasL        : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
            BrstType    : IN STD_LOGIC;
            Rd_En       : IN STD_LOGIC;
            Wr_En       : IN STD_LOGIC;
            BrstLngt    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);

            Address     : IN STD_LOGIC_VECTOR (23 DOWNTO 0);
            DataIn      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);

            InitDone    : OUT BOOLEAN;
            rDone       : OUT BOOLEAN;
            wDone       : OUT BOOLEAN;
            RefrBusy    : OUT STD_LOGIC;

            DataOut     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END Interface;
```

Από το παραπάνω Interface μπορούν εύκολα να περιγραφούν κάποια από τα βασικά χαρακτηριστικά του Ελεγκτή. Αρχικά καθίσταται φανερή η χρήση τριών ρολογιών. Τα δύο από αυτά (CLK και CLK_N) όπως έχει αναφερθεί και προηγουμένως είναι και τα ρολόγια λειτουργίας της μνήμης ενώ το τρίτο (CLK90) είναι ένα ρολόι μετατοπισμένο κατά 90^{o} του CLK και κρίθηκε απαραίτητη η χρήση του για την επίτευξη του συγχρονισμού των σημάτων όπως αναφέρεται και στις παραγράφους που ακολουθούν.

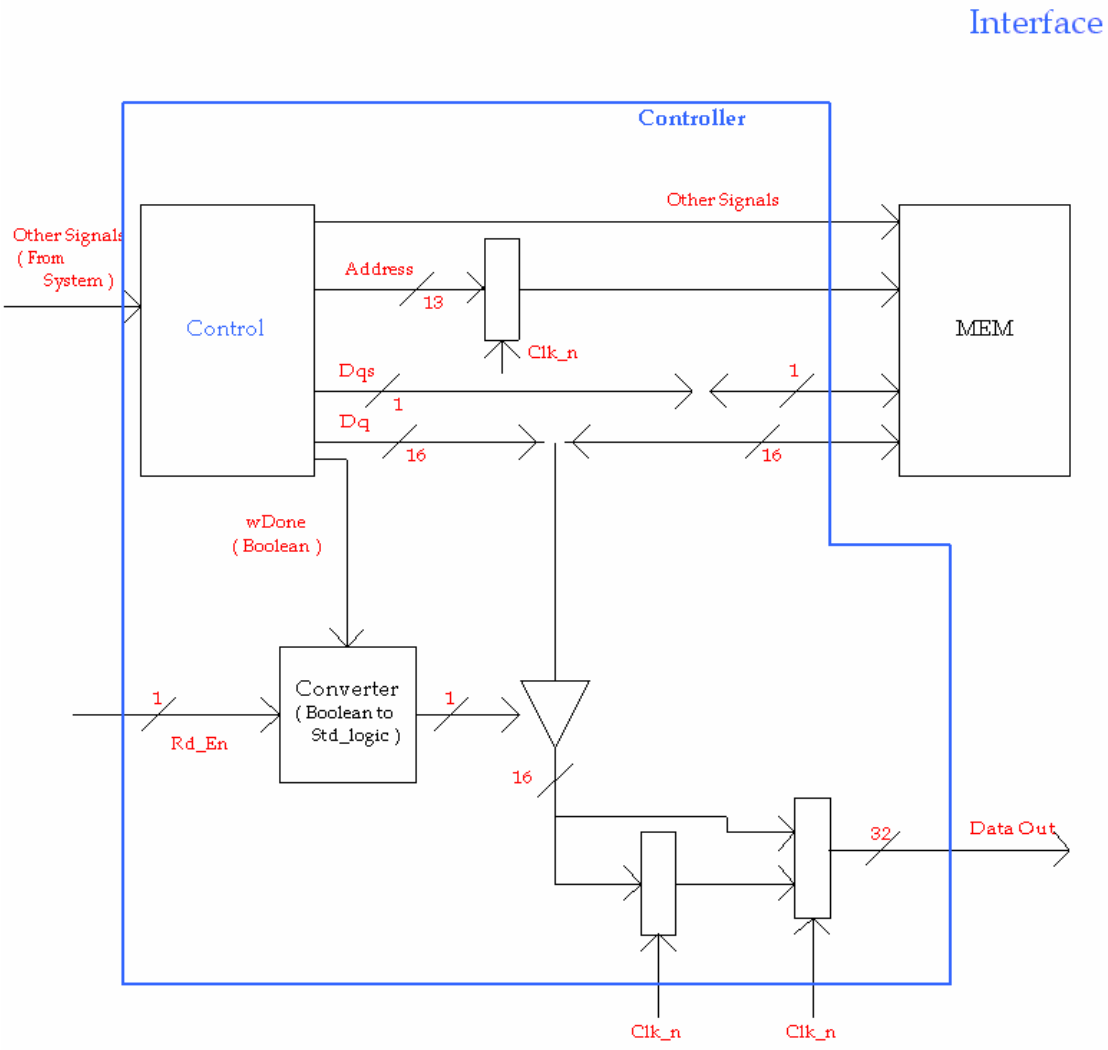
Στη συνέχεια διακρίνονται τα σήματα Start και CasL (CasLatency) BrstType (BurstType) . Το σήμα Start ορίζει την εκκίνηση της αρχικοποίησης της μνήμης ενώ τα σήματα CasL και BrstType προγραμματίζουν την τιμή με την οποία θα λειτουργούν οι αντίστοιχες παράμετροι, που αναφέρθηκαν στην **Παράγραφο** 2.4.3, μετά την αρχικοποίηση της μνήμης. Και τα τρία αυτά σήματα λαμβάνουν τιμή με την εισαγωγή της εντολής Initialize από το TestBench.

Ακολουθούν τα σήματα Rd_En, Wr_En, BrstLngt, Address και DataIn. Αυτά τα σήματα λαμβάνουν τις τιμές τους με την εισαγωγή εντολής ανάγνωσης (Read) ή εγγραφής (Write) από το TestBench και δηλώνουν το είδος της εντολής, το μέγεθος του Burst της εντολής, τη διεύθυνση εκκίνησης της εκτέλεσης της εντολής και σε περίπτωση εντολής εγγραφής τα δεδομένα που επιθυμεί ο χρήστης να εγγραφούν στη μνήμη.

Έπειτα ορίζονται τέσσερα σήματα ελέγχου InitDone, rDone, wDone RefrBusy. Το πρώτο από αυτά δηλώνει την ολοκλήρωση της αρχικοποίησης της μνήμης και καθιστά, με την επιστροφή του, το σύστημα σε κατάσταση ετοιμότητας για την εκτέλεση εντολών. Τα rDone και wDone δηλώνουν την ολοκλήρωση μίας εντολής ανάγνωσης ή εγγραφής αντίστοιχα. Με την επιστροφή ενός τέτοιου σήματος δηλώνεται επίσης και η δυνατότητα εισαγωγής της επόμενης εντολής στο σύστημα. Το σήμα RefrBusy δηλώνει , όταν λαμβάνει την τιμή '1', ότι η μνήμη πραγματοποιεί Refresh και δεν επιτρέπεται εωσότου λάβει την τιμή '0' να εισαχθεί κάποια άλλη εντολή στο σύστημα γιατί δεν επιτρέπεται να διακοπεί το Refresh.

Τέλος παρατηρείται το σήμα εξόδου 32 bit των δεδομένων τα οποία επιστρέφονται στον χρήστη (αλλιώς το σήμα βρίσκεται σε High 'Z' State) μόνο σε περίπτωση εντολής ανάγνωσης από τη μνήμη.

3.2.1.2 Block Diagram **Του Ελεγκτή**



Με μία πρώτη ματιά το Block Diagram του Ελεγκτή δείχνει αρκετά απλό, αυτό όμως συμβαίνει διότι η αρκετά πολύπλοκη λογική παρουσιάζεται σαν μια Control Unit της οποίας ανάλυση θα γίνει αργότερα. Από αυτό το Block Diagram μπορούμε να κρατήσουμε τα αμφίδρομα Buses της μνήμης τόσο για το σήμα Dqs όσο και για τα δεδομένα (Dq). Ακόμη το ότι η μνήμη λειτουργεί όπως έχει προαναφερθεί με δεδομένα των 16 bit αλλά τόσο στην είσοδο του συστήματος όσο και στην έξοδο τα δεδομένα είναι των 32 bit εφόσον σε ένα κύκλο ρολογιού πραγματοποιούνται 2 μεταφορές δεδομένων. Για αυτό το λόγο και διακρίνουμε στο Block Diagram έναν τρικατάστατο οδηγητή και τους δύο καταχωρητές.

Ο τρικατάστατος οδηγητής λαμβάνει τα δεδομένα από το Bus του Dq. Λόγω του ότι αυτό το Bus είναι αμφίδρομο μεταφέρει δεδομένα είτε προς είτε από τη μνήμη. Συνεπώς μεταφέρει δεδομένα που εισέρχονται στον τρικατάστατο οδηγητή είτε σε περίπτωση ανάγνωσης είτε σε περίπτωση εγγραφής. Στην έξοδο όμως του συστήματος είναι επιθυμητό να περνάνε τα δεδομένα μόνο σε περίπτωση ανάγνωσης όπου και η μνήμη πρέπει να επιστρέψει τα δεδομένα που ζητήθηκαν από το χρήστη. Για αυτό το λόγω ο έλεγχος του τρικατάστατου οδηγητή αφορά ακριβώς αυτό το ζήτημα. Λαμβάνει το σήμα Rd_En που διατηρείται ενεργό καθ' όλη τη διάρκεια της εκτέλεσης μιας εντολής ανάγνωσης και το Boolean σήμα wDone που δηλώνει ότι η εκτέλεση μίας εντολής εγγραφής έχει ολοκληρωθεί. Μετατρέπει (Converter) το wDone σε std_logic και επιτρέπει (από το συνδυασμό των δύο εισόδων του) την είσοδο του τρικατάστατου οδηγητή να περάσει στην έξοδο του, μόνο στην περίπτωση που έχουμε ανάγνωση και συνεπώς επιστροφή δεδομένων από τη μνήμη στο χρήστη.

Τόσο η είσοδος όσο και η έξοδος του τρικατάστατου οδηγητή είναι 16 bit. Η μνήμη εξάγει τα 16 πρώτα bit δεδομένων στο πρώτο μισό του κύκλου (θετική ακμή ρολογιού) και τα 16 δεύτερα bit στο δεύτερο μισό του κύκλου (αρνητική ακμή ρολογιού). Για αυτό το λόγο χρησιμοποιείται ένας 16 bit καταχωρητής ο οποίος κρατάει τα πρώτα 16 bit πληροφορίας για μισό κύκλο και τα περνάει στην έξοδό του στο δεύτερο μισό του κύκλου. Η έξοδος αυτού του καταχωρητή εισάγεται, σαν δεύτερη είσοδος, σε έναν δεύτερο καταχωρητή ο οποίος δέχεται σαν πρώτη είσοδο τα δεδομένα που τη συγκεκριμένη στιγμή έχουν περάσει στην έξοδο του τρικατάστατου οδηγητή. Στη συνέχεια στην επόμενη αρνητική ακμή του ρολογιού περνάνε στην έξοδο αυτού του καταχωρητή τα επιθυμητά 32 bit δεδομένων. Με αυτόν τον τρόπο δηλαδή συγχρονίζονται τα δύο 16 bit δεδομένα έτσι ώστε να φθάσουν στον χρήστη σαν μία λέξη των 32 bit.

Τέλος υπάρχει ακόμη ένας καταχωρητής των 13 bit ο οποίος χρησιμοποιείται έτσι ώστε να καθυστερήσει την ανάθεση της διεύθυνσης (που είναι επιθυμητό να προσπελαστεί) στη μνήμη κατά μισό κύκλο χάριν συγχρονισμού των σημάτων και ευκολότερης παρατήρησης της λειτουργίας του CL (θα μπορούσε να παραληφθεί).

3.3 F(inite) S(tate) M(achine) - (FSM)

Για την υλοποίηση του Ελεγκτή, το πρώτο βήμα ήταν η δημιουργία μίας FSM (Finite State Machine) η οποία αποτελεί και την βάση του όλου συστήματος. Για την σωστή λειτουργία της η μνήμη απαιτεί πρώτα από όλα σωστή αρχικοποίηση. Αρχικά λοιπόν ο ελεγκτής βρίσκεται εν αναμονή μίας εντολής Initialize από το χρήστη. Όταν αυτή δοθεί ακολουθεί μία σειρά από καταστάσεις και ακολουθίες εντολών οι οποίες είναι απαραίτητες για τη σωστή αρχικοποίηση, και συνεπώς στη συνέχεια λειτουργία, της μνήμης. Αφού παρέλθει ο χρόνος και οι εντολές που απαιτούνται για την αρχικοποίηση επιστρέφεται ένα σήμα InitDone το οποίο και δηλώνει ότι η μνήμη είναι έτοιμη για χρήση και η FSM περνάει σε μία κατάσταση αναμονής (Wait State) εντολής (Read ή Write).

3.3.1 **Αρχικοποίηση Της Μνήμης**

Για την τροφοδοσία της μνήμης πρέπει αρχικά να δοθούν ταυτόχρονα τα V_{DD} και V_{DDQ} (DQ Power Supply) και στη συνέχεια (ειδάλλως μπορεί να προκληθεί μόνιμη βλάβη της συσκευής) το Vref (Τάση Αναφοράς).

Εκτός του σήματος CKE κανένα άλλο δεν αναγνωρίζεται ως Valid πριν η συσκευή τροφοδοτηθεί με το Vref. Στη συνέχεια πρέπει να βεβαιωθούμε ότι η τιμή του CKE παραμένει LOW κατά τη διάρκεια της τροφοδοσίας έτσι ώστε να βεβαιωθούμε ότι τα σήματα DQ και DQS θα παραμείνουν στην κατάσταση 'Z' όπου και θα παραμείνουν μέχρι να έρθει μία εντολή ανάγνωσης.

Αφού τροφοδοτηθεί σωστά η μνήμη και τα σήματα τροφοδοσίας σταθεροποιηθούν (όπως επίσης και το ρολόι), πρέπει να παρέλθει ένα χρονικό διάστημα 200 μs προτού εισαχθεί μία εκτελέσιμη εντολή. Από άποψη υλοποίησης αυτό πραγματοποιήθηκε με χρήση ενός μετρητή ο οποίος αυξανόταν κατά 1 σε κάθε κύκλο του ρολογιού. Ο μετρητής αυτός βρισκότανε στην πρώτη κατάσταση της FSM μέσα σε μία NOP εντολή. Εφόσον η διάρκεια ενός κύκλου του ρολογιού μίας τέτοιας μνήμης είναι 7.5 ns χρειάσθηκαν 200000 / 7.5 = 26666.66 => 26667 κύκλοι ρολογιού έτσι ώστε να παρέλθει ο απαιτούμενος χρόνος αδράνειας των 200 μs.

Εφόσον το χρονικό αυτό διάστημα περάσει, πρέπει να εισαχθεί μια εντολή NOP στην οποία θα ενεργοποιηθεί (HIGH) το CKE και να ακολουθήσει μία PRECHARGE ALL εντολή. Στη συνέχεια πρέπει να εισαχθεί μία LOAD MODE REGISTER εντολή για το EXTENDED MODE REGISTER (όπου BA1 LOW και BA0 HIGH) για την ενεργοποίηση του DLL ακολουθούμενη από άλλη μία (όπου BA1 και BA0 LOW) για να γίνει RESET το DLL και να προγραμματιστούν οι παράμετροι της μνήμης που έχουν αυτή τη δυνατότητα. Ο Ελεγκτής είναι κατασκευασμένος έτσι ώστε ο χρήστης να έχει τη δυνατότητα να προγραμματίσει (με την εντολή της αρχικοποίησης) το Cas Latency (CL 2, 2.5 ή 3), καθώς και το είδος του Burst (Sequential ή Interleaved) με τα οποία θα λειτουργεί η μνήμη. Τέλος είναι κατασκευασμένος / προγραμματισμένος έτσι ώστε το BurstLength της μνήμης να είναι 2.

Στη συνέχεια πρέπει να περάσουν 200 κύκλοι ρολογιού (υπολογίζεται επίσης με τη χρήση του μετρητή) προτού εισαχθεί μία εκτελέσιμη εντολή (από τη στιγμή που έγινε RESET το DLL). Στο ενδιάμεσο χρονικό διάστημα εισάγεται μία PRECHARGE ALL εντολή έτσι ώστε όλα τα Banks να εισέλθουν σε κατάσταση αναμονής / αδράνειας. Εφόσον αυτό ολοκληρωθεί πρέπει να εισαχθούν δύο εντολές AUTO REFRESH και μία LOAD MODE REGISTER με το bit όμως που κάνει RESET στο DLL ανενεργό (αυτό γιατί είναι απαραίτητο να προγραμματιστούν οι παράμετροι της μνήμης χωρίς το DLL να γίνει RESET). Αν ακολουθηθούν ορθά οι παραπάνω περιορισμοί, η μνήμη πλέον είναι έτοιμη για την ορθή λειτουργία της.

Κάθε μία από τις παραπάνω εντολές αποτελεί και μία κατάσταση της FSM η οποία αποστέλλει τα αντίστοιχα σήματα στη μνήμη. Όταν εισαχθούν και οι τελευταίες εντολές και παρέλθει και το διάστημα των 200 κύκλων, επιστρέφεται ένα σήμα που δηλώνει ότι η αρχικοποίηση πραγματοποιήθηκε (InitDone = True) και η επόμενη κατάσταση της FSM είναι μία κατάσταση αναμονής (Wait State) που περιμένει να εισαχθεί μία εντολή ανάγνωσης ή εντολής από το χρήστη.

3.3.2 Wait State

Εφόσον το σύστημα έχει εισέλθει σε αυτή την κατάσταση περιμένει μία εντολή ανάγνωσης η εγγραφής από το χρήστη. Όταν εισέρχεται μία τέτοια εντολή στο σύστημα ενεργοποιείται το κατάλληλο σήμα έτσι ώστε να καθορισθεί η επόμενη κατάσταση της FSM. Στην περίπτωση εντολής εγγραφής δίνεται Wr_En = 1 (διατηρώντας Rd_En = 0) ενώ σε περίπτωση εντολής ανάγνωσης δίνεται Rd_En = 1 (διατηρώντας Wr_En = 0).

Μία απλή περίπτωση θα ήταν ακολουθώντας το αντίστοιχο σήμα η FSM να μπαίνει σε μία καινούρια κατάσταση (Εντολή Active) στην οποία θα ενεργοποιούσε (Activation) την επιθυμητή σειρά σε ένα συγκεκριμένο κάθε φορά Bank για να καταστεί δυνατή η πρόσβαση σε αυτό για εγγραφή ή ανάγνωση αντίστοιχα και στη συνέχεια να ακολουθούσε η εκτέλεση της ίδιας της εντολής. Χάριν βελτιστοποίησης της απόδοσης τα πράγματα δεν είναι τόσο απλά.

Γίνεται χρήση ενός σήματος ελέγχου και ενός μετρητή ακόμη. Εφόσον εκτελεστεί μία εντολή το σήμα ελέγχου διατηρεί πληροφορία για το Bank στο οποία υπήρξε πρόσβαση από την εντολή αυτή ενώ ο μετρητής μετράει τους κύκλους που παρεμβάλλονται από τη στιγμή που ολοκληρώθηκε η εκτέλεση της τελευταίας εντολής, οπότε και εισάχθηκε το Auto Precharge. Για την σωστή ολοκλήρωση του Auto Precharge χρειάζεται να περάσουν τέσσερις κύκλοι ρολογιού. Ο μετρητής αυξάνει την τιμή του κατά ένα σε κάθε κύκλο ρολογιού που περνάει. Με αυτό τον τρόπο διαχωρίζονται οι παρακάτω περιπτώσεις :

α) Κάθε φορά που το σύστημα εισέρχεται σε αυτή την κατάσταση, ο Ελεγκτής κρίνει αν έχει φθάσει η χρονική στιγμή που είναι απαραίτητη η διεξαγωγή Refresh της μνήμης. Αν απαιτείται Refresh τότε το σύστημα εισάγεται στην αντίστοιχη κατάσταση για τη διεξαγωγή του, ειδάλλως αναμένει μία εντολή ανάγνωσης ή εγγραφής.

β) Αν εισέλθει κάποια εντολή (είτε ανάγνωσης, είτε εγγραφής) γίνεται έλεγχος αν πρόκειται για πρόσβαση στο ίδιο ή σε διαφορετικό Bank από την τελευταία εντολή. Αν πρόκειται για πρόσβαση σε διαφορετικό Bank τότε η επόμενη κατάσταση είναι αυτή που ξεκινάει τη διαδικασία εκτέλεσης της εντολής με την ενεργοποίηση του προς πρόσβαση Bank.

γ) Αν η εντολή που εισέρχεται απαιτεί πρόσβαση στο Bank που χρησιμοποιήθηκε και από την τελευταία εντολή, γίνεται έλεγχος για το αν έχει παρέλθει ο χρόνος που απαιτείται για το Precharge του Bank (4 κύκλοι του ρολογιού). Αν ο μετρητής έχει τιμή που δηλώνει ότι ο χρόνος που έχει παρέλθει από το πέρας της προηγούμενης εντολής έχει καλύψει τις ανάγκες του Precharge τότε ξεκινάει αμέσως η εκτέλεση της εισαχθείσας εντολής.

δ) Σε διαφορετική περίπτωση η επόμενη κατάσταση είναι απλά μία εντολή NOP κατά την οποία ο μετρητής σε κάθε κύκλο του ρολογιού αυξάνει κατά 1. Αυτό πραγματοποιείται εωσότου καλυφθεί ο χρόνος περάτωσης του Auto Precharge της τελευταίας εντολής έτσι ώστε να μπορέσει να εισαχθεί κανονικά η επόμενη. Όπως είναι φανερό η κατάσταση αυτή μπορεί να διαρκέσει για έναν, δύο ή τρεις κύκλους ρολογιού προτού καλυφθεί ο απαιτούμενος χρόνος και μπορέσει να ξεκινήσει η εκτέλεση της νέας εντολής.

(Για καλύτερη κατανόηση των παραπάνω βλέπε **Κεφάλαιο** 3.3.7)

3.3.3 Active State

Όπως έχει προαναφερθεί, σε περίπτωση εισαγωγής μίας νέας εντολής ανάγνωσης ή εγγραφής κρίνεται απαραίτητη η ενεργοποίηση του Bank για την πρόσβαση σε αυτό οπότε και η επόμενη της Wait State κατάσταση είναι μία εντολή ενεργοποίησης της σειράς του Bank που πρόκειται να χρησιμοποιηθεί. Η ενεργοποίηση αυτή απαιτεί τρεις κύκλους ρολογιού. Στη συνέχεια η FSM οδηγεί στην αντίστοιχη διεργασία.

3.3.4 Read

Σε περίπτωση που το σύστημα έχει περάσει στην κατάσταση εκτέλεσης μίας απλής εντολής ανάγνωσης αποστέλλεται αρχικά στη μνήμη το Bank και η στήλη από την οποία επιθυμεί ο χρήστης να επιστραφούν τα δεδομένα και στη συνέχεια παρέρχεται ο χρόνος που χρειάζεται μέχρι να επιστραφούν αυτά. Τότε αποστέλλεται από τον ελεγκτή ένα σήμα (rDone = True) που δηλώνει ότι η ανάγνωση ολοκληρώθηκε με επιτυχία. Χάριν βελτιστοποίησης όμως η μνήμη υποστηρίζει τη δυνατότητα εγγραφών με μεγαλύτερο του 1 Burst. Έτσι όταν εισάγεται από το χρήστη μία εντολή ανάγνωσης, εκτός από τη διεύθυνση από την οποία επιθυμεί να αναγνώσει τα δεδομένα εισάγει και την τιμή μίας ακόμη παραμέτρου (BurstSize). Με τη νέα αυτή παράμετρο, δηλώνει πόσες συνεχόμενες διευθύνσεις επιθυμεί να προσπελασθούν και να του επιστραφούν τα δεδομένα τους.

Λόγω του προγραμματισμού του BurstLength σε 2 η διεύθυνση (στηλών) σε ένα Burst αυξάνεται κατά 2 κάθε φορά. Κάθε στήλη έχει 16 bit δεδομένα και όταν ζητείται ανάγνωση (σειριακή πάντα) από μία διεύθυνση επιστρέφονται τα δεδομένα δύο συνεχόμενων στηλών (Sequential Burst : αν ζητηθεί η ανάγνωση των δεδομένων από τη διεύθυνση 0, θα επιστραφούν τα δεδομένα των διευθύνσεων 0 και 1 ενώ αν ζητηθεί η ανάγνωση των δεδομένων από τη διεύθυνση 1, θα επιστραφούν τα δεδομένα των διευθύνσεων 1 και 0). Συνεπώς όταν ο χρήστης εισάγει ένα BurstSize που επιθυμεί, η ανάγνωση θα εκκινήσει από τη διεύθυνση που έχει ορισθεί στην εντολή και θα συνεχίζει κάνοντας τόσες αναγνώσεις όσες ορίζει το BurstSize.

Για την υλοποίηση αυτού, έπειτα από κάθε ολοκληρωμένη ανάγνωση, μειώνεται το BurstSize κατά 1 και αυξάνεται η διεύθυνση ανάγνωσης κατά 2. Αυτό συνεχίζεται μέχρι να ολοκληρωθεί ο αριθμός των αναγνώσεων που ορίστηκε από το χρήστη οπότε και επιστρέφεται το σήμα που ορίζει την ολοκλήρωση της ανάγνωσης (rDone = True). Όπως είναι εμφανές, ο μέγιστος αριθμός του Burst που μπορεί να δοθεί είναι 256 αφού ο αριθμός των στηλών μίας σειράς (η οποία είναι Activated) είναι 512 και η διεύθυνση αυξάνει κάθε φορά κατά 2. Με αυτό τον τρόπο επιτυγχάνεται ανάγνωση πολλών δεδομένων από τη μνήμη χωρίς να παρεμβάλλεται η παραμικρή χρονική καθυστέρηση.

3.3.5 Write

Η λειτουργία της εγγραφής είναι πανομοιότυπη με αυτή της ανάγνωσης μόνο που ο χρήστης εισάγει και τα δεδομένα που επιθυμεί να εγγραφούν στη μνήμη στην διεύθυνση που ορίζει κάθε φορά.

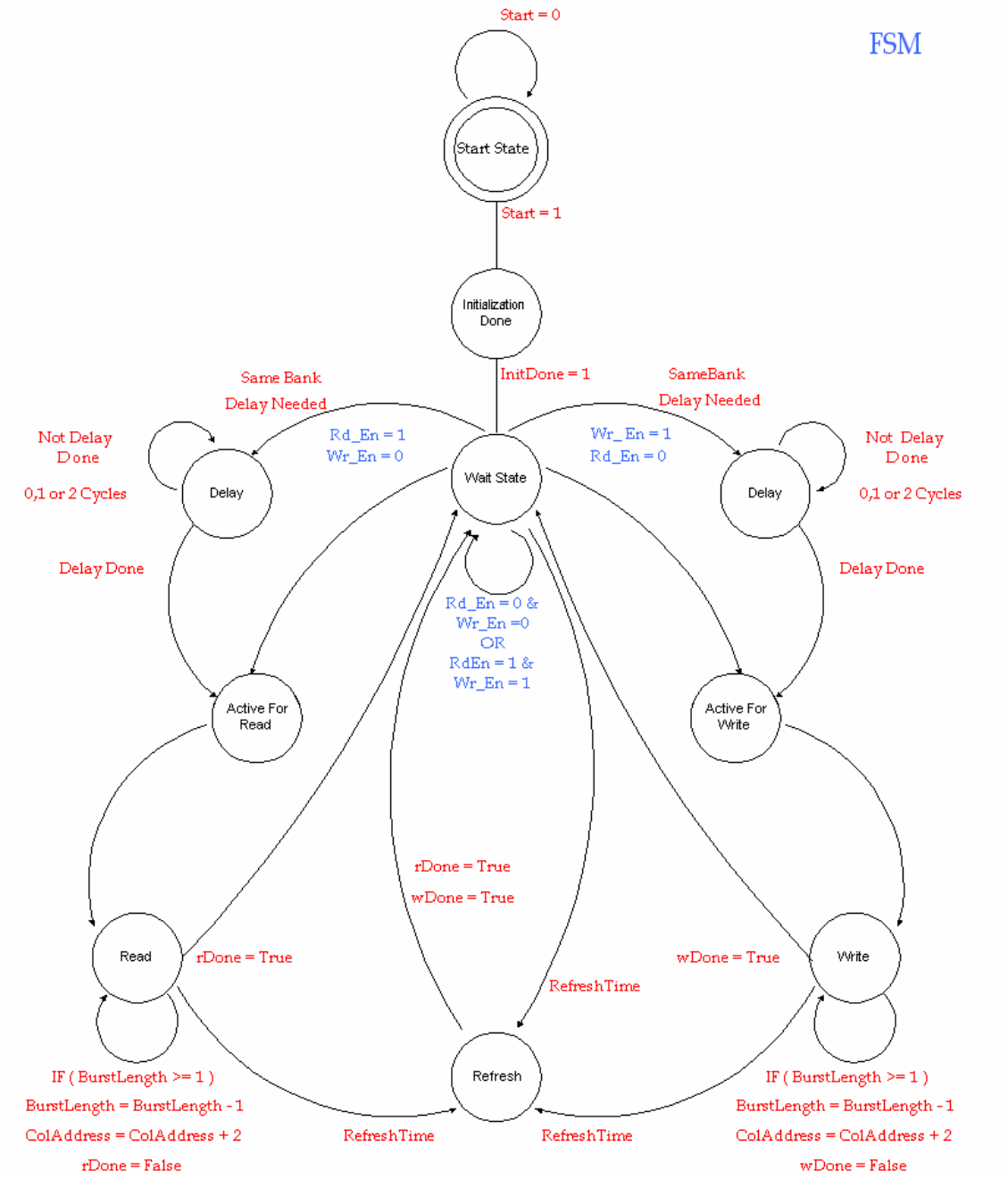
3.3.6 Refresh

Όπως έχει προαναφερθεί, σε μία DRAM είναι απαραίτητη η εκτέλεση Refresh ανά τακτά χρονικά διαστήματα. Στη συγκεκριμένη μνήμη υπολογίζεται ότι ο μέγιστος χρόνος που μπορεί να παρεμβληθεί ανάμεσα σε δύο Refresh είναι 7.8125 μs, χωρίς όμως ένα Refresh να μπορεί να διακόψει μία διεργασία που βρίσκεται ήδη σε εξέλιξη.

Για την υλοποίηση του Auto Refresh της μνήμης χρησιμοποιήθηκε ένας μετρητής ο οποίος από τη στιγμή που ολοκληρώνεται η αρχικοποίηση της μνήμης αρχίζει να μετράει με κάθε κύκλο του ρολογιού. Λόγω του ότι επιτρέπεται μέγιστο BurstSize 256 και η μία εντολή Refresh δεν πρέπει να διακόψει μία εντολή που ήδη βρίσκεται σε εξέλιξη, θα πρέπει να ληφθεί υπ' όψιν η χειρότερη περίπτωση (από άποψη καθυστέρησης). Σε αυτή την περίπτωση πρόκειται για εγγραφή που έχει BurstSize 255. Για να ολοκληρωθεί αυτή η εντολή χρειάζονται 261 κύκλοι ρολογιού => 261 x 7.5 = 1957.5 ns. Συνεπώς ο έλεγχος για Auto Refresh θα πρέπει να γίνεται στα 7.8125 – 1.964 = 5.8485 μs => 779 κύκλοι ρολογιού. Έτσι ακόμη και στη χειρότερη περίπτωση δεν υπάρχει ενδεχόμενο να μην πραγματοποιηθεί το απαιτούμενο Refresh πριν παρέλθει το επιτρεπτό χρονικό διάστημα.

Ο έλεγχος για Refresh γίνεται στο τέλος κάθε εντολής εγγραφής ή ανάγνωσης αλλά και στην κατάσταση αναμονής. Όπως είναι λογικό, ακριβώς επειδή το Refresh θα πραγματοποιείται συνήθως σε χρονικό διάστημα μικρότερο του μέγιστου δυνατού (και διαρκεί 9 κύκλους του ρολογιού) θα υπεισέρχεται κάποια καθυστέρηση στο σύστημα, αλλά το κέρδος από τη δυνατότητα υποστήριξης του Burst Size είναι σημαντικότερο από αυτή την απώλεια.

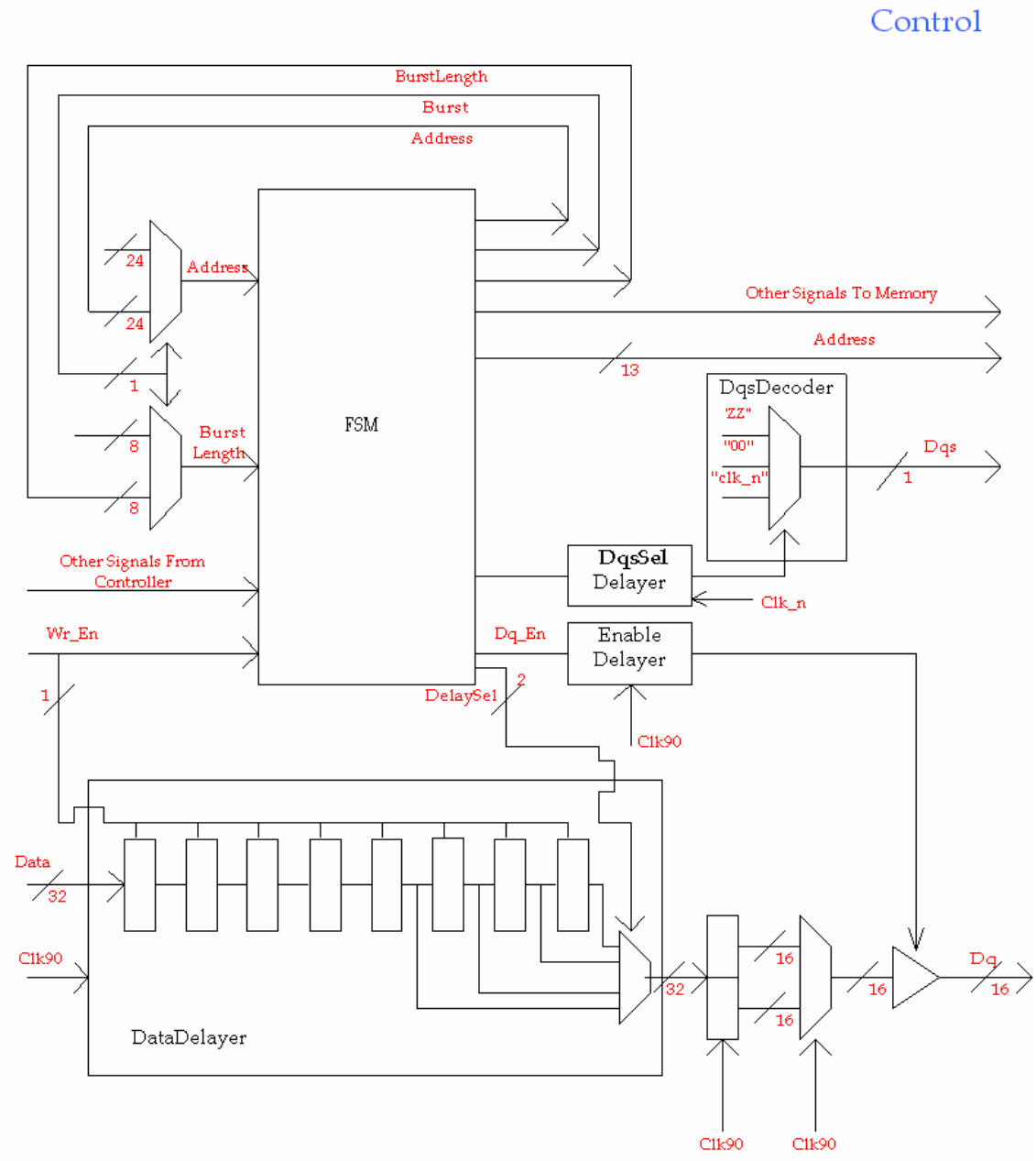
3.3.7 **Διάγραμμα Καταστάσεων**



3.4 Control Unit

Η μονάδα αυτή είναι αυτή που κατέχει την πιο πολύπλοκη λογική με βάση την οποία πραγματοποιούνται οι βελτιστοποιήσεις και οι συγχρονισμοί των σημάτων. Ως βάση αυτής της μονάδας χρησιμοποιείται η FSM που περιγράφηκε στην **Παράγραφο** 3.3.

3.4.1 Block Diagram **Της** Control Unit

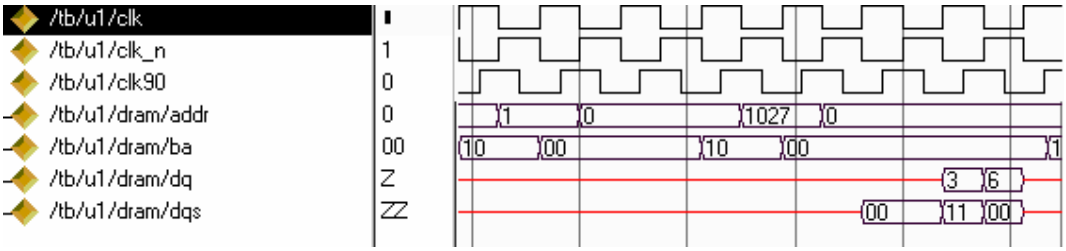


3.4.2 **Περιγραφή Των Βασικών Μονάδων Του** Block Diagram

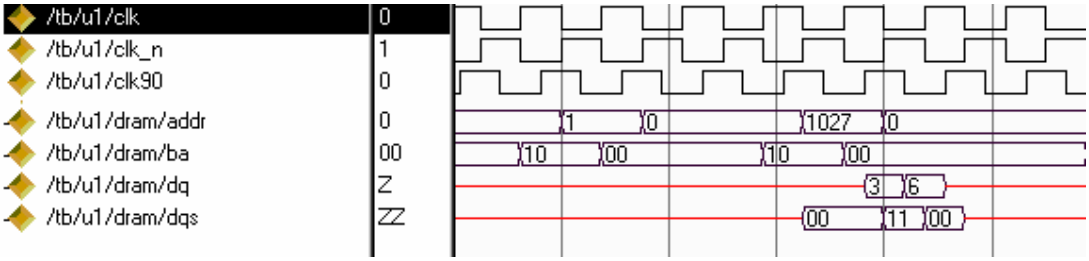
3.4.2.1 Dqs SelDelayer

Όπως έχει ήδη προαναφερθεί, το σήμα (Bus) Dqs της μνήμης είναι αμφίδρομο, επομένως έχει τη δυνατότητα να μεταφέρει πληροφορία σε αυτή από τον χρήστη (μέσω του Ελεγκτή) αλλά και να επιστρέψει πληροφορία στον χρήστη. Σε περίπτωση ανάγνωσης η μνήμη από μόνη της είναι αυτή που στέλνει τα δεδομένα και τα συγχρονίζει κατάλληλα με το σήμα Dqs ενώ σε περίπτωση εγγραφής ο Ελεγκτής είναι αυτός που πρέπει να συγχρονίσει το σήμα Dqs με την πληροφορία για να φθάσει αυτή Valid στη μνήμη. Σε αντίθεση με την περίπτωση ανάγνωσης όπου η μνήμη συγχρονίζει το σήμα DQS απόλυτα με τα δεδομένα, στην περίπτωση εγγραφής πρέπει η αλλαγή της τιμής του σήματος να γίνει στο κέντρο των δεδομένων (Βλέπε **Σχήμα** 3.2 και 3.3).

Σχήμα 3.2 - **Περίπτωση μίας απλής ανάγνωσης από τη μνήμη**



Σχήμα 3.3 - **Περίπτωση μίας απλής εγγραφής προς τη μνήμη**



Για να επιτευχθεί ο συγχρονισμός αποστολής δεδομένων και σήματος Dqs σε περίπτωση εγγραφής χρησιμοποιήθηκε η μονάδα Dqs Sel Delayer. Έτσι όταν η FSM σε κάθε κατάσταση αποστέλλει την τιμή του Dqs Sel (το οποίο στη μονάδα Dqs Decoder καθορίζει κάθε φορά την τιμή του Dqs που θα περνάει στην έξοδο) η μονάδα αυτή απλά καθυστερεί το σήμα κατά μισό κύκλο. Αυτό γιατί όπως γίνεται φανερό και από το παραπάνω σχήμα, η τιμή του Dqs πρέπει να αλλάζει στα μισά της μεταφοράς κάθε τιμής της πληροφορίας προκειμένου αυτά να φθάνουν Valid στη μνήμη.

3.4.2.2 DqsDecoder

Αυτή η μονάδα όπως είναι εμφανές και από το Block Diagram αποτελείται από ένα πολυπλέκτη τριών εισόδων. Σαν έλεγχος σε αυτό τον πολυπλέκτη, ο οποίος καθορίζει και το ποια από όλες τις εισόδους θα περάσει στην έξοδο, είναι όπως έχει προαναφερθεί η έξοδος του Dqs Sel Delayer. Σαν εισόδους λαμβάνει την κατάσταση ‘’ZZ’’, την κατάσταση ‘’00’’ και σαν τελευταία είσοδο την τιμή του ρολογιού Clk_n.

Η πρώτη είσοδος (‘’ZZ’’) μας ενδιαφέρει να περνάει στην έξοδο του πολυπλέκτη όταν στον Ελεγκτή δεν πραγματοποιείται καμία διεργασία ή σε περίπτωση ανάγνωσης οπότε και στο Bus του Dqs θέλουμε να ‘’κυριαρχήσει’’ η τιμή του Dqs που αποστέλλει η μνήμη χωρίς να επηρεάζεται από κάποια άλλη τιμή από τον Ελεγκτή.

Η δεύτερη είσοδος (‘’00’’) μας ενδιαφέρει να περνάει στην έξοδο μόνο σε περίπτωση εκκίνησης μίας εντολής εγγραφής. Αυτό διότι όπως είναι φανερό και από το **Σχήμα** 3.3, σε περίπτωση εγγραφής, πριν ακόμη περάσουν τα δεδομένα προς τη μνήμη, το σήμα Dqs λαμβάνει την τιμή ‘’00’’ για ένα κύκλο του ρολογιού.

Στη συνέχεια μας ενδιαφέρει να περνάει στην έξοδο η τιμή του ανάστροφου ρολογιού (για κάθε bit του Dqs). Σε αυτή την περίπτωση η τιμή που περνάει στην έξοδο του Dqs Decoder αλλάζει μαζί με την τιμή του ρολογιού κάθε μισό κύκλο εναλλάσσοντας τις τιμές ‘’00’’ και ‘’11’’. Αυτό σε συνδυασμό με την καθυστέρηση του σήματος ελέγχου από τη μονάδα Dqs Sel Delayer οδηγούν στον απαιτούμενο συγχρονισμό των δεδομένων με το σήμα Dqs. Με αυτό τον τρόπο, σε περιπτώσεις εντολών εγγραφής με μεγάλο Burst, η εναλλαγή της τιμής του σήματος καλύπτει την εκτέλεσή Burst του απλά ακολουθώντας την τιμή του ρολογιού.

3.4.2.3 Enable Delayer

Εκτός από το Dqs ο Ελεγκτής εξάγει ακόμη ένα σήμα το οποίο απαιτεί κάποια χρονική καθυστέρηση. Αυτό το σήμα ονομάζεται Dq_En. Το σήμα αυτό αποτελεί τον έλεγχο του τρικατάστατου οδηγητή που περνάει στην έξοδό του τα δεδομένα προς εγγραφή (βλέπε **Παράγραφο** 3.4.1). Όπως το Dqs Bus, έτσι και το Dq Bus της μνήμης (Bus μεταφοράς των δεδομένων) είναι αμφίδρομο. Αυτό σημαίνει ότι σε περίπτωση εγγραφής τα δεδομένα που θα μεταφέρει θα κατευθύνονται από τον Ελεγκτή προς τη μνήμη, ενώ σε περίπτωση ανάγνωσης από τη μνήμη προς τον Ελεγκτή. Για να επιτευχθεί αυτός ο διαχωρισμός είναι και απαραίτητη η χρήση του σήματος Dq_En το οποίο χρειάζεται την χρονική καθυστέρηση που αναφέρθηκε.

Σε περίπτωση που η μνήμη δεν πραγματοποιεί κάποια λειτουργία το Dq_En λαμβάνει την τιμή '0' έτσι ώστε στην έξοδο του τρακατάστατου οδηγητή να μην περνάνε ''σκουπίδια''. Επίσης σε περίπτωση ανάγνωσης από τη μνήμη το σήμα λαμβάνει την ίδια τιμή έτσι ώστε η έξοδος του τρικατάστατου οδηγητή να είναι ''Z'' και στο Bus να ''κυριαρχεί'' η πληροφορία (δεδομένα) που αποστέλλει η μνήμη χωρίς να επηρεάζεται από κάποια άλλη τιμή από τον Ελεγκτή.

Σε περίπτωση εγγραφής όπου και η μνήμη εξάγει στο σήμα Dq την τιμή ''Z'' η τιμή του Dq_En γίνεται ''1'' έτσι ώστε να περνάνε στην έξοδο του τρικατάστατου οδηγητή τα δεδομένα που φθάνουν στην είσοδό του. Έτσι τα δεδομένα αυτά καταλαμβάνουν το Bus και φθάνουν στη μνήμη.

Όπως όμως φαίνεται από το **Σχήμα** 3.3 ούτε τα δεδομένα είναι απόλυτα συγχρονισμένα με το κανονικό ρολόι. Έτσι πρέπει να καθυστερήσουν κατά 1/4 του ρολογιού για να συγχρονιστούν απόλυτα με το σήμα Dqs. Για αυτό το λόγο και χρησιμοποιήθηκε ένα ακόμη ρολόι μετατοπισμένο κατά 1/4 (ή αλλιώς κατά 90^{o}) από το κανονικό. Αυτό το ρολόι ονομάστηκε Clk90 και με βάση αυτό πρέπει να καθυστερήσει το σήμα Dq_En για να επιτευχθεί η σωστή καθυστέρηση. Για αυτό το λόγο στη μονάδα Enable Delayer εισάγεται απλά το σήμα Dq_En και καθυστερεί να περάσει στην έξοδο κατά 1/4 του ρολογιού.

3.4.2.4 Data Delayer

Η τελευταία ξεχωριστή μονάδα που διακρίνεται στο Block Diagram της **Παραγράφου** 3.4.1 είναι αυτή που είναι και υπεύθυνη για την καθυστέρηση των δεδομένων. Όπως διακρίνεται και από το Block Diagram τα δεδομένα που ο χρήστης επιθυμεί να εγγράψει στη μνήμη δεν περνάνε καθόλου μέσα από την FSM αλλά ακολουθούν μία δική τους διαδρομή.

Όταν εισάγεται μία εντολή εγγραφής από το χρήστη, καθορίζονται από την ίδια την εντολή τα δεδομένα που πρόκειται να γραφτούν στη μνήμη και η συγκεκριμένη θέση στην οποία πρόκειται να γίνει η εγγραφή. Η επεξεργασία όμως και εκτέλεση της εντολής από τη μνήμη δεν είναι το ίδιο απλή. Είναι απαραίτητη πρώτα η ενεργοποίηση (Activation) της σειράς του Bank στην οποία πρόκειται να γίνει η εγγραφή και πρέπει να παρέλθει ο χρόνος που απαιτείται μέχρι αυτή να ολοκληρωθεί σωστά (3 κύκλοι του ρολογιού). Στη συνέχεια και εφόσον έχει αρχίσει η ίδια η διαδικασία εγγραφής ο πρώτος κύκλος του ρολογιού χρησιμοποιείται για να ορίσει το Bank και τη στήλη στην οποία πρόκειται να γραφτούν τα δεδομένα και στον επόμενο κύκλο αυτά περνάνε προς τη μνήμη. Οι παραπάνω καθυστερήσεις σε συνδυασμό με έναν κύκλο καθυστέρησης που εισάγει ο Ελεγκτής μέσω της FSM ώστε να περάσει από την Κατάσταση Αναμονής στο Activation προσθέτουν αρχικά συνολική χρονική καθυστέρηση 5 κύκλων του ρολογιού.

Για να μην χαθούν τα δεδομένα που είναι προς εγγραφή αλλά και για να συγχρονιστούν με όλα τα υπόλοιπα σήματα που αποστέλλονται από την FSM σε περίπτωση εγγραφής εισάγονται μέσω αυτής της μονάδας 5 Καταχωρητές των 32 bit οι οποίοι περνάνε την είσοδο τους στην έξοδο λειτουργώντας, για τους λόγους που αναφέρθηκαν στην **Παράγραφο** 3.4.2.3, με το μετατοπισμένο κατά 90^{o} ρολόι.

Όλα όσα αναφέρθηκαν μέχρι στιγμής στην συγκεκριμένη παράγραφο αφορούν την απλούστερη περίπτωση όπου η εντολή εγγραφής που εισάγεται μπορεί να εκκινήσει αμέσως την εκτέλεσή της (περιπτώσεις **(β)** και **(γ)** της **Παραγράφου** 3.3.2).

Στην περίπτωση **(δ)** όμως της ίδιας παραγράφου που είναι και η πιο σύνθετη είδαμε ότι ο Ελεγκτής καλύπτει και την περίπτωση στην οποία δύο ‘’συνεχόμενες’’ εντολές αφορούν το ίδιο Bank. Σε αυτή την περίπτωση είναι δυνατόν να χρειάζεται να παρεμβληθεί μία καθυστέρηση το πολύ 3 κύκλων ρολογιού έτσι ώστε εφόσον οι δύο εντολές έχουν εισαχθεί χωρίς να παρεμβληθεί καθόλου κενός χρόνος να ολοκληρωθεί το Precharge του Bank από το τέλος της πρώτης εντολής πριν εκκινήσει την εκτέλεσή της η δεύτερη. Αυτή αποτελεί και την χειρότερη περίπτωση. Υπάρχουν ακόμη δύο περιπτώσεις στις οποίες η επόμενη εντολή εισάγεται με κάποια καθυστέρηση σε σχέση με την πρώτη, χωρίς όμως να έχει καλυφθεί ο χρόνος εκτέλεσης του Precharge. Για αυτό το λόγο και εισάγονται τρεις ακόμη καταχωρητές έτσι ώστε να καλυφθούν όλες οι δυνατές περιπτώσεις και να πραγματοποιηθεί στην κάθε περίπτωση η επιθυμητή καθυστέρηση των δεδομένων για το συγχρονισμό τους με τα υπόλοιπα σήματα.

Για αυτό το λόγο υπάρχει ένας πολυπλέκτης τεσσάρων 32 bit εισόδων ο οποίος ανάλογα με ένα σήμα ελέγχου που έρχεται από την FSM περνάει στην έξοδο την αντίστοιχη είσοδο.

Σε περίπτωση εγγραφών με μεγάλο Burst ένα ένα τα δεδομένα όπως εισέρχονται για εγγραφή περνάνε από καταχωρητή σε καταχωρητή χωρίς πρόβλημα και χρησιμοποιούνται με τη σειρά χωρίς κανένα ενδιάμεσο κύκλο καθυστέρησης, κάτι που προσδίδει μεγάλη ταχύτητα στο σύστημα.

3.4.3 **Υπόλοιπες Υπομονάδες Του Συστήματος**

Εκτός από τις παραπάνω βασικές μονάδες που διακρίνονται στο Block Diagram της **Παραγράφου** 3.4.1, μπορούν να διακριθούν και κάποιες άλλες μικρότερες όχι όμως λιγότερο σημαντικές.

Μία από αυτές είναι και ο Καταχωρητής μιας 32 bit εισόδου και δύο 16 bit εξόδων. Αυτός λαμβάνει τα 32 bit δεδομένα που εξάγονται από τη μονάδα καθυστέρησης των δεδομένων και τα διαχωρίζει σε δύο των 16 bit. Ο λόγος ύπαρξης αυτού του καταχωρητή είναι η δυνατότητα της μνήμης να αποθηκεύει μεν σε ένα κύκλο του ρολογιού 32 bit πληροφορίας αλλά με την μορφή δύο δεκαεξάδων (μία σε κάθε μισό του κύκλου). Έτσι λειτουργώντας πάντα με το μετατοπισμένο κατά 90^{o} ρολόι ο καταχωρητής αυτός διαχωρίζει τα δεδομένα. Στη συνέχεια αυτά εισέρχονται ως δύο διακριτές είσοδοι σε έναν πολυπλέκτη ο οποίος περνάει σε κάθε μισό του κύκλου διαφορετική είσοδο στην έξοδό του. Αυτό διότι σαν έλεγχος του πολυπλέκτη εισάγεται το Clk90. Συνεπώς στην αρνητική ακμή του ρολογιού περνάει στην έξοδο του την πρώτη είσοδο ενώ στη θετική τη δεύτερη. Επιτυγχάνεται έτσι η μεταφορά δυο δεδομένων των 16 bit σε ένα κύκλο του ρολογιού πληρώντας τις ανάγκες της DDR.

Τέλος, μία ακόμη υπομονάδα που διακρίνεται στο Block Diagram είναι αυτή που δίνει τιμή στην είσοδο της FSM τόσο για τη διεύθυνση όσο και για το Burst Size. Αυτή η υπομονάδα αποτελείται από δύο πολυπλέκτες. Έναν για τη διεύθυνση και έναν για το μέγεθος του Burst. Στην απλούστερη περίπτωση όπου εισάγεται μία καινούρια εντολή απλά περνάνε στην είσοδο της FSM η διεύθυνση και το μέγεθος του Burst που εισάγεται με την εντολή από το χρήστη.

Σε περίπτωση τώρα που το μέγεθος του Burst είναι μεγαλύτερο από ‘’1’’ πρέπει να πραγματοποιηθούν πολλές συνεχόμενες εκτελέσεις της εντολής με το δοθέν Burst Size. Για αυτό το λόγο και όπως έχει αναφερθεί και στην **Παράγραφο** 3.3 μετά την εκτέλεση της πρώτης εντολής πραγματοποιείται μείωση του μεγέθους του Burst κατά ‘’’1’’ και αύξηση της διεύθυνσης, στην οποία πρόκειται να πραγματοποιηθεί η επόμενη πρόσβαση, κατά ‘’2’’. Αυτές οι νέες τιμές εξάγονται από την FSM και εισέρχονται σαν είσοδοι στους αντίστοιχους πολυπλέκτες όπως διακρίνεται και από το Block Diagram. Ακόμη εξάγεται ένα ακόμη σήμα το οποίο ονομάζεται Burst. Αυτό το σήμα ελέγχει απλά πότε θα ολοκληρωθεί η εκτέλεση του Burst και εισέρχεται σαν είσοδος ελέγχου στους δύο πολυπλέκτες. Όσο το Burst βρίσκεται σε εξέλιξη η αντίστοιχη μεταβλητή διατηρεί την τιμή ‘’1’’. Έτσι από τους πολυπλέκτες περνάνε στην είσοδο της FSM οι δύο νέες επιθυμητές τιμές των Address και Burst Size. Όταν ολοκληρωθεί η εκτέλεση του συγκεκριμένου Burst η μεταβλητή λαμβάνει την τιμή ‘’0’’ και επομένως η FSM στην επόμενη εντολή θα λάβει τιμές για τη Διεύθυνση και το μέγεθος του Burst από το χρήστη.

Τα υπόλοιπα σήματα στα οποία δεν παρεμβάλλεται κάποια λογική, απλά αποστέλλονται στη μνήμη χωρίς να χρειασθούν περαιτέρω επεξεργασία. Αυτά αποτελούν ουσιαστικά και τη βάση με την οποία απαιτείται να συγχρονιστούν όλα τα υπόλοιπα σήματα για τον συγχρονισμό όλων.

3.5 **Απολογισμός**

Όπως έχει ήδη αναφερθεί τόσο σε προηγούμενα Κεφάλαια όσο και σε αυτό της περιγραφής της υλοποίησης του συστήματος σκοπός και στόχος του Ελεγκτή που υλοποιήθηκε δεν ήταν μόνο η βέλτιστη απόδοση αλλά και η απλότητά του. Οι δυνατότητες που παρέχει μία τέτοια μνήμη είναι αρκετές. Άλλες από αυτές πολύ σημαντικές (όσον αφορά πάντα την απόδοση του συστήματος) και άλλες λιγότερο σημαντικές. Στην υλοποίηση αυτή έγινε προσπάθεια να αξιοποιηθούν όσο το δυνατόν περισσότερες από αυτές χωρίς όμως τελικά να αξιοποιηθούν όλες. Σε αυτή την παράγραφο θα γίνει ένας τελικός απολογισμός ο οποίος θα αναφέρει κάποιες από τις βασικές δυνατότητες / ιδιότητες της μνήμης που αξιοποιήθηκαν και κάποιες που δεν αξιοποιήθηκαν.

Μία βασική ιδιότητα της μνήμης που δεν αξιοποιήθηκε ήταν η αυτή της υποστήριξης τριών διαφορετικών προγραμματιζόμενων Burst Size (2, 4 και 8). Ο λόγος που αυτό δεν πραγματοποιήθηκε είναι η ‘’υπερκάλυψη’’ αυτής της δυνατότητας με τη χρήση της υποστήριξης μεγάλου Burst Length από το χρήστη. Για να γίνει το παραπάνω πιο εύκολα κατανοητό αρκεί να περιγραφεί ο τρόπος λειτουργίας των δύο ιδιοτήτων. Το Burst Size είναι μία μεταβλητή που προγραμματίζεται κατά την αρχικοποίηση της μνήμης και δηλώνει σε μία εντολή πόσες μεταφορές δεδομένων θα πραγματοποιηθούν. Θα μπορούσαν έτσι να μεταφερθούν το πολύ 8 δεδομένα με μία εντολή. Όταν όμως προγραμματιστεί η μνήμη με Burst Size 8 κάθε εντολή θα μεταφέρει 8 δεδομένα και θα διαρκεί συνεπώς 4 κύκλους. Τι γίνεται όμως σε περίπτωση που ο χρήστης επιθυμεί να μεταφέρει μόνο 2 δεδομένα? Είτε χάνονται 3 κύκλοι είτε πρέπει να εισαχθεί μία εντολή Burst Terminate που θα τερματίζει την εντολή πιο νωρίς. Και στις δύο παραπάνω περιπτώσεις χάνεται χρόνος. Για αυτό το λόγο εισήχθηκε μία Burst Length μεταβλητή την οποία εισάγει με κάθε εντολή ο χρήστης (1 < Burst Length < 255) δηλώνοντας έτσι ο ίδιος πόσα δεδομένα επιθυμεί να μεταφερθούν στη συγκεκριμένη εντολή. Με αυτόν τον τρόπο μπορούν να μεταφερθούν πολλά δεδομένα και χωρίς να χάνεται καθόλου χρόνος ανάμεσά τους. Για τον παραπάνω λόγο σε συνδυασμό με την σημαντικά πολλυπλοκότερη λογική που θα εισερχόταν στην FSM για την υποστήριξη των διαφορετικών Burst Size επιλέχθηκε η βασική τιμή 2 και με βάση αυτή υλοποιήθηκε και ο Ελεγκτής.

Μία ακόμη βασική ιδιότητα της μνήμης που δεν υποστηρίχθηκε στο σύστημα είναι η δυνατότητα Ενεργοποίησης (Activate) πολλών διαφορετικών Bank χωρίς να πραγματοποιείται απαραίτητα η εκτέλεση μίας εντολής σε κάθε ένα από αυτά τη συγκεκριμένη στιγμή. Με αυτό τον τρόπο μπορεί να υπάρξει σημαντικό κέρδος στην απόδοση του συστήματος. Αυτό διότι παρέχεται με αυτή την ιδιότητα η δυνατότητα Ενεργοποίησης π.χ. του πρώτου Bank, στη συνέχεια του δεύτερου Bank και όσο το δεύτερο Bank ενεργοποιείται παράλληλα να εκτελείται μία εντολή στο πρώτο που είναι ήδη ενεργοποιημένο. Ακόμη μπορεί αυτό να συνεχίσει για όλα τα Bank και στο τέλος να εισαχθεί μία εντολή Precharge All η οποία θα κάνει Precharge σε όλα τα Activated Bank μαζί.

Ο λόγος που δεν υλοποιήθηκε αυτή η σημαντική ιδιότητα είναι ότι εξαρχής το σύστημα υλοποιήθηκε από την πλευρά του χρήστη οπότε ήταν εξαιρετικά δύσκολη, χρονοβόρα και πολύπλοκη η υλοποίησή του. Με βάση αυτή την οπτική, τόσο η FSM όσο και το Test Bench έχουν υλοποιηθεί με βάση την οπτική του χρήστη. Αυτός γνωρίζει ότι θέλει να εκτελέσει κάποια εντολή είτε ανάγνωσης είτε εγγραφής. Για αυτό το λόγο του ‘’επιβλήθηκε’’ απλά να αρχικοποιήσει τη μνήμη με μία εντολή Initialize και στη συνέχεια εκτελεί μόνο τις επιθυμητές εντολές (Read και Write). Με την είσοδο στην FSM, κάθε μία από αυτές τις εντολές πραγματοποιεί την αντίστοιχη ενεργοποίηση (Activation) του Bank. Σε διαφορετική περίπτωση θα έπρεπε να δίνεται στο χρήστη η ικανότητα εισαγωγής εντολών Active, Precharge, Precharge All, Refresh, Auto Refresh κ.ο.κ. Τα παραπάνω θα καθιστούσαν αρκετά πολύπλοκη τη χρήση του συστήματος τόσο από πλευράς Test Bench όσο και από πλευράς FSM του Ελεγκτή η οποία θα γινόταν αρκετά πιο πολύπλοκη. Ακόμη ο χρήστης θα ήταν απαραίτητο να κατανοήσει πολύ περισσότερα πράγματα για τη λειτουργία της μνήμης, κάτι που μάλλον δεν επιθυμεί. Για τους παραπάνω λόγους και για τις μεγάλες δυσκολίες που εισάγονται στην υλοποίηση της παραπάνω ιδιότητας δεν καλύφθηκε αυτή η δυνατότητα της μνήμης.

Τέλος όπως έχει αναφερθεί και στην περιγραφή της υλοποίησης αξιοποιήθηκαν οι δυνατότητες Auto Precharge και Auto Refresh της μνήμης για εξοικονόμηση ταχύτητας και απλότητας του συστήματος (πάντα όσον αφορά τη χρήση του από κάποιον χρήστη) και η δυνατότητα εκτέλεσης μίας εντολής αμέσως μόλις εισαχθεί στο σύστημα εφόσον δεν επηρεάζει την εκτέλεση της πρώτης (Βλέπε **Παράγραφο** 4.5.2) .

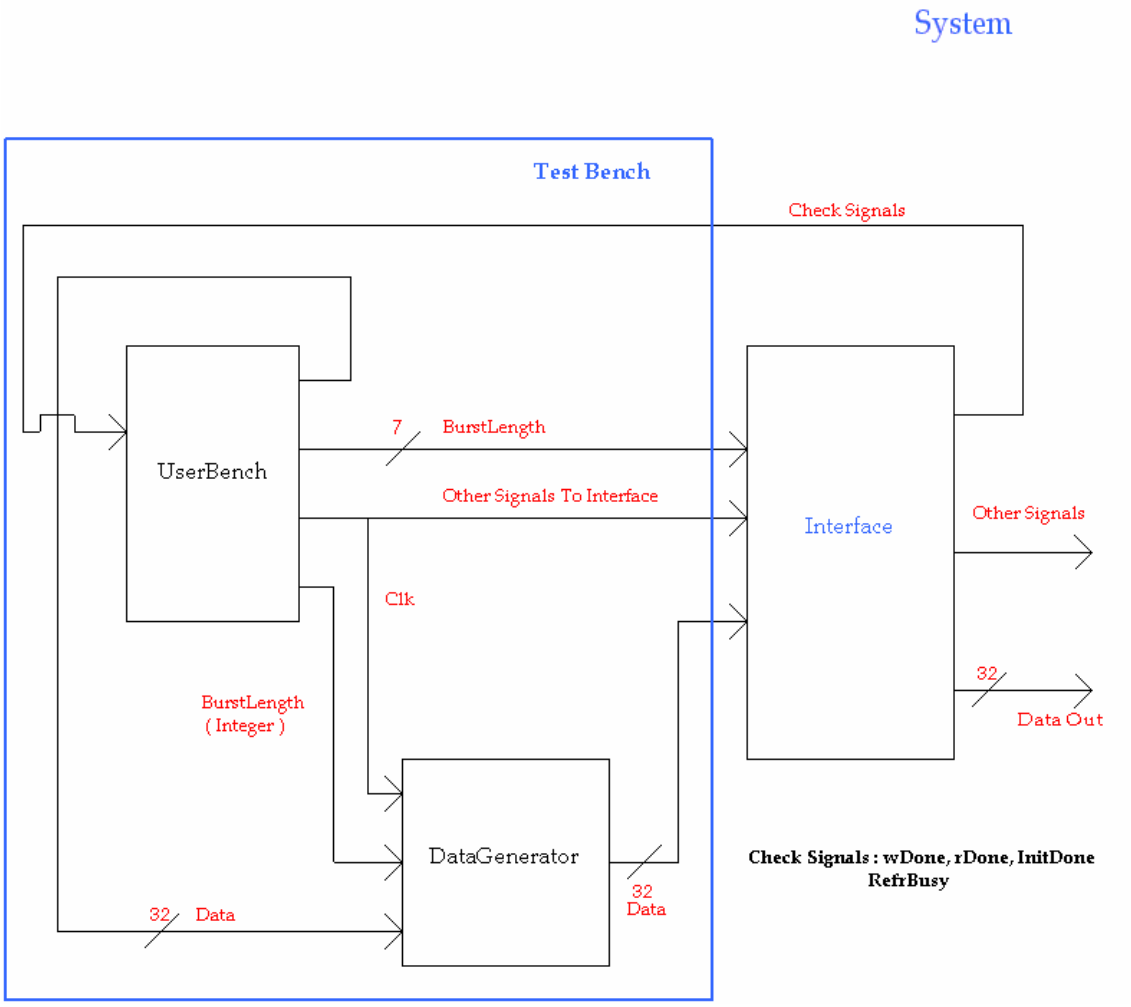
4. Testing

Όπως έχει αναφερθεί και στην **Παράγραφο** 3.1 και διακρίνεται από το **Σχήμα** 3.1 το σύστημα που υλοποιήθηκε μέσα από αυτή την εργασία έχει ένα πολύ απλό Interface έτσι ώστε να μπορεί ο χρήστης να το κατανοήσει και να το χειριστεί όσο πιο εύκολα γίνεται. Κατά την υλοποίηση της κάθε επιμέρους μονάδας του συστήματος κρινόταν απαραίτητη η τακτική προσομοίωση και ο Έλεγχος της σωστής λειτουργίας της. Αυτό πραγματοποιούνταν μέσα από το εργαλείο Model Sim και με τα λεγόμενα Do Files. Για τον έλεγχο όμως της σωστής λειτουργίας του Ελεγκτή στο σύνολό του τα πράγματα δεν είναι τόσο απλά.

Οι έλεγχοι που χρειάσθηκε να πραγματοποιηθούν ήταν αρκετοί και αρκετά σύνθετοι. Για αυτό το λόγο χρειάσθηκε η δημιουργία μία μονάδας, η οποία συνηθίζεται να αποκαλείται Test Bench, και μέσω αυτής πραγματοποιούνται όσοι έλεγχοι επιθυμεί ο χρήστης για να ελέγξει τη σωστή λειτουργία της μνήμης. Με αυτό τον τρόπο το Test Bench καθίσταται όχι μόνο το μέσο μέσα από το οποίο κάποιος χρήστης μπορεί να κάνει χρήση του Συστήματος αλλά και ένα βασικότατο εργαλείο μέσα από το οποίο κατά την δημιουργία του Ελεγκτή πραγματοποιούνταν ο έλεγχος για τη σωστή λειτουργία του συστήματος στο σύνολό του. Ακόμη μέσα από αυτό πραγματοποιήθηκαν πολλά διαφορετικά Simulation. Μέσα από αυτά πραγματοποιήθηκαν πολλοί διαφορετικοί και σύνθετοι συνδυασμοί περιπτώσεων για την απόκτηση μίας τελικής και ολοκληρωμένης εικόνας για την σωστή λειτουργία του συστήματος.

Στη συνέχεια ακολουθεί ένα Block Diagram του Test Bench και ανάλυσή του έτσι ώστε να κατανοηθεί καλύτερα ο τρόπος λειτουργίας του ενώ οι συνδυασμοί που προσομοιώθηκαν και βοηθούν στην κατανόηση της λειτουργίας του συστήματος δίνονται στη συνέχεια του κεφαλαίου και πιο συγκεκριμένα στην **Παράγραφο** 4.5.

4.1 Block Diagram **Του Συστήματος Ελέγχου Λειτουργίας**



4.2 **Περαιτέρω Ανάλυση Των Υπομονάδων Του** Test Bench

4.2.1 User Bench

Η λογική του Block Diagram δείχνει αρκετά απλή. Αποτελείται από δύο βασικές μονάδες. Το Test Bench που αποτελεί και τη λογική με βάση τη οποία πραγματοποιήθηκαν οι απαραίτητοι έλεγχοι της μνήμης και τη μονάδα Interface η οποία όπως είχε φανεί από το Block Diagram της **Παραγράφου** 3.2.1.2 αποτελείται από τον ίδιο τον Ελεγκτή και τη μνήμη. Η ανάλυση αυτών των δύο έχει γίνει παραπάνω οπότε σε αυτή την παράγραφο σκοπός είναι η περιγραφή της λογικής της μονάδας του Test Bench.

Αρχικά λοιπόν υλοποιήθηκε η μονάδα User Bench. Σε αυτή προγραμματίζεται η λειτουργία και ο συγχρονισμός των τριών υπαρχόντων ρολογιών (Clk, Clk_n και Clk90) τα οποία και υποδηλώνουν πότε ξεκινάει η διαδικασία της προσομοίωσης αλλά και πότε αυτή θα σταματήσει, αφού είναι έτσι προγραμματισμένα ώστε εφόσον όλες οι δοθείσες εντολές εκτελεστούν η διαδικασία της προσομοίωσης να τελειώσει.

Μέσα στην μονάδα αυτή ορίζονται σε Procedures τρεις εντολές. Μία για ανάγνωση, μία για εγγραφή και μία για την αρχικοποίηση της μνήμης. Ο χρήστης λοιπόν που θέλει να ελέγξει τη λειτουργία του συγκεκριμένου Ελεγκτή δεν έχει παρά να εισάγει τις κατάλληλες εντολές. Η μορφή με την οποία κάθε μία από αυτές πρέπει να εισαχθεί από τον χρήστη δίνεται στην **Παράγραφο** 4.3.

Για να μπορέσει ο χρήστης να χειριστεί τη μνήμη πρέπει πριν προσπαθήσει να εισάγει κάποια εντολή ανάγνωσης ή εγγραφής να αρχικοποιήσει τη μνήμη. Αν η μνήμη δεν αρχικοποιηθεί σωστά, όποια εντολή και να εισαχθεί σε αυτή παραβλέπεται. Αν η μνήμη με την κατάλληλη εντολή λάβει σήμα για την εκκίνηση της αρχικοποίησής της, τότε συνεχίζει (μέσα από την FSM) χωρίς να επηρεάζεται από τυχόν άλλες εντολές που εισάγονται χωρίς η αρχικοποίηση να έχει ολοκληρωθεί. Όταν αυτή ολοκληρωθεί, επιστρέφει ένα σήμα στη μονάδα User Bench το οποίο δηλώνει την περάτωση της αρχικοποίησης και η επόμενη σε σειρά δοθείσα εντολή από το χρήστη μπορεί να ξεκινήσει την εκτέλεσή της.

Αν πρόκειται για μία εντολή ανάγνωσης δηλώνεται μαζί με την εντολή από τον χρήστη και η διεύθυνση εκκίνησης της ανάγνωσης αλλά και το μέγεθος του Burst που επιθυμεί ο χρήστης. Από την άλλη σε περίπτωση εγγραφής προς τη μνήμη εισάγεται από το χρήστη η διεύθυνση εκκίνησης εγγραφής, τα δεδομένα που επιθυμεί ο χρήστης να εγγραφούν στη μνήμη και το μέγεθος του Burst που αυτός επιθυμεί. Για καλύτερη κατανόηση των παραπάνω Procedures και του τρόπου που αυτά λειτουργούν βλέπε **Παράγραφο** 4.3.

4.2.2 Data Generator

Στην περίπτωση της μονάδας Data Generator τα πράγματα είναι πιο απλά. Η πρόσβαση σε αυτή τη μονάδα πραγματοποιείται μόνο σε περίπτωση εγγραφής, όπου και χρειάζεται αποστολή δεδομένων από τον Tester προς την μνήμη, και μόνο σε περίπτωση που το μέγεθος του Burst είναι μεγαλύτερο του ‘’1’’. Όταν ο χρήστης εισάγει μία απλή εντολή εγγραφής όπου το Burst Size είναι ‘’1’’ τα δεδομένα αποστέλλονται απευθείας από τη μονάδα User Bench στο Interface. Σε περίπτωση που το δοθέν, από το χρήστη, μέγεθος του Burst είναι μεγαλύτερο του ‘’1’’, τα πρώτα δεδομένα, όπου και εισάγονται από τον ίδιο το χρήστη, στέλνονται μεν απευθείας από τη μονάδα User Bench στο Interface αλλά αποστέλλονται και στη μονάδα Data Generator.

Ο λόγος δημιουργίας αυτής της μονάδας είναι η γρήγορη παραγωγή διαφορετικών τιμών δεδομένων των 32 bit, έτσι ώστε να μπορεί να πραγματοποιηθεί η εκτέλεση πολλών συνεχόμενων εγγραφών μέσα από ένα μεγάλο Burst. Ακριβώς επειδή το μέγεθος του Burst προγραμματίζεται από το χρήστη κάθε φορά ανάλογα με το τι επιθυμεί να ελέγξει, δεν ήταν εύκολο ανάλογα με το μέγεθος του Burst ο ίδιος ο χρήστης να μπορεί να εισάγει και τον αντίστοιχο αριθμό δεδομένων προς εγγραφή. Σκοπός όμως του Test Bench είναι απλά ο έλεγχος του λειτουργίας του συστήματος που υλοποιήθηκε και συνεπώς αν αυτό μπορεί να υποστηρίξει εγγραφές με μεγάλο ρυθμό άφιξης. Επομένως δεν είχε τόσο μεγάλη σημασία το τι δεδομένα θα εγγράφονταν σε μία τέτοια περίπτωση, αλλά να εξακριβωθεί ότι τα όποια δεδομένα έφθαναν με τέτοιο ρυθμό θα εγγράφονταν σωστά στη μνήμη από τον Ελεγκτή.

Για αυτό το λόγο, η παραγωγή των δεδομένων είναι που έχει σημασία και όχι το πως αυτή πραγματοποιείται. Στη συγκεκριμένη μονάδα παραγωγής δεδομένων, τα 32 bit πληροφορίας που εισάγονται, χωρίζονται σε δύο δεδομένα των 16 bit και σε κάθε ένα από αυτά, προστίθεται ο αριθμός του Burst Size. Με αυτό τον τρόπο σε κάθε νέα εγγραφή ο αριθμός του Burst Size έχει μειωθεί κατά ‘’1’’. Ο νέος κάθε φορά αριθμός του μεγέθους του Burst προστίθεται τόσο στα 16 MS (Most Significant) bit της πληροφορίας όσο και στα 16 LS (Less Significant) με αποτέλεσμα να παράγονται κάθε φορά 32 bit νέας πληροφορίας και να περνάνε σαν δεδομένα στην είσοδο του Interface. Αυτό συνεχίζεται μέχρι να ολοκληρωθεί η εκτέλεση του Burst.

4.3 **Εκτέλεση Εντολών**

4.3.1 Initialize

Με την εισαγωγή αυτής της εντολής από το χρήστη ανατίθεται στον Ελεγκτή η αρχικοποίηση της μνήμης. Όπως φαίνεται παρακάτω και από τη σύνταξη της εντολής, ο χρήστης έχει τη δυνατότητα να προγραμματίσει τόσο το Burst Type όσο και το Cas Latency στο οποίο επιθυμεί να λειτουργεί η μνήμη. Με την εισαγωγή αυτής της εντολής ενεργοποιείται στην FSM μία μεταβλητή Start η οποία και ορίζει την εκκίνηση της αρχικοποίησης επιτρέποντας στην FSM να εισέλθει στην πρώτη κατάστασή της. Όταν η αρχικοποίηση ολοκληρωθεί με επιτυχία επιστρέφεται μία Boolean μεταβλητή InitDone που υποδηλώνει την ολοκλήρωση της διαδικασίας αρχικοποίησης και επιτρέπει να εισαχθεί στη μνήμη η πρώτη εντολή ανάγνωσης ή εγγραφής που έχει δηλωθεί από το χρήστη.

Initialize (‘BurstType’, "CasLatency");

Π.χ. Initialize (‘0’, "010");

-- Όπου ορίζεται αρχικοποίηση της μνήμης με το Burst Type
-- Sequential και το Cas Latency να προγραμματίζεται στην τιμή 2.

Οι δυνατές τιμές που μπορούν να λάβουν οι παράμετροι Burst Type και Cas Latency υπενθυμίζονται στο **Σχήμα** 4.1.

Σχήμα 4.1

M3	Burst Type
0	Sequential
1	Interleaved

M6	M5	M4	CAS Latency	(DDR400) CAS Latency
0	0	0	Reserved	Reserved
0	0	1	Reserved	Reserved
0	1	0	2	2
0	1	1	Reserved	3
1	0	0	Reserved	Reserved
1	0	1	Reserved	Reserved
1	1	0	2.5	2.5
1	1	1	Reserved	Reserved

4.3.2 Read

Με την εισαγωγή μίας εντολής ανάγνωσης εισάγεται στην FSM η διεύθυνση εκκίνησης της ανάγνωσης αλλά και το μέγεθος του Burst που υποδηλώνει τον αριθμό των αναγνώσεων που επιθυμεί ο χρήστης να πραγματοποιήσει. Έτσι όπως έχει περιγραφεί εκτενώς στην **Παράγραφο** 3.3 εισάγονται στον Ελεγκτή, και επομένως και στην FSM, οι δύο αυτές αρχικές τιμές και αυτός κρίνει πως θα συνεχίσει η εκτέλεση της εντολής. Δηλαδή θα διακρίνει αν πρόκειται για εντολή με μεγάλο Burst, αν θα χρειασθεί αρχικοποίηση της διεύθυνσης που πρόκειται να πραγματοποιηθεί η πρόσβαση και όταν ολοκληρωθεί η εκτέλεση της εντολής αν κρίνεται απαραίτητη η εκτέλεση μίας Auto Refresh εντολής.

Read (Burst Size, "Address");

Π.χ. Read (2, "000000000000110000000011");

-- Πρόσβαση στη μνήμη για ανάγνωση. Η διεύθυνση εκκίνησης της
-- ανάγνωσης είναι στο Bank ‘’10’’ (δηλαδή στο δεύτερο Bank) στη
-- Σειρά ‘’0000000000001’’ (δηλαδή στην πρώτη Σειρά) και στη
-- Στήλη ‘’000000011’’ (δηλαδή στην τρίτη Στήλη). Το μέγεθος
-- του Burst ορίζεται στην τιμή 2 (δηλαδή πρόκειται να
-- πραγματοποιηθούν 2 αναγνώσεις). Κάθε εντολή ανάγνωσης,
-- πραγματοποιεί πρόσβαση για ανάγνωση σε δύο γειτονικές
-- στήλες (DDR) και λόγω της Σειριακής (Sequential) πρόσβασης
-- που έχει προγραμματιστεί θα πραγματοποιηθεί ουσιαστικά
-- ανάγνωση στις στήλες 3,2 και 5,4 και θα επιστραφούν τα
-- αντίστοιχα δεδομένα που ήταν αποθηκευμένα στη μνήμη στις
-- συγκεκριμένες διευθύνσεις.

4.3.3 Write

Σε περίπτωση εγγραφής προς τη μνήμη ακολουθούνται αντίστοιχες διαδικασίες με αυτές της ανάγνωσης με τη διαφορά ότι ο χρήστης δηλώνει με την εισαγωγή και τα πρώτα δεδομένα που επιθυμεί να εγγραφούν στην αρχικά ορισμένη διεύθυνση. Από εκεί και έπειτα η μονάδα Data Generator αναλαμβάνει την παραγωγή των υπολοίπων δεδομένων ανάλογα με το δοθέν μέγεθος του Burst και η FSM την ανάθεση των νέων διευθύνσεων στις οποίες πρόκειται να πραγματοποιηθεί η εγγραφή.

Write (Burst Size, "Data", "Address");

Π.χ. Write (2, “00000000000000010000000000000010”,
"00000000000011100000010");

-- Πρόσβαση στη μνήμη για εγγραφή. Η διεύθυνση εκκίνησης της
-- εγγραφής είναι στο Bank ‘’11’’ (δηλαδή στο τέταρτο Bank) στη
-- Σειρά ‘’0000000000001’’ (δηλαδή στην πρώτη Σειρά) και στη
-- Στήλη ‘’000000010’’ (δηλαδή στην δεύτερη Στήλη). Το μέγεθος
-- του Burst ορίζεται στην τιμή 2 (δηλαδή πρόκειται να
-- πραγματοποιηθούν 2 εγγραφές). Κάθε εντολή εγγραφής,
-- πραγματοποιεί πρόσβαση για εγγραφή σε δύο γειτονικές
-- στήλες (DDR). Λόγω της Σειριακής (Sequential) πρόσβασης
-- που έχει προγραμματιστεί θα πραγματοποιηθεί ουσιαστικά
-- εγγραφή στις στήλες 2,3 και 4,5.

-- Θα εγγραφούν στη στήλη 2 τα16 πρώτα bit του Data
-- ‘’0000000000000001’’ (τιμή ‘1’), στη στήλη 3 τα 16 επόμενα
-- bit του Data ‘’0000000000000010’’ (τιμή ‘2’) ενώ στις στήλες 4
-- και 5 τα δεδομένα που πρόκειται να εγγραφούν θα παραχθούν
-- από τη μονάδα Data Generator και με βάση όσα περιγράφηκαν
-- στην **Παράγραφο** 4.2.2. Με βάση αυτά λοιπόν στήλη 4 θα
-- εγγραφούν δεδομένα με τιμή ‘’000000000000011’’ (τιμή ‘3’)
-- ενώ στην στήλη 5 δεδομένα με τιμή ‘’000000000000100’’ (τιμή ‘4’).

4.4 **Συμπληρωματικά Στοιχεία**

Συμπληρωματικά, αξίζει να αναφερθεί ότι τόσο σε περίπτωση εγγραφής όσο και σε περίπτωση ανάγνωσης, η διεύθυνση που δίνεται από το χρήστη στη μνήμη είναι 24 bit. Από αυτά, τα 13 πρώτα καθορίζουν τη σειρά στην οποία θα πραγματοποιηθεί η πρόσβαση στη μνήμη και αυτή τα λαμβάνει όταν Ενεργοποιεί (εντολή Active) τη συγκεκριμένη σειρά, τα 2 επόμενα bit καθορίζουν το Bank στο οποίο θα πραγματοποιηθεί η πρόσβαση και η τιμή του δίνεται στη μνήμη τόσο κατά την ενεργοποίηση όσο και κατά την εκτέλεση της ίδιας της εντολής ενώ τα υπόλοιπα 9 bit ορίζουν τη συγκεκριμένη στήλη στην οποία πραγματοποιείται η διεργασία και εισάγεται στη μνήμη με την εισαγωγή της ίδιας της διεργασίας. Η μνήμη με τη σειρά της δέχεται Διεύθυνση 13 bit. Σε περίπτωση λοιπόν που κατά τη διάρκεια της Ενεργοποίησης της δίνεται η διεύθυνση σειράς είναι καλυμμένη, σε περίπτωση όμως που τις δίνεται η διεύθυνση στήλης πρέπει να συμπληρωθούν τα 4 MS bit της. Για αυτό το λόγο είτε προστίθενται 4 μηδενικά (‘’0000’’) μπροστά από τη δοθείσα διεύθυνση, είτε στη θέση του τρίτου από αυτά τα μηδενικά μπαίνει ‘’1’’ (‘’0010’’). Αυτή είναι και η περίπτωση εκτέλεσης της εντολής Auto Precharge η οποία πραγματοποιείται όταν στην είσοδο μίας εντολής, το 3 MS bit της διεύθυνσης που εισάγεται έχει την τιμή 1. Όπως έχει ήδη αναφερθεί, στην υλοποίηση αυτού του Ελεγκτή προτιμήθηκε η δεύτερη περίπτωση.

Ακόμη, για τον έλεγχο της σωστής λειτουργίας της μνήμης / προσομοίωση, οι εντολές όλες εισάγονται η μία μετά την άλλη χωρίς να ορίζεται με κάποιο τρόπο κάποια χρονική καθυστέρηση ανάμεσα σε αυτές. Για αυτό το λόγο και ενώ εισάγονται ουσιαστικά συνεχόμενα, η μία δεν επηρεάζει την άλλη. Εκτελούνται με τη σειρά που εισέρχονται (FIFO) και η δεύτερη μπορεί να ξεκινήσει την εκτέλεσή της μόνο εφόσον έχει ολοκληρωθεί η εκτέλεση της πρώτης.

Κάθε μία από τις εντολές Read και Write μετά από την ορθή και ολοκληρωμένη εκτέλεσή τους αποστέλλουν μέσω της FSM ένα Boolean σήμα (rDone και wDone αντίστοιχα) το οποίο δηλώνει ότι η αντίστοιχη εντολή έχει ολοκληρωθεί με επιτυχία και μπορεί να εισέλθει στο σύστημα η επόμενη προς εκτέλεση εντολή.

4.5 Παραδείγματα Testing

Σκοπός αυτής της παραγράφου είναι η παράθεση μερικών παραδειγμάτων για την καλύτερη και ευκολότερη κατανόηση όσων αναφέρθηκαν σε αυτό το κεφάλαιο. Για αυτό το λόγο θα γίνει προσπάθεια κάλυψης κάποιων βασικών περιπτώσεων τόσο από άποψη εισαγωγής των εντολών στον Tester όσο και από άποψη λειτουργίας του όλου συστήματος με την παράθεση διαγραμμάτων χρονισμού.

Ο βασικός τρόπος που εισάγονται οι εντολές στο Test Bench έχει παρουσιαστεί στην **Παράγραφο** 4.3. Σε εκείνη την παράγραφο έχει περιγραφεί και το τι αντιπροσωπεύει κάθε μία από τις μεταβλητές που εμφανίζονται σαν παράμετροι στην κάθε εντολή. Η μονάδα User Bench έχει υλοποιηθεί με τέτοιο τρόπο ώστε να συγχρονίζει κατάλληλα τα ρολόγια του συστήματος και κάθε είδος εντολής να αποτελεί ένα Procedure. Για το λόγο αυτό το μόνο που χρειάζεται να γίνει για να πραγματοποιηθούν διαφορετικοί έλεγχοι για το σύστημα είναι απλά ο χρήστης να εισάγει την κατάλληλη και επιθυμητή ακολουθία εντολών.

4.5.1 **Παράδειγμα** 1

Σκοπός του πρώτου αυτού παραδείγματος είναι να φανεί απλά ο συγχρονισμός των σημάτων σε περίπτωση μίας απλής εντολής εγγραφής όπως επίσης και μίας απλής εντολής ανάγνωσης. Για αυτό το λόγο μέσα στη μονάδα User Bench πρέπει φυσικά να αρχικοποιηθεί η μνήμη και στη συνέχεια να εισαχθούνε οι δύο εντολές. Πρέπει λοιπόν ο χρήστης να εισάγει με τον παρακάτω τρόπο τις τρεις εντολές :

```
BEGIN

    Initialize ( '0', "010" );

    Write (1, "00000000000000011000000000000001", "000000000001101000000011");

    Read  (1,                                    "000000000001101000000011");

    stim_done <= true;

    WAIT;

END PROCESS;
```

Με τον παραπάνω τρόπο ο χρήστης αρχικοποιεί τη μνήμη με Burst Type = **Σειριακό** και Cas Latency = 2 και στη συνέχεια δηλώνει ότι θέλει να πραγματοποιήσει μία απλή εγγραφή με Burst Size = 1, να μεταφέρει 32 bit δεδομένων που αποτελούν **δύο** 16 bit **δεδομένα με τιμές** 1 **και** 3 και να τα αποθηκεύσει στο Bank = 01 στη **Σειρά** = 3 και στη **Στήλη** = 3 (ενώ τα δεύτερα δεδομένα εφόσον η εγγραφή ξεκινάει στη στήλη 3 θα εγγραφούν στη στήλη 2). Στη συνέχεια ο χρήστης ζητάει να αναγνωσθούν τα δεδομένα που μόλις έγραψε. Εφόσον οι δύο εντολές εισέρχονται στο σύστημα χωρίς να παρεμβληθεί ανάμεσά τους χρονική καθυστέρηση και απευθύνονται στο ίδιο Bank, όπως έχει αναφερθεί και στην **Παράγραφο** 3.3.2 θα πρέπει να υπάρξει μία επιπλέον καθυστέρηση τριών κύκλων ανάμεσα στην εκτέλεση των δύο εντολών έτσι ώστε να ολοκληρωθεί το Precharge του Bank πριν εισαχθεί η δεύτερη εντολή. Το διάγραμμα χρονισμού που θα προκύψει από το παράδειγμα αυτό θα είναι της παρακάτω μορφής του Σχήματος 4.2.

Από το διάγραμμα αυτό, εφόσον αποτελεί και το πρώτο παράδειγμα, μπορούν να εξαχθούν αρκετά σημαντικά συμπεράσματα για τον τρόπο λειτουργίας της μνήμης και να κατανοηθούν καλύτερα όσα περιγράφηκαν στο κεφαλαίο αυτό για την υλοποίηση του συστήματος. Αρχικά μπορεί να καταστεί εμφανής ο συγχρονισμός των τριών ρολογιών που σε επίπεδο προσομοίωσης παράγονται από τη μονάδα User Bench. Στη συνέχεια και εφόσον έχει ολοκληρωθεί η αρχικοποίηση της μνήμης παρατηρείται η ενεργοποίηση (τιμή = True) του σήματος InitDone που δηλώνει ότι η αρχικοποίηση ολοκληρώθηκε και ταυτόχρονα εισάγεται στο σύστημα η πρώτη εντολή που αναμένει την εκτέλεσή της.

Εφόσον είναι η πρώτη προς εκτέλεση εντολή εκκινεί αμέσως η εκτέλεσή της. Παρατηρείται η μεταφορά των προς εγγραφή δεδομένων μέσα από τους πέντε καταχωρητές εωσότου δοθούν στη μνήμη συγχρονισμένα με το σήμα Dqs. Ακόμη διακρίνονται οι τρεις κύκλοι της αρχικοποίησης του Bank και οι τρεις που απαιτούνται για την ολοκλήρωση της εγγραφής. Αν προστεθεί ο ένας κύκλος της μετακίνησης από την κατάσταση αναμονής στην ενεργοποίηση του Bank έχουμε το σύνολο των εφτά κύκλων που απαιτούνται για την ολοκλήρωση μίας εγγραφής.

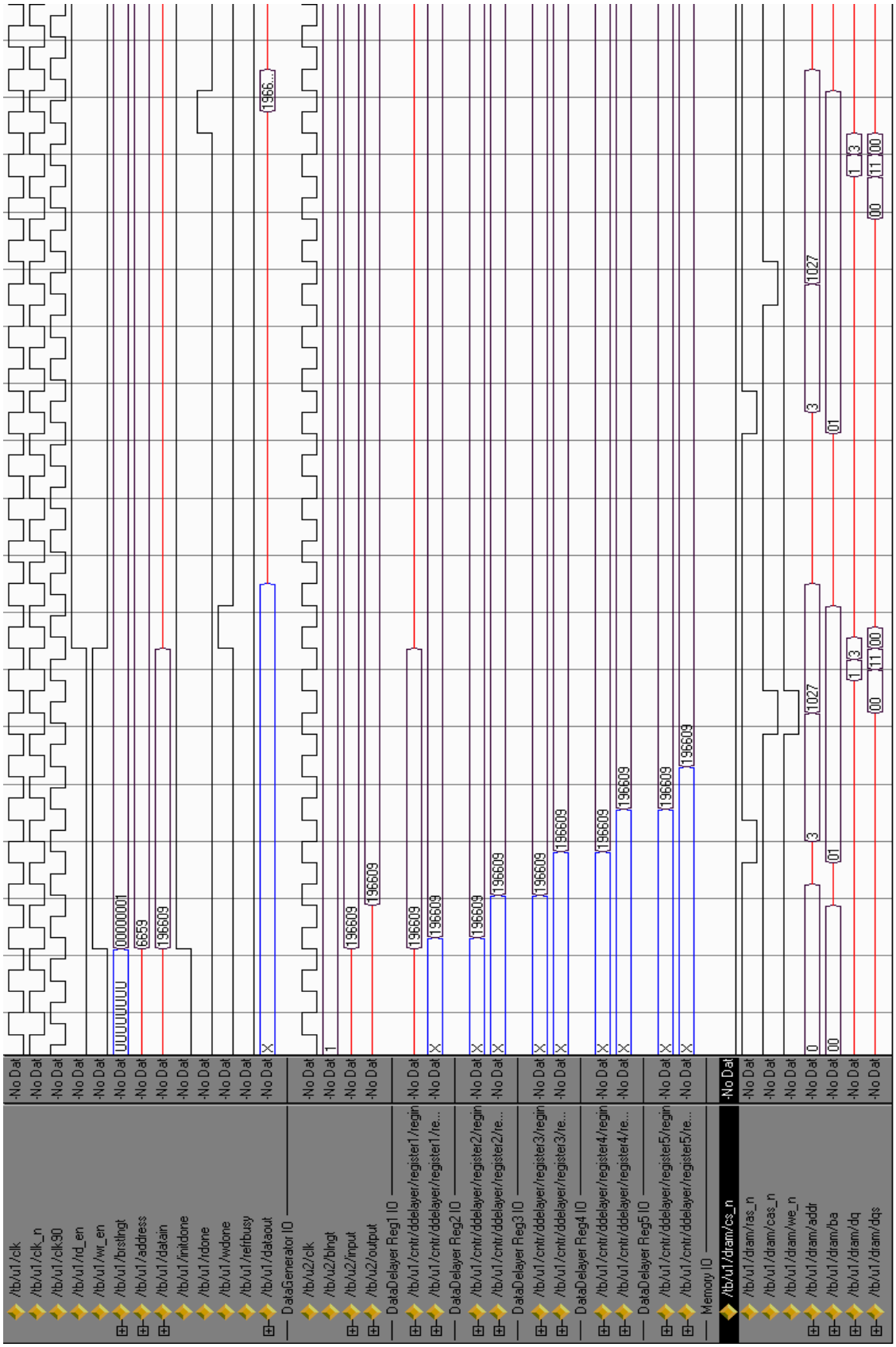
Με την ολοκλήρωση της εγγραφής υψώνεται το σήμα wDone (wDone = True) που δηλώνει ότι η εκτέλεση της εντολής ολοκληρώθηκε και μπορεί να εισαχθεί η επόμενη προς εκτέλεση εντολή. Εφόσον όμως η επόμενη εντολή απευθύνεται στο ίδιο Bank αναμένει για τρεις ακόμη κύκλους προτού εκκινήσει την εκτέλεσή της για να ολοκληρωθεί (δεν επιτρέπεται να διακοπεί) η εκτέλεση του Precharge του Bank από την πρώτη εντολή.

Στη συνέχεια εκκινεί την εκτέλεσή της η εντολή ανάγνωσης. Αυτή απαιτεί 9 συνολικά κύκλους ρολογιού για την εκτέλεσή της (μαζί με το Activation και τον κύκλο μετάβασης από την κατάσταση αναμονής). Όταν ολοκληρώσει και αυτή την εκτέλεσή της (rDone = True) το σύστημα περιμένει για 3 κύκλους ρολογιού και στη συνέχεια ολοκληρώνεται η διαδικασία προσομοίωσης.

Αξίζει να παρατηρηθεί ακόμη η σωστή λήψη των διευθύνσεων και των δεδομένων από τη μνήμη. Τόσο η διεύθυνση Σειράς όσο και η επιλογή του Bank μπορεί να παρατηρηθεί από το διάγραμμα ότι ταιριάζουν με αυτά που εισήγαγε ο χρήστης. Στην περίπτωση της διεύθυνσης Στήλης παρατηρείται ότι αντί για την τιμή 3 υπάρχει η τιμή 1027. Αυτό φυσικά δεν είναι σφάλμα αλλά αποτελεί την διεύθυνση 3 σε συνδυασμό με την εντολή Auto Precharge. Η μνήμη λαμβάνει διεύθυνση 13 bit. Η διεύθυνση στήλης που αποστέλλεται είναι 9 bit της μορφής ‘’000000011’’. Έτσι προστίθενται μπροστά από τη διεύθυνση τέσσερα ακόμη bit με το 2 LS από αυτά να έχει την τιμή ‘1’ για την εισαγωγή της εντολής του Auto Precharge με την ολοκλήρωση της εκτέλεσης της εντολής. Έτσι τα bit που φθάνουν στη μνήμη είναι ‘’0010’’ &’’000000011’’ = ‘’0010000000011’’. Η παραπάνω τιμή σε δεκαεξαδική μορφή αναπαριστά τον αριθμό 1027.

Τέλος αξίζει να παρατηρηθεί η σωστή λειτουργία τόσο της εγγραφής όσο και της ανάγνωσης αφού τα δεδομένα εγγράφονται σωστά στη μνήμη οπότε και στη συνέχεια αναγνώσκονται ορθά από αυτή και περνάνε στην έξοδο του συστήματος με την μορφή πλέον όχι δύο δεδομένων των 16 bit αλλά ενός των 32 όπως και εισήχθησαν.

Σχήμα 4.2


/tb/u1/clk
/tb/u1/clk_n
/tb/u1/clk90
/tb/u1/rd_en
/tb/u1/wr_en
/tb/u1/brstlngt
/tb/u1/address
/tb/u1/datain
/tb/u1/initdone
/tb/u1/rdone
/tb/u1/wdone
/tb/u1/refbusy
/tb/u1/dataout
DataGenerator IO
/tb/u2/clk
/tb/u2/blngt
/tb/u2/input
/tb/u2/output
DataDelayer Reg1 IO
/tb/u1/cntr/ddelayer/register1/regin
/tb/u1/cntr/ddelayer/register1/re...
DataDelayer Reg2 IO
/tb/u1/cntr/ddelayer/register2/regin
/tb/u1/cntr/ddelayer/register2/re...
DataDelayer Reg3 IO
/tb/u1/cntr/ddelayer/register3/regin
/tb/u1/cntr/ddelayer/register3/re...
DataDelayer Reg4 IO
/tb/u1/cntr/ddelayer/register4/regin
/tb/u1/cntr/ddelayer/register4/re...
DataDelayer Reg5 IO
/tb/u1/cntr/ddelayer/register5/regin
/tb/u1/cntr/ddelayer/register5/re...
Memory IO
/tb/u1/dram/cs_n
/tb/u1/dram/ras_n
/tb/u1/dram/cas_n
/tb/u1/dram/we_n
/tb/u1/dram/addr
/tb/u1/dram/ba
/tb/u1/dram/dq
/tb/u1/dram/dqs
-No Dat
UUUUUUUU
00000001
6659
196609
1966...
1
3
1027
01
00
11
1_13

4.5.2 **Παράδειγμα** 2

Σκοπός αυτού του δεύτερου παραδείγματος είναι η παρουσίαση τριών εντολών. Και οι τρεις εντολές θα έχουν πάλι Burst Size = 1 που αποτελεί την απλούστερη περίπτωση διότι σκοπός του παραδείγματος είναι η εξοικονόμηση των τριών κύκλων που χάνονται για την αναμονή ολοκλήρωσης του Precharge. Για αυτό το λόγο θα χρησιμοποιηθούν δύο εντολές εγγραφής και μία εντολή ανάγνωσης. Η πρώτη εντολή εγγραφής και η εντολή ανάγνωσης θα αφορούν το ίδιο Bank.

Στο πρώτο παράδειγμα η ακολουθία των εντολών θα είναι η ακόλουθη :

```
BEGIN

    Initialize ( '0', "010" );

    Write (1, "00000000000000011000000000000001", "000000000001101000000011");

    Read  (1,                                    "000000000001101000000011");

    Write (1, "00000000000001010000000000000100", "00000000000000000000000");

    stim_done <= true;

    WAIT;

END PROCESS;
```

Στην δεύτερη περίπτωση οι εντολές θα είναι οι ίδιες με μόνη διαφορά ότι η εντολή ανάγνωσης θα πραγματοποιηθεί μετά την εκτέλεση και της δεύτερης εντολής εγγραφής :

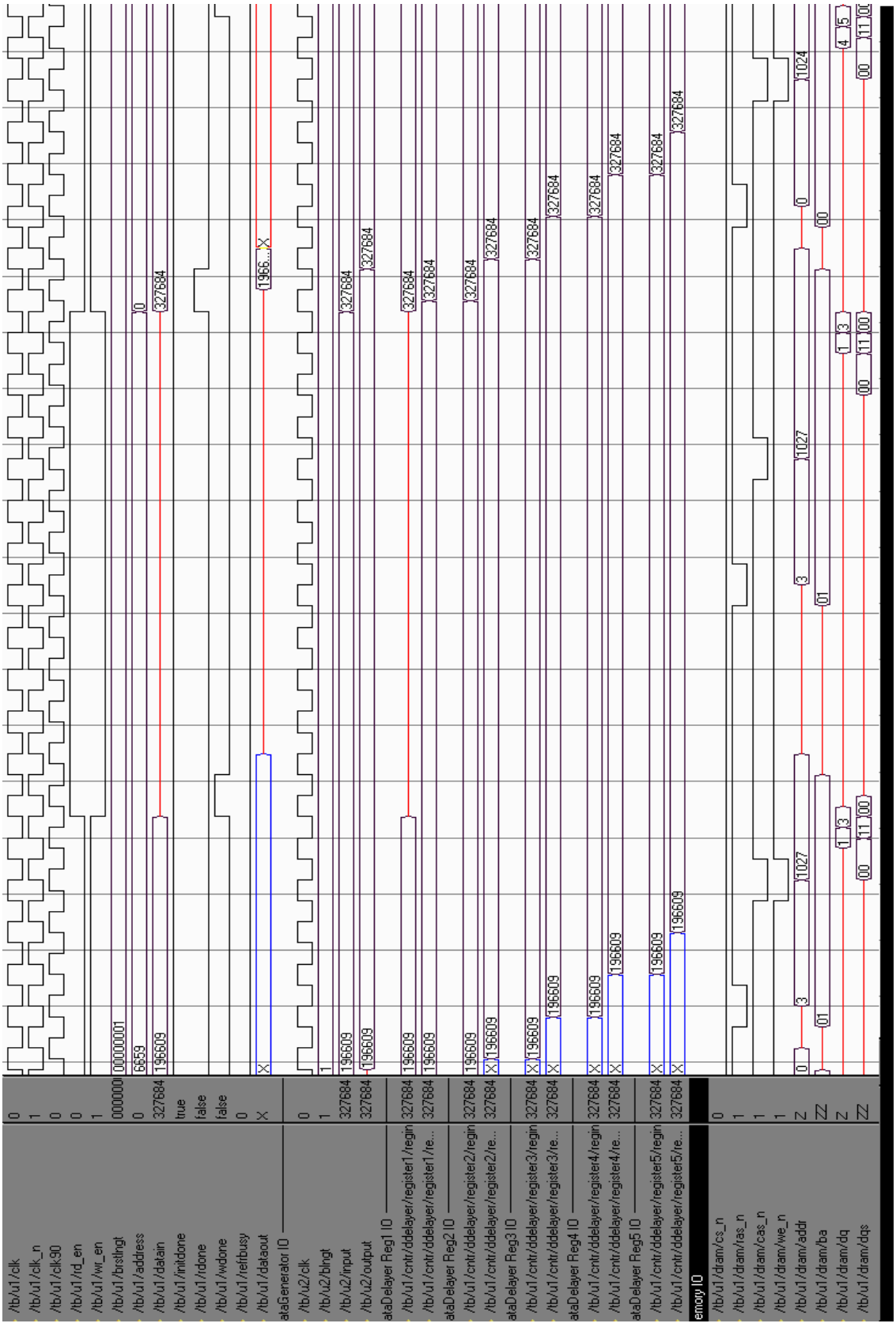
```
BEGIN

    Initialize ( '0', "010" );

    Write (1, "00000000000000011000000000000001", "000000000001101000000011");

    Write (1, "00000000000001010000000000000100", "00000000000000000000000");

    Read  (1,                                    "000000000001101000000011");

    stim_done <= true;

    WAIT;

END PROCESS;
```

Με βάση όσα αναφέρθηκαν στο κεφάλαιο αυτό, στην πρώτη περίπτωση, εφόσον οι εντολές εισάγονται στο σύστημα χωρίς κενό και οι δύο πρώτες αφορούν το ίδιο Bank, ανάμεσα στην πρώτη και τη δεύτερη εντολή θα παρεμβληθεί ένα κενό τριών κύκλων για να ολοκληρωθεί το Precharge του Bank πριν εισαχθεί σε αυτό η επόμενη εντολή. Στην δεύτερη περίπτωση, ανάμεσα στις δύο εντολές που απευθύνονται στο ίδιο Bank παρεμβάλλεται μία εντολή που αφορά κάποιο άλλο Bank. Με αυτόν τον τρόπο όσο ολοκληρώνεται εσωτερικά στη μνήμη το Precharge του πρώτου Bank μπορεί να εκκινήσει την εκτέλεσή της η επόμενη εντολή. Με αυτόν τον τρόπο επιτυγχάνεται εσωτερικά στη μνήμη παράλληλη λειτουργία με αποτέλεσμα όταν πλέον εισαχθεί στη μνήμη η εντολή που αφορά το πρώτο Bank, ο χρόνοε ολοκλήρωσης του Precharge να έχει παρέλθει και να εκκινήσει και αυτή η εντολή αμέσως τη λειτουργία της.

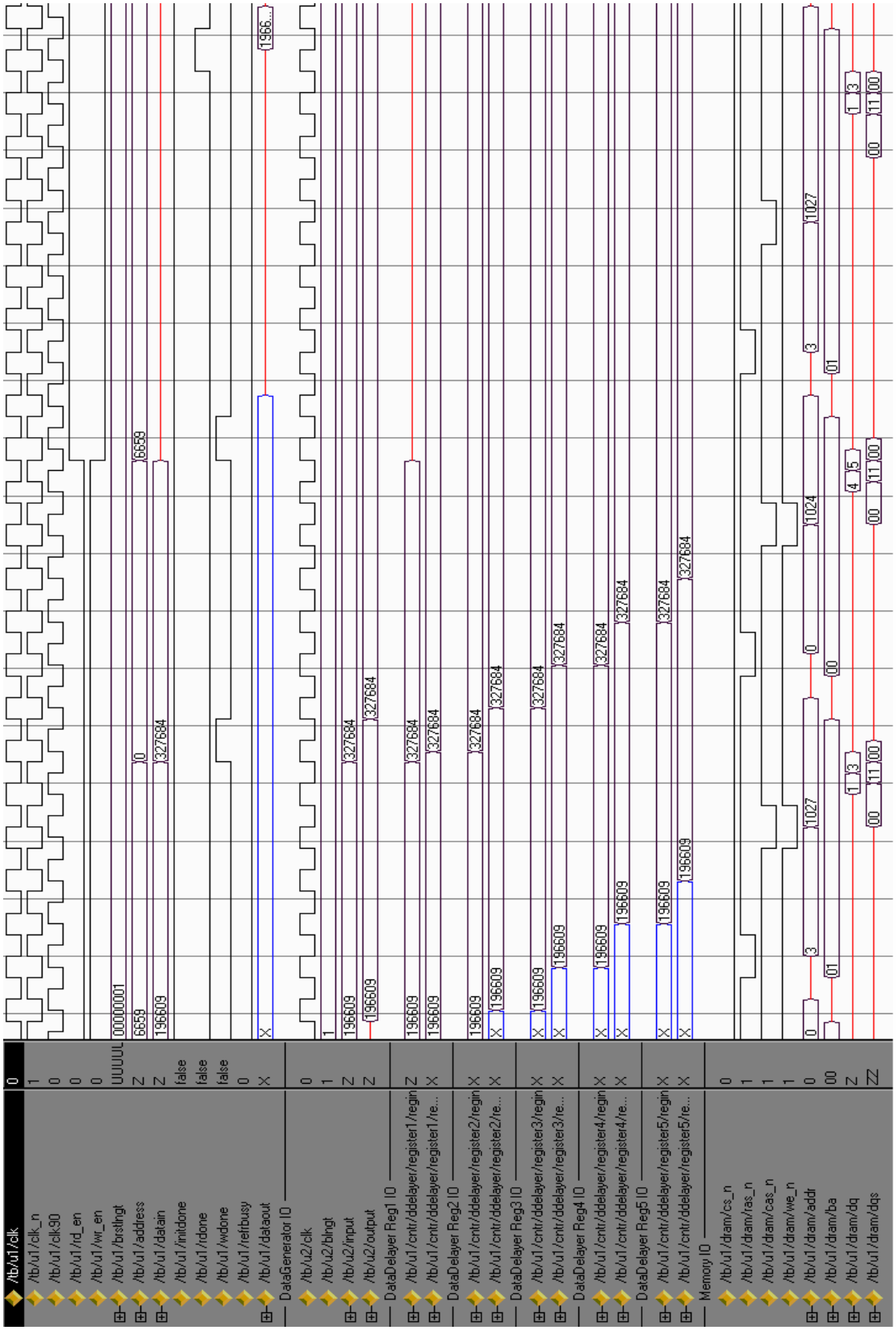
Με αυτό τον απλό τρόπο είναι δυνατή η εξοικονόμηση τριών κύκλων ρολογιού ανάμεσα στην εκτέλεση μόλις τριών εντολών. Στην περίπτωση που δεν παρείχε η μνήμη τη δυνατότητα εκτέλεσης παράλληλων διεργασιών ή στην περίπτωση που ο Ελεγκτής δεν αξιοποιούσε αυτή τη δυνατότητα, ανάμεσα σε κάθε ζευγάρι εντολών θα παρεμβάλλονταν τρεις επιπλέον κύκλοι καθυστέρησης.

Όσα αναφέρθηκαν παραπάνω διακρίνονται ευκολότερα από τα παρακάτω διαγράμματα. Στην πρώτη περίπτωση το διάγραμμα που προκύπτει παρουσιάζεται στο **Σχήμα** 4.3, ενώ στη δεύτερη περίπτωση το διάγραμμα που προκύπτει παρουσιάζεται στο **Σχήμα** 4.4.

Σχήμα 4.3


/tb/u1/clk
/tb/u1/clk_n
/tb/u1/clk90
/tb/u1/rd_en
/tb/u1/wr_en
/tb/u1/brstlngt
/tb/u1/address
/tb/u1/datain
/tb/u1/initdone
/tb/u1/rdone
/tb/u1/wdone
/tb/u1/refrbusy
/tb/u1/dataout
dataGenerator IO
/tb/u2/clk
/tb/u2/blngt
/tb/u2/input
/tb/u2/output
dataDelayer Reg1 IO
/tb/u1/cntr/ddelayer/register1/regin
/tb/u1/cntr/ddelayer/register1/re...
dataDelayer Reg2 IO
/tb/u1/cntr/ddelayer/register2/regin
/tb/u1/cntr/ddelayer/register2/re...
dataDelayer Reg3 IO
/tb/u1/cntr/ddelayer/register3/regin
/tb/u1/cntr/ddelayer/register3/re...
dataDelayer Reg4 IO
/tb/u1/cntr/ddelayer/register4/regin
/tb/u1/cntr/ddelayer/register4/re...
dataDelayer Reg5 IO
/tb/u1/cntr/ddelayer/register5/regin
/tb/u1/cntr/ddelayer/register5/re...
emory IO
/tb/u1/dram/cs_n
/tb/u1/dram/ras_n
/tb/u1/dram/cas_n
/tb/u1/dram/we_n
/tb/u1/dram/addr
/tb/u1/dram/ba
/tb/u1/dram/dq
/tb/u1/dram/dqs

Σχήμα 4.4


/tb/u1/clk
/tb/u1/clk_n
/tb/u1/clk90
/tb/u1/rd_en
/tb/u1/wr_en
/tb/u1/brstlngt
/tb/u1/address
/tb/u1/datain
/tb/u1/initdone
/tb/u1/rdone
/tb/u1/wdone
/tb/u1/refbusy
/tb/u1/dataout
DataGenerator IO
/tb/u2/clk
/tb/u2/blngt
/tb/u2/input
/tb/u2/output
DataDelayer Reg1 IO
/tb/u1/cntr/ddelayer/register1/regin
/tb/u1/cntr/ddelayer/register1/re...
DataDelayer Reg2 IO
/tb/u1/cntr/ddelayer/register2/regin
/tb/u1/cntr/ddelayer/register2/re...
DataDelayer Reg3 IO
/tb/u1/cntr/ddelayer/register3/regin
/tb/u1/cntr/ddelayer/register3/re...
DataDelayer Reg4 IO
/tb/u1/cntr/ddelayer/register4/regin
/tb/u1/cntr/ddelayer/register4/re...
DataDelayer Reg5 IO
/tb/u1/cntr/ddelayer/register5/regin
/tb/u1/cntr/ddelayer/register5/re...
Memory IO
/tb/u1/dram/cs_n
/tb/u1/dram/ras_n
/tb/u1/dram/cas_n
/tb/u1/dram/we_n
/tb/u1/dram/addr
/tb/u1/dram/ba
/tb/u1/dram/dq
/tb/u1/dram/dqs

Συγκρίνοντας τα δύο διαγράμματα γίνονται εύκολα κατανοητά όσα αναφέρθηκαν παραπάνω και παρατηρείται ότι ενώ στην πρώτη περίπτωση από τη στιγμή εκκίνησης εκτέλεσης της πρώτης εντολής έως την ολοκλήρωση της εκτέλεσης της τελευταίας παρέρχονται 26 κύκλοι ρολογιού, στην περίπτωση του δεύτερου διαγράμματος χρειάζονται μόλις 23.

4.5.3 **Παράδειγμα** 3

Το τρίτο παράδειγμα έχει σαν σκοπό να διαχωρίσει τις δύο ακραίες περιπτώσεις όπου δύο διαδοχικές εντολές εγγραφής απευθύνονται σε διαφορετικά η στο ίδιο Bank. Στην περίπτωση όπου οι δύο εντολές απευθύνονται σε διαφορετικά Bank τότε η δεύτερη μπορεί να εκκινήσει την εκτέλεσή της με την ολοκλήρωση της εκτέλεσης της πρώτης, ενώ στην περίπτωση που απευθύνονται στο ίδιο Bank πρέπει η δεύτερη εντολή να περιμένει τρεις επιπλέον κύκλους έτσι ώστε να ολοκληρωθεί το Precharge της πρώτης εντολής.

Στην πρώτη περίπτωση τα προς εγγραφή δεδομένα πρέπει να καθυστερήσουν μόλις 5 κύκλους για να συγχρονιστούν απόλυτα με τα υπόλοιπα σήματα έτσι ώστε να πραγματοποιηθεί με επιτυχία η εγγραφή ενώ στην δεύτερη περίπτωση πρέπει να καθυστερήσουν για τρεις επιπλέον κύκλους. Αυτό γίνεται εμφανές και από τα διαγράμματα που θα ακολουθήσουν.

Στην πρώτη περίπτωση θα υπάρχει μία ακολουθία εντολών της μορφής :

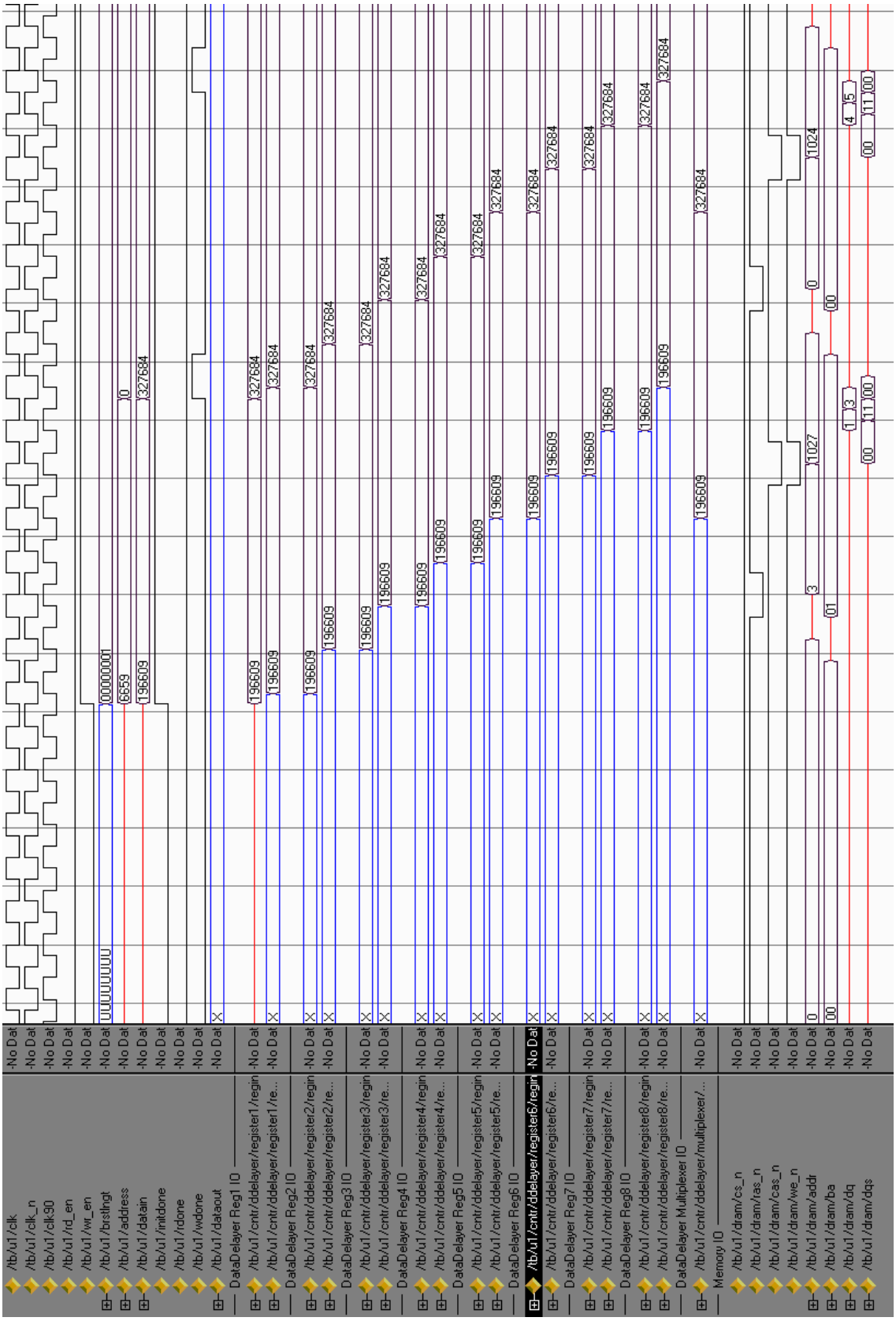
```
BEGIN

    Initialize ( '0', "010" );

    Write (1, "00000000000000011000000000000001", "000000000001101000000011");

    Write (1, "00000000000001010000000000000100", "000000000000000000000000");

    stim_done <= true;

    WAIT;

END PROCESS;
```

Στην δεύτερη περίπτωση η ακολουθία των εντολών θα είναι :

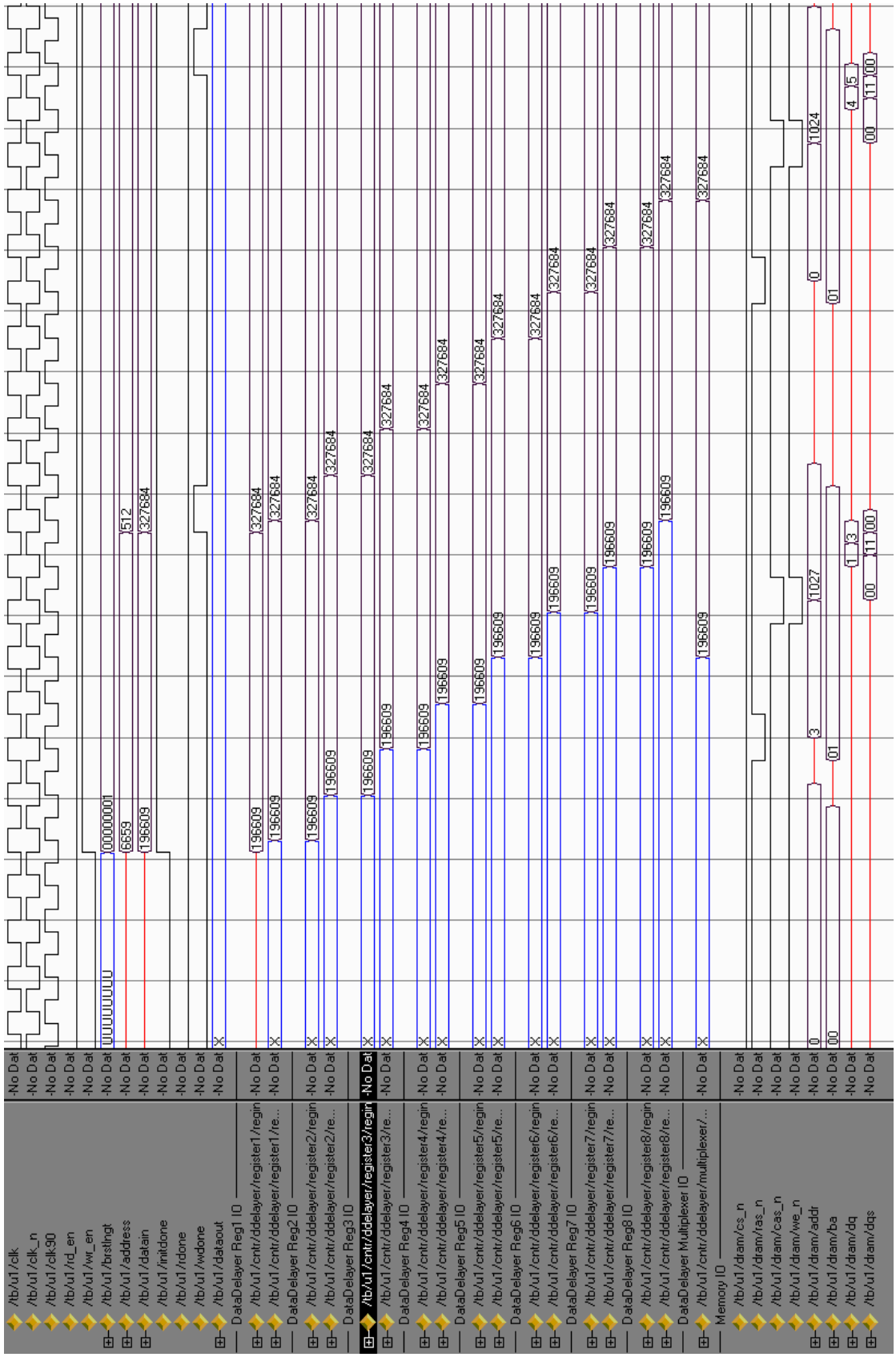
```
BEGIN
    Initialize ( '0', "010" );
    Write (1, "00000000000000011000000000000001", "00000000000011010000000011");
    Write (1, "00000000000001010000000000000100", "00000000000000010000000000");
    stim_done <= true;
    WAIT;
END PROCESS;
```

Στην περίπτωση που οι δύο εγγραφές πραγματοποιούνται σε διαφορετικά Bank το διάγραμμα χρονισμού θα είναι αυτό που διακρίνεται στο **Σχήμα** 4.5 ενώ σε περίπτωση που οι δύο εγγραφές πραγματοποιούνται στο ίδιο Bank το διάγραμμα χρονισμού θα είναι όπως παρουσιάζεται στο **Σχήμα** 4.6.

Σχήμα 4.5



Σχήμα 4.6


/tb/u1/clk
/tb/u1/clk_n
/tb/u1/clk90
/tb/u1/rd_en
/tb/u1/wl_en
/tb/u1/brstlngt
/tb/u1/address
/tb/u1/datain
/tb/u1/initdone
/tb/u1/rdone
/tb/u1/wdone
/tb/u1/dataout
DataDelayer Reg1 IO
/tb/u1/cntr/ddelayer/register1/regin
/tb/u1/cntr/ddelayer/register1/re...
DataDelayer Reg2 IO
/tb/u1/cntr/ddelayer/register2/regin
/tb/u1/cntr/ddelayer/register2/re...
DataDelayer Reg3 IO
/tb/u1/cntr/ddelayer/register3/regin
/tb/u1/cntr/ddelayer/register3/re...
DataDelayer Reg4 IO
/tb/u1/cntr/ddelayer/register4/regin
/tb/u1/cntr/ddelayer/register4/re...
DataDelayer Reg5 IO
/tb/u1/cntr/ddelayer/register5/regin
/tb/u1/cntr/ddelayer/register5/re...
DataDelayer Reg6 IO
/tb/u1/cntr/ddelayer/register6/regin
/tb/u1/cntr/ddelayer/register6/re...
DataDelayer Reg7 IO
/tb/u1/cntr/ddelayer/register7/regin
/tb/u1/cntr/ddelayer/register7/re...
DataDelayer Reg8 IO
/tb/u1/cntr/ddelayer/register8/regin
/tb/u1/cntr/ddelayer/register8/re...
DataDelayer Multiplexer IO
/tb/u1/cntr/ddelayer/multiplexer/...
Memory IO
/tb/u1/dram/cs_n
/tb/u1/dram/ras_n
/tb/u1/dram/cas_n
/tb/u1/dram/we_n
/tb/u1/dram/addr
/tb/u1/dram/ba
/tb/u1/dram/dq
/tb/u1/dram/dqs
-No Dat
UUUUUUUU
00000001
6659
196609
512
327684
3
01
1027
1024
4
5
11
00

Στα παραπάνω διαγράμματα διακρίνονται τόσο οι οκτώ καταχωρητές όσο και η έξοδος του πολυπλέκτη της μονάδας Data Delayer που έχει περιγραφεί στην **Παράγραφο** 4.2.4. Με αυτό τον τρόπο στην πρώτη περίπτωση όπως και ήταν αναμενόμενο διακρίνεται ότι ο πολυπλέκτης περνάει στην έξοδό του την τιμή του πέμπτου καταχωρητή αφού τόση κρίνεται ότι πρέπει να είναι η καθυστέρηση των δεδομένων. Από την άλλη, στην δεύτερη περίπτωση η καθυστέρηση πρέπει να είναι οκτώ κύκλων (για τους λόγους που έχουν αναφερθεί και παραπάνω) για αυτό και στην έξοδο του πολυπλέκτη περνάει η έξοδος του τελευταίου καταχωρητή.

4.5.4 **Παράδειγμα** 4

Το τέταρτο παράδειγμα έχει σαν σκοπό να παρουσιάσει στον αναγνώστη μέσα από το πιο απλό δυνατό παράδειγμα την προσφορά, στον τομέα της απόδοσης και της ταχύτητας, της υποστήριξης μεγάλου αριθμού Burst από τον Ελεγκτή. Θα πραγματοποιηθεί λοιπόν εγγραφή 2 δεδομένων των 32 bit με δύο ξεχωριστές εντολές εγγραφής (κάθε μία θα εγγράψει 32 bit δεδομένων στη μνήμη) και στη συνέχεια θα πραγματοποιηθεί εγγραφή των ίδιων δεδομένων μέσα από μία μόνο εντολή εγγραφής με Burst Size = 2.

Για την απλούστευση της πρώτης περίπτωσης οι δύο εγγραφές θα πραγματοποιηθούν σε δύο διαφορετικά Bank έτσι ώστε όταν μετρηθεί η εξοικονόμηση χρόνου από την δυνατότητα υποστήριξης μεγάλων Burst να μην συνυπολογίζονται και οι τρεις κύκλοι για την ολοκλήρωση του Precharge του Bank της πρώτης εντολής όπως παρατηρήθηκε και από το **Παράδειγμα** 2.

Στην πρώτη περίπτωση θα υπάρχει μία ακολουθία δύο εντολών όπως φαίνεται παρακάτω :

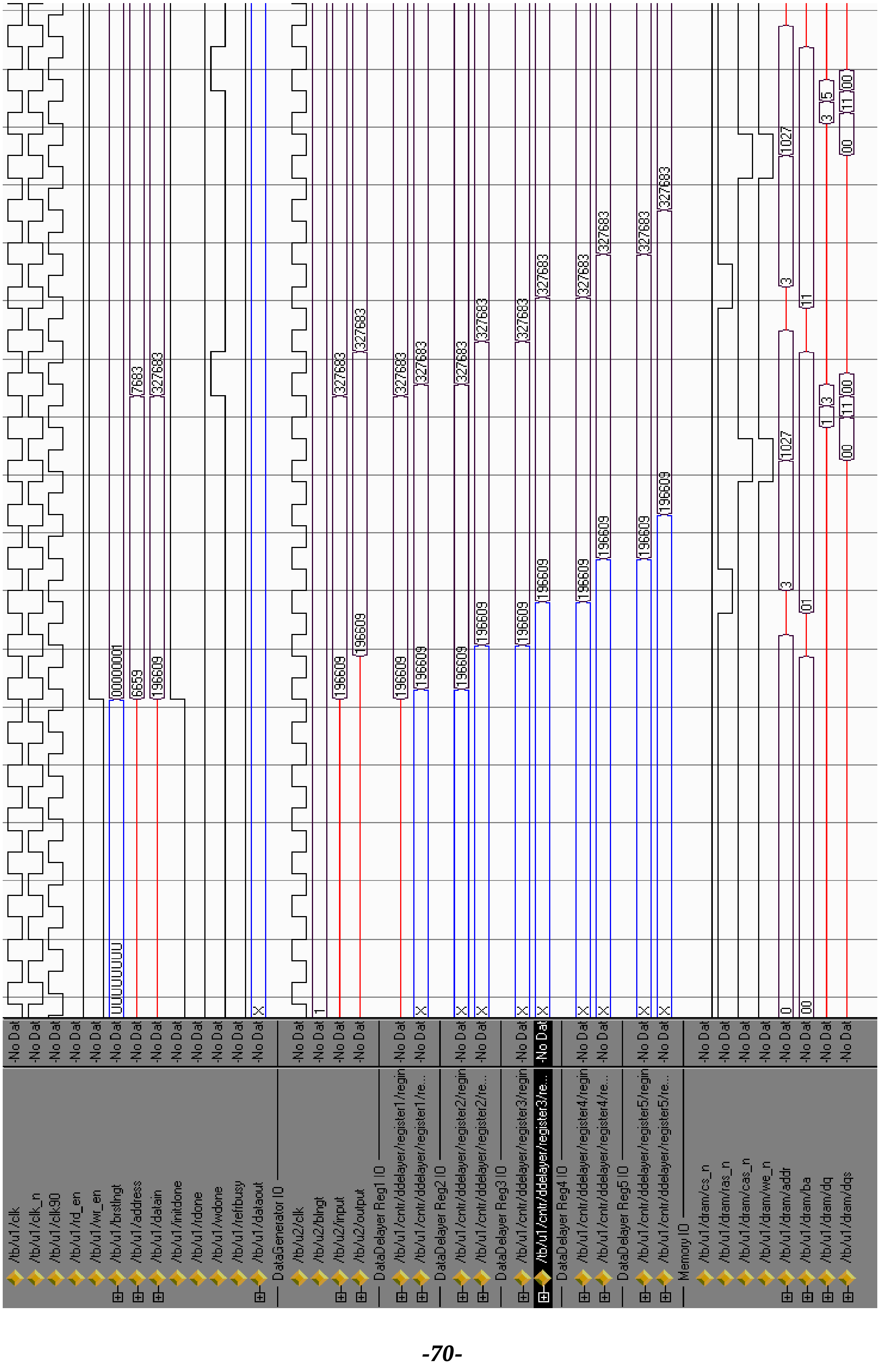
```
BEGIN

    Initialize ( '0', "010" );

    Write (1, "00000000000000011000000000000001", "00000000000011010000000011");

    Write (1, "00000000000000101000000000000011", "00000000000011110000000011");

    stim_done <= true;

    WAIT;

END PROCESS;
```

Στην δεύτερη περίπτωση η εντολή θα είναι όπως προαναφέρθηκε μία, με Burst Size = 2 :

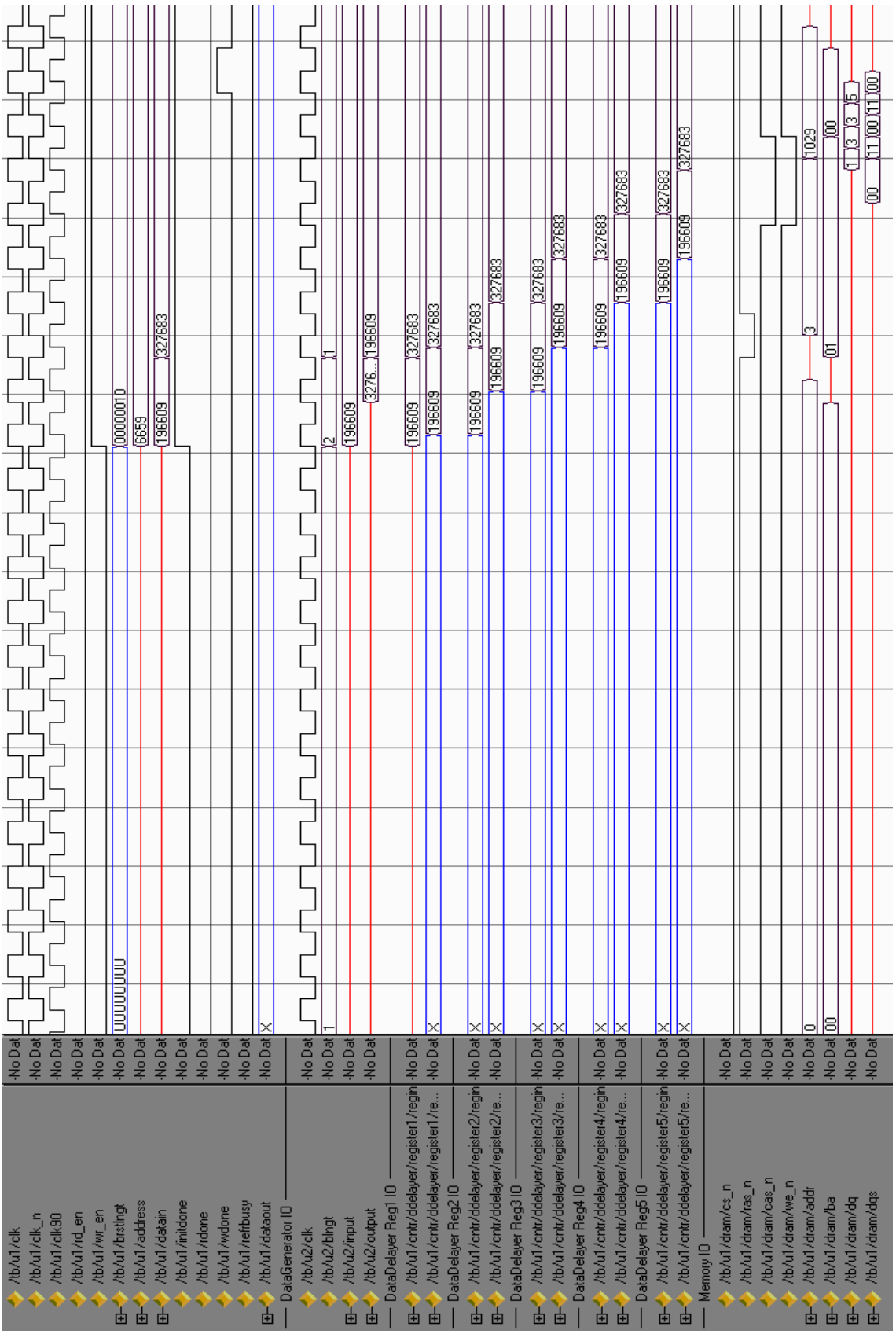
```
BEGIN

    Initialize ( '0', "010" );

    Write (2, "00000000000000011000000000000001", "00000000000011010000000011");

    stim_done <= true;

    WAIT;

END PROCESS;
```

Το διάγραμμα που προκύπτει στην πρώτη περίπτωση δίνεται στο **Σχήμα** 4.7. Το αντίστοιχο της δεύτερης περίπτωσης δίνεται στο **Σχήμα** 4.8.

Σχήμα 4.7


/tb/u1/clk
/tb/u1/clk_n
/tb/u1/clk90
/tb/u1/rd_en
/tb/u1/wr_en
/tb/u1/brstlngt
/tb/u1/address
/tb/u1/datain
/tb/u1/initdone
/tb/u1/rdone
/tb/u1/wdone
/tb/u1/refrbusy
/tb/u1/dataout
DataGenerator IO
/tb/u2/clk
/tb/u2/blngt
/tb/u2/input
/tb/u2/output
DataDelayer Reg1 IO
DataDelayer Reg2 IO
DataDelayer Reg3 IO
DataDelayer Reg4 IO
DataDelayer Reg5 IO
Memory IO
/tb/u1/dram/cs_n
/tb/u1/dram/ras_n
/tb/u1/dram/cas_n
/tb/u1/dram/we_n
/tb/u1/dram/addr
/tb/u1/dram/ba
/tb/u1/dram/dq
/tb/u1/dram/dqs

Σχήμα 4.8


/tb/u1/clk
/tb/u1/clk_n
/tb/u1/clk90
/tb/u1/rd_en
/tb/u1/wr_en
/tb/u1/brstlngt
/tb/u1/address
/tb/u1/datain
/tb/u1/initdone
/tb/u1/rdone
/tb/u1/wdone
/tb/u1/refrbusy
/tb/u1/dataout
DataGenerator IO
/tb/u2/clk
/tb/u2/blngt
/tb/u2/input
/tb/u2/output
DataDelayer Reg1 IO
/tb/u1/cntr/ddelayer/register1/regin
/tb/u1/cntr/ddelayer/register1/re...
DataDelayer Reg2 IO
/tb/u1/cntr/ddelayer/register2/regin
/tb/u1/cntr/ddelayer/register2/re...
DataDelayer Reg3 IO
/tb/u1/cntr/ddelayer/register3/regin
/tb/u1/cntr/ddelayer/register3/re...
DataDelayer Reg4 IO
/tb/u1/cntr/ddelayer/register4/regin
/tb/u1/cntr/ddelayer/register4/re...
DataDelayer Reg5 IO
/tb/u1/cntr/ddelayer/register5/regin
/tb/u1/cntr/ddelayer/register5/re...
Memory IO
/tb/u1/dram/cs_n
/tb/u1/dram/ras_n
/tb/u1/dram/cas_n
/tb/u1/dram/we_n
/tb/u1/dram/addr
/tb/u1/dram/ba
/tb/u1/dram/dq
/tb/u1/dram/dqs
-No Dat
UUUUUUUU
00000010
6659
196609
327683
3276...
1029
01
00
11
3
5

Όπως διακρίνεται και από τα παραπάνω διαγράμματα το κέρδος που προσφέρει στο σύστημα η δυνατότητα υποστήριξης εντολών με μεγάλο Burst είναι πολύ σημαντική. Στην περίπτωση εγγραφής μόλις δύο δεδομένων στην πρώτη περίπτωση απαιτούνται 14 κύκλοι ρολογιού ενώ στη δεύτερη μόλις 8. Δηλαδή, ενώ στην πρώτη περίπτωση απαιτούνται δύο απλές εντολές εγγραφής των 7 κύκλων, στην δεύτερη περίπτωση κάθε επιπλέον δεδομένα προς εγγραφή απαιτούν μόλις έναν ακόμη επιπλέον κύκλο. Για κάθε εγγραφή λοιπόν που μπορεί να καλυφθεί στο Burst έχουμε εξοικονόμηση 6 κύκλων. Αντίστοιχα σε περίπτωση ανάγνωσης η εξοικονόμηση για κάθε επιπλέον ανάγνωση δεδομένων που μπορεί να καλυφθεί στο Burst είναι 8 κύκλοι.

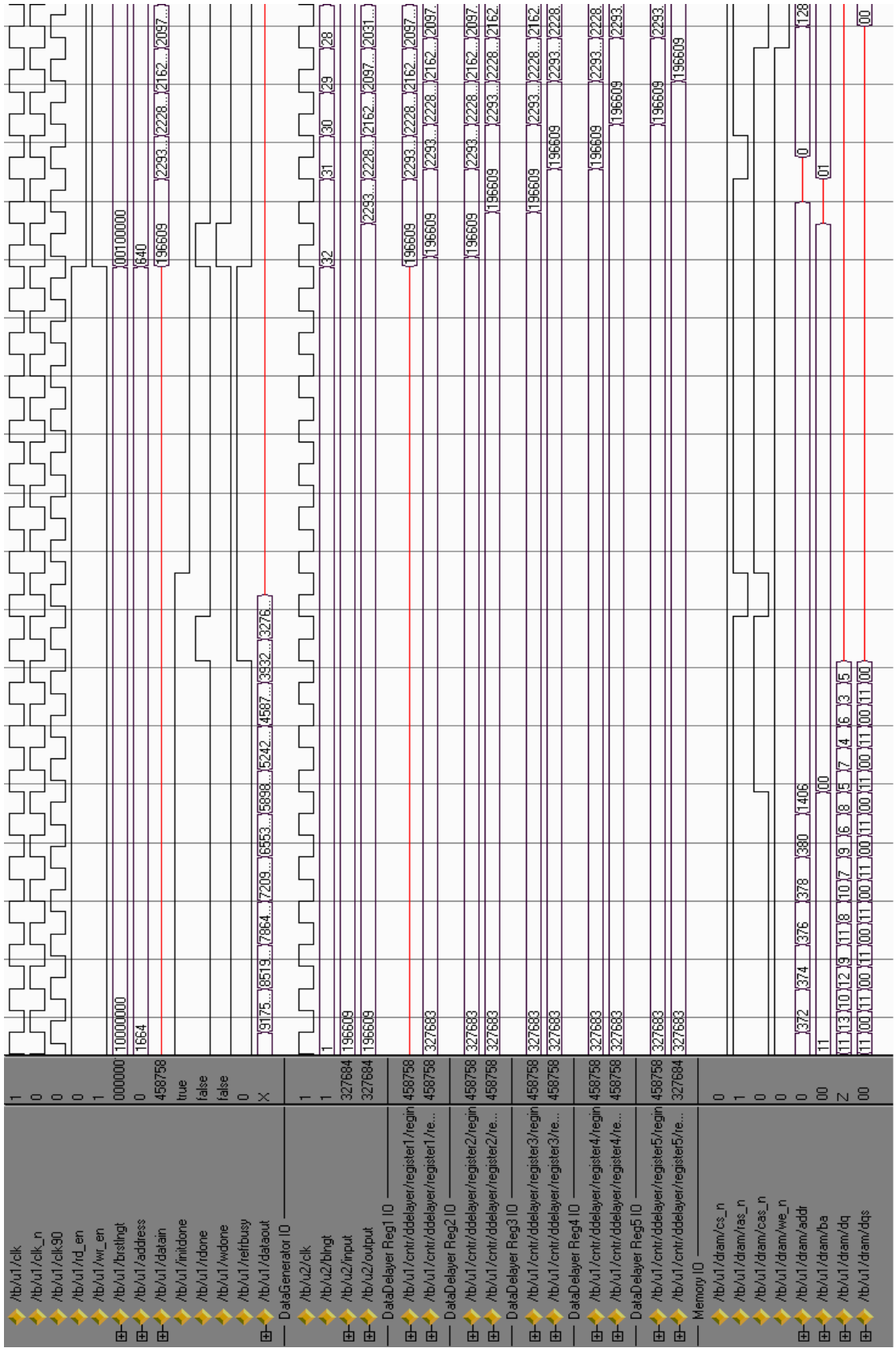
Αν αναλογιστεί κάποιος ότι η μέγιστη τιμή για το Burst Size που υποστηρίζει ο Ελεγκτής (δεδομένου ότι ο αριθμός των Στηλών σε μία Σειρά είναι 512 και η μνήμη λειτουργεί με Burst Length = 2) 256 (για την ακρίβεια 255) το κέρδος που προσδίδει στο σύστημα αυτή η δυνατότητα είναι πάρα πολύ μεγάλο! Στη χειρότερη περίπτωση όπου έχουμε 255 ξεχωριστές εγγραφές, σε εναλλάξ Bank, θα απαιτούνταν 1785 **κύκλοι** ρολογιού για την ολοκλήρωσή τους, ενώ στην περίπτωση όπου αυτές οι 256 εγγραφές μπορέσουν να πραγματοποιηθούν μέσα από μία εντολή εγγραφής με Burst Size = 255, ο χρόνος που θα απαιτούνταν για την ολοκλήρωση της εγγραφής θα ήταν μόλις 261 **κύκλοι** ρολογιού!

Τέλος αξίζει να παρατηρηθεί στο διάγραμμα της δεύτερης περίπτωσης η εισαγωγή νέων δεομένων στους καταχωρητές τα οποία παράγονται, όπως αναφέρθηκε και στην **Παράγραφο** 4.2.2, από τη μονάδα Data Generator καθώς και η ‘’αυτόματη’’ μείωση του μεγέθους του Burst (κατά 1) με την περάτωση κάθε εγγραφής δεδομένων εωσότου ολοκληρωθεί ο αριθμός των εγγραφών που καθορίζει το Burst.

4.5.5 **Παράδειγμα** 5

Το τελευταίο παράδειγμα έχει σαν μοναδικό σκοπό να παρουσιάσει στον αναγνώστη την λειτουργία του Refresh στο σύστημα. Αδιαφορώντας για την ακολουθία των εντολών που οδήγησαν το σύστημα μέχρι τη χρονική στιγμή που κρίνεται αναγκαία η εκτέλεση του Refresh παρουσιάζεται από το διάγραμμα που ακολουθεί (στο **Σχήμα** 4.9) μόνο η εισαγωγή του συστήματος στην εκτέλεση του και τα αντίστοιχα σήματα ελέγχου.

Σχήμα 4.9



Με την ολοκλήρωση της εκτέλεσης κάθε εντολής πραγματοποιείται και έλεγχος για το αν είναι απαραίτητη η πραγματοποίηση Refresh στο σύστημα (Αυτό διότι το Refresh δεν επιτρέπεται να διακόψει μία εντολή που έχει ήδη εκκινήσει την εκτέλεσή της). Όταν αυτή κριθεί απαραίτητη, η επόμενη προς εκτέλεση εντολή δεν εισάγεται στο σύστημα αλλά περιμένει εωσότου ολοκληρωθεί η διαδικασία του Refresh.

Όταν το σύστημα εκκινήσει τις διαδικασίες για την εκτέλεση του Refresh υψώνει στην τιμή ‘1’ ένα σήμα ελέγχου (RefrBusy) το οποίο εφόσον κατέχει αυτή την τιμή δεν επιτρέπει στο σύστημα την εισαγωγή νέας εντολής. Όταν το Refresh ολοκληρωθεί (έπειτα από 9 κύκλους ρολογιού) το σήμα RefrBusy ξαναπαίρνει την τιμή ‘0’ και πλέον μπορεί να εισαχθεί στο σύστημα η επόμενη εντολή.

5. Μέτρηση Απόδοσης Συστήματος

5.1 Περί Της Απόδοσης Του Συστήματος

Όπως έχει αναφερθεί αρκετές φορές στην πορεία αυτού του κειμένου, σκοπός της διπλωματικής εργασίας δεν είναι μόνο η επίτευξη της υλοποίησης ενός Ελεγκτή που θα επικοινωνεί απλά με τη μνήμη και θα μπορεί να πραγματοποιήσει μία απλή εγγραφή και μία απλή ανάγνωση με επιτυχία. Σκοπός είναι και η, κατά το δυνατόν, βελτιστοποίηση της ταχύτητας λειτουργίας και της απόδοσης του όλου συστήματος.

Μπορεί ο Ελεγκτής που υλοποιήθηκε να μην εκμεταλλεύεται στο μέγιστο τις δυνατότητες της μνήμης και για αυτό το λόγο μπορεί να μην οδηγεί στην μέγιστη δυνατή απόδοση του συστήματος. Αυτό όμως συνέβη αφ' ενός διότι όπως έχει προαναφερθεί το όλο σύστημα εξετάστηκε από την πλευρά του χρήστη οπότε και θα έπρεπε να είναι απλό στην κατανόηση και τον τρόπο λειτουργίας του και αφ' ετέρου διότι κάποιες ακόμη βελτιστοποιήσεις που παραλήφθηκαν κρίθηκαν (Βλέπε **Παράγραφο** 3.5) αρκετά σύνθετες από άποψη υλοποίησης και όχι τόσο χρήσιμες στο σύστημα από την οπτική του χρήστη.

Όλα αυτά σε συνδυασμό με την υποστήριξη εκτέλεσης εντολών με μεγάλο Burst, που η συνεισφορά του στην απόδοση του συστήματος είναι πολύ μεγάλη (Βλέπε **Παράγραφο** 4.5.4) μπορεί να μην οδηγούν στη μέγιστη δυνατή απόδοση του συστήματος, οδηγούν όμως σε μία πολύ καλή απόδοση και υλοποιούν ένα σύστημα αρκετά γρήγορο. Για την καλύτερη κατανόηση της απόδοσης του συστήματος κρίθηκε χρήσιμη η εξαγωγή μίας συνάρτησης η οποία θα μπορεί να μετρήσει με βάση κάποιες κύριες παραμέτρους του συστήματος, το χρόνο που χρειάζεται το σύστημα για την εκτέλεση κάποιων δοθεισών εντολών οπότε και κατά μέσο όρο το χρόνο εκτέλεσης μίας εντολής μέσα από το σύστημα που υλοποιήθηκε.

5.2 Εξαγωγή Της Συνάρτησης Μέτρησης Ταχύτητας

Για την εξαγωγή μίας τέτοιας συνάρτησης πρέπει να συνυπολογιστούν οι βασικές παράμετροι του συστήματος που επηρεάζουν την ταχύτητα του συστήματος.

Για αυτό το λόγο πρέπει αρχικά να γίνει διαχωρισμός του συνόλου / αριθμού των εντολών εγγραφής που εισάγονται στο σύστημα από τον αριθμό των εντολών ανάγνωσης. Αυτό διότι όπως έχει αναφερθεί και σε προηγούμενες παραγράφους ο χρόνος εκτέλεσης μίας απλής εντολής ανάγνωσης σε σχέση με τον χρόνο εκτέλεσης μίας απλής εντολής εγγραφής διαφέρουν κατά δύο κύκλους ρολογιού, παράγοντας αρκετά σημαντικός αν αναλογιστεί κανείς ότι σε μία μόλις εντολή υπάρχει διαφορά 2 κύκλων.

Ένας ακόμη παράγοντας που επηρεάζει σημαντικά την ταχύτητα του συστήματος είναι το μέγεθος του Burst. κάθε εντολή (είτε πρόκειται για εντολή ανάγνωσης είτε για εντολή εγγραφής) πραγματοποιεί τη μεταφορά τουλάχιστον μίας λέξης (32 bit) δεδομένων ανάλογα με το Burst Size της εντολής.

Όπως έχει περιγραφεί αναλυτικότατα στο **Κεφάλαιο** 3, υπάρχει η δυνατότητα διαμεσολάβησης από ένα έως τρεις κενούς κύκλους ρολογιού ανάμεσα σε δύο διαδοχικές εντολές. Αυτή δεν είναι μία σταθερή παράμετρος οπότε μπορεί να δοθεί στην εξίσωση σαν μία ‘’πιθανή’’ καθυστέρηση τριών κύκλων στο σύστημα. Έτσι θα εισαχθεί σαν μία καθυστέρηση τριών κύκλων συναρτήσει της πιθανότητας δύο διαδοχικές εντολές που θα εισαχθούν στο σύστημα να αφορούν το ίδιο Bank. Λαμβάνεται μόνο η περίπτωση των τριών κύκλων διότι θεωρείται ότι στην εισαγωγή των εντολών δεν παρεμβάλλονται κενοί χρόνοι και γενικά δεν δίνονται στη μνήμη κενές εντολές. Για αυτό το λόγο και η συνάρτηση λειτουργεί με βάση την αρχή ότι όλες οι εντολές εισάγονται στη μνήμη χωρίς κενό μεταξύ τους και εξυπηρετούνται από τη μνήμη, με τη σειρά που εισήχθησαν, αμέσως μόλις αυτό είναι δυνατό.

Πρέπει σε αυτό το σημείο να γίνει κατανοητό ότι ακόμη και αν υπήρχανε κενοί χρόνοι ανάμεσα σε δύο εντολές, η καθυστέρηση του 1, των 2 ή των 3 κύκλων ρολογιού όπως περιγράφηκε στην **Παράγραφο** 3.3 αφορά τη μεταφορά των δεδομένων σε περίπτωση εγγραφής αφού έτσι και αλλιώς η καθυστέρηση που αφορά τη μνήμη είναι στην ελάχιστη περίπτωση σταθερή και ίση με τρεις κύκλους.

Μία τελευταία παράμετρος που επηρεάζει την ταχύτητα του συστήματος είναι το Refresh. Αυτό κρίνεται απαραίτητο, όπως έχει αναλυθεί διεξοδικά στην **Παράγραφο** 3.3.6, κάθε 779 κύκλους του ρολογιού και διαρκεί για 9 κύκλους.

Με βάση όλες τις παραπάνω παραμέτρους προκύπτει και η παρακάτω συνάρτηση :

Για την εκτέλεση μίας εντολής εγγραφής απαιτούνται Wi κύκλοι ρολογιού:

$$Wi = 6 + \#Word_w + 3P$$

Ομοίως για την εκτέλεση μίας εντολής ανάγνωσης απαιτούνται Ri κύκλοι ρολογιού:

$$Ri = 8 + \#Word_r + 3P$$

Στις παραπάνω εξισώσεις το P αποτελεί την πιθανότητα η εντολή i να απευθύνεται στο ίδιο Bank με την εντολή i-1 οπότε και πρέπει να καθυστερήσει τρεις κύκλους πριν εκκινήσει την εκτέλεσή της ενώ η παράμετρος $\#Word_i$ ορίζει το μέγεθος του Burst του κάθε είδους και της αντίστοιχης εντολής. Τα 6 και 8 είναι οι σταθεροί κύκλοι, εκτός του τελευταίου που πραγματοποιεί και τη μεταφορά των δεδομένων, στην περίπτωση εγγραφής και ανάγνωσης αντίστοιχα.

Ανάγοντας τις παραπάνω συναρτήσεις για τον αριθμό Nr εντολών ανάγνωσης προκύπτει η εξίσωση :

$$fr = \sum_{i=1}^{Nr} Ri = \sum_{i=1}^{Nr} (8 + \#Word_r + 3P) =>$$

$$fr = 8Nr + 3PNr + \sum_{i=1}^{Nr} \#Word_r$$

Αντίστοιχα για την περίπτωση Nw εντολών εγγραφής προκύπτει η εξίσωση :

$$fw = \sum_{i=1}^{Nw} Wi = \sum_{i=1}^{Nw} (6 + \#Word_w + 3P) =>$$

$$fw = 6Nw + 3PNw + \sum_{i=1}^{Nw} \#Word_w$$

Συνδυάζοντας τις δύο περιπτώσεις έτσι ώστε να εξαχθεί μόνο μία συνάρτηση που θα εξάγει την συνολική χρονική καθυστέρηση (σε κύκλους ρολογιού) N εντολών, όπου **N** = **N**r + Nw (Nr ο αριθμός των εντολών ανάγνωσης και Nw ο αριθμός των εντολών εγγραφής σε ένα παράδειγμα) προκύπτει η ακόλουθη συνάρτηση :

$$f = fr + fw =>$$

$$f = 8Nr + 6Nw + 3P(Nr + Nw) + \sum_{i=1}^{N} \#Word =>$$

$$f = 8Nr + 6Nw + 3PN + \sum_{i=1}^{N} \#Word$$

όπου $\sum_{i=1}^{N} \#Word = \sum_{i=1}^{Nr} \#Word_r + \sum_{i=1}^{Nw} \#Word_w$

Τέλος για να εισαχθεί στην παραπάνω εξίσωση και ο παράγοντας του Refresh μπορεί απλά να προκύψει από την απλή μέθοδο των τριών ότι, εφόσον στους 779 κύκλους ρολογιού κρίνεται απαραίτητη η διενέργεια Refresh (Βλέπε **Παράγραφο** 3.3.6) τότε στους 779 + 9 = 788 κύκλους ρολογιού οι 9 απαιτούνται για το Refresh. Συνεπώς στην περίπτωση f κύκλων του ρολογιού οι κύκλοι που θα απαιτούνται για Refresh θα είναι :

$$x = \frac{9f}{788}$$

Προσθέτοντας την παραπάνω παράμετρο στην συνάρτηση f προκύπτει και η τελική συνάρτηση μέτρησης της ταχύτητας του συστήματος. Έτσι :

$$f = 8Nr + 6Nw + 3PN + \sum_{i=1}^{N} \#Word + x \Rightarrow$$

$$f = 8Nr + 6Nw + 3PN + \sum_{i=1}^{N} \#Word + \frac{9f}{788} \Rightarrow$$

$$f\left(1 - \frac{9}{788}\right) = 8Nr + 6Nw + 3PN + \sum_{i=1}^{N} \#Word \Rightarrow$$

$$f = \left(8Nr + 6Nw + 3PN + \sum_{i=1}^{N} \#Word\right) / 0.99$$

Αξίζει να αναφερθεί ότι ο συνδυασμός του αριθμού των εντολών σε συνδυασμό με τον αριθμό των λέξεων που μεταφέρονται μέσα από τις εντολές δηλώνουν τη μέση τιμή του μεγέθους του Burst στο σύστημα.

Για την επαλήθευση της ορθής λειτουργίας της παραπάνω εξίσωσης χρησιμοποιήθηκε από άποψη προσομοίωσης ένα παράδειγμα αρκετά σύνθετο ώστε να συνδυάζει όλες τις παραπάνω παραμέτρους. Αρχικά έγινε υπολογισμός, με βάση την εξίσωση που προέκυψε, της θεωρητικής τιμής του αριθμού των κύκλων του ρολογιού που απαιτούνται για την ολοκλήρωση της εκτέλεσης του προγράμματος και στη συνέχεια η τιμή αυτή συγκρίθηκε με το αποτέλεσμα που προέκυψε από το διάγραμμα χρονισμού κατά την εκτέλεση / προσομοίωση του παραδείγματος.

5.2.1 **Παράδειγμα Επαλήθευσης Της Εξίσωσης**

```
BEGIN
    Initialize ( '0', "010" );
    Write (1,  "00000000000000110000000000000001",  "000000000001101000000011");
    Write (2,  "00000000000001010000000000000100",  "000000000000000000000000");
    Write (8,  "00000000000010000000000000000010",  "000000000000001000000000");
    Read  (2,                                       "000000000000000000000000");
    Read  (1,                                       "000000000001101000000011");
    Write (16, "00000000000000110000000000000001", "000000000000000010000000");
    Read  (1,                                       "000000000000000010000000");
```

```
    Read  (8,                                                    "00000000000000001000000000");
    Read  (16,                                                   "00000000000000000010000000");
    Write (192, "00000000000000011000000000000000001", "00000000000000001010000000");
    Read  (192,                                                  "00000000000000001010000000");
    Write (128, "00000000000000011000000000000000001", "00000000000000011010000000");
    Read  (128,                                                  "00000000000000011010000000");
    Write (32,  "00000000000000011000000000000000001",  "00000000000000010010000000");
    Read  (32,                                                   "00000000000000010010000000");
    Write (1,   "00000000000000011000000000000000001",   "00000000000000010010000000");
    Read  (1,                                                    "00000000000000010010000000");
    stim_done <= true;
    WAIT;
  END PROCESS;
```

Με βάση το παραπάνω σύνολο εντολών διακρίνεται εύκολα ότι ο αριθμός των εντολών εγγραφής προς τη μνήμη είναι 8 και διακρίνονται με κόκκινο χρώμα (Nw = 8), ο αριθμός των εντολών ανάγνωσης είναι 9 και φαίνονται με μαύρο χρώμα στο παράδειγμα(Nr = 9). Το σύνολο των εντολών είναι 17 (N = 17) ενώ η πιθανότητα δύο διαδοχικές εντολές να απευθύνονται στο ίδιο Bank είναι 6 / 17 (P = 6 / 17) όπως διακρίνεται και με μπλε χρώμα από το παράδειγμα. Τέλος ο συνολικός αριθμός των λέξεων που θα μεταφερθούν είτε από, είτε προς τη μνήμη σε αυτό το παράδειγμα προκύπτει εύκολα αν προστεθούν όλα τα μεγέθη Burst όλων των εντολών. Με αυτό τον τρόπο προκύπτει ότι ο συνολικός αριθμός των λέξεων είναι 761 (#Word = 761) και η μέση τιμή του μεγέθους του Burst στο παράδειγμα είναι 761 / 17 = 45.

Με βάση τα παραπάνω στοιχεία, από την εξίσωση που παράχθηκε σε αυτό το κεφάλαιο προκύπτει ότι ο συνολικός χρόνος καθυστέρησης από την εκκίνηση εκτέλεσης της πρώτης εντολής του δοθέντος παραδείγματος εωσότου ολοκληρωθεί και η εκτέλεση της τελευταίας εντολής θα είναι :

$$f = \text{Clock Cycles} = \left(6*8 + 8*9 + 3*\frac{6}{17}*16 + 761 \right) / 0.99 = 908 \textbf{ κύκλοι ρολογιού}$$

Το παραπάνω αποτέλεσμα συμβαδίζει με τις τιμές που προκύπτουν κατά την εκτέλεση της προσομοίωσης του παραδείγματος. Το διάγραμμα χρονισμού που προκύπτει από την εκτέλεση της προσομοίωσης του παραδείγματος είναι πολύ μεγάλο για να χωρέσει στο κείμενο αυτό, οπότε και απλά αναφέρονται οι τιμές που προκύπτουν κατά την εκτέλεσή του.

Η εκκίνηση της εκτέλεσης της πρώτης εντολής αρχίζει στα 201671250 ps (εφόσον τότε ολοκληρώνεται η αρχικοποίηση του συστήματος), ενώ η τελευταία εντολή ολοκληρώνει την εκτέλεσή της στα 208481250 ps. Αξίζει να αναφερθεί ότι με την ολοκλήρωση της εκτέλεσης της εντολή ανάγνωσης με Burst Size 128 εκκινεί η διαδικασία του Refresh (207641250 ps).

Από τα παραπάνω αποτελέσματα προκύπτει ότι η διάρκεια εκτέλεσης του προγράμματος είναι 208481250 – 201671250 = 6810 ns που σε κύκλους ρολογιού αντιστοιχεί σε 6810 / 7.5 = 908 **κύκλοι ρολογιού**. Παρατηρείται λοιπόν ότι τα θεωρητικά αποτελέσματα συμβαδίζουν απόλυτα με τα πειραματικά και πιστοποιείται η σωστή εξαγωγή και λειτουργία της εξίσωσης (Φυσικά τα παραδείγματα που χρησιμοποιήθηκαν για την επαλήθευση της σωστής λειτουργίας της εξίσωσης ήταν περισσότερα. Το συγκεκριμένο είναι απλά ένα από αυτά που να καλύπτει όλες τις παραμέτρους).

Ακόμη αξίζει να παρατηρηθεί ότι η εντολή Refresh εισάγεται στο σύστημα έπειτα από 207641250 – 201671250 = 5970 ns που αντιστοιχούν σε 5970 / 7.5 = 796 κύκλους ρολογιού. Συνεπώς ο έλεγχος (η τιμή του Refresh Counter > 779) διαπίστωσε ότι κρίνεται απαραίτητη η διεξαγωγή Refresh στο σύστημα με την ολοκλήρωση της εντολής (εφόσον δεν επιτρέπεται να διακοπεί μία εντολή που εκτελείται ήδη και εφόσον 796 > 779) και πέρασε στην εκτέλεση του Refresh καθυστερώντας την εκκίνηση εκτέλεσης της επόμενης εντολής κατά 9 κύκλους. Με την ολοκλήρωση της εκτέλεσης του Refresh πέρασε στο σύστημα η επόμενη προς εκτέλεση εντολή.

Το τελικό αλλά και πολύ σημαντικό συμπέρασμα που μπορεί να προκύψει από το παραπάνω παράδειγμα είναι ο μέσος χρόνος που χρειάζεται για τη μεταφορά, είτε από είτε προς τη μνήμη, 32 bit δεδομένων (μίας λέξης) μέσα από το σύστημα που υλοποιήθηκε. Αν διαιρεθεί ο συνολικός αριθμός των κύκλων του ρολογιού που προέκυψε στο παραπάνω παράδειγμα με το συνολικό αριθμό των λέξεων που μεταφέρθηκαν σε αυτό το χρονικό διάστημα τότε προκύπτει **ο μέσος αριθμός κύκλων ρολογιού για την μεταφορά μίας λέξης** 908 / 761 = 1.2 **κύκλοι ρολογιού**. Επίσης ο μέσος αριθμός κύκλων που απαιτήθηκαν για την εκτέλεση μίας εντολής είναι 908 / 17 = 53 κύκλοι ρολογιού.

Στη συνέχεια ακολουθούν κάποια παραδείγματα που παρουσιάζουν την ταχύτητα του συστήματος (σε κύκλους ρολογιού) συναρτήσει των παραμέτρων της εξίσωσης ή του συνδυασμού κάποιων από αυτές τις παραμέτρους. Σκοπός των παραδειγμάτων αυτών είναι η καλύτερη κατανόηση της επιρροής κάθε μίας από τις παραμέτρους αυτές, και σε ορισμένες περιπτώσεις ο συνδυασμός τους, στην ταχύτητα του συστήματος.

5.3 **Παραδείγματα**

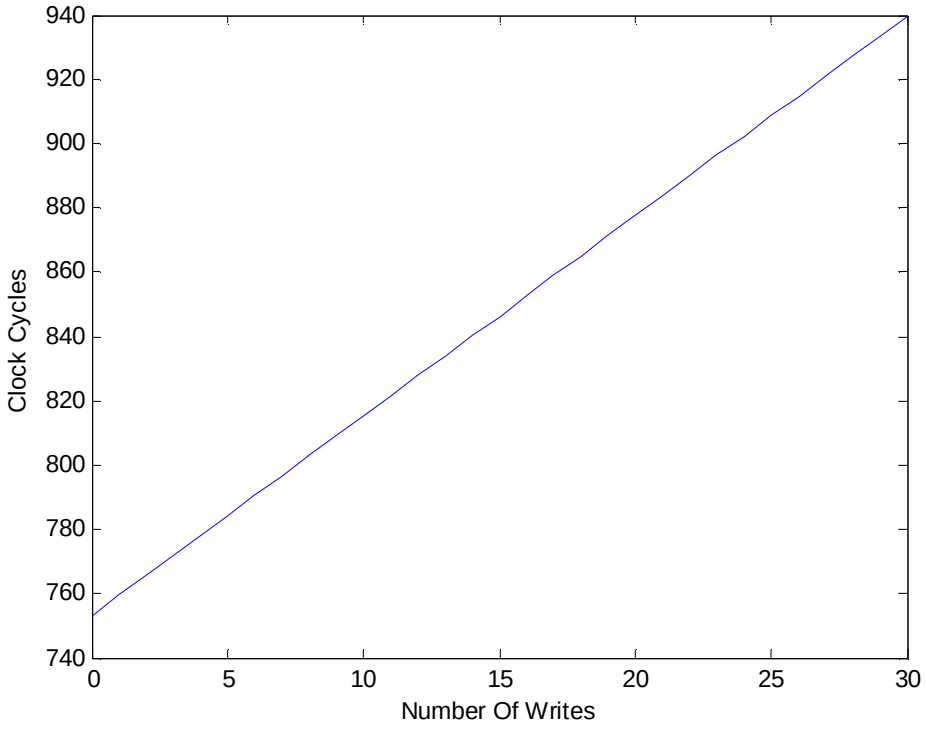
Τα παραδείγματα που ακολουθούν όπως έχει ήδη αναφερθεί παρουσιάζουν με διαγράμματα την επίπτωση της μεταβολής των παραμέτρων της εξίσωσης στην ταχύτητα του συστήματος και υλοποιήθηκαν μέσα από το εργαλείο της Matlab.

5.3.1 **Μεταβολή Του** Nw **ή του** Nr

Ίσως όχι τόσο σημαντικός έλεγχος, όσο αυτοί που θα ακολουθήσουν, αλλά η σύγκριση της μεταβολής στην ταχύτητα του συστήματος με την μεταβολή του αριθμού των εντολών ανάγνωσης (Nr) σε σχέση με τη μεταβολή των εντολών εγγραφής (Nw) είναι άξια προσοχής :

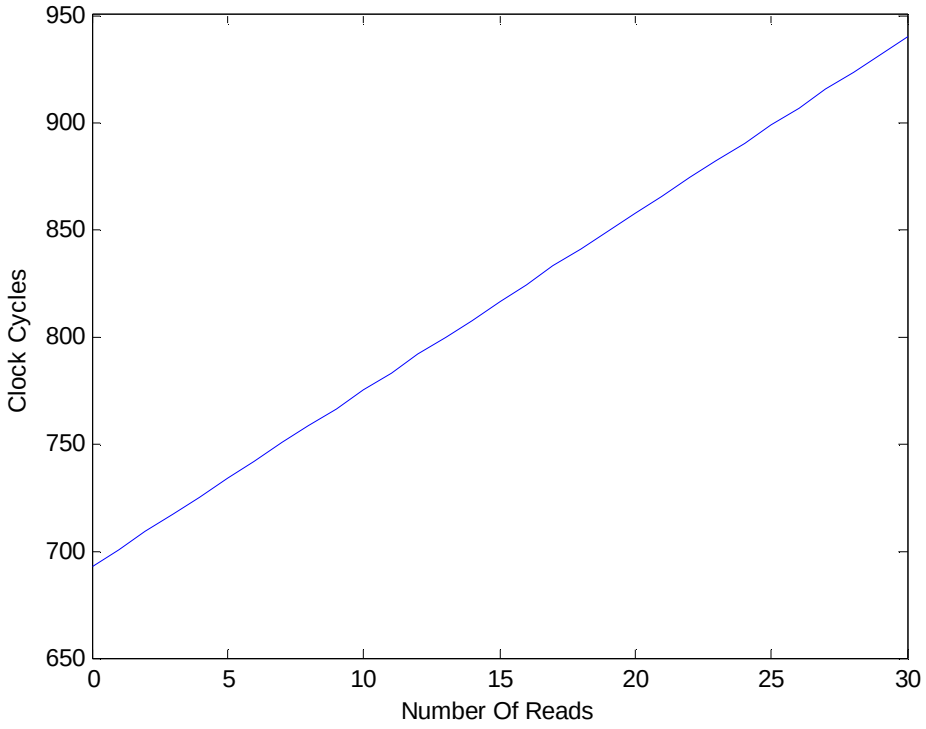
Μεταβολή Αριθμού Εντολών Εγγραφής :

```
Nr = 30;
Word = 500;
P = 1/20;
Nw = 0:30;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(Nw,F);
xlabel('Number Of Writes');
ylabel('Clock Cycles');
```



Μεταβολή Αριθμού Εντολών Ανάγνωσης :

```
Nw = 30;
Word = 500;
P = 1/20;
Nr = 0:30;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(Nr,F);
xlabel('Number Of Reads');
ylabel('Clock Cycles');
```

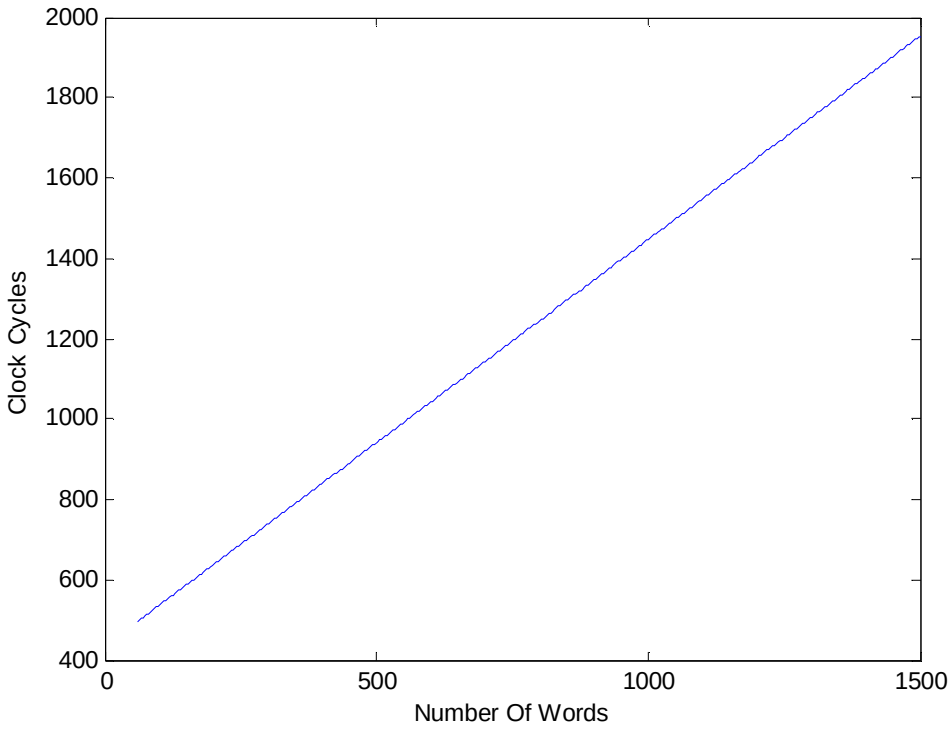


Διατηρώντας στην ίδια τιμή τις υπόλοιπες παραμέτρους τις εξίσωσης είναι εύκολο να διακρίνει κανείς ότι όσο μεγαλύτερος ο αριθμός των αναγνώσεων σε σχέση με τον αριθμό των εγγραφών , τόσο μεγαλύτερη και η καθυστέρηση του συστήματος εφόσον κάθε εντολή ανάγνωσης απαιτεί 2 επιπλέον κύκλους για την εκτέλεσή της σε σχέση με μία εντολή εγγραφής. Αυτό γίνεται πολύ εύκολα αντιληπτό από την μεγαλύτερη κλίση της δεύτερης καμπύλης. Επίσης αν γίνει σύγκριση των ακραίων περιπτώσεων όπου η μεταβαλλόμενη παράμετρος λαμβάνει την τιμή ‘0’ ενώ η σταθερή την τιμή ‘30’, στην πρώτη περίπτωση ο αριθμός των κύκλων του ρολογιού που απαιτούνται για την εκτέλεση των 30 αναγνώσεων είναι περίπου 753 (πρώτο διάγραμμα) ενώ στην περίπτωση εκτέλεσης 30 εγγραφών) 692 (δεύτερο διάγραμμα).

5.3.2 **Μεταβολή Του Αριθμού Των Λέξεων** (#Word)

Στην περίπτωση αυτή, η παράμετρος που μεταβάλλεται είναι ο αριθμός των λέξεων που πρέπει να μεταφερθούν είτε προς είτε από τη μνήμη κατά τη διάρκεια της εκτέλεσης του προγράμματος. Όπως είναι λογικό και αυτή η παράμετρος από μόνη της δεν αποτελεί κάποιο ιδιαίτερο στοιχείο αφού είναι αναμενόμενο, εφόσον όλες οι υπόλοιπες παράμετροι είναι σταθερές, με την αύξηση του αριθμού των λέξεων κατά ‘1’ αυξάνεται κατά ένα και ο αριθμός των κύκλων του ρολογιού :

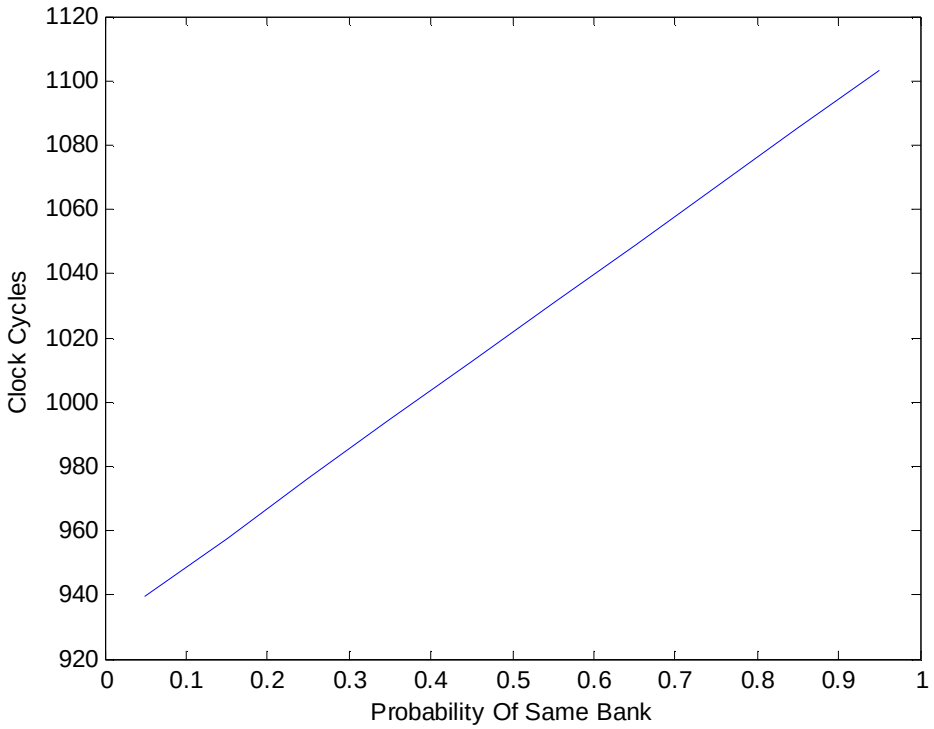
```
Nw = 30;
Word = 60:1:1500;
P = 1/20;
Nr = 30;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(Word,F);
xlabel('Number Of Words');
ylabel('Clock Cycles');
```



5.3.3 **Μεταβολή Της Πιθανότητας (P)**

Στην περίπτωση αυτή, η παράμετρος που μεταβάλλεται είναι η πιθανότητα δύο συνεχόμενες / διαδοχικές εντολές να απευθύνονται στο ίδιο Bank. Εφόσον όλες οι υπόλοιπες παράμετροι είναι σταθερές, η μεταβολή της πιθανότητας αυτής, όπως διακρίνεται και από το παρακάτω διάγραμμα είναι υψίστης σημασίας στον τομέα της απόδοσης. Όσο μεγαλύτερη είναι αυτή η πιθανότητα τόσο μεγαλύτερη και η χρονική καθυστέρηση για την εκτέλεση των εντολών. Παρατηρείται ότι για την εκτέλεση 30 εντολών Εγγραφής και 30 εντολών Ανάγνωσης και συνολική μεταφορά 500 λέξεων, αν η πιθανότητα αυτή είναι 1/20, η εκτέλεση των εντολών θα ολοκληρωθεί σε 940 κύκλους ρολογιού ενώ αν η πιθανότητα αυτή είναι 1/2 (δηλαδή σχεδόν κάθε δεύτερη εντολή να απευθύνεται στο ίδιο Bank), η εκτέλεση των εντολών θα ολοκληρωθεί σε 1022 κύκλους!

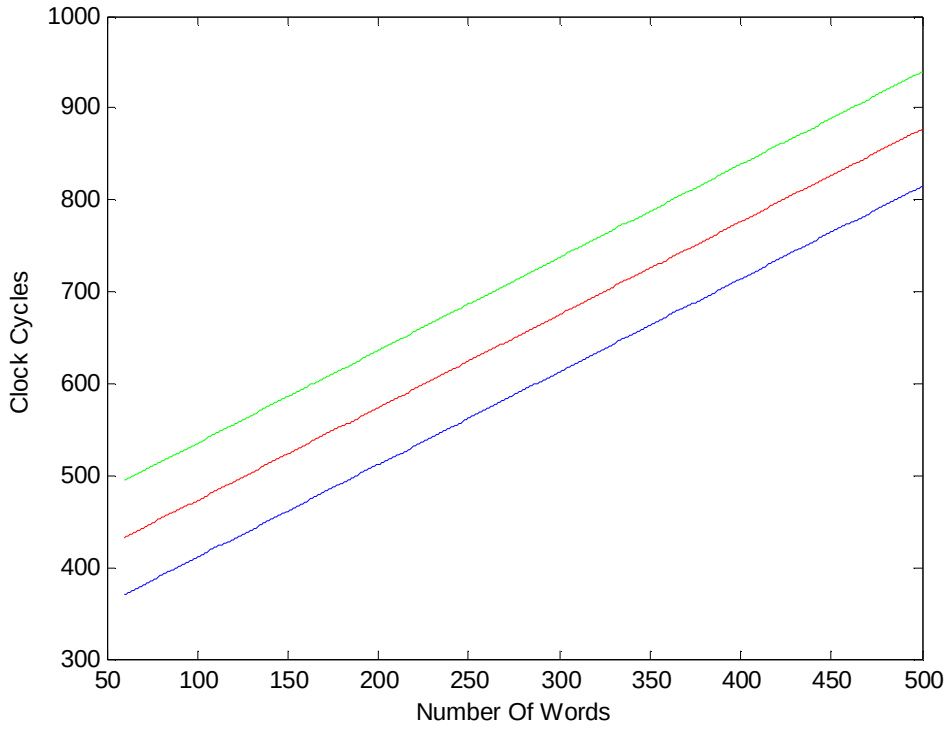
```
Nw = 30;
Word = 500;
P = 1/20:0.1:1;
Nr = 30;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(P,F);
xlabel('Probability Of Same Bank');
ylabel('Clock Cycles');
```



5.3.4 **Μεταβολή Του Αριθμού Των Λέξεων (#Word) Συναρτήσει Της Μεταβολής Του Αριθμού Των Εντολών Εγγραφής (Nw)**

Σε αυτό το παράδειγμα διακρίνονται μέσα στο διάγραμμα που προκύπτει τρεις διαφορετικές καμπύλες. Οι καμπύλες αυτές αναπαριστούν τον αριθμό των κύκλων του ρολογιού που απαιτούνται σε περίπτωση μεταβολής του αριθμού των προς μεταφορά λέξεων σε σχέση όμως με το συνολικό αριθμό των εντολών μέσα από τις οποίες θα πραγματοποιηθεί η μεταφορά. Ο συνδυασμός αυτών των δύο παραμέτρων είναι και αυτός που καθορίζει το μέσο μέγεθος Burst με το οποίο θα πραγματοποιηθούν οι προσβάσεις στη μνήμη. Στην πρώτη περίπτωση ο αριθμός των εντολών είναι 10, στην δεύτερη 20 και στην τρίτη 30. Με αυτόν τον τρόπο διακρίνεται και η συμβολή του Burst στο σύστημα :

```
Word = 60:500;
P = 1/20;
Nr = 30;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(Word,F);
xlabel('Number Of Words');
ylabel('Clock Cycles');
```



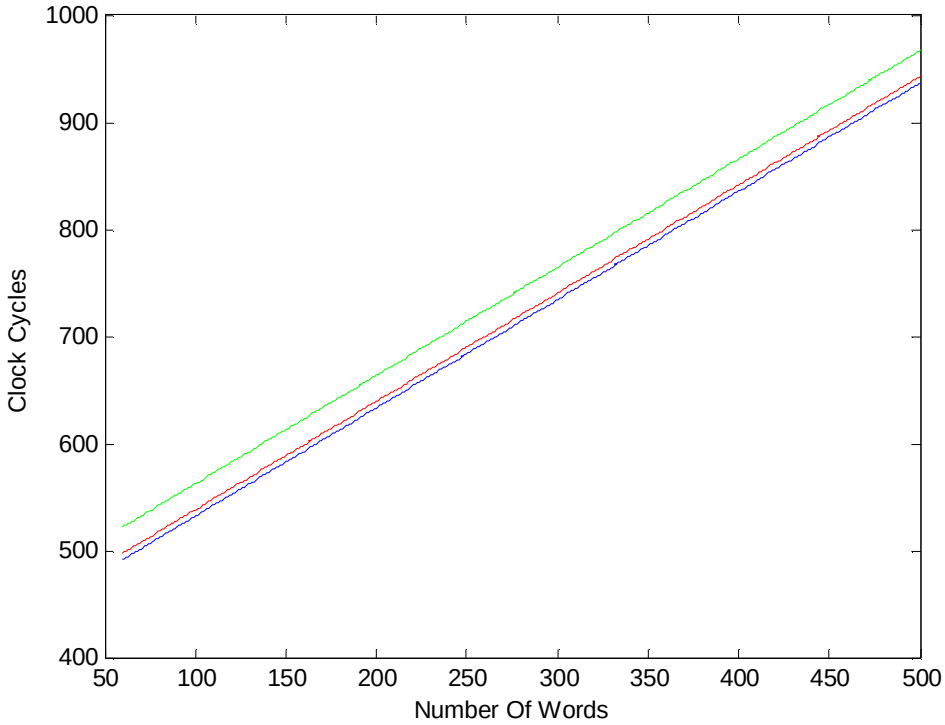
Όπου Nw =10, Nw = 20 και Nw = 30.

Διακρίνεται με αυτό τον τρόπο ότι στην περίπτωση που για παράδειγμα επιθυμείται η μεταφορά 500 λέξεων, αν αυτό πραγματοποιηθεί με 30 + 10 = 40 εντολές απαιτούνται περίπου 815 κύκλοι ρολογιού ενώ στην περίπτωση που αυτό πραγματοποιείται μέσα από 30 + 30 = 60 εντολές (μικρότερος μέσος όρος μεγέθους Burst) απαιτούνται 929 κύκλοι ρολογιού!

5.3.5 **Μεταβολή Του Αριθμού Των Λέξεων (#Word) Συναρτήσει Της Μεταβολής Της Πιθανότητας (P)**

Σε αυτό το παράδειγμα επίσης διακρίνονται μέσα στο διάγραμμα που προκύπτει τρεις διαφορετικές καμπύλες. Οι καμπύλες αυτές αναπαριστούν τον αριθμό των κύκλων του ρολογιού που απαιτούνται σε περίπτωση μεταβολής του αριθμού των προς μεταφορά λέξεων σε σχέση όμως με την πιθανότητα δύο διαδοχικές εντολές να αφορούν το ίδιο Bank. Στην πρώτη περίπτωση η πιθανότητα είναι αρκετά μικρή έχοντας την τιμή 1 / 30, στη συνέχεια λαμβάνει την τιμή 1 / 15 και τέλος την τιμή 1 / 5 :

```
Nw = 30;
Word = 60:500;
Nr = 30;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(Word,F);
xlabel('Number Of Words');
ylabel('Clock Cycles');
```



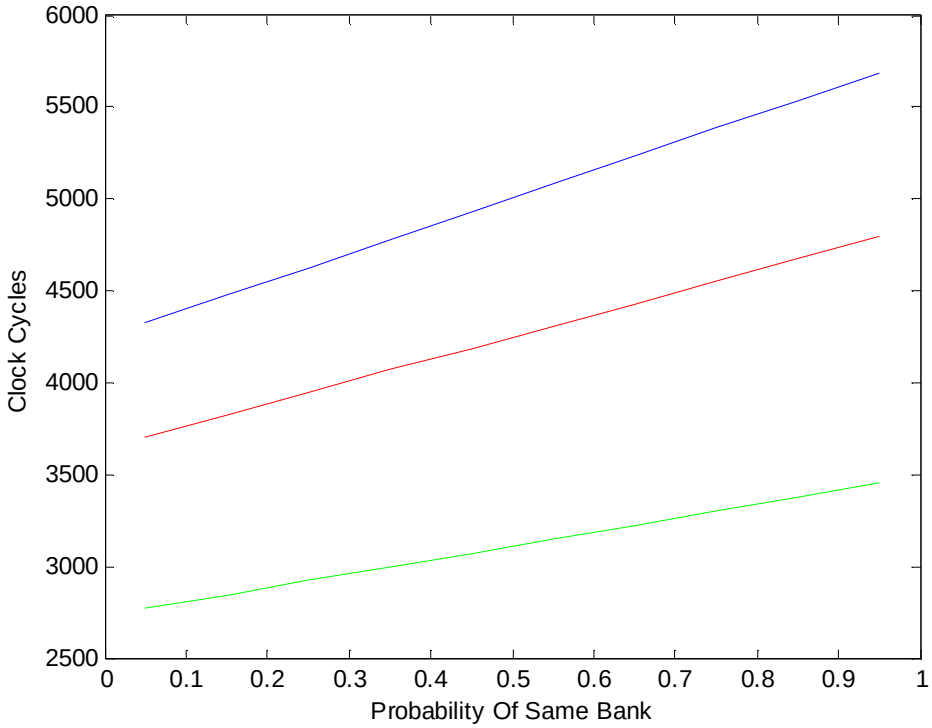
Όπου P =1 / 30, P = 1 / 15 και P = 1 / 5.

Όπως ήταν και αναμενόμενο, από το παραπάνω διάγραμμα διακρίνεται καθαρά ότι όσο μεγαλύτερη η πιθανότητα δύο διαδοχικές εντολές να απευθύνονται στο ίδιο Bank (δηλαδή όσο περισσότερο πλησιάζει η πιθανότητα την τιμή 1), τόσο πιο ευκρινής είναι και η καθυστέρηση του συστήματος!

5.3.6 Μεταβολή Της Πιθανότητας (P) Συναρτήσει Της Μεταβολής Του Αριθμού Των Εντολών Εγγραφής (Nw)

Στο τελευταίο αυτό παράδειγμα διακρίνονται επίσης μέσα στο διάγραμμα που προκύπτει τρεις διαφορετικές καμπύλες. Οι καμπύλες αυτές αναπαριστούν τον αριθμό των κύκλων του ρολογιού που απαιτούνται σε περίπτωση μεταβολής της πιθανότητας δύο διαδοχικές εντολές να αφορούν το ίδιο Bank σε σχέση όμως με τον συνολικό αριθμό των εντολών. Αξίζει εδώ να παρατηρηθεί ότι αυτή αποτελεί και τη μοναδική περίπτωση που η μεταβολή της καμπύλης δεν αποτελεί απλά μία παράλληλη μετακίνησή της στον κάθετο άξονα αλλά επηρεάζεται και η κλίση της καμπύλης. Αυτό διότι δεν πρόκειται πλέον για μεταβολή δύο ανεξάρτητων παραμέτρων που στη συνέχεια απλά προστίθενται αλλά μεταβάλλονται δύο παράμετροι που στην εξίσωση πολλαπλασιάζονται μεταξύ τους. Όσο λοιπόν η πιθανότητα αυξάνει και πλησιάζει την τιμή 1 και όσο ο αριθμός του συνόλου των εντολών αυξάνει, τόσο παίρνει σημαντικότερες διαστάσεις η εισαγωγή αυτής της καθυστέρησης στο σύστημα, κάτι που μπορεί να γίνει εύκολα διακριτό και μέσα από το διάγραμμα. Στην πρώτη περίπτωση ο αριθμός των εντολών είναι 300, στην δεύτερη 200 και στην τρίτη 50 :

```
Word = 800;
P = 1/20:0.1:1;
Nr = 200;

F = (6*Nw + 8*Nr + Word + 3*P*(Nr + Nw))/0.98857868;
plot(P,F);
xlabel('Probability Of Same Bank');
ylabel('Clock Cycles');
```



Όπου Nw =300, Nw = 200 και Nw = 50.

6. Επίλογος

Στην διπλωματική αυτή εργασία έγινε προσπάθεια υλοποίησης ενός Ελεγκτή που θα μπορούσε να χειριστεί με αξιώσεις μία DDR DRAM μνήμη και τις δυνατότητές της. Για τον λόγο αυτό ήταν απαραίτητη η κατανόηση και μελέτη τόσο του τρόπου λειτουργίας της μνήμης όσο και των δυνατοτήτων της. Μία τέτοια μνήμη έχει αρκετά σύνθετο τρόπο λειτουργίας προκειμένου να παρέχει τη δυνατότητα μεταφοράς δύο δεδομένων πληροφορίας σε ένα μόνο κύκλο ρολογιού.

Ένα από τα πιο σύνθετα προβλήματα αποτέλεσε και η κατανόηση του τρόπου λειτουργίας όλων των σημάτων της μνήμης. Σε κάθε εντολή και κάθε στιγμή της μνήμης αυτά τα σήματα έπρεπε να συνδυάζονται κατάλληλα έτσι ώστε η πληροφορία που θα φτάνει στη μνήμη να είναι χρήσιμη. Για αυτό το λόγο από άποψη υλοποίησης ένα από τα δυσκολότερα σημεία ήταν εφόσον κατανοήθηκε στο μέγιστο δυνατό η λειτουργία των σημάτων, να μπορέσουν αυτά να συνδυαστούν και να συγχρονιστούν κατάλληλα έτσι ώστε να φθάνουν με σωστή σειρά και την κατάλληλη χρονική στιγμή στη μνήμη για να μπορέσει αυτή να ''κατανοήσει'' την λειτουργία που της ζητείται και να την εκτελέσει. Σε διαφορετική περίπτωση δεν θα ήταν προβλέψιμος / προσδιορίσιμος ο τρόπος λειτουργίας της μνήμης οπότε και δεν θα λειτουργούσε σωστά.

Οι παραπάνω λόγοι σε συνδυασμό με την αρκετά σύνθετη λειτουργία του συστήματος μπορεί να μην οδήγησαν στην υλοποίηση του καλύτερου δυνατού Ελεγκτή της μνήμης. Θα μπορούσε ίσως να κατασκευαστεί ένας πιο αποδοτικός Ελεγκτής που θα αξιοποιούσε όλες τις δυνατότητες της μνήμης. Παρ' όλα αυτά ο Ελεγκτής που υλοποιήθηκε είναι αρκετά αποδοτικός σε σχέση με έναν απλό Ελεγκτή βασικής διαχείρισης της μνήμης για απλές εγγραφές και αναγνώσεις και αρκετά απλός στην κατανόηση του τρόπου λειτουργίας του.

Στο Κείμενο της περιγραφής του συστήματος έγινε προσπάθεια περιγραφής της λειτουργίας τόσο της μνήμης και του Ελεγκτή όσο και ολόκληρου του συστήματος με τον απλούστερο δυνατό τρόπο. Αυτό διότι βασικός σκοπός είναι η καλύτερη και ευκολότερη κατανόηση της διαχείρισης και λειτουργίας της υλοποίησης. Με αυτόν τον τρόπο το παραπάνω κείμενο μπορεί να φανεί χρήσιμο τόσο κάποιον αναγνώστη / μελετητή που ενδιαφέρεται απλά να ενημερωθεί και να κατανοήσει τον τρόπο λειτουργίας του συστήματος, όσο και σε κάποιον ερευνητή. Στην περίπτωση του δεύτερου τίθενται και γερές βάσεις για ενδεχόμενη χρήση ή ακόμη και επέκταση ή βελτιστοποίηση του Ελεγκτή της DDR DRAM μνήμης (Πιθανή Μελλοντική Εργασία).

Πιο συγκεκριμένα μια μελλοντική εργασία θα μπορούσε, χρησιμοποιώντας ως βάση τον Ελεγκτή που υλοποιήθηκε σε αυτή τη διπλωματική εργασία, να επεκτείνει την λειτουργία του Ελεγκτή συμπεριλαμβάνοντας και όλες τις υπόλοιπες δυνατότητες τις μνήμης που δεν αξιοποιήθηκαν (**Παράγραφος** 3.5) έτσι ώστε να δημιουργηθεί ένας ''Βέλτιστος'' Ελεγκτής. Τόσο το κείμενο της εργασίας όσο και η περιγραφή της υλοποίησης μπορούν να δώσουν αρκετές πληροφορίες για το πώς λειτουργεί η μνήμη και πως θα πρέπει να λειτουργεί ο Ελεγκτής έτσι ώστε να επιτευχθεί αυτό.

Σε αυτή την περίπτωση λοιπόν η παραπάνω εργασία προσφέρει και σημαντική βοήθεια στην κατανόηση της βασικής λειτουργίας και της φιλοσοφίας ενός τέτοιου συστήματος. Μία ακόμη δυνατή επέκταση του υλοποιηθέντος συστήματος είναι η προσαρμογή του ώστε να επεκταθεί η χρήση του Ελεγκτή σε DIMMs. Ο Ελεγκτής που υλοποιήθηκε έχει τη δυνατότητα να επικοινωνεί με ένα Chip μίας DDR DRAM μνήμης. Θα ήταν λοιπόν δυνατή η επέκτασή του έτσι ώστε να ελέγχει τη λειτουργία ενός συνόλου από τέτοια Chip τα οποία υπάρχουν πάνω σε ένα DIMM μνήμης.

References

1. www.micron .com

2. William Stallings - Οργάνωση & Αρχιτεκτονική Υπολογιστών (6η Έκδοση)

3. Stephen Brown & Zvonko Vranesic – Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL

4. http://en.wikipedia.org/wiki/Dynamic_Random_Access_Memory

5. http://www.pantherproducts.co.uk/Articles/What_is/What_is_RAM.shtml

6. http://apprais.tripod.com/TIPS/42WHAT%20IS%20RAM.htm

7. http://kb.iu.edu/data/ahty.html?cust=12464

8. http://www.tech-faq.com/ram.shtml

9. http://www.pantherproducts.co.uk/Articles/What_is/What_is_RAM.shtml

10. http://inventors.about.com/library/weekly/aa100898.htm

11. http://www.ddrmemoryupgrades.com/what_is_ram.html

12. http://www.ivitex.com/faq1.htm

13. http://www.necel.com/memory/english/products/sram/ssram-info.html

14. www.eetimes.de/story/showArticle.jhtml?articleID=19502732

15. http://www.webopedia.com/TERM/R/RAM.html

16. http://searchwinsystems.techtarget.com/originalContent/ 0,289142,sid68_gci1073862,00.html

17. http://www.qdrsram.com/news/1_10_2000.asp

18. http://www.pcguide.com/ref/ram/typesDRAM-c.html

19. http://computer.howstuffworks.com/ram4.htm

20. http://www.ftc.gov/os/2002/06/rambuscmp.htm

21. http://www.pcguide.com/ref/ram/typesDRAM-c.html

22. http://www.pricecooler.com/faq.asp#8