
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ


# «Τεχνικές Μείωσης της Κατανάλωσης Ισχύος σε Ασύρματα Δίκτυα Αισθητήρων»

Μεταπτυχιακή Εργασία


Αποστολίδης Απόστολος

Τσιάμης Δημήτριος

**Επιβλέπων:**
Σταμούλης Γεώργιος
Καθηγητής Πανεπιστημίου Θεσσαλίας



Βόλος, Ιούνιος 2016

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ**

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

# «Τεχνικές Μείωσης της Κατανάλωσης Ισχύος σε Ασύρματα Δίκτυα Αισθητήρων»

Μεταπτυχιακή Εργασία

Αποστολίδης Απόστολος

Τσιάμης Δημήτριος

**Επιβλέπων:**
Σταμούλης Γεώργιος
Καθηγητής Πανεπιστημίου Θεσσαλίας

Εγκρίθηκε από την τριμελή επιτροπή

| (Υπογραφή) | (Υπογραφή) | (Υπογραφή) |
|---|---|---|
| ..... | ..... | ..... |
| Σταμούλης Γεώργιος | Ευμορφόπουλος Νέστωρ | Πλέσσας Φώτιος |
| Καθηγητής Π.Θ. | Επ. Καθηγητής Π.Θ. | Αν. Καθηγητής Π.Θ. |

Βόλος, Ιούνιος 2016

# Περιεχόμενα

## Εισαγωγή

Το θέμα της μείωσης κατανάλωσης ισχύος σε wireless sensor networks (WSN) αποτελεί εδώ και περισσότερο από 20 χρόνια ένα από τα πιο επίμαχα προβλήματα στη βιομηχανική παραγωγή σχεδίασης ολοκληρωμένων κυκλωμάτων και των τηλεπικοινωνιών. Υπάρχει μια τεράστια βιβλιογραφία όπου εκατοντάδες επιστήμονες σε όλο το κόσμο έχουν δημοσιεύσει άρθρα στα οποία προτείνουν διάφορες "έξυπνες" τεχνικές με τις οποίες εξοικονομείται ενέργεια σε μικρο-αισθήτηρες "μιας χρήσης" με σκοπό να παραταθεί όσο το δυνατόν περισσότερο ο χρόνος ζωής τους με το όσο το δυνατόν χαμηλότερο κόστος. Το ζήτημα είναι πολύ σύνθετο όχι μόνο εξαιτίας της εφαρμογής των WSN σε ένα ευρύ φάσμα εφαρμογών (στρατιωτικός τομέας, έλεγχος αγροτικών περιοχών για πυροπροστασία, ιατρικές εφαρμογές, βιομηχανικός έλεγχος θερμοπυρηνικών διεργασιών κ.ο.κ) αλλά κυρίως επειδή η ουσιαστική επίλυση του απαιτεί την συνεργασία πολλών και ξεχωριστών περιοχών της ηλεκτρονικής, της πληροφορικής και των τηλεπικοινωνιών επειδή η φύση του προβλήματος αγκαλιάζει όλα τα επίπεδα της λειτουργίας ενός WSN από την επεξεργασία σε επίπεδο υλικού (hardware processing), την επεξεργασία και τη μετάδοση σημάτων σε ασύρματο τηλεπικοινωνιακό δίκτυο (signal processing & wireless networking), ακόμα και τους ίδιους τους αλγορίθμους επεξεργασίας σε επίπεδο software.

Η εκπόνηση της παρούσας μεταπτυχιακής εργασίας αποσκοπεί σε 3 στόχους:

1. να αξιοποιήσει και να ταξινομήσει ένα όσο το δυνατόν μεγαλύτερο κομμάτι της ήδη υπάρχουσας βιβλιογραφίας ώστε να εκτιμήσει και να αναλύσει τις καινοτομίες αλλά και τις αδυναμίες των σημαντικότερων τεχνικών μείωσης της κατανάλωσης ισχύος σε WSN που έχουν προταθεί και εφαρμοστεί μέχρι τώρα.

2. να αναδείξει νέα ζητήματα που τίθενται και απαιτούν την άμεση επίλυσή τους από τους σχεδιαστές WSN προτείνοντας ερευνητικές κατευθύνσεις

3. να συμβάλλει στην επίλυση ενός συγκεκριμένου και πρακτικού ζητήματος των WSN σε επίπεδο hardware processing, περιγράφοντας όλη τη διαδικασία ανάπτυξης του προβλήματος σε θεωρητικό και πρακτικό επίπεδο, παρουσιάζοντας αποτελέσματα από εργαστηριακές μετρήσεις.

Ενότητα πρώτη

# Γενικές κατηγορίες τεχνικών μείωσης της κατανάλωσης ισχύος σε WSN

## 1. Το ζήτημα σε γενικές γραμμές

Τα ασύρματα δίκτυα αισθητήρων (WSN) σε γενικές γραμμές προσπαθούν να επιτύχουν την αλληλεπίδραση με το φυσικό περιβάλλον, και λαμβάνοντας δείγματα μέτρησης από αυτό (πχ θερμοκρασία, πίεση, υγρασία, κ.ο.κ) να τα επεξεργαστούν με υπολογιστικούς μηχανισμούς καταλήγοντας σε διάφορα συμπεράσματα που ενδιαφέρουν τον κάτοχο ή χρήστη. Σήμερα, οι ερευνητές και οι επαγγελματίες χρησιμοποιούν κόμβους χαμηλής ισχύος οι οποίοι αποτελούνται από αισθητήρες, ολοκληρωμένα κυκλώματα ασύρματης ραδιοεκπομπής (RF) και υπολογιστικές μονάδες σε μια ευρεία γκάμα εφαρμογών στην ιατρική, το στρατό, τη βιολογία, την υγεία κ.ο.κ. Ιδανικά, οι ερευνητές θα ήθελαν να επιτύχουν μια πιο βαθιά ενσωμάτωση των κόμβων αυτών μέσα στο φυσικό κόσμο και να τροφοδοτούν ενεργειακά τις συσκευές τους μέσα στο ατμοσφαιρικό περιβάλλον όπου τις περισσότερες φορές ο άνθρωπος δεν μπορεί εύκολα να έχει πρόσβαση για διαφόρους λόγους (δύσβατες ή δασώσεις γεωγραφικές περιοχές, ραδιενεργές ζώνες, στρατιωτικό πεδίο μάχης κλπ). Για να επιτευχθούν τέτοιες χαμηλές ενεργειακές απαιτήσεις, σχεδιάζονται σε διάφορα ερευνητικά εργαστήρια ειδικοί επεξεργαστές πολύ χαμηλής ισχύος (ultra-low-power-processors) αποκλειστικά για εφαρμογές WSN.

Όμως αντικειμενικά, η σχεδίαση ultra-low-power συστημάτων επιβάλλει μια συνολική προσέγγιση των προδιαγραφών και των απαιτήσεων της εκάστοτε σχεδίασης. Οι αποφάσεις του σχεδιαστή για την αρχιτεκτονική του (system architecture) πρέπει να συναρτώνται απόλυτα με την εξέταση και διερεύνηση του προβλήματος που καλείται να επιλύσει (σ. το σύστημα) επιλέγοντας τις ανάλογες κυκλωματικές τεχνικές μείωσης της ισχύος. Τα WSNs λαμβάνουν μετρήσεις και αντιδρούνε αναλόγως στα διάφορα φυσικά φαινόμενα που παρατηρούνε. Συνεπώς, οι απαιτήσεις της λειτουργίας και εκτέλεσης των WSNs systems μπορεί να κυμαίνονται σε ένα πολύ μεγάλο εύρος -δηλαδή να λαμβάνονται δείγματα κάθε λίγα λεπτά (π.χ μετρήσεις θερμοκρασίας σε έναν αγρό ή δάσος) μέχρι χιλιάδες δείγματα το δευτερόλεπτο (συστήματα σεισμικών μετρήσεων, επεξεργασίας ήχου και εικόνας). Η μετάδοση για το κάθε δείγμα του κάθε αισθητήρα με ραδιοεκπομπή όχι μόνο θα κατανάλωνε όλο το bandwidth των ασύρματων καναλιών του δικτύου αλλά πολύ γρήγορα (απελπιστικά γρήγορα θα λέγαμε) θα εξαντλούσε τη διαθέσιμη σε αυτούς αποθηκευμένη ενέργεια. Για αυτό το λόγο λοιπόν, πολλά WSNs systems που έχουν να διαχειριστούν τεράστιες ποσότητες δειγμάτων με περιορισμένους ενεργειακούς πόρους "φιλτράρουν" τα δεδομένα πάνω στους κόμβους-αισθητήρες (data filtering) επομένως μεταδίδεται πάνω από το ασύρματο κανάλι με ραδιοεκπομπή μόνο η "ουσιαστική" πληροφορία. Το δίλλημα που γεννιέται ανάμεσα στο "επικοινωνιακό" και το "υπολογιστικό" κόμματι της λειτουργίας του WSNs βαραίνει

αναμφισβήτητα προς το δεύτερο και ως επί το πλείστων δίνεται έμφαση στις τεχνικές μείωσης της ισχύος πάνω στον κόμβο-αισθητήρα (on-processing).

Σε αυτή την ενότητα θα εξετάσουμε τις διάφορες κατηγορίες εφαρμογών και θα αναλύσουμε διάφορες circuit techniques και system architectures που μειώνουν σημαντικά την κατανάλωση ισχύος on-processing. Μια πολύ σημαντική κατηγορία κυκλωματικών τεχνικών που χρησιμοποιείται ευρέως είναι η αξιοποίηση των πλεονεκτημάτων που προσφέρουν τα ασύγχρονα κυκλώματα ως προς την αποφυγή του clock overhead που για τα κλασικά σύγχρονα ψηφιακά κυκλώματα είναι η "αχίλλειος πτέρνα" ως προς την συνολική κατανάλωση ισχύος και ενέργειας. Μια άλλη, πιο απλή όμως όχι λιγότερο διαδεδομένη, κατηγορία είναι η ρύθμιση των τάσεων τροφοδοσίας ($V_{dd}$) χαμηλότερα από την τάση κατωφλίου (Voltage threshold). Τέλος, υπάρχει και μια ακόμα κατηγορία τεχνικών που παρέχει την ανάλογη υποστήριξη για κυκλώματα που διακόπτουν από την τάση τροφοδοσίας αχρησιμοποίητα κομμάτια στην προσπάθεια τους να μειωθεί το leakage current (αξίζει να σημειωθεί ότι όσο οι διαστάσεις των transistor μειώνονται, το leakage current αυξάνεται και παύει να έχει αμελητέα επίδραση) .

## 1.1 Σχεδιαστικές τεχνικές σε επίπεδο κυκλώματος

Σε επίπεδο κυκλώματος υπάρχουν σημαντικές δυνατότητες για να μειώσουμε την κατανάλωση ισχύος. Ένα πρόβλημα που όλο και περισσότερο πλέον απασχολεί τους σχεδιαστές στη βιομηχανία είναι η μεγάλη επίδραση του leakage current στην συνολική κατανάλωση ενέργειας κυρίως σε WSNs με χαμηλές υπολογιστικές απαιτήσεις που το μεγαλύτερο μέρος του χρόνου τους βρίσκονται σε idle κατάσταση. Όπως είπαμε και λίγο πιο πάνω το leakage current αυξάνεται σημαντικά όσο μειώνονται οι διαστάσεις των VLSI τεχνολογιών. Σε αυτή την υποενότητα θα περιγράψουμε τις κυκλωματικές τεχνικές που χρησιμοποιούνται για να μειωθεί η κατανάλωση ενέργειας σε ένα αισθητήρα-κόμβο.

### 1.1.1 Voltage and Frequency Scaling and Subthreshold Design

Όπως είναι αυτονόητο, στην συντριπτική πλειοψηφία των εφαρμογών τους, τα δίκτυα αισθητήρων δεν μπορούν να συνδεθούν καλωδιακά με ισχυρές πηγές ενεργειακής τροφοδοσίας, επομένως η συντήρηση της ενεργειακής κατανάλωσης του συστήματος είναι το πρωταρχικό σχεδιαστικό μας μέλημα. Η ενέργεια μπορεί να εκφραστεί ως το άθροισμα της ενέργειας που οφείλεται στην μεταγωγική δραστηριότητα του κυκλώματος (active switching energy) συν την ενέργεια του leakage current [1] :

$$E_{total} = V_{dd}(\alpha C_{sw} V_{dd} + I_{leak} \Delta t_{op})$$

όπου :

α : είναι η μεταγωγική δραστηριότητα ανά δευτερόλεπτο

$\Delta t_{op}$ : η χρονική διάρκεια που απαιτείται για την ολοκλήρωσης της εργασίας

Είναι προφανές από την παραπάνω σχέση ότι για εφαρμογές χαμηλού κύκλου εργασιών, η ενέργεια του leakage current θα κυριαρχεί επειδή ο συντελεστής α θα είναι

μικρός. Ένας από τους πιο αποτελεσματικούς τρόπους για να εξοικονομηθεί ενέργεια σε αυτή τη περίπτωση είναι να ρυθμίσουμε την τάση τροφοδοσίας ($V_{dd}$ scaling). Όταν η $V_{dd}$ ρυθμίζεται προς τα κάτω, η active switching energy μειώνεται τετραγωνικά ($E_{sw} = \alpha C_{sw} V_{dd}^2$). Όπως σημειώνεται στη βιβλιογραφία, το σημείο βελτιστοποίησης της ενέργειας του $V_{dd}$ στα περισσότερα κυκλώματα βρίσκεται χαμηλότερα από την τάση κατωφλίου. Δυστυχώς όμως οι παραδοσιακές μνήμες SRAMs δεν λειτουργούνε αξιόπιστα με τάση χαμηλότερη του $V_t$. Αν και τα τελευταία χρόνια έχουν προταθεί σε κάποια ερευνητικά άρθρα πρωτότυπες ιδέες σχεδίασης subthreshold SRAMs, ωστόσο η βιομηχανία δεν είναι ακόμα έτοιμη για μαζική παραγωγή [2].

Το μεγαλύτερο πρόβλημα που υπάρχει όταν τα κυκλώματα λειτουργούν με τάση υποκατωφλίου είναι η καθυστέρηση γιατί, όπως είναι γνωστό, το ρεύμα οδήγησης εξαρτάται εκθετικά από την τάση τροφοδοσίας όταν το transistor βρίσκεται στην περιοχή του υποκατωφλίου. Συνεπώς, η καθυστέρηση μιας CMOS πύλης στο subthreshold μπορεί να εκφραστεί με την παρακάτω σχέση:

$\Delta t_{op}$ ανάλογο του $e^{-kV_{dd}}$

όπου :

k : σταθερά που εξαρτάται από την τεχνολογία του κατασκευαστή και την θερμοκρασία.

Επειδή λοιπόν η καθυστέρηση του κυκλώματος θα μεταβάλλεται εκθετικά με κάθε μεταβολή της θερμοκρασίας ή της βιομηχανικής τεχνολογίας, είναι δύσκολο να επιλεγεί μια σταθερή συχνότητα ρολογιού στην οποία το chip θα λειτουργεί αξιόπιστα. Όπως άλλωστε αναφέραμε και στην εισαγωγή, οι προδιαγραφές κάθε ξεχωριστής WSN εφαρμογής επηρεάζονται από διάφορες τάξεις μεγεθών, πόσο μάλλον αυτών που σχεδιάζονται να δρουν σε ακραίες περιβαλλοντικές συνθήκες οπότε και καθορίζουν ελάχιστα περιθώρια ανοχής. Τα σύγχρονα subthreshold systems που τρέχουν σε χαμηλές συχνότητες ρολογιού ίσως να μην μπορούν να ανταπεξέλθουν σε απαιτήσεις μεσαίων και υψηλών ρυθμών υπολογιστικής επεξεργασίας δειγμάτων. Η λειτουργία υποκατωφλίου παρέχει σημαντικές δυνατότητες για εξοικονόμηση ενέργειας με κόστος στην απόδοση και είναι κατάλληλη μόνο για WSNs των οποίων ο φόρτος εργασίας δεν έχει αυξημένες υπολογιστικές ανάγκες στη μονάδα του χρόνου.

### 1.1.2 Σχεδίαση ασύγχρονων κυκλωμάτων

Τα ασύγχρονα κυκλώματα δεν βασίζονται σε global περιοδικά σήματα ρολογιού, αλλά χρειάζεται να λειτουργούν κάτω από επιπρόσθετους περιορισμούς σχεδίασης. Όλα τα ασύγχρονα κυκλώματα πρέπει να είναι glitch-free (μέσα σε καθορισμένα χρονικά μοντέλα). Τα ασύγχρονα κυκλώματα επίσης απαιτούν καθορισμένα σήματα χειρισμού (handshake signals) ανάμεσα στα κυκλωματικά blocks. Το handshaking κάνει εύκολη τη σύνδεση των κυκλωματικών blocks επειδή ο συγχρονισμός και οι ανάγκες ροής των δεδομένων επιτυγχάνονται με σαφήνεια. Επιτρέπει επίσης το σύστημα να αποκρίνεται στις μεταβολές της καθυστέρησης του critical path παρέχοντας πληροφορίες ως προς το συγχρονισμό του εφόσον οι παράγοντες λειτουργίας του αλλάξουν (π.χ θερμοκρασία, τάση τροφοδοσίας, μεταβολές που οφείλονται στον κατασκευαστή). Επειδή τα ασύγχρονα κυκλώματα δεν οδηγούνται από σήματα ρολογιού όσο ο επεξεργαστής είναι ανενεργός, η κατανάλωση ενέργειας μειώνεται σημαντικά σε low-duty cycle εφαρμογές.

## 2.3 VDD-Gating

Η τεχνική του gating έχει χρησιμοποιηθεί για να μειώσει το leakage current που καταναλώνεται σε ένα σύστημα [4]. Το μειονέκτημα αυτής της τεχνικής είναι η απώλεια της κυκλωματικής κατάστασης του block που βγαίνει off από το σύστημα τροφοδοσίας. Αυτό που χρειάζεται είναι υποστήριξη hardware σε επίπεδο μικροαρχιτεκτονικής που θα μπορεί να διαχειριστεί αποτελεσματικά το gating των transistors ανάλογα με τις προδιαγραφές της εκάστοτε WSN εφαρμογής.

Ενότητα Δεύτερη

# Συστήματα hardware και αρχιτεκτονικές για ασύρματα δίκτυα αισθητήρων με στόχο τη μείωση της κατανάλωσης ισχύος

Την τελευταία δεκαετία έχουν προταθεί, υλοποιηθεί και αναπτυχθεί συστήματα WSN που αξιοποιούν σε κάποιο βαθμό τη γνώση και τεχνογνωσία που υπάρχει και προέρχεται από τους ερευνητικούς τομείς του VLSI, του computer architecture, των ασύρματων επικοινωνιών και πρωτοκόλλων καθώς και της τεχνολογίας αισθητήρων, τεχνογνωσία που εφαρμόστηκε με επιτυχία στην ευρύτερη βιομηχανία των υπολογιστών και των καταναλωτικών ηλεκτρονικών. Ωστόσο η έρευνα σε επίπεδο εξειδικευμένων τεχνικών με στόχο την ανάπτυξη και υλοποίηση μοντέλων αρχιτεκτονικής για τη μείωση της κατανάλωσης ισχύος βρίσκεται ακόμα σε νηπιακό στάδιο και αφορά κυρίως τα μεγάλα projects επιφανών πανεπιστημιακών ιδρυμάτων των ΗΠΑ. Η βιβλιογραφία δεν είναι επαρκής και πολλές φορές ξεφεύγει από τους στόχους ενώ δεν φαίνεται ακόμα να έχει επιτευχθεί αξιόλογος συντονισμός ανάμεσα στον ακαδημαϊκό κόσμο και στη βιομηχανία ανάπτυξης συστημάτων WSN. Σε αυτή την ενότητα θα προσπαθήσουμε να περιγράψουμε και να αναλύσουμε όσο το δυνατόν καλύτερα και με σαφήνεια τις οικογένειες αρχιτεκτονικών σε επίπεδο hardware των WSN κάνοντας, προς ευκολία του αναγνώστη, μια υποτυπώδη κατηγοριοποίηση και διευκρινίζοντας φυσικά ότι ακόμα το τοπίο είναι ρευστό και δεν υπάρχει επίσημη προτυποποίηση των τεχνικών/αρχιτεκτονικών.

Κατηγοριοποιούμε τα συστήματα WSN *σε επίπεδο hardware.*

## 2.1 Εμπορικά συστήματα γενικού σκοπού

Υπάρχουν πλατφόρμες δικτύων αισθητήρων σχεδιασμένες πάνω σε microcontrollers της αγοράς, όπως ο TI MSP430 ή ο Atmel ATMega 128L. Αυτοί οι processors είναι σχεδιασμένοι για λειτουργίες χαμηλής ισχύος σε μια ευρεία γκάμα εφαρμογών, όμως βασίζονται σε συμβατικές υπολογιστικές μονάδες γενικού σκοπού που δεν είναι και τόσο κατάλληλες για τη "φύση" των WSN. Ενώ τέτοιοι processors μπορούν να υποστηρίξουν idle καταστάσεις χαμηλής ισχύος (με κατανάλωση μικρότερη από 5 μA στην περίπτωση του MSP430), *απενεργοποιούν ολόκληρο το process και το "ξυπνάνε" στο επόμενο interrupt*, οπότε γι' αυτό το λόγο η χρήση τους *περιορίζεται σε WSN εφαρμογές αλληλεπιδραστικού τύπου***.** Η συμβατική φύση αυτών των αρχιτεκτονικών αποκλείει επίσης την ευελιξία στο χρονισμό μεμονωμένων υπομονάδων του processor.

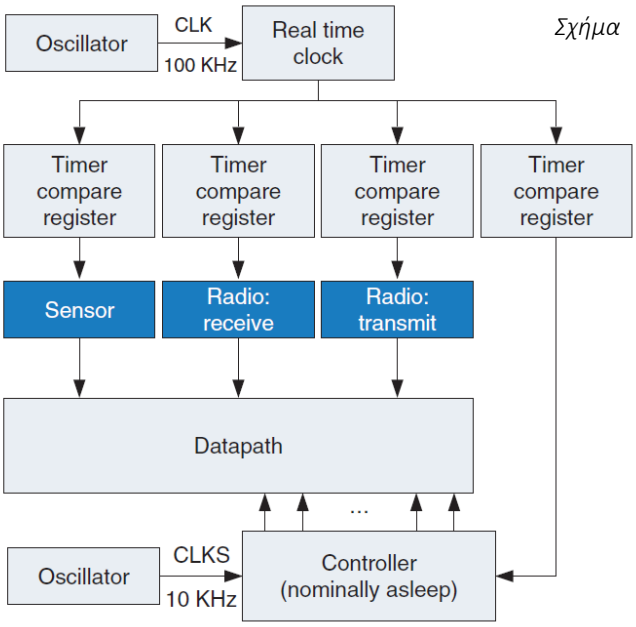
Το Mica2 mote - που βασίζεται στην αρχιτεκτονική του microcontroller Atmel ATMega 128L - έχει χρησιμοποιηθεί ευρέως στα συστήματα ερευνητικού σκοπού. Αποτελείται από έναν ATmega 128L processor των 7.3 MHz με code memory των 128KB και data memory των 4KB. Οι μετρήσεις πάνω στο Mica2 mote δείχνουν ότι καταναλώνεται κατά μέσο όρο ρεύμα των 8 mA όταν είναι ενεργό και 100 μA όταν είναι σε κατάσταση χαμηλής ισχύος [5], χοντρικά δηλαδή καταναλώνονται 3.2 nJ ανά εντολή.

Ένας άλλος σημαντικός microcontroller που χρησιμοποιείται για πλατφόρμες WSN είναι ο MSP430 από την Texas Instruments. Έχει 16-bit databus, 10KB RAM και 48KB Flash. Καταναλώνει 2mA στα 8 MHz και 3V ενώ σε κατάσταση "ύπνου" καταναλώνει λίγα microamps ρεύματος. Αυτά αντιστοιχούν σε μια κατανάλωση ενέργειας περίπου στα 750 pJ ανά εντολή.

Αρκετά λειτουργικά συστήματα έχουμε γραφτεί για WSN εφαρμογές που παρέχουν δυνατότητες χειρισμού στο χρήστη τα οποία βασίζονται σε εμπορικούς microcontroller. Το TinyOS χρησιμοποιείται ευρέως από τους κατασκευαστές δικτύων αισθητήρων και είναι συμβατό και σε Atmel ATmega 128L και σε TI MSP430.

## 2.2 Smart Dust

Ένας από τους πρώτους WSN microcontrollers σχεδιάστηκε για την πλατφόρμα Smart Dust στο πανεπιστήμιο του Berkeley της Καλιφόρνιας [6],[7]. Εννοιολογικά, ένα "έξυπνο" dust mote αποτελείται από MEMS αισθητήρες, ένα ηλιακό κύτταρο, οπτική επικοινωνία και έναν microcontroller. Ο microcontroller είναι ένας load-store RISC επεξεργαστής με την αρχιτεκτονική του Harvard (ξεχωριστές μνήμες εντολών και δεδομένων). Το σχήμα 2.1 απεικονίζει ένα λειτουργικό διάγραμμα του συστήματος. Η σχεδίαση αποτελείται από έναν ταλαντωτή χαμηλής-ταχύτητας που τροφοδοτεί 5 timers για δειγματοληψία σήματος αισθητήρα, ραδιοεκπομπές και λήψεις, και πράξεις στο datapath. Όταν οι timers ξεκινήσουν, ανοίγει ένας γρηγορότερος ταλαντωτής που τροφοδοτεί το datapath και το ADC (Analog to Digital Converter). Εφόσον η δομή της smart Dust αποτελείται από ανεξάρτητα υποσυστήματα (καθένα από τα οποία τροφοδοτείται από το δικό του timer) το σύστημα έχει τη δυνατότητα για clock-gate σε ανενεργά blocks. Το σύστημα *δεν είναι εγγενώς αλληλεπιδραστικό* και γι' αυτό χρησιμοποιεί τους timers ώστε να *συγκεντρώνει περιοδικά* τα δεδομένα που συλλέγει από τους αισθητήρες του. Αυτό το σύστημα σχεδιάστηκε σε τεχνολογία 0.25μm και επομένως δεν τίθεται το πρόβλημα του leakage current στο επίπεδο της αρχιτεκτονικής του. Για 1.0 V και 500 kHz το σύστημα καταναλώνει 12pJ ανά εντολή.




*Σχήμα 2.1. Smart dust microarchitecture. Reprinted with permission from [23], B.A.W arneke and K.S.J.Pister , An ultra-low energy microcontroller for smart dust wireless sensor networks. ISSCC, January (2004).© 2004.*

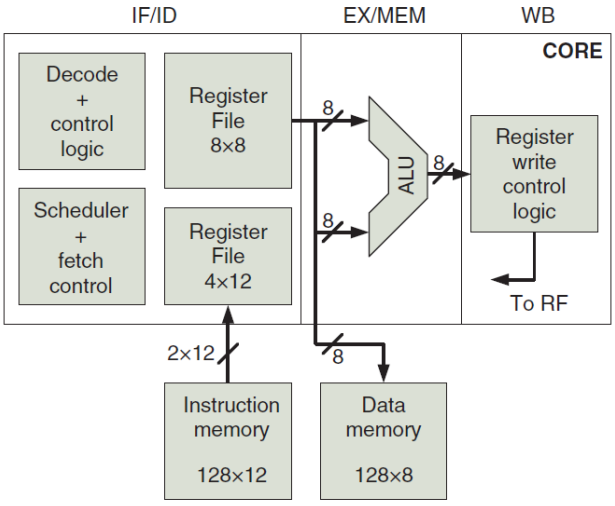
## 2.3 Συστήματα υποκατωφλίου (Subthreshold systems)

Όπως έχουμε αναφέρει, οι υπολογιστικές απαιτήσεις των WSN κόμβων είναι διαφορετικές ανάλογα με την περίπτωση και τη φύση του προβλήματος. Για συστήματα που απαιτούν χαμηλή συχνότητα δειγματοληψίας (λιγότερη από 100 Hz), εύκολα εξασφαλίζεται σε πραγματικό χρόνο η εξυπηρέτηση του φόρτου εργασίας σε κενές χρονικές περιόδους μεγάλης διάρκειας. Αρκετοί ερευνητές έχουν σχεδιάσει κυκλώματα που λειτουργούν χαμηλότερα της τάσης κατωφλίου για μείωση της κατανάλωσης ισχύος. Στη βιβλιογραφία έχουν προταθεί πολλά υποσυστήματα hardware που λειτουργούν στην περιοχή υποκατωφλίου, συμπεριλαμβάνοντας και τον αλγόριθμο FFT [8].

Οι ερευνητές του Michigan έχουν σχεδιάσει έναν από τους πρώτους ολοκληρωμένους επεξεργαστές για WSNs που λειτουργούν στην περιοχή υποκατωφλίου, τον *Subliminal Processor.* Όλες οι versions του συστήματος επικεντρώνονται γύρω από το γενικό σκοπό της harvard αρχιτεκτονικής. Διαφέρουν όμως σε επιμέρους παραμέτρους της αρχιτεκτονικής όπως είναι το μέγεθος των καταχωρητών, το πλάτος των διαύλων, CISC/RISC, και ο αριθμός των pipeline σταδίων. Για κάθε μία από αυτές τις versions υπολογίζεται το μέγεθος κώδικα, ο αριθμός των κύκλων ανά εντολή (CPI), και η ενέργεια [9].

Η αρχιτεκτονική της version 2 του Subliminal processor παρουσιάζεται στο σχήμα 2.2. Το σύστημα αποτελείται από ένα datapath των 8 bit με ξεχωριστές μνήμες εντολών και δεδομένων, έναν αποκωδικοποιητή, το fetch στάδιο, και μια ψηφιακή λογική χρονοπρογραμματισμού των διαδικασιών.

Δύο από τις version του Subliminal processor έχουν υλοποιηθεί σε τεχνολογία CMOS των 130nm. Επειδή, όπως έχουμε ήδη πει, οι παραδοσιακές μνήμες SRAM δεν λειτουργούν με αξιοπιστία στην περιοχή υποκατωφλίου χρησιμοποιούνται μνήμες που βασίζονται σε λογική πολυπλέκτη (mux-based logic). Και στις δύο versions **μετρήθηκε σημαντική αύξηση της καθυστέρησης στη λειτουργία υποκατωφλίου**. Η κανονικοποιημένη μέγιστη συχνότητα λειτουργίας της πρωτότυπης version κυμαίνεται από 0.6 σε 1.8 στα 260 mV σε μια κατανομή από 26 chips [10]. Η μέση κατανάλωση ενέργειας ενός subliminal processor είναι 2.6 με 3.5 pJ ανά εντολή.




*Σχήμα 2.2. Block diagram of the subliminal processor (University of Michigan).Reprinted with permission from [8], S.Hanson et al., Performance and variability optimization strategies in a Sub-200 mV, 3.5 pJ/inst, 11 nW subthreshold processor. IEEE Symposium on VLSI Circuits (VLSI-Symp), June (2007).© 2007.*

## 2.4 Ασύγχρονος - SNAP

Ο επεξεργαστής SNAP από το Cornell είναι ένας *ασύγχρονος* επεξεργαστής των 16-bit βασισμένος στην αρχιτεκτονική RISC και σχεδιασμένος ειδικά για την διαχείριση του φόρτου εργασίας και δεδομένων σε δίκτυα αισθητήρων [11], [12], [13]. Ο SNAP στην ουσία είναι ένας *αλληλεπιδραστικός* επεξεργαστής και αποτελείται από 2 επιταχυντές (accelerators) οι οποίοι παράγουν *ειδικά συμβάντα* για WSN. Ο SNAP χρησιμοποιεί κυκλώματα QDI (*quasi delay-insensitive*). Αφού λοιπόν ο SNAP σχεδιάζεται αποκλειστικά με ασύγχρονα κυκλώματα, είναι δυνατόν να τρέξει σε ένα ευρύ πεδίο τιμών της τάσης τροφοδοσίας από 1.8V μέχρι 0.6V.

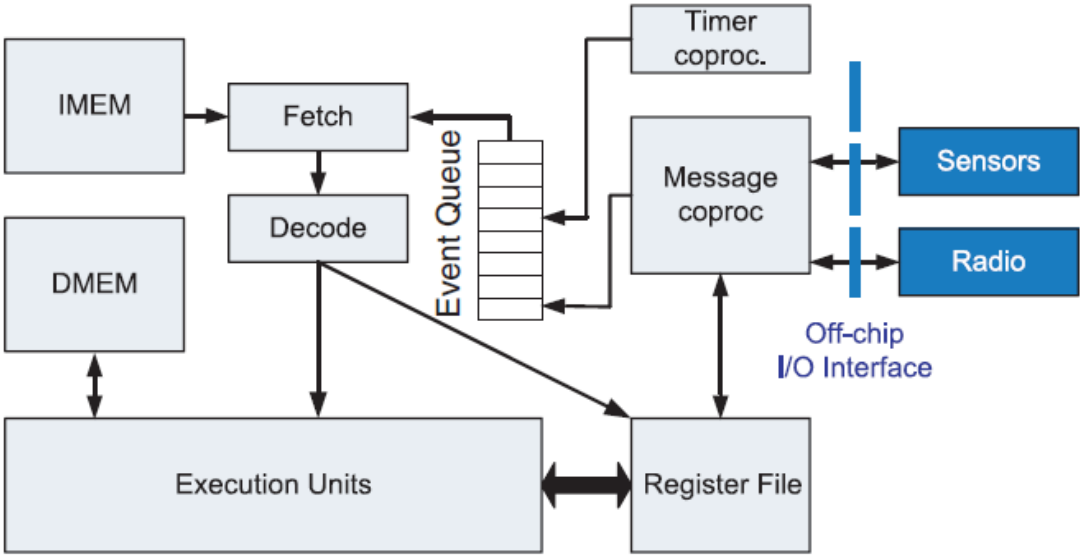
Στο σχήμα 2.3 παρουσιάζεται ένα απλοποιημένο διάγραμμα βαθμίδας του SNAP/LE, η πρωταρχική υλοποίηση της SNAP αρχιτεκτονικής. *Το ιδιαίτερο χαρακτηριστικό του SNAP είναι ότι δεν εκτελεί τους υπολογισμούς συνεχόμενα αλλά αντ' αυτού ανταποκρίνεται σε διακριτά εισερχόμενα συμβάντα*. Το σύστημα περιλαμβάνει μια *ουρά συμβάντων* που περιέχει *tokens* καθένα από τα οποία ορίζει ένα συγκεκριμένο συμβάν. Όταν το ένα *token* φτάσει στην κεφαλή της ουράς, ο SNAP φέρνει το αντίστοιχο *σήμα χειρισμού συμβάντων* από τη μνήμη εντολών (IMEM). Ο αποκωδικοποιητής επιλέγει την κατάλληλη μονάδα εκτέλεσης, από το αντίστοιχο opcode και παίρνονται οι τελεστέοι από τη μνήμη δεδομένων (DMEM) και τον φάκελο καταχωρητών (Register File). Οι εντολές εκτελούνται *σειριακά* και μέχρι η τελευταία εκτελεσμένη εντολή να φτάσει να δείχνει το τέλος του *χειριστή συμβάντων. Αν η ουρά συμβάντων είναι άδεια, και αφού έχουμε φτάσει στην τελευταία εντολή ο SNAP διακόπτεται και περιμένει το νέο event token να μπει στην ουρά συμβάντων.*

Ο SNAP περιλαμβάνει 2 components που παράγουν συμβάντα, τον *timer coprocessor* και τον *message coprocessor*. Ο timer coprocessor αποτελείται από τρεις timer registers που αυτο-μειώνονται: όταν οι καταχωρητές φτάσουν στο μηδέν τότε "γεννιέται" ένα συμβάν. Ο message coprocessor λειτουργεί ως διεπαφή ανάμεσα στον SNAP και τις συσκευές που βρίσκονται από το chip παρέχοντας δεδομένα εισόδου/εξόδου για τους αισθητήρες και τα ραδιοκυκλώματα. Ο SNAP επικοινωνεί με τον message comprocessor μέσα από έναν απλό καταχωρητή. Κάθε στιγμή που 1 byte φτάνει από τα ραδιοκυκλώματα, ο message comprocessor φτιάχνει ένα νέο συμβάν στην ουρά συμβάντων. Ο message coprocessor δεν εκτελεί υπολογισμούς πάνω στα δεδομένα που έρχονται από τις ραδιολήψεις αλλά τα προωθεί στο datapath.

Πολλοί εμπορικοί microcontrollers περιέχουν λειτουργίες χαμηλής κατανάλωσης ισχύος που τοποθετούν την μονάδα επεξεργασίας σε κατάσταση "ύπνου": η ποσότητα του χρόνου που χρειάζεται το σύστημα για να "ξυπνήσει" από την κατάσταση "ύπνου" είναι πολύ σημαντική. Από τη φύση τους τα ασύγχρονα κυκλώματα δεν χρειάζονται ρολόι. *Υποστηρίζεται από τους ερευνητές ότι ο SNAP μπορεί να πέσει σε κατάσταση ύπνου μέσα σε nanoseconds επειδή σταματάνε όλα τα switching του κυκλώματος όταν η ουρά συμβάντων είναι άδεια. Αυτή η τεχνική είναι αποδοτική για τεχνολογίες των 180 nm όπου το leakage current είναι μικρό.* Καθώς όμως το leakage current αυξάνεται, τα συστήματα SNAP χρειάζεται να ενσωματώσουν τεχνικές μείωσης του leakage.

Ο BitSNAP, η τελευταία version της SNAP αρχιτεκτονικής, αντικαθιστά το παράλληλο datapath του SNAP/LE με ένα σειριακό datapath που αν και χρειάζεται περισσότερο χρόνο για να υπολογίσει τις πράξεις, εντούτοις *καταναλώνει 70% λιγότερη ενέργεια από το SNAP/LE* [12]. Ο BitSNAP συμπεριλαμβάνει *dynamic significance compression με το οποίο προσαρμόζει επιλεκτικά τον αριθμό των επαναλήψεων που τρέχουν πάνω στο σειριακό datapath ο οποίος*

*εξαρτάται από τον αριθμό των significant bits στους τελεστέους.* Τα σειριακά datapaths έχουν σημαντικό ενδιαφέρον από θέμα κατανάλωσης ισχύος επειδή αποτελούνται από λιγότερα transistors σε σχέση με τα παράλληλα datapaths και αυτό έχει ως αποτέλεσμα μικρότερο αποτελεσματικό πλάτος πάνω στο leakage current path.



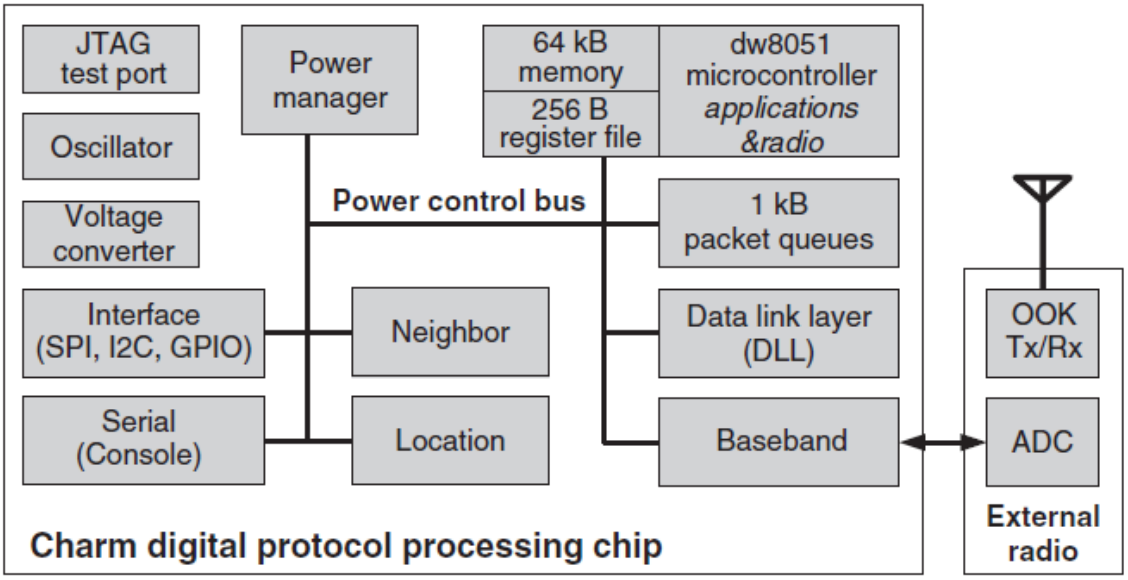

*Σχήμα 2.3. Simplified block diagram of the SNAP processor for WSN.System includes separate instruction and data memories, a timer coprocessor, and a message processor which provides a FIFO interface to the off-chip radio and sensors.Reprinted with permission from [4], V.Ekanayak e et al., An ultra low-power processor for sensor networks. Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, October (2004). © 2004.*

## 2.5 Charm - Network Stack Acceleration

Το Charm είναι σύστημα ψηφιακής επεξεργασίας της στοίβας πρωτοκόλλου δικτύου ο οποίος σχεδιάστηκε στο Berkeley Wireless Research Center και *ενσωματώνει στο hardware ένα μεγάλο κομμάτι του radio stack που αντιστοιχεί στο μοντέλο αναφοράς OSI* [14]. Το σύστημα Charm υλοποιεί όλη τη στοίβα πρωτοκόλλων, δηλαδή το επίπεδο application, το επίπεδο δικτύου, το επίπεδο ζεύξης δεδομένων και το φυσικό επίπεδο της ψηφιακής διαμόρφωσης του OSI radio stack *με ολοκληρωμένο hardware που απαρτίζεται από αυτοτελή υποσυστήματα.* Στο σχήμα 2.4 βλέπουμε την μικροαρχιτεκτονική του Charm με τα block για κάθε ένα από τα κύρια υποσυστήματα.

Εξαιτίας της σχεδιαστικής του δομής, το Charm έχει σημαντικές δυνατότητες διαχείρισης της ισχύος αφού το κάθε ένα από τα υποσυστήματα της στοίβας πρωτοκόλλου έχει *ξεχωριστή κατανάλωση ισχύος* που δημιουργείται ως επί το πλείστων από το leakage current και εξαρτάται από τη συχνότητα λειτουργίας του. Στη βιβλιογραφία η ισχύς που καταναλώνει το κάθε υποσύστημα ξεχωριστά αναφέρεται ως *power domain.* Όπως είπαμε παραπάνω, η τεχνική που έχει προταθεί και αναπτυχθεί για τη μείωση του leakage current είναι το VDD-gating. Η καινοτομία ως προς τη διαχείριση ισχύος που αξιοποιεί την σχεδιαστική δομή του hardware έχει ως εξής: *αντί να γίνεται το gating κλασικά και με τον ίδιο τρόπο σε*

*όλο το σύστημα hardware, το κάθε power domain του Charm περιλαμβάνει switches τα οποία επιλέγουν ανάμεσα σε 2 διαφορετικές τάσεις τροφοδοσίας, την $VDD_{hi}$ και την $VDD_{low}$.* Η $VDD_{low}$ είναι η χαμηλότερη τάση που απαιτείται για να διατηρηθεί η λογική κατάσταση του ψηφιακού κυκλώματος. Η εκάστοτε κατάσταση switch ρυθμίζεται από έναν **διαχειριστή ισχύος** που δέχεται αιτήσεις από τα power domains οι οποίες μεταφέρονται μέσα από ένα κεντρικό δίαυλο.




*Σχήμα 2.4. The charm protocol processor microarchitecture. Reprinted with permission from [19], M.Sheets et al., A power-managed protocol processor for wireless sensor networks. VLSI, June (2006).© 2006.*

Το Charm καταναλώνει 250.1 μW από το leakage current όταν όλα τα power domains βρίσκονται στο $VDD_{hi}$ (1.0V). Αντίστοιχα όταν τα power domains βρίσκονται στο $VDD_{low}$ (0.3V) καταναλώνονται 10μW από το leakage current και 43.6μW από τη λειτουργία των κυκλωμάτων. *Δηλαδή όταν χρησιμοποιούμε την τεχνική των switched power domains μπορούμε να καταναλώσουμε μέχρι και 5 φορές μικρότερη ισχύ*. Αν το σύστημα δειγματοληπτεί τις ραδιολήψεις *με περίοδο 100ms* η μέση κατανάλωση ισχύος του Charm είναι *132μW.* Όμως δεν μπορούμε να μετρήσουμε την *ενέργεια ανά εντολή* επειδή η πρωταρχική μονάδα των υπολογισμών είναι ράδιο-πακέτα.

Σε μία γενική εκτίμηση το σύστημα Charm παρέχει επαρκή ενέργεια για την επιτάχυνση της λειτουργίας του radio stack όμως για άλλες ζωτικές λειτουργίες των WSN όπως είναι το data filtering και το compression (φιλτράρισμα και συμπίεση δεδομένων) βασίζεται σε κλασσικά συστήματα επεξεργασίας γενικού σκοπού τα οποία εκτός από ενεργοβόρα δεν είναι και απολύτως "ταιριαστά" σε εξειδικευμένες ανάγκες. Δυστυχώς ή ευτυχώς οι προδιαγραφές και τα επίπεδα απαιτήσεων των WSN radio stack αλλάζουν διαρκώς και η σχεδίαση ειδικευμένου hardware για radio stack δεν είναι μία εύκολη υπόθεση.

## 2.6 Η αρχιτεκτονική "Event Driven" του Πανεπιστημίου Harvard (αλληλεπιδραστική)

Ομάδα από το πανεπιστήμιο του Harvard εισήγαγε μια νέα τεχνική προσέγγιση "αλληλεπιδραστικού" τύπου που προσπαθεί να βελτιστοποιήσει από τη μία τις τυπικές λειτουργίες των δικτύων αισθητήρων με hardware acceleration, και από την άλλη να μειώσει το πρόβλημα του leakage current μέσω κατάλληλων εφαρμογών που ελέγχουν το VDD-gating (application-controlled fine-grained VDD-gating) [15]. Αυτή η προσέγγιση συναντάται συχνά στη βιβλιογραφία ως *event-driven* (αλληλεπίδραση μέσω συμβάντων). Το "αλληλεπιδραστικό" σύστημα του Harvard για δίκτυα αισθητήρων χρησιμοποιεί λοιπόν τρεις κατηγορίες τεχνικών για να μειώσει την κατανάλωση ενέργειας.

- *χειρισμός συμβάντων απευθείας από το hardware για μείωση φόρτου από λογισμικό* - Ο χειρισμός των εισερχόμενων συμβάντων/interrupts γίνεται ως επί το πλείστον από έναν εξειδικευμένο event processor. Με αυτό το τρόπο απαλλασσόμαστε από το φόρτο λογισμικού που χρειάζεται για το χειρισμό συμβάντων σε έναν general-purpose processor.

- *Hardware acceleration για τις τυπικές εφαρμογές του WSN* - χρησιμοποιούνται για να ολοκληρώσουν τις τακτικές διαδικασίες ενός κόμβου-αισθητήρα όπως είναι το φιλτράρισμα δεδομένων (data filtering) και η δρομολόγηση μηνυμάτων (message routing).

- *Application-controlled fine-grained VDD-gating* - διαχειρίζονται το πρόβλημα του leakage current με υποστήριξη αρχιτεκτονικής VDD-gating απενεργοποιώντας από την τροφοδοσία τα blocks που δεν χρησιμοποιούνται την τρέχουσα χρονική περίοδο.

Πριν μιλήσουμε αναλυτικά και με λεπτομέρειες για τα επιμέρους κομμάτια της αρχιτεκτονικής αυτής σκόπιμο είναι να κάνουμε μια συνολική επισκόπηση. Το σύστημα της αρχιτεκτονικής απεικονίζεται στο σχήμα 2.5 και αποτελείται από 2 διακριτά τμήματα που διαχωρίζονται με βάση το κατά πόσο έχουν τα components τη δυνατότητα να ελέγξουν το σύστημα bus. Τα components που έχουν τον πλήρη έλεγχο στις διευθυνσιοδοτήσεις του συστήματος αναφέρονται ως *master components* ενώ τα υπόλοιπα στοιχεία/blocks που δεν μπορούν να συμβάλλουν στις διαδικασίες ροής δεδομένων του databus *accelerator components.* Το σύστημα του bus αποτελείται από 3 τμήματα: ένα *interrupt bus* (bus για τις εξαιρέσεις/διακοπές) και ένα *data bus* (bus για τα δεδομένα) όπως και στις κλασικές αρχιτεκτονικές που γνωρίζουμε, και επίσης από ένα τμήμα *με γραμμές για τον έλεγχο της ισχύος* (power control lines). Τα *accelerator components* διαβάζουν και γράφουν αιτήσεις από και προς την *master* πλευρά του data bus, και αυτό επιτρέπει στα *master components* να διαβάζουν τις πληροφορίες και να ελέγχουν τους accelerators. Το master κομμάτι αποτελείται από έναν *event processor*, ο οποίος αποτελείται από μικρές μηχανές καταστάσεων (FSM), και από έναν *μικροελεγκτή γενικού σκοπού*, ο οποίος χρησιμοποιείται σπάνια.

Το πλεονέκτημα που έχει η δομική αρχιτεκτονική σχεδίασης είναι η δυνατότητα που μας δίνει να κάνουμε διαχείριση ισχύος σε κάθε ένα ξεχωριστό component με ατομικό τρόπο (και στους masters και στους accelerators). Αν θέλουμε μπορούμε να "σβήσουμε" κάποια components και με τη χρήση του VDD-gating να ελαχιστοποίησουμε το leakage current του συστήματος. Για παράδειγμα, ο μικροελεγκτής γενικού σκοπού μπορεί να είναι σχετικά

περίπλοκος και ενεργοβόρος όταν είναι ενεργός, όμως μπορούμε να τον έχουμε σε κατάσταση VDD-gated τον περισσότερο χρόνο που είναι ανενεργός (idle). Ο event processor διαχειρίζεται όλα τα interrupts, κατανέμει τις εργασίες στους accelerators και ξυπνάει τον μικροελεγκτή μόνο εφόσον χρειάζεται (δηλαδή σπάνια).

Τώρα θα περιγράψουμε μερικά από τα πιο ενδιαφέροντα components του συστήματος με μεγαλύτερη λεπτομέρεια.

- *Σύστημα του bus.* Όπως ήδη έχουμε πει το system bus αποτελείται από το data bus, το interrupt bus, και τις γραμμές διαχείρισης ισχύος. Το data bus έχει διεύθυνση, δεδομένα και σήματα ελέγχου που ξεκλειδώνουν λειτουργίες ανάγνωσης και γραφής.

- *Μικροελεγκτής.* Ο μικροελεγκτής είναι μια απλή CPU γενικού σκοπού που εκτελεί το σετ εντολών z80. Από το να σχεδιάσουμε έναν επεξεργαστικό πυρήνα "άτσαλα" και με διάφορα τεχνικά και λειτουργικά λάθη, καλύτερο είναι να τροποποιήσουμε έναν ήδη υπαρκτό επεξεργαστικό πυρήνα ο οποίος θα εκτελεί υπολογισμούς γενικού σκοπού. Ο μικροελεγκτής χρησιμοποιείται κυρίως για να διαχειρίζεται ανώμαλες καταστάσεις του συστήματος.

- *Event processor.* Ο event processor (επεξεργαστής συμβάντων) είναι μια προγραμματιζόμενη μηχανή καταστάσεων που μπορεί να επεξεργαστεί βασικές μεταφορές δεδομένων και εντολές διαχείρισης ισχύος. Στην ουσία είναι ο "ενορχηστρωτής" για τις εργασίες που αφορούν όλα τα υπόλοιπα components του συστήματος. Δεν εκτελεί υπολογισμούς, αλλά δουλεύει *αναδραστικά με ρουτίνες εξυπηρέτησης διακοπών (Interrupt Service Routines)* που μεταφέρουν δεδομένα ανάμεσα στα blocks και αρχικοποιούν τα accelerator components (ή τον επεξεργαστή γενικού σκοπού) για να κάνουν υπολογισμούς και να εξυπηρετούν τα τρέχοντα interrupts. Οι ρουτίνες εξυπηρέτησης διακοπών περιέχουν επίσης εντολές διαχείρισης ισχύος που επιτρέπουν στον event processor να κάνει απευθείας διαχείριση ισχύος σε όλα τα άλλα components του συστήματος.

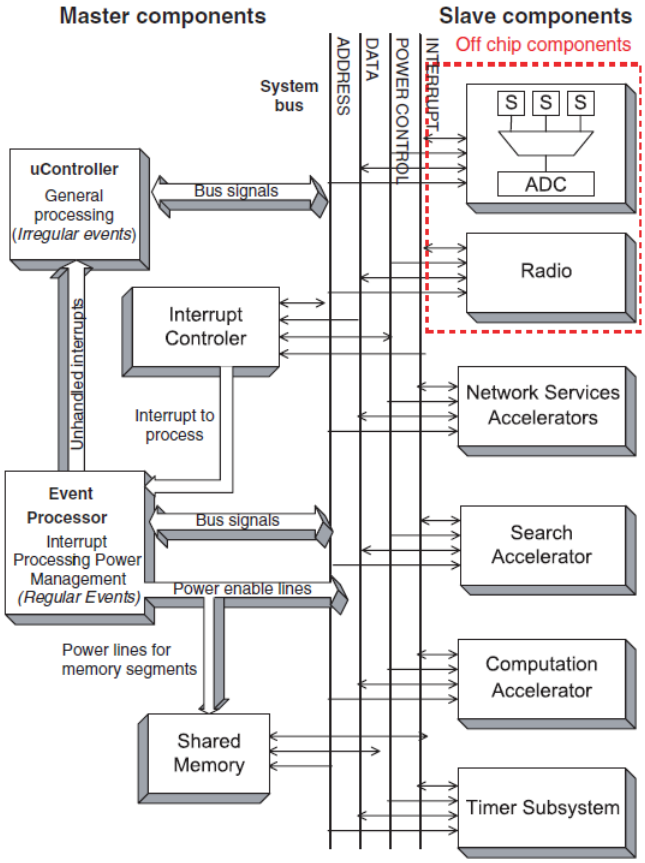
- *Ελεγκτής διακοπών.* Ο ελεγκτής διακοπών επιλέγει για εξυπηρέτηση μία από τις γραμμές διακοπών. Παρέχει απλή επιλογή με προτεραιότητα και ικανότητα "μεταμφίεσης" των διακοπών.

- *Υποσύστημα timer.* Η περιοδική φύση των εφαρμογών στα δίκτυα αισθητήρων χρειάζεται πολλαπλούς και παραμετροποιημένους timers. Το υποσύστημα timer υποστηρίζει τέσσερεις μετρητές των 16-bit που μπορούν να χρησιμοποιηθούν ώστε να "ξυπνάει" ο event processor για να διαχειρίζεται διαφορετικά συμβάντα.

Οι προσομοιώσεις του συστήματος βασίζονται στο μοντέλο αρχιτεκτονικής SystemC το οποίο εμφανίζει 10 φορές καλύτερη απόδοση για την ίδια εφαρμογή σε σχέση με ένα αντίστοιχο σύστημα που βασίζεται στο εμπορικό μοντέλο Mica2 mote. Αυτή η βελτιστοποίηση για τακτικές εργασίες επιτρέπει το σύστημα είτε να τρέχει με ρολόι 10 φορές πιο αργά είτε να τρέχει γρηγορότερα αφήνοντας περισσότερο χρόνο στο VDD-gate του hardware των accelerators.

Το σύστημα έχει υλοποιηθεί σε τεχνολογία CMOS των 130nm, έχει 4KB μνήμη και τρέχει σε τροφοδοσία ισχύος από 0.55V μέχρι 1.2V. Οι μετρήσεις που έγιναν στο σύστημα δείχνουν

εξοικονόμηση leakage power κατά 100 φορές με VDD-gating και μια συνολική ωφέλιμη ενέργεια των 680 pJ ανά εργασία (μια εργασία είναι ισοδύναμη με 1500 εντολές στον Atmel ATmega 128L). Το σύστημα τρέχει μέχρι τα 12.5 MHz σε τροφοδοσία 0.55V και υψηλότερες συχνότητες απαιτούν υψηλότερες τάσεις τροφοδοσίας.




*Σχήμα 2.5. Block diagram of the Harvard event-driven system*

| System | Arch style | Data path width | Event driven (y/n) | Circuit techniques | Accelerators | Memory (KB) | Process | Voltage (V) | Throughput (MIPS) | Energy (pJ/ins) |
|---|---|---|---|---|---|---|---|---|---|---|
| Atmel ATmega128L | GP Off-the-shelf | 8 | N | N | N | 132 KB | 350 nm | 3.0 V | 7.3 MHz | 3200 |
| TI MSP430 | GP Off-the-shelf | 16 | N | N | N | 10 KB | NA | 3.0 | 8 MHz | 750 |
| SNAP/LE | GP RISC | 16 | Y | Asyncronous | Timer, message interface | 8 KB | 180 nm | 1.8<br>0.6 | 200<br>23 | 218<br>24 |
| BitSNAP | GP RISC. Bit-serial datapath | 16 | Y | Asyncronous | Timer, message interface | 8 KB | 180 nm | 1.8<br>0.6 | 54<br>6 | 152<br>17 |
| Smart Dust | GP RISC | 8 | N | Syncronous-two clocks | None | 3.125 KB | 250 nm | 1.0 | 0.5 (500 kHz) | 12 |
| Charm | Protocol processor | NA | N | Two power domains | Custom radio stack | 68 KB | 130 nm | 1.0 V (high)<br>0.3–1.0 V (low) | 8 MHz | 150 $\mu$W<br>53.6 $\mu$W leakage |
| Michigan 1 | GP | 8 | Y | Subthreshold | None | 0.25 KB | 130 nm | 0.360 | 833 kHz | 2.6 |
| Michigan 2 | GP | 8 | Y | Subthreshold | None | 0.3125 | 130 nm | 0.350 | 354 kHz | 3.52 |
| Harvard | Event driven accelerator | 8 | Y | VDD-gating | Timer, filter, message proc | 4 KB | 130 nm | 0.55–1.2 | 12.5 MHz | 680 pJ/task |

*Πίνακας 2.1. Σύνοψη των hardware systems για Wireless Sensor Networks*

Ενότητα Τρίτη

# Διαχείριση μνήμης και caching για τη μείωση της κατανάλωσης ενέργειας και ισχύος σε WSN κόμβους

### Γενικά

Ένα πολύ σημαντικό πρόβλημα που έχουν τα παραπάνω hardware συστήματα WSN κόμβων είναι η μεγάλη κατανάλωση ενέργειας κατά την μεταφορά δεδομένων από το σύστημα μνήμης στον processor και αντίστροφα. Στις αρχές της προηγούμενης δεκαετίας έρευνες έδειξαν ότι τα μικρής κλίμακας λειτουργικά συστήματα πραγματικού χρόνου που χρησιμοποιούνται σε κόμβους αισθητήρων, TinyOS και ambientRT, αυξάνουν κατά πολύ την κατανάλωση ενέργειας σε εφαρμογές που απαιτούν εντατικούς υπολογισμούς και συχνές επικοινωνιακές μεταδόσεις [16]. Για αυτό το λόγο αναπτύχθηκαν, περίπου κατά τα μέσα της προηγούμενης δεκαετίας, νέες πλατφόρμες για κόμβους αισθητήρων σε αρχιτεκτονικές επεξεργαστών των 16 και των 32 bit έτσι ώστε να αντιμετωπιστεί το πρόβλημα των "ενεργοβόρων" εφαρμογών [17].Το 2003 μια άλλη ομάδα ερευνητών έδειξε ότι *ο processor από μόνος του καταναλώνει το 35% της συνολικής διαθέσιμης ενέργειας στην πλατφόρμα MICA2 όταν τρέχει Surge* [18]. Αξίζει επίσης να αναφερθεί ότι για μια σειρά δεδομένων καταναλώνεται υψηλότερη ενέργεια για το compression (συμπίεση) απ' ότι για το transmission (μετάδοση) [19].

Μετά από μελέτη των αποτελεσμάτων που προκύπτουν από τις διάφορες προσομοιώσεις και τα testbed, βρεθήκανε 2 πολύ σημαντικά χαρακτηριστικά που αφορούν τα δεδομένα των WSN. Το πρώτο είναι πως υπάρχει *τοπικότητα των τιμών μέσα σε σχετικά μεγάλες χρονικές περιόδους* και το δεύτερο η *ομοιότητα των τιμών των δεδομένων.*

1. **Χρονική τοπικότητα των τιμών:**

   Οι εφαρμογές των δικτύων αισθητήρων έχουνε *περιοδική* συμπεριφορά. Ειδικά, οι εφαρμογές ελέγχου, όπως το *Surge,* δουλεύουν σε σταθερά δεδομένα και σταθερές περιοχές μνήμης για μεγάλες διάρκειες. Γι' αυτό το λόγο οι εντολές αποθήκευσης (store instructions) στις WSN εφαρμογές είναι κατά μεγάλο ποσοστό "άσκοπες" (δηλαδή οι εντολές γράφουν τιμές που ταιριάζουν απόλυτα με τις ήδη αποθηκευμένες τιμές στις διευθύνσεις μνήμης που αναφέρονται).

2. **Ομοιότητα τιμών των δεδομένων:**

   έχει αποδειχθεί επίσης ότι υπάρχουν αρκετές ίδιες τιμές δεδομένων που κινούνται σε όλο τον επεξεργαστικό κύκλο της αρχιτεκτονικής εντολών (datapath).

## 3.1 Αρχιτεκτονική MoteCache

Τώρα θα εξηγήσουμε αναλυτικά την δομή της αρχιτεκτονικής της MoteCache (για συντομία από δω και στο εξής θα το λέμε MC) που στην ουσία δεν είναι τίποτα περισσότερο

από ένας απλός buffer που αξιοποιεί την χρονική τοπικότητα των δεδομένων στις WSN εφαρμογές. Στην MC, κάθε σειρά αποτελείται από buffers με αποθηκευμένα δεδομένα και μια *συσχετιστική μνήμη* (*associative content-addressable memory*) που κρατάει τις ετικέτες που αντιστοιχούν στις διευθύνσεις αυτών των buffers. Η βασική ιδέα είναι να διατηρούνται οι τιμές των δεδομένων των N πιο πρόσφατων προσπελάσεων διευθύνσεων σε μια MC των N-bytes [20].

### 3.1.1 Τα βασικά μοντέλα της MoteCache αρχιτεκτονικής

Οι τρεις τύποι/μοντέλα της MoteCache που συναντάμε κατά κόρον είναι:

1. MoteCache άμεσης απεικόνισης (DMMC από το direct-mapped MoteCache): Πρόκειται για την υιοθέτηση σε επίπεδο MoteCache της πιο κλασικής και της πιο συνηθισμένης τεχνικής caching που χρησιμοποιείται σε ευρεία κλίμακα αρχιτεκτονικών υπολογιστικών συστημάτων. Εφόσον η SRAM της πλατφόρμας MICA2 περιέχει αποθηκευτικό χώρο ενός και μόνο byte σε κάθε γραμμή, η cache άμεσης απεικόνισης προσαρμόζεται να περιέχει ένα byte από κάθε σύνολο. Η τεχνική DMMC έχει το μικρότερο latency μεταξύ όλων των τεχνικών, αφού μπορούμε να διαβάσουμε τα δεδομένα παράλληλα με τις συγκρίσεις των ετικετών για όσο χρόνο οι διευθύνσεις είναι διαθέσιμες.

2. Συσχετιστική MoteCache συνόλου (SAMC από Set-Associative MoteCache):
   Στην προσπάθεια μας να ελαττώσουμε τον ρυθμό αστοχίας που στο μοντέλο DMMC είναι σημαντικός και οφείλεται κυρίως σε αλληλοεκτοπίσεις δεδομένων της ίδιας θέσης cache, μπορούμε να επιλέξουμε την τεχνική SAMC η οποία είναι παραπλήσια με την γνωστή τεχνική της συσχετιστικής cache συνόλου, όμως απαιτεί περισσότερες συγκρίσεις για μια απλή πρόσβαση μνήμης.

3. Πλήρως συσχετιστική MoteCache (FAMC από Fully-Associative MoteCache):
   Η τεχνική αυτή ενεργοποιεί όλους τους συγκριτές ετικετών/διευθύνσεων για κάθε πρόσβαση στη μνήμη. Όμως καταναλώνει περισσότερη ισχύ συγκριτικά με τις άλλες τεχνικές, ενώ το latency δεν είναι καλύτερο από ότι στην DMMC εφόσον υπάρχει πρόσθετη καθυστέρηση πολυπλεξίας.

### 3.1.2 Πρόσβαση σε MoteCache

Μια πρόσβαση ανάγνωσης σε μια δομή MC εξελίσσεται ως εξής:

Πρώτος κύκλος ρολογιού. Σύγκριση ετικετών στη MoteCache:

Μόλις υπολογιστεί η διεύθυνση, το μέρος της ετικέτας συγκρίνεται συσχετιστικά με τους αριθμούς που συσχετίζονται με τα περιεχόμενα της MC. Εάν υπάρχει ευστοχία (MC-hit), αναβάλλεται η προσχεδιασμένη ανάγνωση της μνήμης, και ενδεχομένως εξοικονομείται η ενέργεια που θα χρειαζόταν για την ανάγνωση της σειράς στη μνήμη. Με άλλα λόγια, η σύγκριση ετικετών και η ανάκτηση των δεδομένων από τη MC ίσως να επικαλύπτονται για να μειωθεί το latency σε μια δομή DMMC.

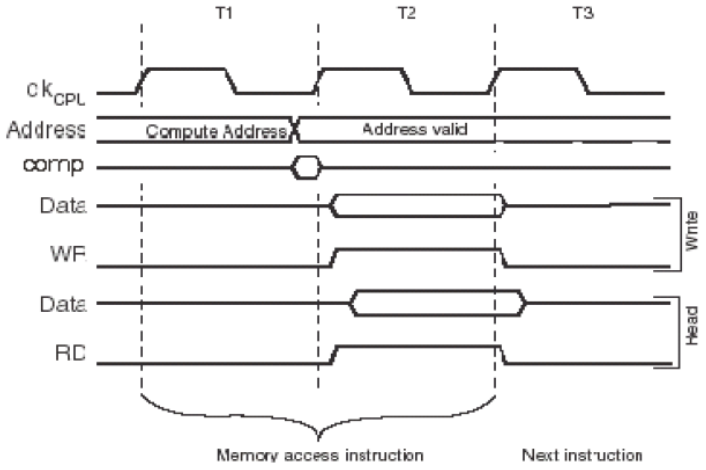
Δεύτερος κύκλος ρολογιού. Χειρισμός δεδομένων:

Στις περιπτώσεις που υπάρχει MC-hit, τα δεδομένα διαβάζονται από την αντίστοιχη καταχώρηση της MC. Όταν όμως υπάρξει MC-miss, η πρόσβαση μνήμης συνεχίζεται σε αυτό το κύκλο ρολογιού και τα δεδομένα που διαβάζονται καταχωρούνται σε μια θέση της MC η οποία επιλέγεται ως "θύμα", σύμφωνα με την LRU πολιτική (Least Recently used).

Το γράψιμο μιας νέας καταχώρησης στην MC γίνεται με τα γνωστά βήματα που ισχύουν στην αρχιτεκτονική υπολογιστών, τα οποία ακολουθεί ένα επιπλέον στάδιο το οποίο εγκαθιστά την καινούργια ετικέτα και το καινούρια δεδομένα στην αντίστοιχη θέση της MC. Αυτή η πολιτική είναι πιο βολική όταν σχεδιάζουμε low power hardware.

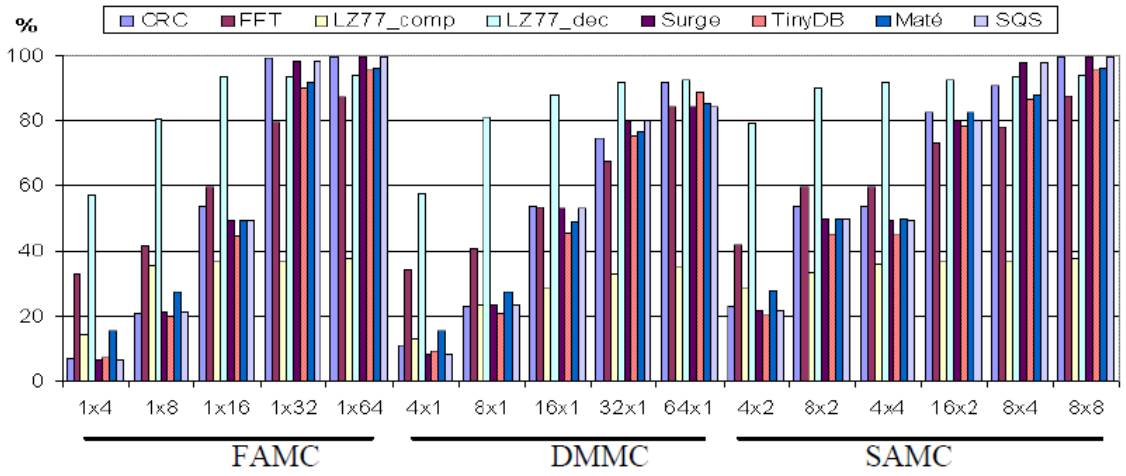
Στο σχήμα 3.1 φαίνεται το τροποποιημένο διάγραμμα χρόνου για την παραπάνω αρχιτεκτονική. Να σημειωθεί ότι, στο τέλος του πρώτου κύκλου ρολογιού T1, υπάρχει μια μικρή καθυστέρηση όπου συγκρίνεται η ετικέτα του συνόλου της MC με την υπολογισμένη διεύθυνση[1]. Αναλόγως του αποτελέσματος που προκύπτει από τη σύγκριση ετικέτας, γίνεται πρόσβαση στην MC ή στην SRAM.

Όπως άλλωστε είπαμε παραπάνω και είναι προφανές, αυτό που εγγυάται παραγωγή ευστοχίας MC-hit είναι η τοπικότητα των αναφορών. Στο σχήμα 3.2 δείχνεται ότι με τεχνική SAMC στο μέγιστο μέγεθος (8x8) ο μέσος ρυθμός ευστοχίας είναι περισσότερο από 89% . Αντιθέτως, ο χαμηλότερος ρυθμός ευστοχίας παρατηρείται στην τεχνική DMMC (4x1) και είναι περίπου 18.4%.




Σχήμα 3.1. *Χρονικό διάγραμμα κύκλων ρολογιού για πρόσβαση στα δεδομένα της SRAM και της MC*

[1] *(Μπορούμε να υποθέσουμε επίσης ότι η διαδικασία υπολογισμού της διεύθυνσης ολοκληρώνεται λίγο νωρίτερα από ότι σε μια κανονική σχεδίαση αρχιτεκτονικής. Αυτό είναι δυνατό, εφόσον οι μικροεπεξεργαστές που χρησιμοποιούνε οι κόμβοι αισθητήρων έχουν πολύ χαμηλή συχνότητα (περίπου 7 MHz). Επομένως, παρέχονται μεγάλοι κύκλοι ρολογιού για μικρές εργασίες τέτοιοι ώστε να είναι εφικτή η σύγκριση ετικέτας μέσα σε ένα κύκλο. Βέβαια αυτή δεν αποτελεί μια αυστηρή προϋπόθεση για να επιτύχουμε το στόχο μας. Πράγματι όμως, ένας κόμβος αισθητήρα μπορεί να είναι ανεκτικός σε μια μικρή αύξηση του χρονικού κύκλου ρολογιού του επεξεργαστή για σειριοποίηση του υπολογισμού της διεύθυνσης και της σύγκρισης ετικέτας στη MoteCache στον κύκλο T1, εφόσον φυσικά η ταχύτητα απόδοσης δεν αποτελεί τη μέγιστη προτεραιότητά μας.)*




*Σχήμα 3.2. Απεικόνιση των διαφόρων τεχνικών MC*

### 3.1.3 Silent-Store Filtering MoteCache

Όταν μια εντολή αποθήκευσης γράφει σε μια θέση μνήμης μια τιμή που είναι ίδια με την ήδη υπάρχουσα σε αυτή τη θέση, η εντολή αυτή λέγεται *silent-store.* Αυτές οι εντολές προφανώς δεν έχουνε καμία επίδραση στην μηχανή καταστάσεων και εφόσον εντοπιστούν μπορούν να απομακρυνθούν, να "φιλτραριστούν" δηλαδή έτσι ώστε να βελτιωθεί ο χρόνος ζωής ενός κόμβου αισθητήρα. Η ανάλυση των silent-store εντολών με τεχνικές εντοπισμού και απομάκρυνσης των εντολών αυτών σε επίπεδο αρχιτεκτονικής, με βάση τα δεδομένα της εποχής εκείνης εννοείται, παρουσιάστηκε για πρώτη φορά το 2001 από ομάδα ερευνητών σε ενιαίο άρθρο του IEEE [21]. Στο σχήμα 3.3 μπορούμε να δούμε ποσοστιαίες κατανομές των silent-store εντολών σε όλα τα συστήματα αναφοράς hardware των WSNs, επομένως μπορούμε να υπολογίσουμε ένα μέσο συνολικό ποσοστό το οποίο είναι ίσο με 81.6%, δηλαδή εξαιρετικά υψηλό.

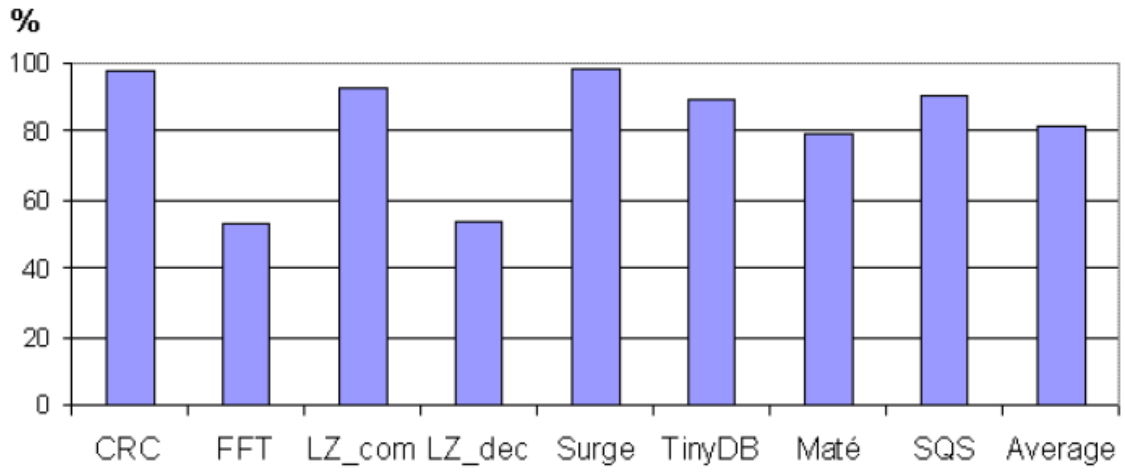
Για τον εντοπισμό και την απομάκρυνση των silent-store εντολών αξιοποιούμε και αναπτύσσουμε τη λειτουργικότητα του *dirty-bit.* Ως εκ τούτου, από δω και στο εξής θα κάνουμε μια μετονομασία του *dirty-bit* σε *dirty&noisy-bit*(DN). Μια τιμή θα γράφεται πίσω στην SRAM μόνο όταν αυτή είναι *βρώμικη* (δηλαδή τροποποιημένη), και επίσης *θορυβώδης* (δηλαδή no silent: αν η νέα τιμή είναι διαφορετική από αυτή που αποθηκεύτηκε προηγουμένως). Η διαδικασία πρόσβασης και εγγραφής με silent-store filtering MC έχει ως εξής:

1. ευστοχία MC και εντοπισμός Silent Store:

   Ύστερα από μία ευστοχία στην MC, η τιμή των δεδομένων που πρόκειται να εγγραφεί συγκρίνεται με την τιμή των δεδομένων στην αντίστοιχη γραμμή της MC. Εάν δεν είναι ίδιες, το DN-bit αυτής της γραμμής στην MC σημαδεύεται ώστε να δείχνει ότι η εντολή αποθήκευσης είναι *θορυβώδης.*

2. αστοχία MC και επαναγραψιμότητα στην SRAM:

Μετά από μια αστοχία στην MC, επιλέγουμε μια γραμμή ως "θύμα" για αντικατάσταση. Αν το DN-bit στην γραμμή "θύμα" είναι σημαδεμένο, τότε ακολουθείται η κανονική διαδικασία της επαναγραψιμότητας από την MC. Διαφορετικά, μπορούμε απλά να αντικαταστήσουμε την γραμμή "θύμα" της MC ακυρώνοντας την διαδικασία επαναγραψιμότητας, εφόσον τα αντίστοιχα δεδομένα της MC και της SRAM είναι ήδη σε συγχρονισμό, δηλαδή είναι τα ίδια.

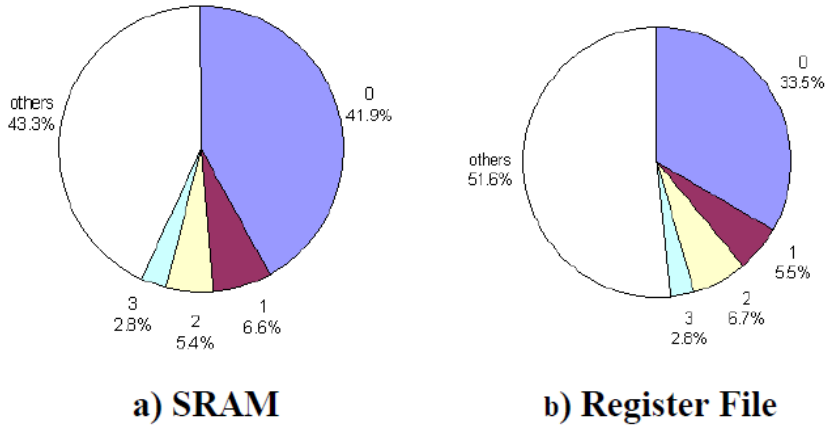



*Σχήμα 3.3. Ποσοστά "σιωπηλών" αποθηκεύσεων στα συστήματα WSN.*

### 3.1.4 Η τεχνική του Content-aware Data Management (CADMA)

Η τεχνική CADMA είναι μια από τις πιο σύγχρονες και πολλά υποσχόμενες τεχνικές στον τομέα της σχεδίασης ultra-low power hardware. Αναπτύχθηκε και εν τέλει προτάθηκε το 2007 από 2 Τούρκους ερευνητές του πανεπιστημίου Yeditepe της Κωνσταντινούπολης.

Κατά τις προσομοιώσεις που κάνανε πάνω στα μέχρι τότε συστήματα/πλατφόρμες για WSNs, παρατήρησαν ότι υπήρχαν κάποιες κοινές τιμές δεδομένων που κυκλοφορούσανε συχνά στο datapath (συγκεκριμένα τα 0,1,2, και 3). Συγκεκριμένα δείξανε ότι το 48.4% των αποθηκευμένων δεδομένων της SRAM και το 56.7% των δεδομένων στους καταχωριτές αποτελούνται από αυτές τις κοινές μεταβλητές δεδομένων [22].




*Σχήμα 3.4. Ποσοστιαία κατανομή των τιμών στην SRAM και στον Register File*

Η όλη φιλοσοφία του CADMA είναι η αξιοποίηση του φαινομένου των συνεχώς εμφανιζόμενων τιμών δεδομένων για τη μείωση της καταναλισκόμενης ενέργειας στην SRAM, την MC και τους καταχωρητές. Πως επιτυγχάνεται αυτό; Με την εισαγωγή ενός επιπλέον bit, που θα το λένε CD bit (από το common data), σε κάθε γραμμή αυτών των δομών αποθηκεύσης. Στο σχήμα 3.5 αποκονίζεται το ψηφιακό κυκλωματικό διάγραμμα σε λογική CMOS που χρειάζεται για τη μεταφορά του CD bit σε ένα byte μνήμης. Η CD διαδικασία γίνεται με τα εξής λειτουργικά βήματα:

1. Εγγραφή δεδομένων και CD Reset:

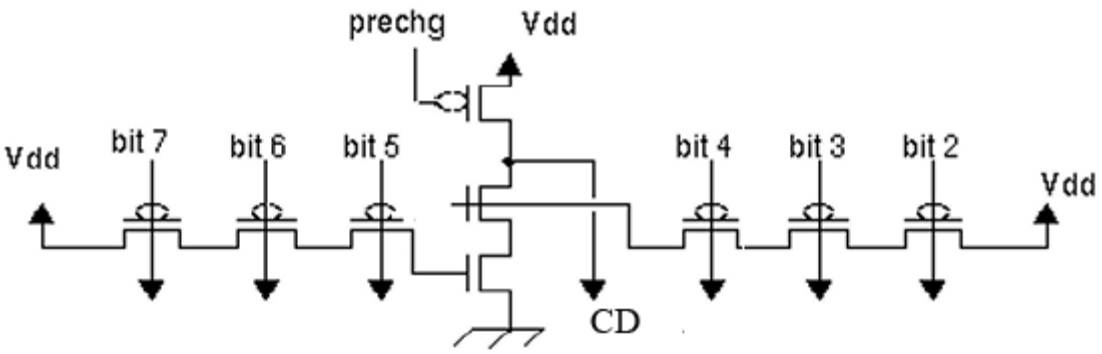
   Το CD bit τοποθετείται στο μηδέν όταν υπάρχει μια εγγραφή σε καταχωρητή ή σε μνήμη η οποία προσπαθεί να αποθηκεύσει μια κοινή τιμή. Σε αυτή τη περίπτωση, το CADMA γράφει μόνο 3 bits (το CD bit και τα 2 λιγότερο σημαντικά bits της κοινής τιμής) για αυτό το byte. Να σημειωθεί ότι για όλες τις κοινές τιμές των δεδομένων (0,1,2 και 3), δεν υπάρχει καμία ανάγκη για κωδικοποίηση και αποκωδικοποίηση, εφόσον ήδη ταιριάζουν μέσα στα 2 λιγότερο σημαντικά bits. Όταν η τιμή των δεδομένων δεν είναι μια από τις τέσσερεις κοινές τιμές, τότε το CADMA γραφεί 9 bits (το CD bit και τα 8 bits της τιμής) καταναλώνοντας *ελαφρώς περισσότερη ισχύ* σε σύγκριση με την συνηθέστερη περίπτωση.

2. Ανάγνωση δεδομένων και CD Probe:

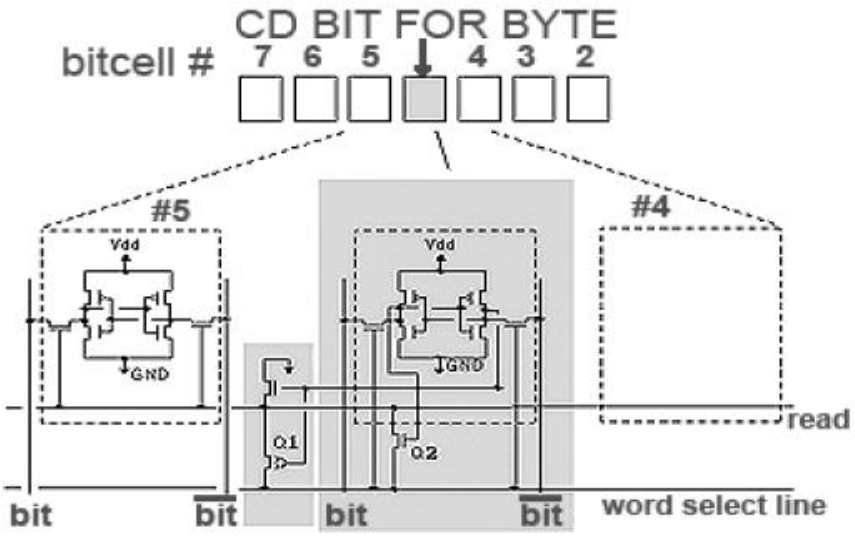
   Η ανάγνωση με CADMA αρχίζει με εξέταση του CD bit. Όταν είναι σε κατάσταση reset (δηλαδή έχει τιμή μηδέν), το CADMA ενεργοποιεί την ανάγνωση μόνο για τα 2 λιγότερο σημαντικά bits. Με αυτή την ακύρωση ανάγνωσης των υπολοίπων 6 bits επιτυγχάνεται σημαντική εξοικονόμηση ισχύος και για τις 3 δομές μνήμης. Σε αυτή τη περίπτωση, το υπόλοιπο κομμάτι του byte (δηλαδή αυτά τα 6 bits) ανασκευάζεται απλώς βάζοντας το λογικό-0 στα εναπομείναντα 6 σημαντικότερα bits. Το κυκλωματικό μοτίβο CMOS λογικής που χρησιμοποιείται για να απενεργοποιήσει την ανάγνωση των 6 περισσότερο σημαντικών bits φαίνεται στο σχήμα 3.6. Το PMOS transistor Q1 και το NMOS transistor Q2 χρησιμοποιούνται για να συνδέσουν την γραμμή ανάγνωσης που αντιστοιχεί στα 6 πιο σημαντικά bits του ενός byte στη word select line εφόσον η τιμή του CD bit είναι 1. Διαφορετικά, η γραμμή ανάγνωσης γειώνεται και τα 6 πιο σημαντικά bits του byte δεν διαβάζονται.

   Αν το CD bit είναι σε κατάσταση set (δηλαδή 1), όπως και στην εγγραφή, καταναλώνεται *ελαφρώς περισσότερη ισχύς* έτσι ώστε να διαβαστούν τα 9 bits.

Σε πειράματα προσομοίωσης που έγιναν σε hardware με ολοκληρωμένο σύστημα CADMA μέσα σε MC, σε SRAM και σε δομές καταχωρητών παρατηρήθηκε ότι το CADMA καταφέρνει εξοικονόμηση ενέργειας στον επεξεργαστή περισσότερο από 10%, και συνολική εξοικονόμηση ενέργειας του κόμβου αισθητήρα 5% σε σύγκριση πάντα με μια MC χωρίς CADMA.




*Σχήμα 3.5. Κύκλωμα CMOS για τη μεταφορά του CD bit σε ένα byte μνήμης*




Σχήμα 3.6 . Κυκλωματικά μοτίβα ελέγχου των 6 περισσότερο σημαντικών στοιχείων bit σε ένα byte από *το CD bit.*

Ενότητα Τέταρτη

# Αλγόριθμος μερικών αθροισμάτων (Prefix Sum algorithm)

### Βασική ιδέα

Ο αλγόριθμος μερικών αθροισμάτων χρησιμοποιείται για να υπολογίσει τα μερικά αθροίσματα δοθέντος ενός πίνακα τυχαίων αριθμών. Πρόκειται για μία φαινομενικά σειριακή διαδικασία, καθώς πρέπει κανείς να προσθέσει μέχρι τα **κ** πρώτα στοιχεία προτού φτάσει στην πρόσθεση του **κ + 1** στοιχείου του πίνακα. Ωστόσο η σειριακή εκτέλεση του αλγορίθμου απαιτεί μεγάλο χρόνο εκτέλεσης και σκοπός είναι να αποφεύγεται έτσι ώστε να αποφύγουμε τις περιττές πράξεις που αυξάνουν την πολυπλοκότητα του αλγορίθμου. Αντ' αυτού προτείνεται η παράλληλη εκτέλεση του αλγορίθμου που όπως θα φανεί στις επόμενες παραγράφους, μειώνει σημαντικά την πολυπλοκότητά του.[24]

Είσοδος

| 0 | ... | | | | | ... κ | κ + 1 |
|---|---|---|---|---|---|---|---|
| 3 | 1 | 7 | 0 | 4 | 1 | 6 | 3 |

Έξοδος

| 0 | ... | | | | | ... κ | κ + 1 |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 11 | 11 | 14 | 16 | 22 | 25 |

*Εικόνα 4.1. Είσοδος στοιχείων στον αλγόριθμο και η έξοδός του (μερικά αθροίσματα).*

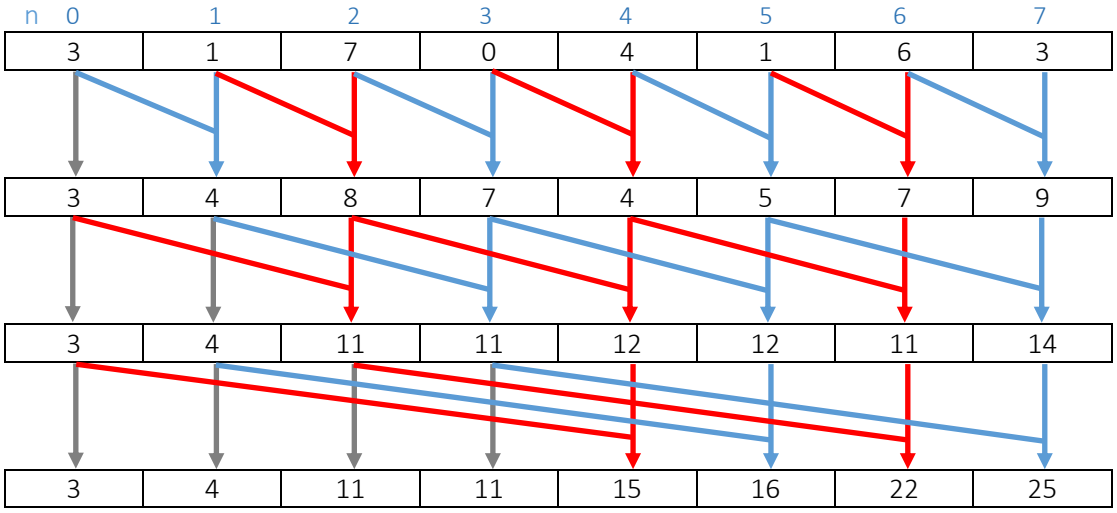
Η χρήση του αλγορίθμου μερικών αθροισμάτων (all-prefix-sums) είναι ευρεία. Παρακάτω φαίνεται η λίστα των περιπτώσεων που χρησιμοποιούν τον αλγόριθμο:

1. Για να γίνει σύγκριση μεταξύ των χαρακτήρων λέξεων (strings) με σκοπό την ταξινόμηση τους. Για παράδειγμα η λέξη η λέξη «παράμετρος» θα πρέπει σε ένα λεξικό να εμφανίζεται πριν από τη λέξη «παρουσία» σε ένα λεξικό.
2. Για την πρόσθεση αριθμών με πολλά δεκαδικά ψηφία ώστε να επιτύχουμε καλύτερη ακρίβεια όπου απαιτείται.
3. Στον υπολογισμό αναδρομικών σχέσεων.
4. Στην εκτέλεση αλγορίθμων όπως ο radix sort και quicksort.
5. Για την διαγραφή μαρκαρισμένων στοιχειών σε έναν πίνακα.
6. Για την εκτέλεση κάποιων πράξεων σε δέντρα όπως την εύρεση του βάθους ενός δέντρου κτλ. [24]

## 4.1 Αρχική εκδοχή του παράλληλου αλγορίθμου πρόσθεσης

Στην εικόνα 4.2 φαίνεται η διαδικασία εκτέλεσης του παράλληλου αλγορίθμου πρόσθεσης. Ο αλγόριθμος προσθέτει παράλληλα τα στοιχεία του πίνακα σε $log_2(n)$ βήματα, όπου $n$ το πλήθος των στοιχείων. Το κάθε στοιχείο προστίθεται με αυτό που βρίσκεται $2^{i-1}$

θέσεις δεξιότερα, όπου *i* είναι ο αριθμός του βήματος. Οι προσθέσεις σε κάθε βήμα είναι $n - 2^{i-1}$ και γνωρίζοντας ότι τα βήματα *(i)* παίρνουν τιμές από 1 έως και $log_2(n)$ στο τελευταίο βήμα, οι προσθέσεις που θα γίνονται σε κάθε βήμα θα έχουν την εξής ακολουθία *(n - 1), (n - 2), (n - 4), … (n – n/2)*.




*Εικόνα 4.2. Τα βήματα εκτέλεσης του αλγορίθμου παράλληλης πρόσθεσης*

```
1: for i ← 1 to log2n do
2:         for all k in parallel do
3:                 if k ≥ 2^i then
4:                         x[k] ← x[k - 2^i] + x[k]
5:                 else
6:                         x[k] ← x[k]
```

*Εικόνα 4.3 : Αλγόριθμος του Horn για τον υπολογισμό των μερικών αθροισμάτων δοθέντος ενός πίνακα x που περιέχει n στοιχεία.* [25]

Οι συνολικές προσθέσεις που κάνει ο αλγόριθμος είναι : $n * log_2(n) - (n-1) \rightarrow O( n * log_2(n) )$

Ο αλγόριθμος μπορεί να πετύχει πολυπλοκότητα $O(log_2 n)$ όταν υπάρχουν *n* ή περισσότεροι επεξεργαστές για παράλληλη επεξεργασία. Όταν όμως το πλήθος των στοιχείων του πίνακα είναι μεγαλύτερο από τον αριθμό των επεξεργαστών που μπορούν να χρησιμοποιηθούν τότε ο αλγόριθμος εκτελείται σειριακά και όχι παράλληλα. Αυτό μπορεί να θεωρηθεί και ως κριτήριο παραλληλισμού του αλγορίθμου.[24] [25]

## 4.2 Παράλληλος αλγόριθμος υπολογισμού μερικών αθροισμάτων προσέγγιση Blelloch

Με σκοπό την μείωση των χρονοβόρων υπολογισμών – πράξεων σε κάθε βήμα ο Blelloch πρότεινε μία εναλλακτική μέθοδο χρησιμοποιώντας δυαδικά δέντρα, κατά την οποία

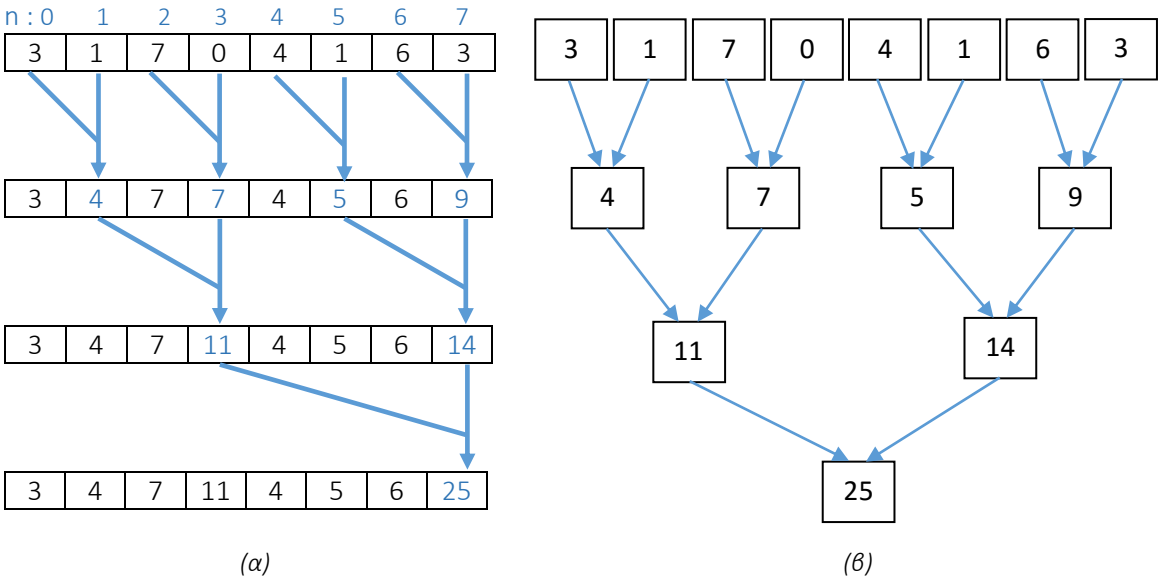
ο αλγόριθμος σπάει σε δύο τμήματα. Το πρώτο τμήμα καλείται trunk-phase και το δεύτερο twig-phase. [24] [25]

## A. Up-Sweep phase

Σε πρώτη φάση ο πίνακας εισόδου μειώνεται από *n* στοιχεία σε *1* όπου είναι και το άθροισμα όλων των στοιχείων του. Σε κάθε βήμα το μέγεθος του πίνακα μειώνεται στο μισό αθροίζοντας μη επικαλύπτοντα γειτονικά ζεύγη των στοιχείων του. Στη εικόνα 4.4 φαίνεται ο ψευδοκώδικας της 1ης φάσης διαπέρασης του δέντρου όπως αυτό διαμορφώνεται από τον πίνακα εισόδου *n* στοιχείων (Up-Sweep phase). Στην εικόνα 4.5 (β) φαίνεται το προκύπτον δυαδικό δέντρο και η εκτέλεση της 1ης φάσης. Όπως φαίνεται στην πρώτη φάση η διαπέραση του δυαδικού δέντρου γίνεται από τα φύλλα του προς τον πατέρα. [24] [25]

```
1: for d ← 1 to log2n do
2:         for i ← 1 to n/2^d - 1 in parallel do
3:                 ad[i] ← ad-1 [2i] + ad-1 [2i+1]
```

*Εικόνα 4.4. Ψευδοκώδικας για την 1η φάση (Up-Sweep phase) διαπέρασης ενός πίνακα με n στοιχεία [24] [25]*




*Εικόνα 4.5. Τα βήματα εκτέλεσης της 1ης φάσης (Up-Sweep phase) (α) και το δυαδικό δέντρο όπως αυτό προκύπτει.*

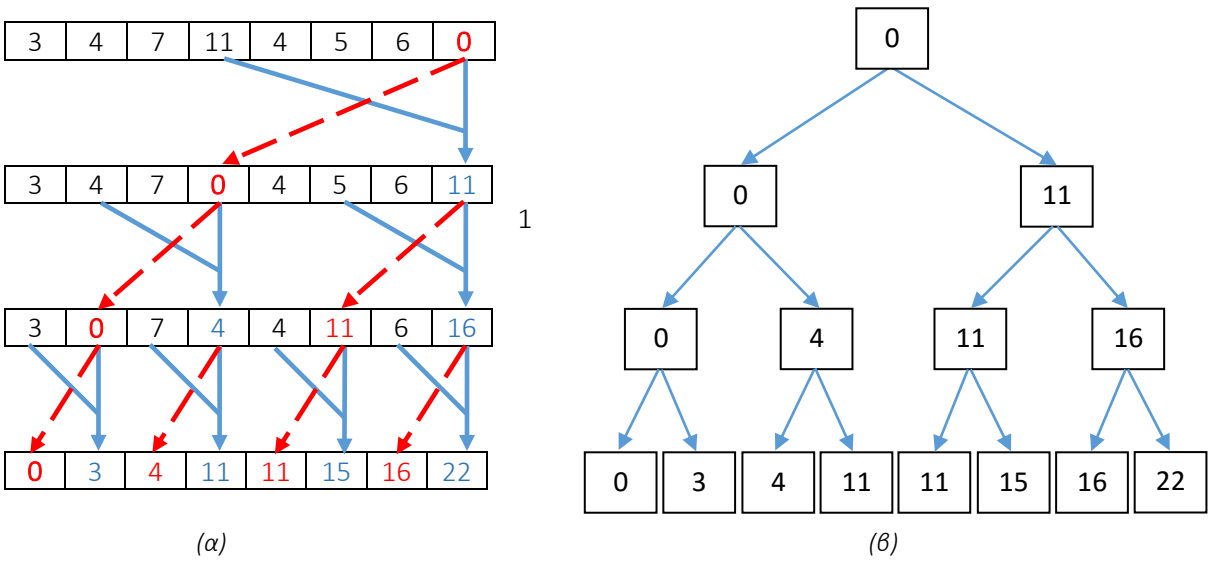
## B. Down-Sweep phase

Στη Δεύτερη φάση του αλγορίθμου (down-sweep phase) τα μερικά αθροίσματα που έχουν προκύψει από την πρώτη φάση (Up-Sweep phase) χρησιμοποιούνται με σκοπό τον υπολογισμό των τελικών μερικών αθροισμάτων. Αυτή τη φορά η διαπέραση του δέντρου γίνεται από τον πατέρα προς τα φύλλα του. Στην εικόνα 4.6 φαίνεται ο ψευδοκώδικας της Down-Sweep φάσης. [24] [25]

```
1: for d ← (log2n) − 1 downto 0 do
2:         for i ← 0 to n/2^d −1 in parallel do
3:                 if i > 0 then
4:                     if (i mod 2) ≠0 then
5:                         a_d[i] ← a_d+1[i/2]
6:                     else
7:                         a_d[i] ← a_d[i] + a_d+1 [(i/2)−1]
8:                 else
9:                     a_d[i] ← a_d[i]
```

*Εικόνα 4.6. Ψευδοκώδικας της 2ης φάσης διαπέρασης του πίνακα n στοιχείων. Όπου d είναι το βάθος του δέντρου και στον πίνακα $a_d$ σώζονται τα τρέχοντα μερικά αθροίσματα.* [24] [25]

Η τιμή της ρίζας του δένδρου που είναι το συνολικό άθροισμα (25) γίνεται μηδέν (0) και είναι η είσοδος του δένδρου. Η διαπέραση του γίνεται με τη μέθοδο preorder. Σε κάθε βήμα η κάθε κορυφή του δένδρου θα δίνει την τιμή που έχει στο εκάστοτε αριστερό παιδί της και το δεξί παιδί θα κληρονομεί το αποτέλεσμα του αθροίσματος, μεταξύ της προκείμενης κορυφής και του αριστερού παιδιού, όπως αυτό είχε υπολογιστεί στην Up-Sweep διαδικασία. Στο παράδειγμα της παρακάτω εικόνας το 0 θα κληρονομηθεί από το αριστερό παιδί της κορυφής. Το δεξί παιδί της ίδιας κορυφής θα κληρονομήσει την τιμή του αποτελέσματος του αριστερού παιδιού του δένδρου που προέκυψε στην 1η φάση εκτέλεσης (11) (Up-Sweep) Σχήμα 4.5 (β), αθροισμένο με την τιμή της κορυφής που εξετάζουμε εκείνη τη στιγμή (0). Επομένως είναι σημαντικό όπως θα δούμε παρακάτω να έχουμε κρατήσει σε κάποιο στοιχείο μνήμης της τιμές των αθροισμάτων που προκύπτουν στην 1η φάση μόνο από τα αριστερά φύλλα-παιδιά.




*Εικόνα 4.7. Βήματα εκτέλεσης της 2ης φάσης (Down-Sweep phase) (α) και το δυαδικό δέντρο (β) όπως προκύπτει.*

Η χρονική πολυπλοκότητα και για τις 2 φάσεις είναι :

$$T_R(n, p) = \lceil n/p \rceil + \lceil \log_2 p \rceil = O(n/p + \log_2 p)$$

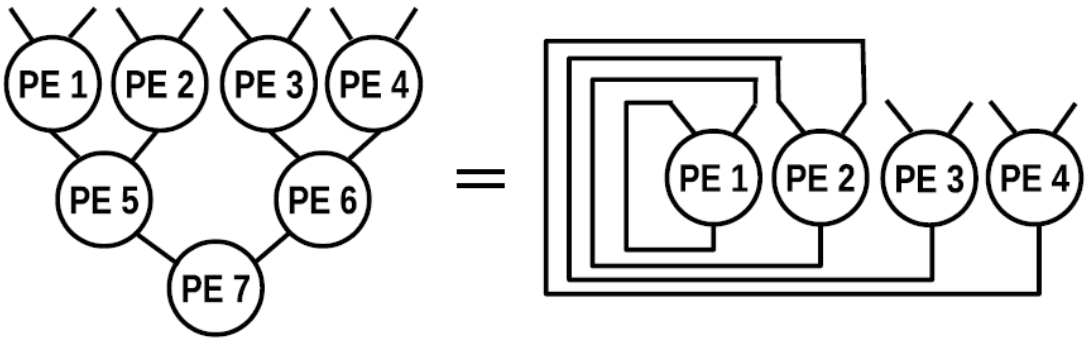
Όταν $n/p \geq \log_2 p$ η πολυπλοκότητα είναι $O(n/p)$ [25]

$p$ : ο αριθμός των επεξεργαστών
$n$ : Πλήθος των στοιχείων του πίνακα

Στόχος της μεταπτυχιακής εργασίας είναι η υλοποίηση του αλγορίθμου μερικών αθροισμάτων σε επίπεδο αρχιτεκτονικής, άλλα και πως μπορούμε μέσω αυτής να μειώσουμε την κατανάλωση ισχύος σε μικροεπεξεργαστές που χρησιμοποιούνται στα σύγχρονα Wireless Sensor Networks. Για να το πετύχουμε αυτό θα πρέπει να θεωρήσουμε τα δένδρα, όπως αυτά παρουσιάστηκαν προηγουμένως, ως ένα ξεχωριστό κύκλωμα που μπορεί να λειτουργεί ως ένα δυαδικό δένδρο. Αυτό σημαίνει ότι το κύκλωμα που προκύπτει εκτελώντας τον αλγόριθμο θα αποτελείται από τέτοια στοιχεία που θα στηρίζονται στην αρχή λειτουργίας του αλγορίθμου μερικών αθροισμάτων και κατά συνέπεια ενός δυαδικού δένδρου. Αυτό μπορεί να επιτευχθεί εφόσον θεωρήσουμε πως οι κορυφές του είναι αυτόνομα Processor Elements (PEs) τα οποία μπορούν να εκτελούν τις πράξεις που απαιτούνται κάθε φορά βασιζόμενοι στον τρόπο εκτέλεσης του αλγορίθμου.

## 4.3 Εμφωλευμένα Δένδρα

Εφόσον έχουμε θεωρήσει τις κορυφές του δυαδικού δένδρου ως PEs (processor elements), η πρώτη σκέψη είναι να δούμε πόσα PEs χρειάζονται για N στοιχεία εισόδου. Όπως είδαμε στην προηγούμενη ανάλυση για N στοιχεία εισόδου απαιτούνται N-1 PEs. Προκειμένου να μειώσουμε τον χώρο που θα απαιτηθεί αλλά και την κατανάλωση ισχύος η ιδέα που προτείνεται είναι η χρήση εμφωλευμένων δένδρων τα οποία θα επαναχρησιμοποιούν κάποια από τα PEs που διαθέτουν. Με αυτό τον τρόπο το πλήθος των PEs που θα χρησιμοποιηθούν μειώνεται στο μισό και το ίδιο ο χώρος που απαιτείται. Επίσης πέρα από τα PEs που επαναχρησιμοποιούνται, γίνεται και επαναχρησιμοποίηση των συνδέσεων εσωτερικά του ολοκληρωμένου κυκλώματος. Η τοπολογία του εμφωλευμένου δένδρου φαίνεται στο σχήμα QQ.



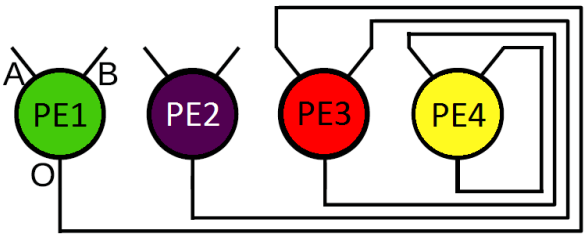

*(α)* *(β)*

*Εικόνα 4.8. Τοπολογία δυαδικού δένδρου ως κύκλωμα χρησιμοποιώντας PEs (α) και δίπλα το ισοδύναμο εμφωλευμένο.*

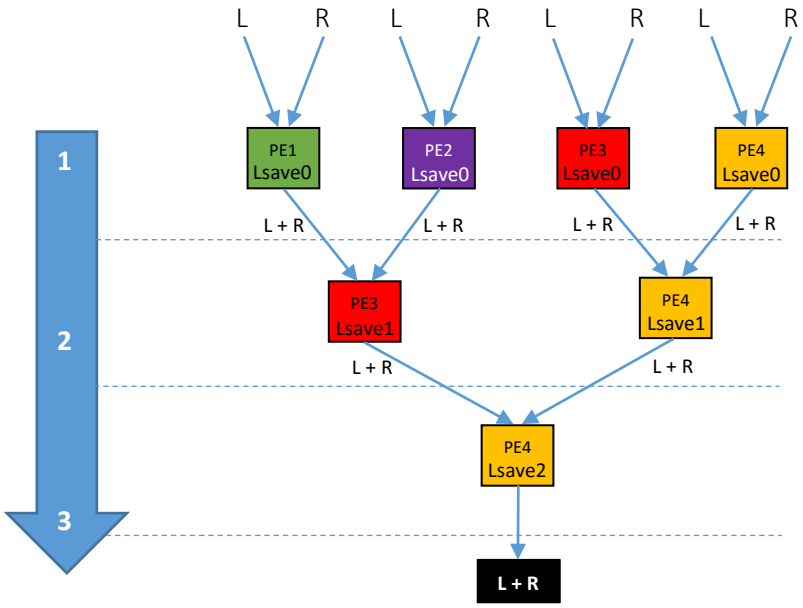
## 4.4 Υλοποίηση του αλγορίθμου σε επίπεδο κυκλώματος με τη χρήση εμφωλευμένων δένδρων

Σε αυτή την παράγραφο θα δείξουμε με ποιον τρόπο μπορεί να υλοποιηθεί ένα δυαδικό δένδρο σε επίπεδο αρχιτεκτονικής – κυκλώματος με τη χρήση εμφωλευμένου δένδρου στηριζόμενοι στον αλγορίθμου του Blelloch για τον υπολογισμό των μερικών αθροισμάτων. Στην εικόνα 4.9 φαίνεται το ανάπτυγμα ενός εμφωλευμένου δένδρου που χρησιμοποιεί 4 PEs.

### A. 1η Φάση (Up-Sweep phase)




*(α)*




*(β)*

*Εικόνα 4.9. Ανάπτυγμα – τοπολογία ενός εμφωλευμένου δένδρου 1ης φάσης (Up-Sweep). Σε κάποια από τα PEs η τιμή του αθροίσματος που προκύπτει από τα προηγούμενα αποθηκεύεται πάνω από μία φορές εφόσον επαναχρησιμοποιείται (PE3, PE4).*

Ο αλγόριθμος μερικών αθροισμάτων (Blelloch) λειτουργεί ως εξής. Κάθε κορυφή (PE) αποθηκεύει την τιμή του αριστερού φύλλου. Την πρώτη φορά θα αποθηκεύσει την τιμή εισόδου και στη συνέχεια τα αποτελέσματα των αθροισμάτων όπως αυτά προκύπτουν σε κάθε

βήμα. Με «Lsave"x"» έχει σημειωθεί πάνω σε κάθε κορυφή ποια τιμή θα αποθηκεύεται (L = Left όπου x είναι το βήμα). Με «L + R» συμβολίζεται το άθροισμα που προκύπτει μεταξύ του αριστερού «L» και δεξιού «R» φύλλου. Η αποθήκευση των αριστερών φύλλων είναι αναγκαίο να γίνεται σε κάθε βήμα στην 1η φάση του αλγορίθμου (Up-Sweep), έτσι ώστε να μπορέσει να πραγματοποιηθεί ο τελικός υπολογισμός των μερικών αθροισμάτων στην 2η φάση (Down-Sweep). Επομένως η χρήση κυκλωματικών στοιχείων μνήμης είναι απαραίτητη. Η πράξη της πρόσθεσης, άλλα και άλλες εφόσον αυτές χρειάζονται, μπορούν να γίνουν με τη χρήση μιας αριθμητικής/λογικής μονάδας ALU η οποία θα υπάρχει σε κάθε PE.

Παρατηρούμε ότι σε κάθε στάδιο η τιμή του αριστερού φύλλου αποθηκεύεται ενώ το άθροισμα που προκύπτει περνάει στο επόμενο PE κοκ, μέχρι να καταλήξουμε στο τελικό αποτέλεσμα που είναι και το άθροισμα των αριθμών εισόδου του δένδρου. Στο 1ο στάδιο γίνεται χρήση όλων των PEs, στο 2ο στάδιο γίνεται επαναχρησιμοποίηση των PE3 και PE4 και στο 3ο στάδιο γίνεται η χρήση μόνο του PE4. Στα PEs που επαναχρησιμοποιούνται η αποθήκευση των τιμών των «L» φύλλων θα πρέπει να γίνονται σε διαφορετικές θέσεις μνήμης, επομένως θα χρειαστεί η προσθήκη στοιχείου μνήμης με περισσότερες από μία διευθύνσεις όπως ένας Register File. Για την ακρίβεια η θέσεις μνήμης που θα χρειαστούν για ένα PE θα είναι τουλάχιστον $log_2(N)$, όπου $N$ είναι ο αριθμός των στοιχείων εισόδου.

Παρακάτω φαίνεται ο πίνακας των εντολών που απαιτούνται για την εκτέλεση του αλγορίθμου μερικών αθροισμάτων.

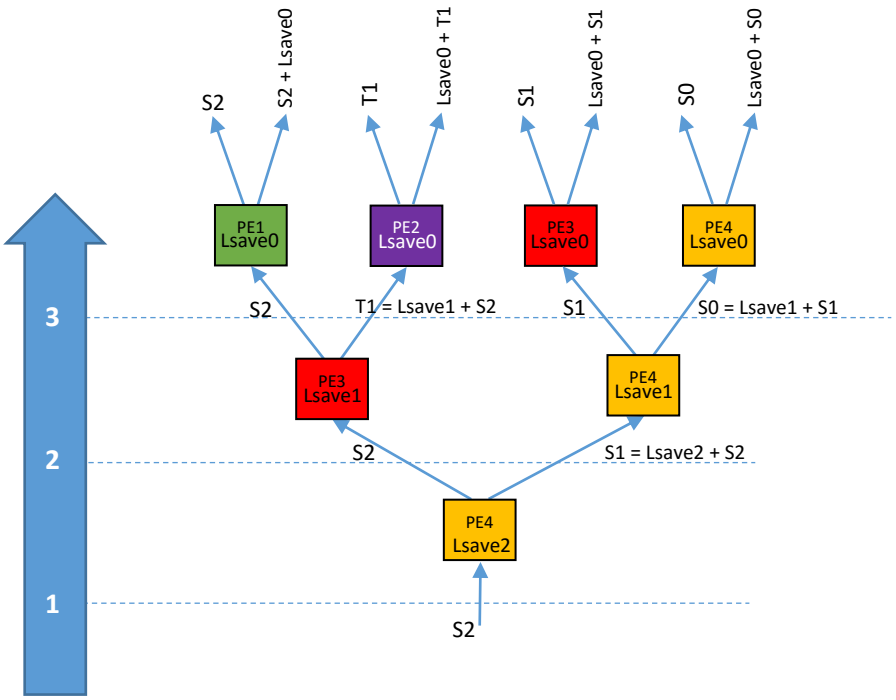
| Label | description | Write RF | | Read RF | | Operation |
|---|---|---|---|---|---|---|
| | | A | B | A | B | |
| 0: **T** | L → Lsave0 & L + R → O | inputA @ 0b00 | inputB @ 0b00 | 0b00 @ Lsave0 | 0b00 @ R | ADD A+B |
| 1: **T** | L → Lsave1 & L + R → O | inputA @ 0b01 | inputB @ 0b01 | 0b01 @ Lsave1 | 0b01 @ R | ADD A+B |
| 2: **T** | L → Lsave2 & L + R → O | inputA @ 0b10 | inputB @ 0b10 | 0b10 @ Lsave2 | 0b10 @ R | ADD A+B |

*Πίνακας 4.1. Πίνακας εντολών εκτέλεσης του αλγορίθμου μερικών αθροισμάτων με τη χρήση εμφωλευμένων δένδρων για την 1η φάση του αλγορίθμου Up-Sweep. [23]*

Ο παραπάνω πίνακας μας δείχνει τη σειρά των εντολών που εκτελούνται κατά το τρέξιμο του αλγορίθμου για κάθε PE ξεχωριστά. Και τα τρία βήματα εκτελούνται μόνο από το PE4. Σε κάθε βήμα η τιμή του αριστερού φύλλου αποθηκεύεται ενώ το άθροισμα περνάει στην έξοδο του PE. Η αποθήκευση και η ανάκτηση των τιμών σε κάθε PE γίνονται σε δύο διαφορετικούς τοπικούς Register Files (RF A και RF B). Για παράδειγμα η εντολή X@0bY σημαίνει ότι η τιμή X θα αποθηκευτεί στην διεύθυνση 0bY ενώ η εντολή 0bY@X φορτώνει το περιεχόμενο της διεύθυνσης Y στο X.

## B. 2η Φάση (Down-Sweep phase)

Στη δεύτερη φάση (down-sweep phase) ο αλγόριθμος διαπερνά το δυαδικό δένδρο με την αντίθετη φορά, δηλαδή από τον πατέρα προς τα φύλλα. Η εισερχόμενη τιμή (το μηδέν «0») εισέρχεται στον πατέρα του δένδρου (PE4) και συμβολίζεται με S. Σύμφωνα με τον αλγόριθμο Blelloch η εισερχόμενη τιμή S περνάει στην είσοδο του αριστερού φύλλου, ενώ στο δεξί φύλλο περνάει το άθροισμα της τιμής S με την τιμή Lsave2 που είχε αποθηκευτεί από την προηγούμενη διαπέραση του δένδρου (Up-Sweep) κοκ. Μία καλύτερη εξήγηση μπορεί να δοθεί κοιτώντας το σχήμα 4.10 όπου εξηγούνται τα βήματα του αλγορίθμου καθώς και το ποια PEs είναι ενεργοποιημένα κάθε φορά.




*Εικόνα 4.10. Ανάπτυγμα – τοπολογία ενός εμφωλευμένου δένδρου 2ης φάσης (Down-sweep). Σε κάποια από τα PEs η τιμή του αθροίσματος που προκύπτει από τα προηγούμενα αποθηκεύεται πάνω από μία φορές εφόσον επαναχρησιμοποιείται (PE3, PE4).*

| PE 4 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: **T** | Identity = S2 → Left | - | - | - | 0b11 @ identity | PASS B | A |
| 1: | (Lsave2+S2=S1) → L&R | - | previous @ 0b11 = S2 | 0b10 @ Lsave2 | 0b11 @ S2 | ADD A+B | A and B |
| 2: | (Lsave1+S1=S0) → L&R | - | previous @ 0b11 = S1 | 0b01 @ Lsave1 | 0b11 @ S1 | ADD A+B | A and B |
| 3: | (Lsave0+S0) → Right | - | previous @ 0b11 = S0 | 0b00 @ Lsave0 | 0b11 @ S0 | ADD A+B | B |

| PE 3 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: **T** | S2 → Left | - | inputO @ 0b11 = S2 | - | 0b11 @ S2 | PASS B | A |
| 1: | (Lsave1+S2=T1) → Right | - | inputO @ 0b10 = S1 | 0b01 @ Lsave1 | 0b11 @ S2 | ADD A+B | B |
| 2: | S1 → Left | - | - | - | 0b10 @ S1 | PASS B | A |
| 3: | (Lsave0+S1) → Right | - | - | 0b00 @ Lsave0 | 0b10 @ S1 | ADD A+B | B |

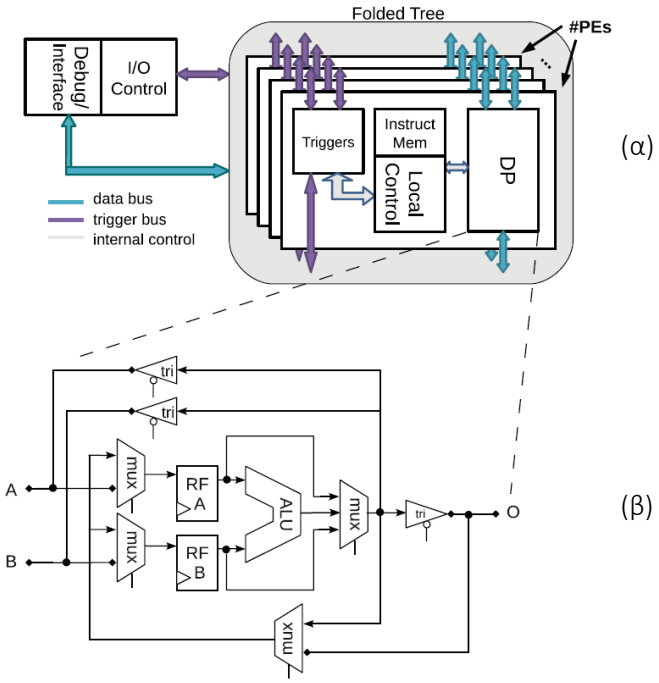
| PE 2 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: **T** | S2 → Left | - | inputO @ 0b01 = S2 | - | 0b01 @ S2 | PASS B | A |
| 1: | (Lsave0+S2) → Right | - | - | 0b00 @ Lsave0 | 0b01 @ S2 | ADD A+B | B |

| PE 1 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: **T** | T1 → Left | - | inputO @ 0b01 = T1 | - | 0b01 @ T1 | PASS B | A |
| 1: | (Lsave0+T1) → Right | - | - | 0b00 @ Lsave0 | 0b01 @ T1 | ADD A+B | B |

*Πίνακας 4.2. Πίνακας εντολών εκτέλεσης του αλγορίθμου μερικών αθροισμάτων με τη χρήση εμφωλευμένων δένδρων για την 2η φάση του αλγορίθμου (Down-Sweep). [23]*

## 4.5 Σχεδίαση κυκλώματος

Στην εικόνα 4.11 φαίνεται η υλοποίηση του κυκλώματος όπως αυτή προτάθηκε στην δημοσίευση με τίτλο «Low-Power Digital Signal Processor Architecture for WSN (2014)» [23]. Το κύκλωμα περιλαμβάνει οκτώ πανομοιότυπα PEs των 16 bit, όπου το καθένα αποτελείται από το data path και μία instruction memory 16 x 36 bit. Η ύπαρξη των triggers βοηθά το κύκλωμα να ενεργοποιεί το εκάστοτε PE μόνο όταν υπάρχουν δεδομένα στην είσοδο του με τέτοιο τρόπο ώστε να επιτυγχάνεται η διαπέραση ενός δυαδικού δένδρου τόσο στην $1^{η}$ φάση του αλγορίθμου όσο και στην $2^{η}$ . Μέσω των πολυπλεκτών (muxes), επιλέγονται κάθε φορά δεδομένα ως είσοδος του κυκλώματος (external data), δεδομένα προς αποθήκευση (stored data) και δεδομένα που προέρχονται ως αποτέλεσμα πράξεων προηγούμενου PE. Κάθε PE περιέχει μία αριθμητική και λογική μονάδα (ALU) τροφοδοτούμενη από δύο ξεχωριστούς Register Files (RF-A και RF-B) για κάθε είσοδο A και B. Οι συγκεκριμένοι καταχωρητές (RFs) είναι τεσσάρων θέσεων και αποτελούν την μνήμη ολόκληρου του συστήματος η οποία είναι κατανεμημένη, αφού κάθε PE έχει τους δικούς του καταχωρητές. Καθώς τα δεδομένα ρέουν μέσα στο δυαδικό δένδρο αποθηκεύονται σύμφωνα με τα βήματα εκτέλεσης του αλγορίθμου όπως αυτή αναπτύχθηκε σε προηγούμενες παραγράφους. Αυτό έχει θετική επίδραση στο κύκλωμα καθώς με αυτό τον τρόπο αποφεύγεται το φαινόμενο “von Neumann bottleneck” κατά το οποίο, λόγω του ότι η μνήμη βρίσκεται σε ξεχωριστό σημείο από την μονάδα επεξεργασίας, δημιουργείται καθυστέρηση και αυξάνεται η ενεργειακή κατανάλωση του κυκλώματος. Προκειμένου να επιτευχθεί η διαπέραση του δένδρου κατά τις δύο φάσεις του αλγορίθμου, κάποιοι από τους αγωγούς του κυκλώματος είναι δύο κατευθύνσεων (by-directional). Είναι οι αγωγοί που όπως φαίνεται στο σχήμα (β) έχουν στα άκρα τους το σχήμα του ρόμβου.




*Εικόνα 4.11. Σχηματικό διάγραμμα του κυκλώματος. (α) Σχέδιο ολοκληρωμένου κυκλώματος που χρησιμοποιεί και τα τέσσερα PEs.(β) Λεπτομερές κύκλωμα ενός PE. [23]*

Οι στόχοι της συγκεκριμένης σχεδίασης είναι η λειτουργία του κυκλώματος σε εύρος συχνοτήτων 20 – 80 MHz για τάση 1.2 volt και η κατασκευή του είναι στα 130 nm με τεχνολογία CMOS.
Στον πίνακα 4.3 φαίνονται οι τιμές της δυναμικής ενέργειας-ισχύος (Dynamic Energy) και διαρροή (leakage Power), για το ένα PE που λειτουργεί με συχνότητα λειτουργίας 20 MHz και 1.2 volt τάση τροφοδοσίας. Η κατανάλωση της ισχύος είναι 42 μW συμπεριλαμβανομένης και της ενέργειας διαρροής (Power leakage 0.03 μW).

| 1 Processing Element | Active PE core | Idle PE core | PE instruction memory |
|---|---|---|---|
| Dynamic energy/instruction (pJ) | 14.6 | 4.7 | 2.10 |
| Leakage power (μW) | 0.03 | 0.03 | 0.01 |
| Total power @ 20 MHz (μW) | 41.7 | 13.5 | 6.0 |

*Πίνακας 4.3. Καταναλισκόμενη ισχύς για ένα PE σε συχνότητα λειτουργίας 20 MHz*

Με σκοπό τον έλεγχο των αποτελεσμάτων της κατανάλωσης του ενός PE στη δημοσίευση [23] υπολογίζεται η κατανάλωση της ενέργειας για 8-PEs βασιζόμενοι στην κατανάλωση του ενός. Στο τέλος γίνεται έλεγχος των αποτελεσμάτων με αυτά που πραγματικά μετρήθηκαν. Για ένα εμφωλευμένο δένδρο που εκτελεί την πρώτη φάση του αλγορίθμου (Up-Sweep) με οκτώ PEs, χρειάζονται 4 στάδια μέχρι το τελικό αποτέλεσμα να εμφανιστεί στη ρίζα του δένδρου. Σε κάθε στάδιο ο αριθμός των PEs που είναι ενεργά μειώνεται στο μισό. Επομένως τα ενεργοποιημένα PEs σε κάθε στάδιο θα είναι 8, 4, 2, 1. Αντίστοιχα ο αριθμός των ανενεργών PEs αυξάνεται σε κάθε βήμα με την ακολουθία 0, 4, 6, 7. Συνολικά λοιπόν για την 1$^{η}$ φάση του αλγορίθμου θα είναι ενεργά 15 PEs και 17 απενεργοποιημένα. Γνωρίζοντας αυτή την πληροφορία υπολογίζεται η καταναλισκόμενη ισχύς και ενέργεια ολόκληρου του δένδρου όπως φαίνεται στον πίνακα RR. Τα αποτελέσματα που βγαίνουν επιβεβαιώνονται σε σχέση με αυτά που μετρήθηκαν.

| Folded Tree (Up-Sweep phase) | Estimate | Measured | Incl. Instr. Memory |
|---|---|---|---|
| Total dynamic energy (pJ) | 298.8 | 289.2 | 356 |
| Leakage power (μW) | 0.25 | 0.25 | 0.35 |
| Total power @ 20 MHz (μW) | 213.7 | 206.8 | 254.9 |

*Πίνακας 4.4.. Καταναλισκόμενη ενέργεια για εμφωλευμένο δένδρο με 8 PEs*

Σύμφωνα με την δημοσίευση [23], το ένα PE χρειάζεται 7 ανερχόμενους κύκλους ρολογιού για την υλοποίηση μιας εντολής. Επομένως γνωρίζοντας την κατανάλωση ενέργειας, όπως μετρήθηκε, ανά εντολή για τα ενεργά, μη ενεργά PEs και για την instruction memory μπορούμε να υπολογίσουμε και τη συνολική καταναλισκόμενη ισχύ για ένα PE. Ο υπολογισμός μπορεί να γίνει ως εξής:

$$P_{PE} = \frac{Energy\ (pJ)}{7\ \text{cycles} \times \text{T(ns)}}$$

Για τον υπολογισμό της συνολικής καταναλισκόμενης ενέργειας του δένδρου όπως αυτό προκύπτει ο υπολογισμός γίνεται ως εξής :

$$E_{total} = (\text{Energy } (1\ PE_{active})) \times (\text{Num. active PEs}) + (\text{Energy } (1\ PE_{inactive})) \times (\text{Num. Inactive PEs})$$

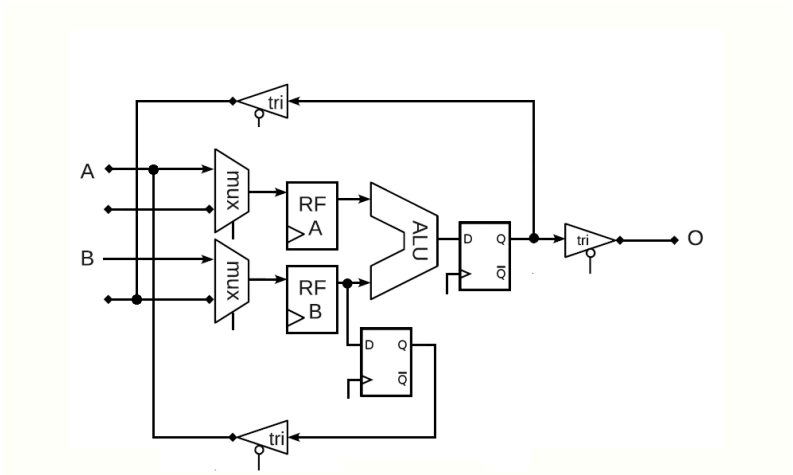
Γνωρίζοντας την συνολική ενέργεια και τη χρονική διάρκεια που απαιτείται για την εκτέλεση μιας εντολής η οποία υπολογίζεται από τον αριθμό των βημάτων εκτέλεσης της φάσης πολλαπλασιαζόμενη με τους κύκλους που χρειάζονται ανά εντολή, μπορούμε να υπολογίσουμε την συνολική κατανάλωση δυναμικής ισχύος.

$$P_{total} = \frac{E_{total}\ (pJ)}{7\ \text{cycles} \times \text{stages} \times \text{T(ns)}}$$

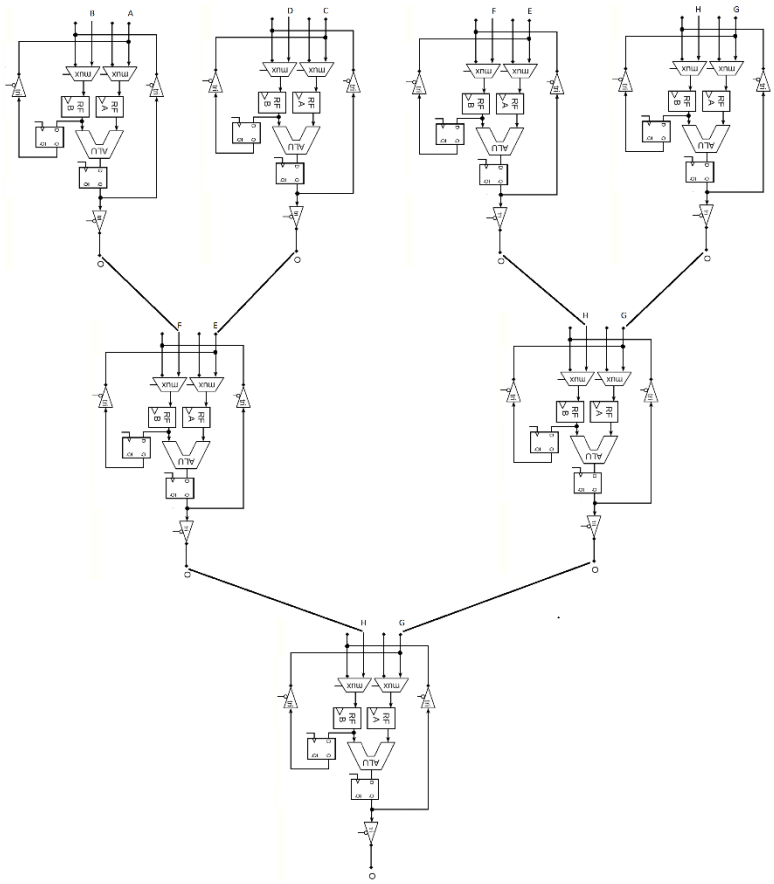
## 4.6 Μείωση της κατανάλωσης ενέργειας του κυκλώματος

Θέλοντας να μειώσουμε την κατανάλωση ενέργειας του κυκλώματος αρχίσαμε να παρατηρούμε τι θα μπορούσαμε να παραλείψουμε σε κυκλωματικό επίπεδο. Αποφασίσαμε να παραλείψουμε την Μνήμη Εντολών (instruction memory), δίνοντας τον ρόλο της στην Μηχανή Πεπερασμένων Καταστάσεων (FSM) του κυκλώματος. Με λίγα λόγια η Μηχανή Πεπερασμένων Καταστάσεων αναλαμβάνει σε κάθε βήμα του αλγορίθμου να ενεργοποιεί τα PEs που εμείς επιθυμούμε, να καθορίζει την διεύθυνση που θα εγγραφούν/διαβαστούν τα δεδομένα καθώς και τον τελεστή της Αριθμητικής Λογικής Μονάδας (ALU) που θα χρησιμοποιείται κάθε φορά. Με αυτό τον τρόπο και σύμφωνα με τις μετρήσεις της δημοσίευσης [23], το κύκλωμα θα καταναλώνει έως και 18% λιγότερη ισχύ.

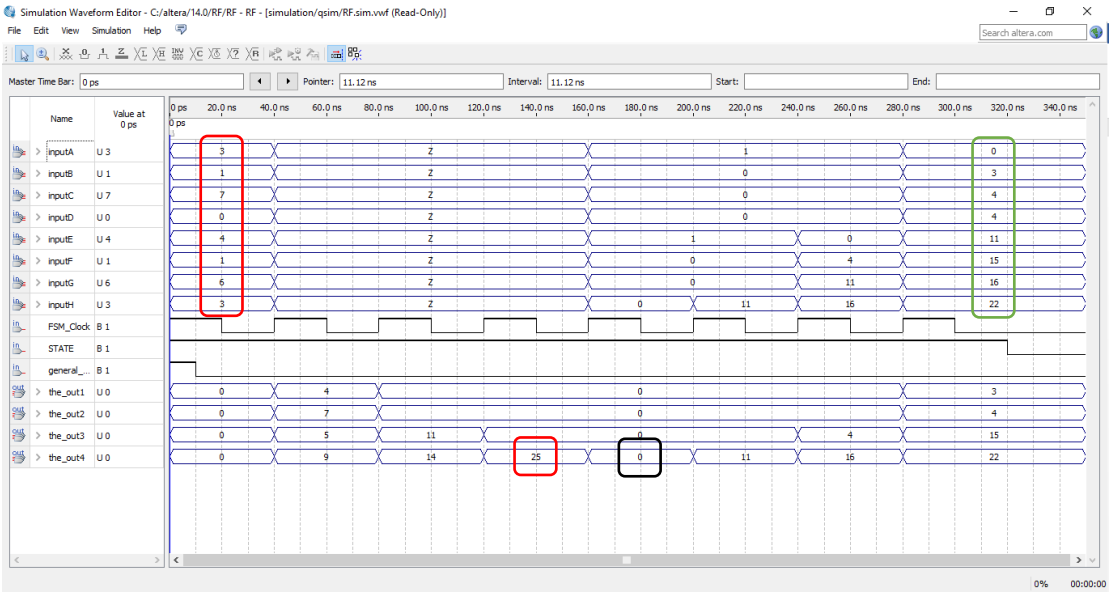
Επίσης όλες οι εντολές θα εκτελούνται σε ένα κύκλο ρολογιού και όχι σε πολλαπλούς κύκλους όπως συνέβαινε στο κύκλωμα της δημοσίευσης [23]. Επομένως οι πράξεις αποθήκευσης, ενεργοποίησης των PEs και ALU θα γίνονται σε έναν μόνο κύκλο ρολογιού, πράγμα το οποίο βοηθά περαιτέρω στη μείωση της συνολικής κατανάλωσης του κυκλώματος. Οι στόχοι της συγκεκριμένης σχεδίασης είναι η λειτουργία του κυκλώματος σε συχνότητα 20 MHz για τάση 1.2 volt στα 45 nm με τεχνολογία CMOS. Αφού περιγράψουμε το κύκλωμα μέσω της γλώσσας VHDL θα κάνουμε και ενεργειακή ανάλυση μέσω κατάλληλου εργαλείου με σκοπό να δείξουμε την μείωση που επιτεύχθηκε στην κατανάλωση (Παράρτημα). Στην εικόνα που ακολουθεί (Σχήμα 4.12) φαίνεται το κύκλωμα που προτείνεται με τις αλλαγές που πραγματοποιήθηκαν σε σχέση με το αρχικό. Στο τελευταίο έχουν αφαιρεθεί κάποιοι από τους πολυπλέκτες και έχουν προστεθεί κάποια D-flip flops. Τα τελευταία είναι απαραίτητα, καθώς λόγω του ότι δεν υπάρχει instruction memory χρειαζόμαστε ένα στοιχείο μνήμης που θα κρατά την τελευταία τιμή που μας ενδιαφέρει σε κάθε χτύπο του ρολογιού με σκοπό την επεξεργασία του στο επόμενο. Στο σχήμα 4.12 φαίνονται επίσης τα μονοπάτια διπλής κατεύθυνσης (by-directional) και μονής με σκοπό το κύκλωμα να μπορεί να χρησιμοποιεί τις εξόδους του ως εισόδους στις φάσεις του αλγορίθμου.




*Σχήμα 4.12. Κύκλωμα ενός PE*



*Σχήμα 4.13. Ανάπτυγμα εμφωλευμένου δένδρου χρησιμοποιώντας ως κορυφές του το κύκλωμα του σχήματος 4.12. Τα κυκλώματα με εισόδους F,E & H,G επαναχρησιμοποιούνται.*




*Σχήμα 4.13. Κυματομορφές των εισόδων & εξόδων του κυκλώματος.*

Στο σχήμα 4.13 φαίνονται οι κυματομορφές του κυκλώματος κατά την εισαγωγή και επεξεργασία οκτώ (8) αριθμών. Κοιτώντας το σχήμα από δεξιά προς τα αριστερά παρατηρούμε ότι σε κάποια χρονική στιγμή στην έξοδο του κυκλώματος εμφανίζονται το συνολικό άθροισμα (25) και στη συνέχεια ο αριθμός μηδέν (0) το οποίο αντικαθιστά το συνολικό άθροισμα μηδέν (25), όπως αυτό αναλύθηκε στην παράγραφο 4.2.Β. Τα μερικά αθροίσματα θα εμφανιστούν στις εισόδους του κυκλώματος μιας και τα μονοπάτια είναι by-directional. Οι καταστάσεις της FSM αλλάζουν βάσει των ανερχόμενων παρυφών του ρολογιού. Το ίδιο συμβαίνει και με τις πράξεις αποθήκευσης και πρόσθεσης σε κάθε επίπεδο.

Τα αποτελέσματα από την ενεργειακή κατανάλωση του κυκλώματος σε σύγκριση με το κύκλωμα της δημοσίευσης [23] συνοψίζονται στον παρακάτω πίνακα. Περισσότερες πληροφορίες δίνονται στο αντίστοιχο πεδίο του παραρτήματος.

| Total Dynamic Power @ 20 MHz (μW) | |
|---|---|
| Κύκλωμα Δημοσίευσης [23] | Κύκλωμα Σχήματος 4.12 & 4.13 |
| 254.9 | 11.30 |

***Πίνακας 4.5. Σύγκριση ενεργειακών καταναλώσεων των δύο κυκλωμάτων.***

Παρόλο που το παραλλαγμένο κύκλωμα σχεδιάστηκε βασιζόμενο σε αρχιτεκτονική των 8 bit σε αντίθεση με το κύκλωμα της δημοσίευσης που είναι στα 16 bit, παρατηρούμε ότι η δυναμική κατανάλωση ισχύος είναι κατά πολύ χαμηλότερη περίπου 95%.

# Παράρτημα

## Περιγραφή του κυκλώματος σε VHDL

### 1. Folded tree with eight elements

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity RF is

        port(
                inputA, inputB : inout std_logic_vector(7 downto 0);
                inputC, inputD : inout std_logic_vector(7 downto 0);
                inputE, inputF : inout std_logic_vector(7 downto 0);
                inputG, inputH : inout std_logic_vector(7 downto 0);
                general_reset  : in std_logic;
                STATE          : in std_logic;
                FSM_Clock      : in std_logic;
                the_out1       : out std_logic_vector(7 downto 0);
                the_out2       : out std_logic_vector(7 downto 0);
                the_out3       : out std_logic_vector(7 downto 0);
                the_out4       : out std_logic_vector(7 downto 0)
             );

end RF;


architecture structure of RF is

-------------------------------------------------------------------
---------------- Signals from FSM ---------------------------------
-------------------------------------------------------------------

signal theSig1 : std_logic_vector(16 downto 0); -- Signal for PE1
signal theSig2 : std_logic_vector(16 downto 0); -- Signal for PE2
signal theSig3 : std_logic_vector(16 downto 0); -- Signal for PE3
signal theSig4 : std_logic_vector(16 downto 0); -- Signal for PE4
-------------------------------------------------------------------

signal PE1toPE3 : std_logic_vector(7 downto 0); ----- PE1 to PE3
signal PE2toPE3 : std_logic_vector(7 downto 0); ----- PE2 to PE3
signal PE3toPE4 : std_logic_vector(7 downto 0); ----- PE3 to PE4
signal PE4toPE4 : std_logic_vector(7 downto 0); ----- PE4 to PE4
-------------------------------------------------------------------

signal PE1toDFF1 : std_logic_vector(7 downto 0); ---- PE1 to DFF1
signal PE2toDFF2 : std_logic_vector(7 downto 0); ---- PE2 to DFF2
signal PE3toDFF3 : std_logic_vector(7 downto 0); ---- PE3 to DFF3
signal PE4toDFF4 : std_logic_vector(7 downto 0); ---- PE4 to DFF4

-------------------------------------------------------------------------------------
--- We are using muxes in order to select the input from user or previous stage --
-------------------------------------------------------------------------------------
signal MUXtoPE1B : std_logic_vector(7 downto 0);
signal MUXtoPE2B : std_logic_vector(7 downto 0);
signal MUXtoPE3A : std_logic_vector(7 downto 0);
signal MUXtoPE3B : std_logic_vector(7 downto 0);
signal MUXtoPE4A : std_logic_vector(7 downto 0);
signal MUXtoPE4B : std_logic_vector(7 downto 0);
```

```
------------------------------------------------------------------
------------------- Tristate Signals -----------------------------
------------------------------------------------------------------

signal trisToA : std_logic_vector(7 downto 0);
signal trisToB : std_logic_vector(7 downto 0);
signal trisToC : std_logic_vector(7 downto 0);
signal trisToD : std_logic_vector(7 downto 0);
signal trisToE : std_logic_vector(7 downto 0);
signal trisToF : std_logic_vector(7 downto 0);
signal trisToG : std_logic_vector(7 downto 0);
signal trisToH : std_logic_vector(7 downto 0);

------------------------------------------------------------------



signal tristate_enable : std_logic;

signal selectMUX  : std_logic;
signal selectMUX1 : std_logic;
signal selectMUX2 : std_logic;
signal selectMUX3 : std_logic_vector(1 downto 0);

signal REGtoDFF1 : std_logic_vector(7 downto 0);
signal REGtoDFF2 : std_logic_vector(7 downto 0);
signal REGtoDFF3 : std_logic_vector(7 downto 0);
signal REGtoDFF4 : std_logic_vector(7 downto 0);

------------------------------------------------------------------
signal DFF1toPE1 : std_logic_vector(7 downto 0);
signal DFF2toPE2 : std_logic_vector(7 downto 0);
signal DFF3toPE3 : std_logic_vector(7 downto 0);
signal DFF4toPE4 : std_logic_vector(7 downto 0);

component PE is

       port (                      A, B : in std_logic_vector(7 downto 0);
-------------------------------------------------------------------------------
               addr_writeA, addr_readA : in std_logic_vector(1 downto 0);
             a  ddr_writeB, addr_readB : in std_logic_vector(1 downto 0);

             write_enA, read_enA, rstA : in std_logic;
             write_enB, read_enB, rstB : in std_logic;

                               sl_alu : in std_logic_vector(2 downto 0);
                               ALU_en : in std_logic;
                              clocker : in std_logic;
-------------------------------------------------------------------------------
                               regOUT : out std_logic_vector(7 downto 0);
                                    F : out std_logic_vector(7 downto 0)
            );

end component;

-------------------------------------------------------------------------------

component FSM is

       port(         reset : in std_logic;      -- reset signal
                      S_in : in std_logic;      -- serial bit Input sequence
                       clk : in std_logic;
                    S_out1 : out std_logic_vector(16 downto 0);
                    S_out2 : out std_logic_vector(16 downto 0);
                    S_out3 : out std_logic_vector(16 downto 0);
                    S_out4 : out std_logic_vector(16 downto 0);
             tristate_en : out std_logic;
                  SL_MUX : out std_logic;
                 SL_MUX1 : out std_logic;
                 SL_MUX2 : out std_logic;
                 SL_MUX3 : out std_logic_vector(1 downto 0)

          );

end component;

-------------------------------------------------------------------------------
```

```
component mux2to1_8bit is
        port ( input_A, input_B : in std_logic_vector(7 downto 0);
                             sl : in std_logic;
                         output : out std_logic_vector(7 downto 0)
             );
end component;

-------------------------------------------------------------------------------------

component DFF_8bit is
        port(     D : in std_logic_vector(7 downto 0);
                clk : in std_logic;
                  Q : out std_logic_vector(7 downto 0)
             );
end component;

-------------------------------------------------------------------------------------
component tristate_8_bit is
        port(   A : in std_logic_vector(7 downto 0);
               en : in std_logic;
                B : out std_logic_vector(7 downto 0)
             );
end component;

-------------------------------------------------------------------------------------

component mux3to1 is
        port(   w0,w1,w2 : in std_logic_vector(7 downto 0);
                       s : in std_logic_vector(1 downto 0);
                 mux_out : out std_logic_vector(7 downto 0)
             );
end component;

-------------------------------------------------------------------------------------

begin

        FSMEIROS : FSM  port map( general_reset, STATE, FSM_Clock,
                                  theSig1,theSig2, theSig3, theSig4,
                                  tristate_enable, selectMUX, selectMUX1,
                                  selectMUX2, selectMUX3
                                );

        the_out1 <= PE1toPE3;
        the_out2 <= PE2toPE3;
        the_out3 <= PE3toPE4;
        the_out4 <= PE4toPE4;

        inputA <= trisToA;
        inputB <= trisToB;

        inputC <= trisToC;
        inputD <= trisToD;

        inputE <= trisToE;
        inputF <= trisToF;

        inputG <= trisToG;
        inputH <= trisToH;

        MUXPE1B : mux2to1_8bit  port map(inputB, inputE, selectMUX1, MUXtoPE1B);
        MUXPE2B : mux2to1_8bit  port map(inputD, inputF ,selectMUX2, MUXtoPE2B);

        MUXPE3A : mux2to1_8bit  port map(inputE, PE1toPE3, selectMUX, MUXtoPE3A);
        MUXPE3B : mux3to1       port map(inputF, PE2toPE3, inputG, selectMUX3, MUXtoPE3B);

        MUXPE4A : mux2to1_8bit  port map(inputG, PE3toPE4, selectMUX, MUXtoPE4A);
        MUXPE4B : mux2to1_8bit  port map(inputH, PE4toPE4, selectMUX, MUXtoPE4B);

        DFF_1 : DFF_8bit   port map(PE1toDFF1, FSM_Clock, PE1toPE3);
        DFF_2 : DFF_8bit   port map(PE2toDFF2, FSM_Clock, PE2toPE3);
        DFF_3 : DFF_8bit   port map(PE3toDFF3, FSM_Clock, PE3toPE4);
        DFF_4 : DFF_8bit   port map(PE4toDFF4, FSM_Clock, PE4toPE4);
```

```
        DFF_REG_1 : DFF_8bit   port map(REGtoDFF1, FSM_Clock, DFF1toPE1);
        DFF_REG_2 : DFF_8bit   port map(REGtoDFF2, FSM_Clock, DFF2toPE2);
        DFF_REG_3 : DFF_8bit   port map(REGtoDFF3, FSM_Clock, DFF3toPE3);
        DFF_REG_4 : DFF_8bit   port map(REGtoDFF4, FSM_Clock, DFF4toPE4);

        --tristate_enable;

        TRI_A   : tristate_8_bit  port map (DFF1toPE1, tristate_enable, trisToA);
        TRI_B   : tristate_8_bit  port map (PE1toPE3,  tristate_enable, trisToB);

        TRI_C   : tristate_8_bit  port map (DFF2toPE2, tristate_enable, trisToC);
        TRI_D   : tristate_8_bit  port map (PE2toPE3,  tristate_enable, trisToD);

        TRI_E   : tristate_8_bit  port map (PE3toPE4,  tristate_enable, trisToF);
        TRI_F   : tristate_8_bit  port map (DFF3toPE3, tristate_enable, trisToE);

        TRI_G   : tristate_8_bit  port map (DFF4toPE4, tristate_enable, trisToG);
        TRI_H   : tristate_8_bit  port map (PE4toPE4,  tristate_enable, trisToH);

        PE1 : PE port map(     inputA, MUXtoPE1B,
                               theSig1(16 downto 15), theSig1(14 downto 13),
                               theSig1(12 downto 11), theSig1(10 downto 9),
                               theSig1(8), theSig1(7), theSig1(6),
                               theSig1(5), theSig1(4), theSig1(3),
                               theSig1(2 downto 0),
                               FSM_Clock, REGtoDFF1, PE1toDFF1
                        );

        PE2 : PE port map(     inputC, MUXtoPE2B,
                               theSig2(16 downto 15), theSig2(14 downto 13),
                               theSig2(12 downto 11), theSig2(10 downto 9),
                               theSig2(8), theSig2(7), theSig2(6),
                               theSig2(5), theSig2(4), theSig2(3),
                               theSig2(2 downto 0),
                               FSM_Clock, REGtoDFF2, PE2toDFF2
                         );

        PE3 : PE port map(     MUXtoPE3A, MUXtoPE3B,
                               theSig3(16 downto 15), theSig3(14 downto 13),
                               theSig3(12 downto 11), theSig3(10 downto 9),
                               theSig3(8), theSig3(7), theSig3(6),
                               theSig3(5), theSig3(4), theSig3(3),
                               theSig3(2 downto 0),
                               FSM_Clock, REGtoDFF3, PE3toDFF3
                        );

        PE4 : PE port map(     MUXtoPE4A, MUXtoPE4B,
                               theSig4(16 downto 15), theSig4(14 downto 13),
                               theSig4(12 downto 11), theSig4(10 downto 9),
                               theSig4(8), theSig4(7), theSig4(6),
                               theSig4(5), theSig4(4), theSig4(3),
                               theSig4(2 downto 0),
                               FSM_Clock, REGtoDFF4, PE4toDFF4
                        );

end structure;
```

## 2. FSM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

--Sequence detector for detecting the sequence "1011".
--Overlapping type.

entity FSM is

        port(         reset : in std_logic;         --reset signal
                      S_in  : in std_logic;         --serial bit Input sequence
                       clk  : in std_logic;
                    S_out1 : out std_logic_vector(16 downto 0); -- Output
                    S_out2 : out std_logic_vector(16 downto 0);
                    S_out3 : out std_logic_vector(16 downto 0);
                    S_out4 : out std_logic_vector(16 downto 0);
                tristate_en : out std_logic;
                     SL_MUX : out std_logic;
                    SL_MUX1 : out std_logic;
                    SL_MUX2 : out std_logic;
                    SL_MUX3 : out std_logic_vector(1 downto 0)
            );

end FSM;

architecture Behavioral of FSM is
--Defines the type for states in the state machine
type state_type is (S0,S1,S2,S3,S4,S5,S6,S7);
--Declare the signal with the corresponding state type.
signal Current_State, Next_State : state_type;
        begin

        process(clk,reset)

                begin
                        if( reset = '1' ) then                  --Synchronous Reset
                                Current_State <= S0;
                        elsif (clk'event and clk = '1') then    --Rising edge of Clock
                                Current_State <= Next_State;
                        end if;
        end process;
-- Combinational Process
        Process(Current_State, S_in)
        Begin
                case Current_State is

                        when S0 =>

------------------------------------------------------ ALL PEs operating, Trunk-Phase

                                S_out1 <= "00000000111111011";
                                S_out2 <= "00000000111111011";
                                S_out3 <= "00000000111111011";
                                S_out4 <= "00000000111111011";
                                tristate_en <= '0';
                                SL_MUX      <= '0';
                                SL_MUX1     <= '0';
                                SL_MUX2     <= '0';
                                SL_MUX3     <= "00";


                                if ( S_in = '0' ) then
                                        Next_State <= S0;
                                else
                                        Next_State <= S1;
                                end if;

                        when S1 =>
```

```
------------------------------------------------------ PE3 & PE4 operating


                                S_out1 <= "00000000001001000";
                                S_out2 <= "00000000001001000";
                                S_out3 <= "01010101111111011";
                                S_out4 <= "01010101111111011";
                                tristate_en <= '0';
                                SL_MUX      <= '1';
                                SL_MUX1     <= '0';
                                SL_MUX2     <= '0';
                                SL_MUX3     <= "01";

                                if ( S_in = '1' ) then
                                        Next_State <= S2;
                                end if;

                        when S2 =>

------------------------------------------------- PE4 operating

                                S_out1 <= "00000000001001000";
                                S_out2 <= "00000000001001000";
                                S_out3 <= "00000000001001000";
                                S_out4 <= "10101010111111011";
                                tristate_en <= '0';
                                SL_MUX      <= '1';
                                SL_MUX1     <= '0';
                                SL_MUX2     <= '0';
                                SL_MUX3     <= "01";


                                if ( S_in = '1' ) then
                                        Next_State <= S3;
                                end if;

                        when S3 =>

--------------------------------------- PE4 operating get the ZERO

                                S_out1 <= "00000000001001000";
                                S_out2 <= "00000000001001000";
                                S_out3 <= "00000000001001000";
                                S_out4 <= "11111111011011111";
                                tristate_en <= '0';
                                SL_MUX      <= '1';
                                SL_MUX1     <= '0';
                                SL_MUX2     <= '0';
                                SL_MUX3     <= "01";

                                if (S_in = '1' ) then
                                        Next_State <= S4;
                                end if;

                        when S4 =>

--------------------------------------- PE4 get the ZERO and add it.

                                S_out1 <= "00000000001001000";
                                S_out2 <= "00000000001001000";
                                S_out3 <= "00000000001001000";
                                S_out4 <= "10101111011011011";
                                tristate_en <= '1';
                                SL_MUX      <= '0';
                                SL_MUX1     <= '1';
                                SL_MUX2     <= '1';
                                SL_MUX3     <= "11";

                                if (S_in = '1' ) then
                                        Next_State <= S5;
                                end if;

                        when S5 =>
```

```
---------------------------------------- PE4 get the ZERO and add it.

                                S_out1 <= "00000000001001000";
                                S_out2 <= "00000000001001000";
                                S_out3 <= "01010101011111011";
                                S_out4 <= "01011010011111011";
                                tristate_en <= '1';
                                SL_MUX      <= '0';
                                SL_MUX1     <= '1';
                                SL_MUX2     <= '1';
                                SL_MUX3     <= "11";

                                if (S_in = '1' ) then
                                        Next_State <= S6;
                                end if;

                        when S6 =>

---------------------------------------- PE4 get the ZERO and add it.

                                S_out1 <= "00000000011111011";
                                S_out2 <= "00000000111111011";
                                S_out3 <= "00000000011111011";
                                S_out4 <= "00000101011111011";
                                tristate_en <= '1';
                                SL_MUX      <= '0';
                                SL_MUX1     <= '1';
                                SL_MUX2     <= '1';
                                SL_MUX3     <= "11";

                                if (S_in = '1' ) then
                                        Next_State <= S7;
                                end if;

                        when S7 =>

---------------------------------------- PE4 get the ZERO and add it.

                                S_out1 <= "00000000011111011";
                                S_out2 <= "00000000111111011";
                                S_out3 <= "00000000011111011";
                                S_out4 <= "00000101011111011";
                                tristate_en <= '1';
                                SL_MUX      <= '0';
                                SL_MUX1     <= '1';
                                SL_MUX2     <= '1';
                                SL_MUX3     <= "11";

                                if (S_in = '1' ) then
                                        Next_State <= S0;
                                end if;

                        when others =>
                        NULL;
                end case;
        end process;
end behavioral;
```

## 3. One PE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity PE is

       port (                      A, B : in std_logic_vector(7 downto 0);
-------------------------------------------------------------------------------
                addr_writeA, addr_readA : in std_logic_vector(1 downto 0);
                addr_writeB, addr_readB : in std_logic_vector(1 downto 0);

              write_enA, read_enA, rstA : in std_logic;
              write_enB, read_enB, rstB : in std_logic;

                                 sl_alu : in std_logic_vector(2 downto 0);
                               clocker : in std_logic;
-------------------------------------------------------------------------------
                                 regOUT : out std_logic_vector(7 downto 0);
                                      F : out std_logic_vector(7 downto 0)
            );

end PE;



architecture structure of PE is

       signal muxA_to_PEA    : std_logic_vector(7 downto 0);
       signal muxB_to_PEB    : std_logic_vector(7 downto 0);

       signal muxC_to_out    : std_logic_vector(7 downto 0);

       signal PEA_to_ALU     : std_logic_vector(7 downto 0);
       signal PEB_to_ALU     : std_logic_vector(7 downto 0);

       signal ALU_to_MUX3to1 : std_logic_vector(7 downto 0);
       signal MUX3to1_OUT    : std_logic_vector(7 downto 0);


       component Register_File is

              port (                 data_in : in std_logic_vector(7 downto 0);
                              addr_wr,addr_rd : in std_logic_vector(1 downto 0);
                        we_en,rd_en,reset,clock : in std_logic;
                                           Fo : out std_logic_vector(7 downto 0)
                    );

       end component;

-------------------------------------------------------------------------------

       component alu8bit is

              port(   alu_A, alu_B : in std_logic_vector(7 downto 0);
                           sel_alu : in std_logic_vector(2 downto 0);
                           out_alu : out std_logic_vector(7 downto 0)
                  );
       end component;

-------------------------------------------------------------------------------

       component mux2to1_8bit is

              port (input_A, input_B : in std_logic_vector(7 downto 0);
                                  sl : in std_logic;
                              output : out std_logic_vector(7 downto 0)
                    );
       end component;

-------------------------------------------------------------------------------
```

```
        component  mux3to1 is

                port (     w0,w1,w2 : in std_logic_vector(7 downto 0);
                                 s : in std_logic_vector(1 downto 0);
                            mux_out : out std_logic_vector(7 downto 0));
        end component;

        ------------------------------------------------------------------------------

        component f_out is

                port (
                        j : in std_logic_vector(7 downto 0);
                        i : out std_logic_vector(7 downto 0)
                      );
        end component;

        ------------------------------------------------------------------------------


        begin

        MUXA_TO_PE_A : Register_File port map (
                                                A, addr_writeA, addr_readA, write_enA,
                                                read_enA, rstA, clocker, PEA_to_ALU
                                              );

        MUXB_TO_PE_B : Register_File port map (
                                                B, addr_writeB, addr_readB, write_enB,
                                                read_enB, rstB, clocker, PEB_to_ALU
                                              );

        ALU          : alu8bit port map (
                                           PEA_to_ALU, PEB_to_ALU, sl_alu,
                                           ALU_to_MUX3to1
                                         );

        TRABA        : f_out port map (ALU_to_MUX3to1, F);
        TRABA_REG    : f_out    port map (PEB_to_ALU, regOUT);

end structure;
```

## A. Register File

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


------------------- entity of Register File circuit ------------------------------

entity Register_File is

        port (                  data_in : in std_logic_vector(7 downto 0);
                          addr_wr,addr_rd : in std_logic_vector(1 downto 0);
                 we_en,rd_en,reset,clock : in std_logic;
                                      Fo : out std_logic_vector(7 downto 0)
             );

end Register_File;
```

```
------------------- entity of Simply register DFF circuit ------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity register_8bit is
       port (            D : in std_logic_vector(7 downto 0);
                clk,resetn : in std_logic;
                         Q : out std_logic_vector(7 downto 0)
            );
end register_8bit;

architecture behavior of register_8bit is
begin
    process(resetn,Clk)
           begin
                  if resetn = '0' then
                         Q <= "00000000";
                  elsif Clk'event and Clk = '1' then
                         Q <= D;
                  end if;
    end process;

end behavior;

--------------------------------------------------------------------------------
----------- entity and architecture of Decoder 2 to 4 16/8 bit with Enable -------
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity DEC_2to4 is

    port (     dst_addr : in std_logic_vector(1 downto 0);
           we, clock_we : in std_logic;
                dec_out : out std_logic_vector(3 downto 0)
         );
end DEC_2to4;

architecture behavior of DEC_2to4 is -------------- architecture of one DEC_2to4
       begin
              process(clock_we, we) --dst_addr,
              begin
                     if clock_we'event and clock_we = '0' then
                            if we = '1' then
                                   case dst_addr is
                                          when "00" =>
                                                 dec_out <= "1000";
                                          when "01" =>
                                                 dec_out <= "0100";
                                          when "10" =>
                                                 dec_out <= "0010";
                                          when "11" =>
                                                 dec_out <= "0001";
                                          when others =>
                                                 dec_out <= "ZZZZ";
                                   end case;
                            else
                                   dec_out <= "0000";
                            end if;
                     end if;
              end process;
end behavior;
```

```
-------------------------------------------------------------------------------------
---------------------- END of DECoder 2 to 1 16 bit with Enable ---------------------
-------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity muxxx is
       port (  w0,w1,w2,w3 : in std_logic_vector(7 downto 0);
                        s : in std_logic_vector(1 downto 0);
                       en : in std_logic;
                  mux_out : out std_logic_vector(7 downto 0)
            );
end muxxx;

architecture behavior of muxxx is
       begin
              process(w0,w1,w2,w3,s,en)
              begin
                     if en = '1' then
                            case s is
                                   when "00" =>
                                          mux_out <= w0;
                                   when "01" =>
                                          mux_out <= w1;
                                   when "10" =>
                                          mux_out <= w2;
                                   when "11" =>
                                          mux_out <= w3;
                                   when others =>
                                     mux_out <= "ZZZZZZZZ";
                            end case;

                     elsif en = '0' then
                            mux_out <= "ZZZZZZZZ";
                     end if;
              end process;
end behavior;







-------------------------------------------------------------------------------------
-------------------------------- REG FILE BOX ---------------------------------------
-------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

architecture structure of Register_File is

       signal dec_to_dff  : std_logic_vector(3 downto 0);
       signal reg_to_mux1 : std_logic_vector(7 downto 0);
       signal reg_to_mux2 : std_logic_vector(7 downto 0);
       signal reg_to_mux3 : std_logic_vector(7 downto 0);
       signal reg_to_mux4 : std_logic_vector(7 downto 0);

---------------------- register_8bit_black_box component ----------------------------

component register_8bit is

       port (           D : in std_logic_vector(7 downto 0);
                clk,resetn : in std_logic;
                         Q : out std_logic_vector(7 downto 0)
             );
end component;
```

```
------------------------------ tristate 8_bit component ------------------------------

component muxxx is
       port (   w0,w1,w2,w3 : in std_logic_vector(7 downto 0);
                         s : in std_logic_vector(1 downto 0);
                        en : in std_logic;
                   mux_out : out std_logic_vector(7 downto 0)
            );
end component;

---------------------------- Decoder 2 to 4 Component --------------------------------

component DEC_2to4 is
       port (    dst_addr : in std_logic_vector(1 downto 0);
                 we,clock_we : in std_logic;
                     dec_out : out std_logic_vector(3 downto 0)
            );
end component;

---------------------------- signal connection with Fo ------------------------------

component f_out is
       port ( j : in std_logic_vector(7 downto 0);
              i : out std_logic_vector(7 downto 0)
            );
end component;

--------------------------------------------------------------------------------------
begin
--Here is the Port map of register file.

ADDRESS_WR  : DEC_2to4     port map (addr_wr, we_en, clock, dec_to_dff);

DATA_IN_I   : register_8bit port map (data_in, dec_to_dff(3), reset, reg_to_mux1);
DATA_IN_II  : register_8bit port map (data_in, dec_to_dff(2), reset, reg_to_mux2);
DATA_IN_III : register_8bit port map (data_in, dec_to_dff(1), reset, reg_to_mux3);
DATA_IN_IV  : register_8bit port map (data_in, dec_to_dff(0), reset, reg_to_mux4);

CONTROL_Fo  : muxxx         port map (
                                      reg_to_mux1, reg_to_mux2, reg_to_mux3,
                                      reg_to_mux4, addr_rd, rd_en, Fo
                                     );
end structure;
```

## B. ALU

```
--------------------------------------------------------------------------------------
---------------------------------- ALU 2 input 8 bit ---------------------------
--------------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity alu8bit is
       port( alu_A, alu_B : in std_logic_vector(7 downto 0);
                  sel_alu : in std_logic_vector(2 downto 0);
                  out_alu : out std_logic_vector(7 downto 0)
           );
end alu8bit;
```

```
architecture behavior of alu8bit is
begin
    process(sel_alu, alu_A, alu_B)
            variable temp: std_logic_vector(7 downto 0);
                    begin
                            case sel_alu is
                                    when "000" =>
                                            temp := "00000000";
                                    when "001" =>
                                            temp := alu_B - alu_A;
                                    when "010" =>
                                            temp := alu_A - alu_B;
                                    when "011" =>
                                            temp := alu_A + alu_B;
                                    when "100" =>
                                            temp := alu_A XOR alu_B;
                                    when "101" =>
                                            temp := alu_A OR alu_B;
                                    when "110" =>
                                            temp := alu_A AND alu_B;
                                    when others =>
                                            temp := "11111111";
                            end case;

                            if temp="11111111" then
                                    out_alu <= "00000000";
                            else
                                    out_alu <= temp;
                            end if;


    end process;

end behavior;

--------------------------------------------------------------------------------------
------------------------------- END of ALU 2 input 8 bit -----------------------------
--------------------------------------------------------------------------------------
```

## C. DFF (D-flip-flop)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity DFF_8bit is
    port (    D : in std_logic_vector(7 downto 0);
            clk : in std_logic;
              Q : out std_logic_vector(7 downto 0)
          );
end DFF_8bit;

architecture behavior of DFF_8bit is
begin
    process(Clk)
            begin
                    if Clk'event and Clk = '1' then
                            Q <= D;
                    end if;
    end process;
end behavior;
```

## D. MUX 2 to 1

```
--------------------------------------------------------------------------------
-------------------- entity and architecture of MUX 2 to 1 8 bit -----------------
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mux2to1_8bit is
    port   (
                input_A, input_B : in std_logic_vector(7 downto 0);
                              sl : in std_logic;
                          output : out std_logic_vector(7 downto 0)
            );
end mux2to1_8bit;


architecture behavior of mux2to1_8bit is ---------- architecture of one MUX 2 to 1

    begin

            process(input_A, input_B, sl)
                    begin
                            case sl is
                                    when '0' =>
                                            output <= input_A;
                                    when '1' =>
                                            output <= input_B;
                                    when others =>
                                            output <= "ZZZZZZZZ";
                            end case;
            end process;

end behavior;
```

## E. f_out

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity f_out is
    port (  j : in std_logic_vector(7 downto 0);
            i : out std_logic_vector(7 downto 0));
end f_out;


architecture fout of f_out is
    begin
            i <= j;
end fout;
```

## F. MUX 3 to 1

```
--------------------------------------------------------------------------------
----------------------------- MUX 3 to 1 8 bit ---------------------------------
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mux3to1 is
    port (  w0,w1,w2 : in std_logic_vector(7 downto 0);
                   s : in std_logic_vector(1 downto 0);
              mux_out : out std_logic_vector(7 downto 0));
end mux3to1;

architecture behavior of mux3to1 is
    begin
            process(w0,w1,w2,s)
            begin
                    case s is
                            when "00" =>
                                    mux_out <= w0;
                            when "01" =>
                                    mux_out <= w1;
                            when "10" =>
                                    mux_out <= w2;
                            when "11" =>
                                    mux_out <= w2;
                            when others =>
                                    mux_out <= "ZZZZZZZZ";
                    end case;
            end process;
end behavior;
```

# Έκθεση Ενεργειακής Ανάλυσης

## Α. Ενεργειακή ανάλυση ολόκληρου του κυκλώματος

```
****************************************
Report : power
        -analysis_effort low
Design : RF
Version: H-2013.03-SP5-4
Date   : Thu Mar 24 19:53:27 2016
****************************************


Library(s) Used:

    NangateOpenCellLibrary (File:
/…/Desktop/libraries/NanGate/NangateOpenCellLibrary_PDKv1_3_v2010_12/Front_End/Liberty/CCS/Nangate
OpenCellLibrary_low_temp_ccs.db)

Operating Conditions: low_temp   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
RF                    5K_hvratio_1_1    NangateOpenCellLibrary

Global Operating Voltage = 1.25
Power-specific unit information :

    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


Cell Internal Power  =   9.3970 uW   (83%)
Net Switching Power  =   1.9076 uW   (17%)
                         ---------
Total Dynamic Power  =  11.3046 uW  (100%)

Cell Leakage Power   =  22.3254 uW

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
---------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.2369            0.0000            0.2369  (   0.71%)
register           8.2867            0.3438        1.3046e+04           21.6760  (  64.76%)
sequential     5.9747e-02        1.5117e-02          300.2714            0.3751  (   1.12%)
combinational      1.0505            1.1525        8.9796e+03           11.1825  (  33.41%)
---------------------------------------------------------------------------------------------------
Total              9.3970 uW         1.7482 uW     2.2325e+04 nW        33.4706 uW
```

## Β. Ενεργειακή ανάλυση των επιμέρους κυκλωματικών στοιχείων

```
Operating Conditions: low_temp   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
RF                    5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.25
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW
```

| Hierarchy | Switch Power | Int Power | Leak Power | Total Power | % |
|---|---|---|---|---|---|
| RF | 1.916 | 9.398 | 2.23e+04 | 33.635 | 100.0 |
| PE4 (PE_0) | 5.92e-02 | 0.000 | 0.000 | 5.92e-02 | 0.2 |
| TRABA_REG (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| TRABA (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| ALU (alu16bit) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| MUXB_TO_PE_B (Register_File) | 8.74e-03 | 0.000 | 0.000 | 8.74e-03 | 0.0 |
| MUXA_TO_PE_A (Register_File) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| PE3 (PE_1) | 5.92e-02 | 0.000 | 0.000 | 5.92e-02 | 0.2 |
| TRABA_REG (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| TRABA (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| ALU (alu16bit) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| MUXB_TO_PE_B (Register_File) | 8.74e-03 | 0.000 | 0.000 | 8.74e-03 | 0.0 |
| MUXA_TO_PE_A (Register_File) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| PE2 (PE_2) | 5.05e-02 | 0.000 | 0.000 | 5.05e-02 | 0.2 |
| TRABA_REG (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| TRABA (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| ALU (alu16bit) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| MUXB_TO_PE_B (Register_File) | 8.74e-03 | 0.000 | 0.000 | 8.74e-03 | 0.0 |
| MUXA_TO_PE_A (Register_File) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| PE1 (PE_3) | 5.05e-02 | 0.000 | 0.000 | 5.05e-02 | 0.2 |
| TRABA_REG (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| TRABA (f_out) | 2.14e-02 | 0.000 | 0.000 | 2.14e-02 | 0.1 |
| ALU (alu16bit) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| MUXB_TO_PE_B (Register_File) | 8.74e-03 | 0.000 | 0.000 | 8.74e-03 | 0.0 |
| MUXA_TO_PE_A (Register_File) | 3.87e-03 | 0.000 | 0.000 | 3.87e-03 | 0.0 |
| TRI_H (tristate_8_bit) | 1.75e-02 | 0.000 | 0.000 | 1.75e-02 | 0.1 |
| TRI_G (tristate_8_bit) | 3.32e-02 | 0.000 | 0.000 | 3.32e-02 | 0.1 |
| TRI_F (tristate_8_bit) | 3.29e-02 | 0.000 | 0.000 | 3.29e-02 | 0.1 |
| TRI_E (tristate_8_bit) | 3.23e-02 | 0.000 | 0.000 | 3.23e-02 | 0.1 |
| TRI_D (tristate_8_bit) | 1.75e-02 | 0.000 | 0.000 | 1.75e-02 | 0.1 |
| TRI_C (tristate_8_bit) | 4.37e-03 | 0.000 | 0.000 | 4.37e-03 | 0.0 |
| TRI_B (tristate_8_bit) | 1.75e-02 | 0.000 | 0.000 | 1.75e-02 | 0.1 |
| TRI_A (tristate_8_bit) | 4.37e-03 | 0.000 | 0.000 | 4.37e-03 | 0.0 |
| DFF_REG_4 (DFF_8bit_0) | 3.69e-03 | 0.954 | 1.55e+03 | 2.511 | 7.5 |
| DFF_REG_3 (DFF_8bit_1) | 3.70e-03 | 0.955 | 1.55e+03 | 2.512 | 7.5 |
| DFF_REG_2 (DFF_8bit_2) | 3.67e-03 | 0.953 | 1.55e+03 | 2.510 | 7.5 |
| DFF_REG_1 (DFF_8bit_3) | 3.68e-03 | 0.954 | 1.55e+03 | 2.510 | 7.5 |
| DFF_4 (DFF_8bit_4) | 3.76e-02 | 0.952 | 1.55e+03 | 2.542 | 7.6 |
| DFF_3 (DFF_8bit_5) | 3.77e-02 | 0.953 | 1.55e+03 | 2.544 | 7.6 |
| DFF_2 (DFF_8bit_6) | 3.70e-02 | 0.952 | 1.55e+03 | 2.542 | 7.6 |
| DFF_1 (DFF_8bit_7) | 3.77e-02 | 0.953 | 1.55e+03 | 2.543 | 7.6 |
| MUXPE4B (mux2to1_8bit_0) | 4.85e-02 | 0.129 | 1.02e+03 | 1.193 | 3.5 |
| MUXPE4A (mux2to1_8bit_1) | 1.75e-02 | 0.130 | 1.02e+03 | 1.164 | 3.5 |
| MUXPE3B (mux3to1) | N/A | 0.223 | 1.14e+03 | 1.354 | 4.0 |
| MUXPE3A (mux2to1_8bit_2) | 1.80e-02 | 0.130 | 1.02e+03 | 1.164 | 3.5 |
| MUXPE2B (mux2to1_8bit_3) | N/A | 0.116 | 1.02e+03 | 1.106 | 3.3 |
| MUXPE1B (mux2to1_8bit_4) | N/A | 0.116 | 1.02e+03 | 1.105 | 3.3 |
| FSMEIROS (FSM) | 0.683 | 0.926 | 3.67e+03 | 5.281 | 15.7 |

# Πηγές


1. *L. P. Alarcon, T.-T. Liu, M. D. Pierson, and J.M. Rabaey, Exploring very low-energy logc: A case study. J. of Low Power Electronics(2007)*

2. *Calhoun and A. Chandrakasan, A 256 kb sub-threshold SRAM in 65 nm CMOS, IEEE International Solid-State Circuit Conference(ISSCC), February(2006)*

3. *A. Davis and S. M. Nowick, An introduction to Asynchronous Circuit Design, Technical Report UUCS-97-013, University of Utah Technical Report, Department of Computer Science, September(2007)*

4. *M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T.N. Vijaykumar, Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories, International Symposium on Low Power Electronics and Design (ISLPED), June (2000).*

5. *Shnayder, M. Hempstead, B.-R. Chen, G. W. Allen, and M. Welsh, Simulating the power consumption of largescale sensor network applications. Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, MD, November(2004).*

6. *Warneke and K. S. J. Pister, An ultra-low energy microcontroller for smart dust wireless sensor networks. ISSCC, January(2004).*

7. *B.A. Warneke, M,D. Scott. B.S. Leibowirz, L. Zhou, C.L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. Pister, An autonomous 16 mm3 solar-powered node for distributed wireless sensor networks. Int'l Conference on Sensors, June(2002)*

8. *Wang and A. Chandrakasan, A 180 mV FFT processor using subthreshold circuit techniques. ISSCC, January(2004)*

9. *L. Nazhandali, B. Zhai, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, T. Austin, and D. Blaauw, Energy optimization of subthreshold-voltage sensor network processors. The 32nd Annual International Symposium on Computer Architecture (ISCA), June(2005)*

10. *Zhai, L. Nazhandali, J. Olson, A. Reeves, M. Minuth, R.Helfand, S.Pant, D.Blaauw, and T. Austin, A 2.60 pJ/Inst subthreshold sensor processor for optimal energy efficiency. IEEE Symposium on VLSI Circuits (VLSI-Symp), June(2006).*

11. *V. Ekanayake, C. Kelly, and R. Manohar, An ultra low-power processor for sensor networks. Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, October(2004).*

12. *V. Ekanayake, C. Kelly, and R. Manohar, BitSNAP: Dynamic significance compression for a low-energy sensor network asynchronous processor. Proceedings of the 11th International Symposium on Asynchronous Circuits and Systems, March (2005)*

13. *C.Kelly, V. Ekanayake, and R. Manohar, SNAP: A sensor-network asynchronous processor. Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems, Vancouver, BC, May(2003)*

14. *M. Sheets, F. Burghardt, T. Karakar, J. Ammer, Y.H. Chee, and J. Rabaey, A power-managed protocol processor for wireless sensor networks, VLSI, June (2006).*

15. *M. Hempstead, N. Tripathi, P. Mauro, G.-Y. Wei, and Brooks, An ultra-low power system architecture for sensor network applications. The 32nd Annual International Symposium on Computer Architecture (ISCA), June (2005).*

16. *Hill J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K., "System Architecture Directions for Networked Sensors", in Proc. of ASPLOS IX, 2000 και AmbientRT: Real-Time Operating System for embedded devices, http://www.ambient-systems.net*

17. *Tmote Sky wireless sensor device, http://www.moteiv.com και imote2 sensor, http://www.intel.com/research/sensornets/*

18. *Conner, W. S., Chhabra, J., Yarvis, M., Krishnamurthy, L., "Experimental Evaluation of Synchronization and Topology Control for In-Building Sensor Network Applications", in Proc. of WSNA '03, Sept. 2003*

19. *Polastre, J.R.., "Design and Implementation of Wireless Sensor Networks for Habitat Monitoring", Research Project, University of California, Berkeley, 2003*

20. *Ghose, K. et al., "Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation, in ISLPED'99, 1999 και Kucuk, G, et al., "Energy-Efficient Register Renaming", in Proc. of PATMOS'03, Torino, Italy, September 2003. Published as LNCS 2799, pp.219-228*

21. *Lepak, K.M., Bell, G.B., and Lipasti M.H., "Silent Stores and Store Value Locality", in IEEE Transactions on Computers, (50)11, Nov. 2011.*

22. *(Gurhan Kucuk and Can Basaran "Reducing Energy Consumption of Wireless Sensor Networks through Processor Optimizations" IEEE JOURNAL OF COMPUTERS, VOL. 2, NO. 5, JULY 2007)*

23. *Low-Power Digital Signal Processor Architecture for Wireless Sensor Nodes. IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 22, NO. 2, FEBRUARY 2014.*

24. *Prefix Sums and Their Applications | Guy E. Blelloch School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213-3890.*

25. *A Work-Efficient Step-Efficient Prefix-Sum Algorithm|*
*Shubhabrata Sengupta Aaron E. Lefohn† John D. Owens‡*