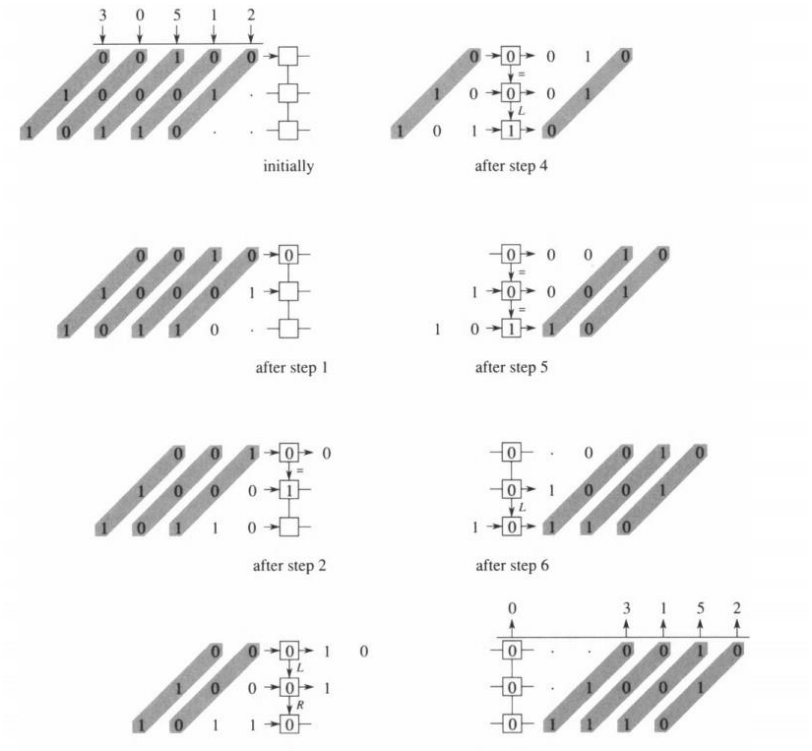
# Parallel Sorting Network : VHDL Implementation

Εργασία για το μάθημα «Παράλληλα και Κατανεμημένα Συστήματα» Ιούλιος 2021

Άγγελος Ψημίτης – Χριστοδουλόπουλος , ΑΜ : 2019-513, ΠΜΣ : ΗΑ

## 1. Εισαγωγή

Στην παρούσα εργασία γίνεται υλοποίηση σε VHDL του linear array comparison αλγορίθμου που παρουσιάζεται στο βιβλίο του Thomson Leighton : "*Introduction to parallel algorithms and architectures (1992)".* Ο αλγόριθμος βασίζεται στην ταξινόμηση αριθμών στο bit model. Στο word model η σύγκριση και η διάδοση ολόκληρων «λέξεων» (πχ 32-bit words) πραγματοποιείται σε ένα βήμα και η πολυπλοκότητα του αλγορίθμου υπολογίζεται μετρώντας τα word steps που χρειάζεται ο αλγόριθμος για να ολοκληρωθεί. Αν οι λέξεις είναι πολύ μεγάλες ή χρειάζεται να ξέρουμε τον αριθμό των transistors και των gates που απαιτούνται ώστε να σχεδιάσουμε ένα τέτοιο device, το μοντέλο αυτό δεν είναι πρακτικό. Αντίθετα το bit model μας δίνει μια πιο ακριβή εκτίμηση της πολυπλοκότητας του αλγορίθμου καθώς οι διεργασίες που εφαρμόζονται στις λέξεις (αριθμούς) διαμοιράζονται σε διεργασίες πάνω σε κάθε bit και ο χρόνος μετριέται σε bit steps. Το βασικό κλειδί του διαμοιρασμού της λειτουργίας του sorting αλγορίθμου σε bit operations είναι η μέθοδος με την οποία πραγματοποιούνται τα comparisons. Η πρώτη μέθοδος χρησιμοποιεί ένα linear array ώστε οι αριθμοί να συγκρίνονται ανα bit, ξεκινώντας απ' τα most significant bits, και ολοκληρώνεται σε k - βήματα (για k-bit words). Η δεύτερη μέθοδος χρησιμοποιεί ένα binary tree network και ολοκληρώνεται σε $logk + 1$ βήματα. Ο linear array αλγόριθμος μπορεί να λειτουργήσει καλύτερα απ' το binary tree με χρήση *pipeline* όπως φαίνεται στο *σχήμα 1*. Στο παράδειγμα αυτό ένα 3-cell array διαλέγει τον μικρότερο αριθμό απ' τους 3,0,5,1,2 (011,000,101,001,010), ενώ οι υπόλοιποι αριθμοί μετατίθενται στην έξοδο. Ύστερα από $N + k - 1$ βήματα (όπου N το πλήθος των λέξεων και k το πλήθος των bits της κάθε λέξης) τα bits του μικρότερου αριθμού θα περιέχονται στο array και τα bits των υπολοίπων αριθμών θα έχουν εξαχθεί σε κλιμακωτή μορφή.




*Σχήμα 1*: Linear array για την αποθήκευση του μικρότερου αριθμού εκ των 3,0,5,1,2.

Χρησιμοποιώντας $N$ $k$ −cell arrays μπορούμε να φτιάξουμε ένα $k \times N$ array (mesh) από bit processors ώστε η πρώτη φάση της ταξινόμησης $N$ $k$ −bit αριθμών να ολοκληρώνεται σε $2N + k - 2$ bit steps. Η δεύτερη φάση της ταξινόμησης (εξαγωγή των αριθμών με αύξουσα σειρά απ' το κύκλωμα) δεν έχει υλοποιηθεί στην εργασία.

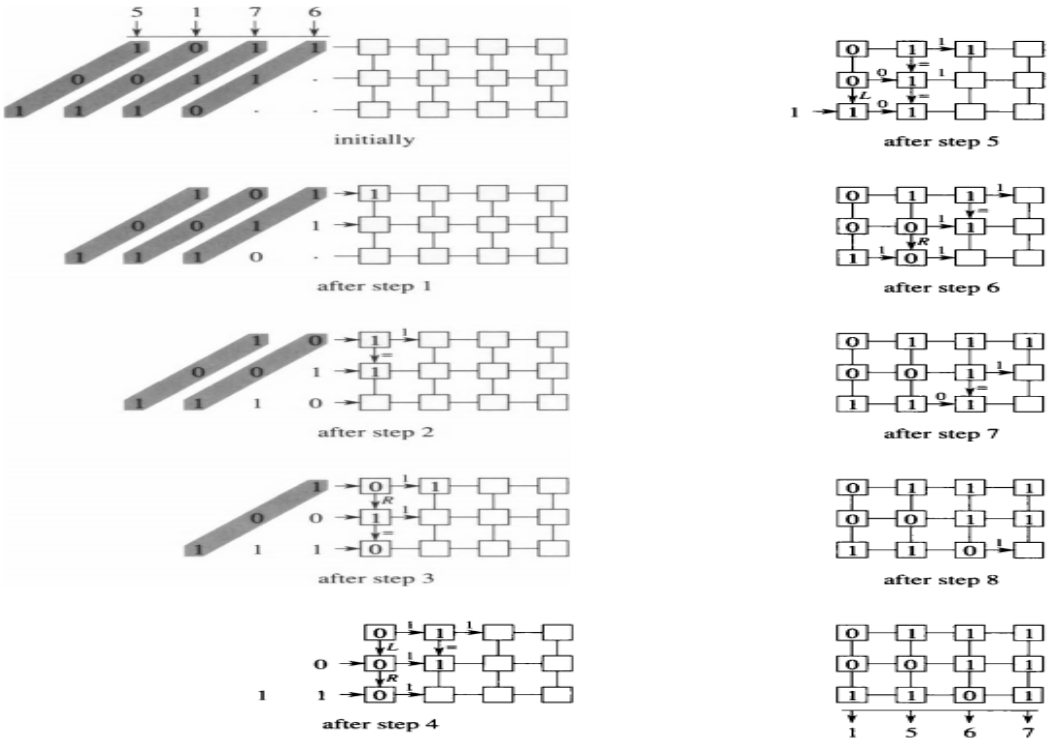
## 2. Περιγραφή της λειτουργίας του αλγορίθμου.

Ο αλγόριθμος χρησιμοποιεί ένα mesh από $k \times N$ πανομοιότυπους bit processors (cells) η λειτουργία των οποίων είναι απλή. Κάθε cell είναι αδρανές μέχρι να λάβει ένα bit απ' τα αριστερά. Το cell τότε «ξυπνάει» και αποθηκεύει το bit. Στα επόμενα βήματα, λαμβάνει ένα bit από αριστερά κι ένα σήμα ("L", "R", ή "=") από πάνω και προχωράει σύμφωνα με τους παρακάτω κανόνες. (Όλα τα cells της πρώτης σειράς λαμβάνουν "=").

- ➔ Αν λάβει "L", τότε στέλνει στα δεξιά το bit που έλαβε απ' τα αριστερά και στέλνει κάτω το σήμα "L".
- ➔ Αν λάβει "R", τότε αποθηκεύει το bit που έλαβε από αριστερά, στέλνει δεξιά το bit που είχε αποθηκευμένο και κάτω το σήμα "R".
- ➔ Αν λάβει "=", τότε συγκρίνει το bit που έλαβε απ' τα αριστερά με το bit που έχει αποθηκευμένο, στέλνει δεξιά το μεγαλύτερο κι αποθηκεύει το μικρότερο, ενώ κάτω στέλνει το κατάλληλο σήμα αναλόγα με το αποτέλεσμα της σύγκρισης.

Το κάθε cell λειτουργεί με αυτό τον τρόπο μέχρι να σταματήσει να λαμβάνει bits απ'τα αριστερά. Στο σημείο αυτό έχει ολοκληρωθεί η πρώτη φάση της ταξινόμησης και το κάθε linear array (column) έχει αποθηκευμένο έναν αριθμό (word). Το πρώτο column έχει τον μικρότερο, το δεύτερο τον αμέσως μεγαλύτερο κ.ο.κ. Τα cells του 1ου row του mesh έχουν τα most significant bits των λέξεων που ταξινομήθηκαν.

Η λειτουργία του αλγορίθμου παρουσιάζεται στο *σχήμα 2*.

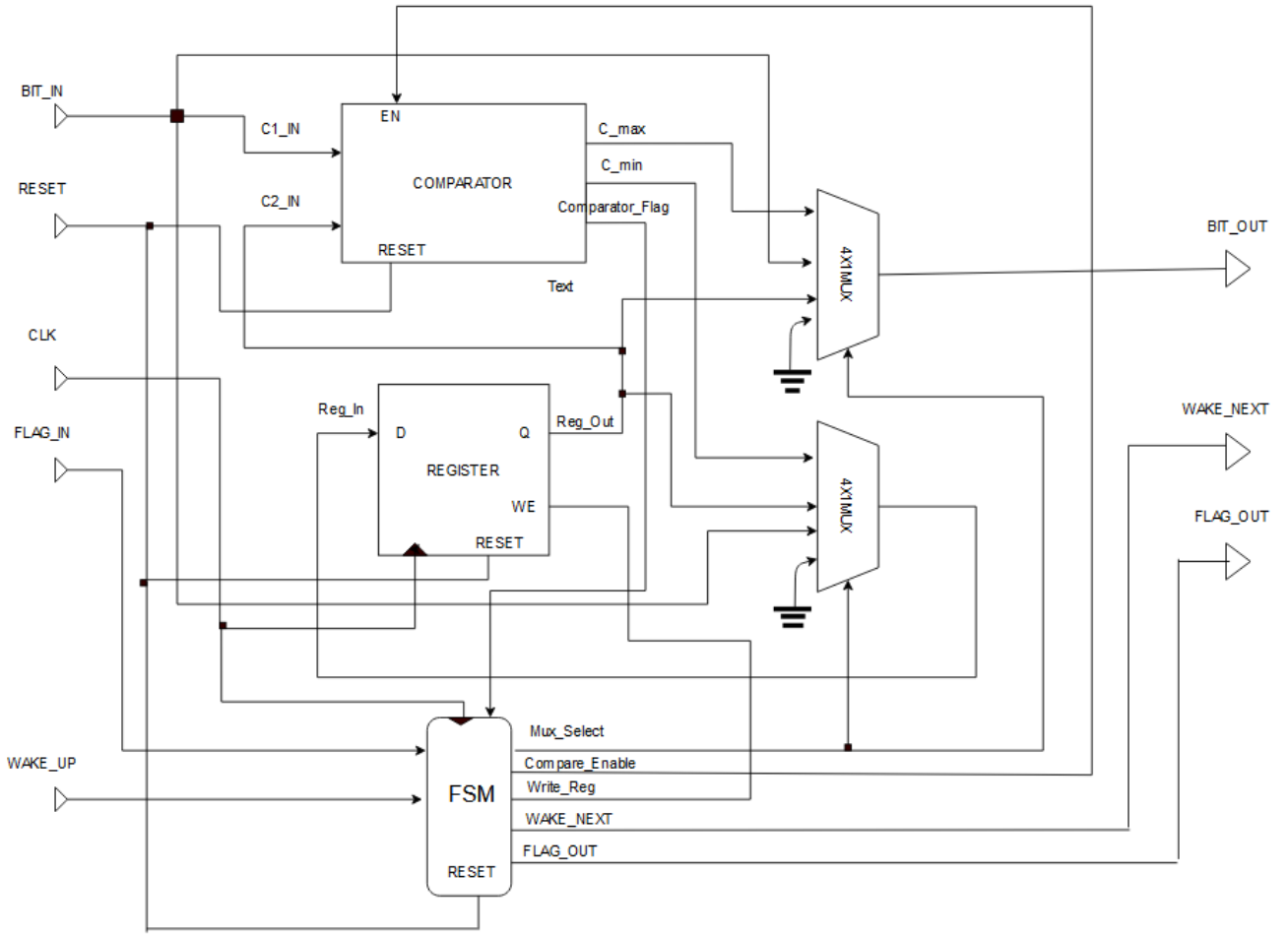



*Σχήμα 2*: Η 1η φάση της ταξινόμησης των αριθμών 5,1,7,6 χρησιμοποιώντας ένα 3×4 array στο bit model.

## 3. Περιγραφή των δομικών στοιχείων και της λειτουργίας του κυκλώματος.

### Processing Element (PE)

Το κάθε cell του πίνακα είναι ένα Processing Element (PE) το οποίο πραγματοποιεί τη λειτουργία που περιγράφηκε στην ενότητα 2. Το κύκλωμα του PE φαίνεται στην εικόνα 3 κι αποτελείται από έναν Comparator, 2 Multiplexers 4×1, ένα Register (D Flip- Flop με σύγχρονο RESET και Write Enable) κι ένα FSM.



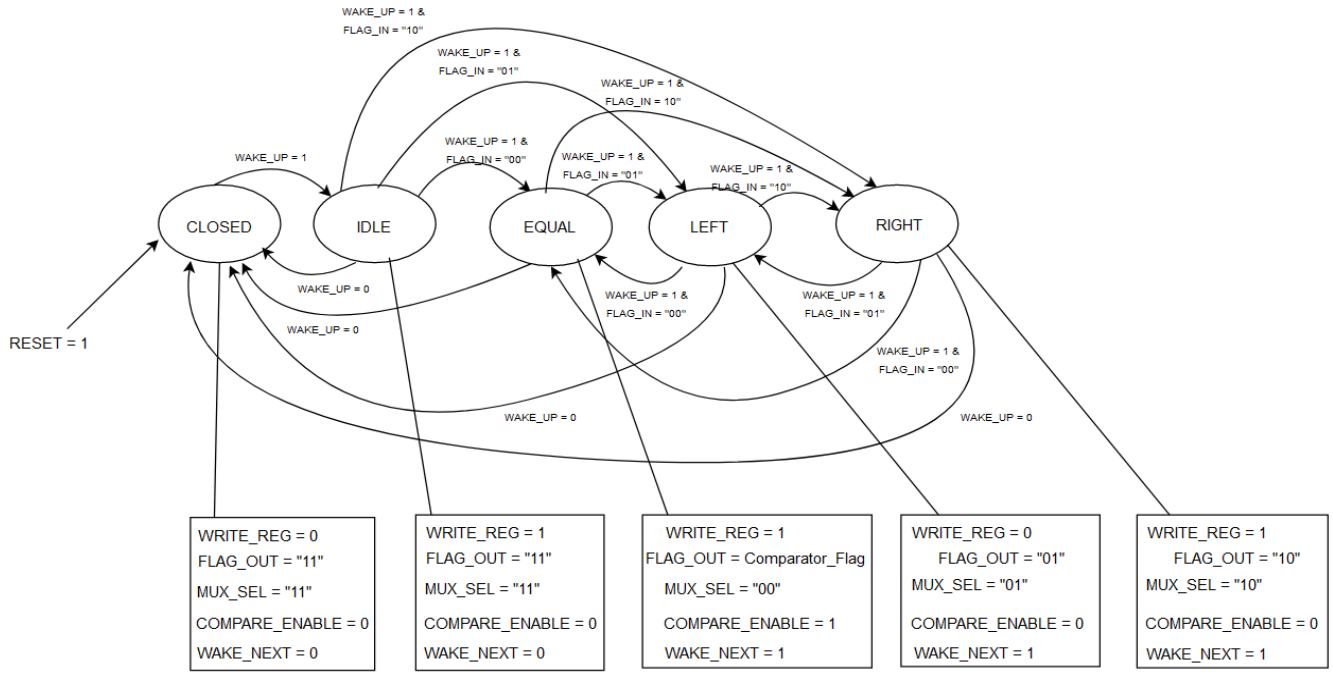

*Σχήμα 3*: Processing Element Diagram

### Finite State Machine (FSM)

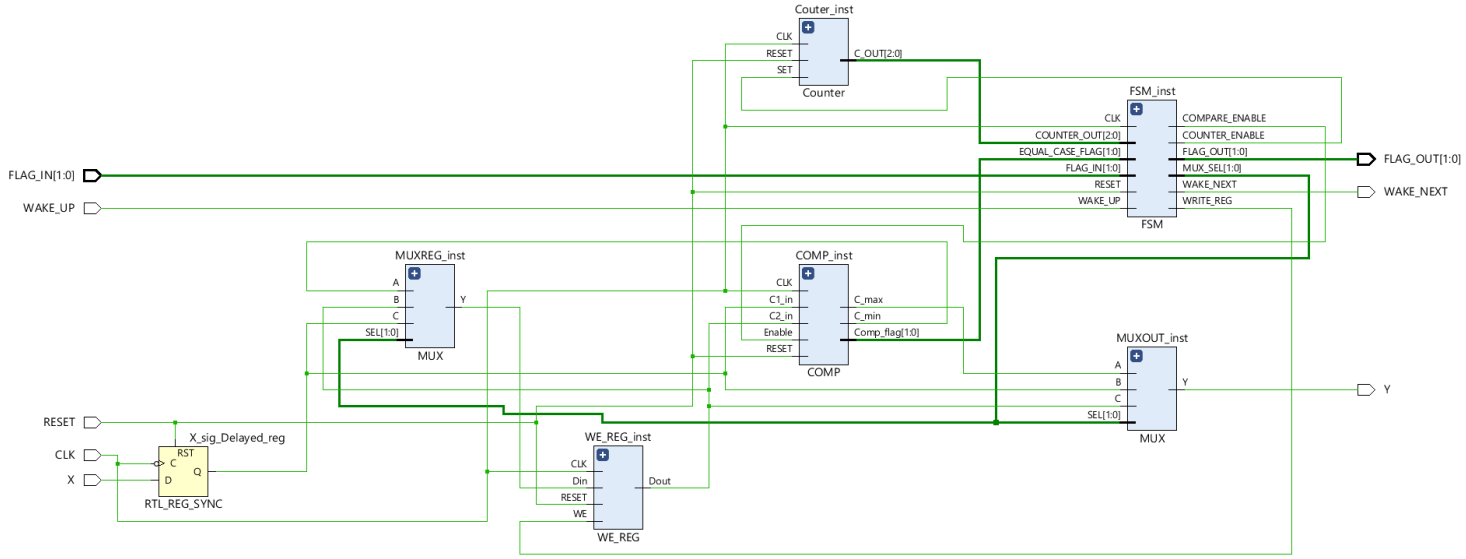
Το FSM αποτελεί τον «ηλεκτρονικό εγκέφαλο» του PE και καθορίζει τις τιμές των σημάτων Mux_Select, Compare_Enable, Write_Reg, Wake_Next και FLAG_OUT, αναλόγως της κατάστασης στην οποία βρίσκεται. Οι κατάστασεις απ' τις οποίες περνάει το FSM καθορίζονται σύμφωνα με τα σήματα WAKE_UP και FLAG_IN όπως φαίνεται στο transition state diagram του *σχήματος 4*. Το διάγραμμα αυτό σχεδιάστηκε έτσι ώστε το PE να μπορεί να επιτελεί τις λειτουργίες που περιγράφηκαν στην ενότητα 2. Το σήμα FLAG_IN κωδικοποιείται ως εξής :

- FLAG_IN = "00" , "=",
- FLAG_IN = "01" , "L",
- FLAG_IN = "10" , "R",
- FLAG_IN = "11" , "No meaning"

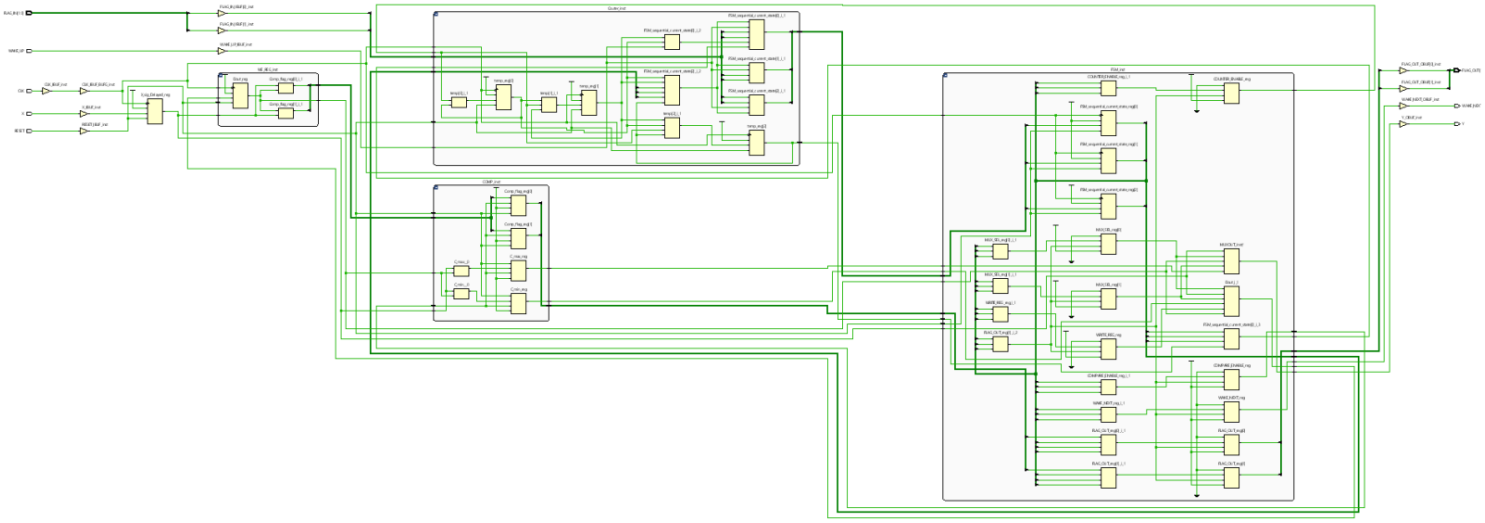
Η κατάσταση «CLOSED» είναι η αρχική κατάσταση στην οποία βρίσκονται όλα τα PEs. Μόλις το σήμα WAKE_UP πάρει την τιμή 1 περνούν στην κατάσταση IDLE όπου το σήμα έγκρισης εγγραφής Write_Reg ενεργοποιείται ώστε το 1ο bit που λαμβάνεται στην είσοδο να αποθηκευτεί στον Register. Στη συνέχεια (και για όσο το σήμα WAKE_UP έχει την τιμή 1) περνούν στις καταστάσεις «EQUAL», «LEFT» ή «RIGHT» αναλόγως της τιμής του σήματος FLAG_IN ( 00, 01 και 10 αντίστοιχα). Για το PE που βρίσκεται στο (i,j)το σήμα FLAG_IN λαμβάνεται από το PE που βρίσκεται ακριβώς από πάνω του ( δηλαδή το (i-1 , j) ), ενώ όλα τα PEs της πρώτης σειράς λαμβάνουν FLAG_IN = "00", που σημαίνει ότι περνούν στην κατάσταση «EQUAL». Όταν ένα PE βρίσκεται στο state «EQUAL» ενεργοποιείται το σήμα Compare_Enable ώστε ο COMPARATOR να συγκρίνει το bit που λαμβάνεται στην είσοδο με το bit που είναι αποθηκευμένο στον register. Το σήμα mux_sel παίρνει την τιμή "00" με αποτέλεσμα ο πολυπλέκτης που συνδέεται με το BIT_OUT να περνάει στην έξοδο το μεγαλύτερο bit κι ο πολυπλέκτης που συνδέεται με την είσοδο του Register να περνάει το μικρότερο. Το σήμα Write_Reg παραμένει ανοιχτό ώστε το μικρότερο bit να εγγραφεί στον Register. Το σήμα εξόδου WAKE_NEXT ενεργοποιείται ώστε να «ξυπνήσει» το γειτονικό PE (το (i , j+1) ) και παραμένει ενεργοποιημένο μέχρις ότου να επιστρέψει στην κατάσταση CLOSED. Το σήμα FLAG_OUT συνδέεται με το σήμα Comparator_Flag (το οποίο αναλόγως του αποτελέσματος της σύγκρισης παίρνει τις τιμές "00", "01", "10") και περνιέται στην έξοδο ώστε να καθορίσει την κατάσταση στην οποία θα βρεθεί το PE που βρίσκεται από κάτω (το (i + 1, j) ). Στην κατάσταση «LEFT» το σήμα Write_Reg παίρνει την τιμή 0 διότι δεν χρειάζεται να ανανεώσει το bit που είναι αποθηκευμένο στον Register, ενώ το σήμα Mux_Sel παίρνει την τιμή "01" ώστε να περαστεί στην έξοδο (BIT_OUT) το bit που λαμβάνεται στην είσοδο. Το σήμα FLAG_OUT παίρνει επίσης την τιμή "01" προκειμένου να ενημερώσει το PE που βρίσκεται από κάτω ότι ο αριθμός αριστερά από αυτόν του οποίου το bit έχει ήδη αποθηκευμένο είναι μεγαλύτερος. Στην κατάσταση «RIGHT» το σήμα Write_Reg παίρνει την τιμή 1 διότι πρέπει να αποθηκευτεί το bit που λαμβάνεται στην είσοδο. Το σήμα mux_sel παίρνει την τιμή "10" ώστε ο πολυπλέκτης να περάσει στην έξοδο το bit που ήταν προηγουμένως αποθηκευμένο στον Register, ενώ το σήμα FLAG_OUT παίρνει την τιμή "10" ώστε να ενημερώσει το PE που βρίσκεται από κάτω ότι ο αριθμός του οποίου το bit είναι ήδη αποθηκευμένο είναι μεγαλύτερος απ' τον αριθμό του οποίου το bit θα λάβει στην είσοδο και να μεταβεί κι αυτό στην κατάσταση «RIGHT». Μόλις το σήμα WAKE_UP κλείσει το PE μεταβαίνει στην κατάσταση CLOSED όπου αδρανοποιεί όλα τα σήματα εξόδου. Μόλις όλα τα PEs επιστρέψουν στην κατάσταση CLOSED η 1η φάση του αλγορίθμου έχει ολοκληρωθεί και οι αριθμοί που δώσαμε στο κύκλωμα είναι αποθηκευμένοι κατά αύξουσα σειρά στους Registers των cells.




*Εικόνα 4* : State Transition Diagram του FSM



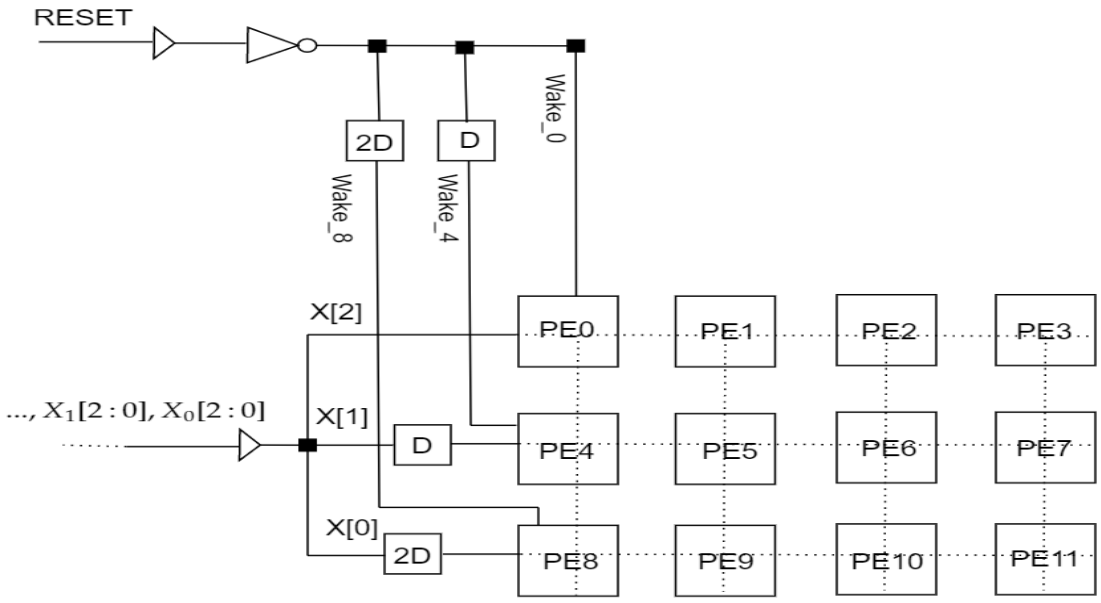

*Εικόνα 5*: RTL του Processing Element



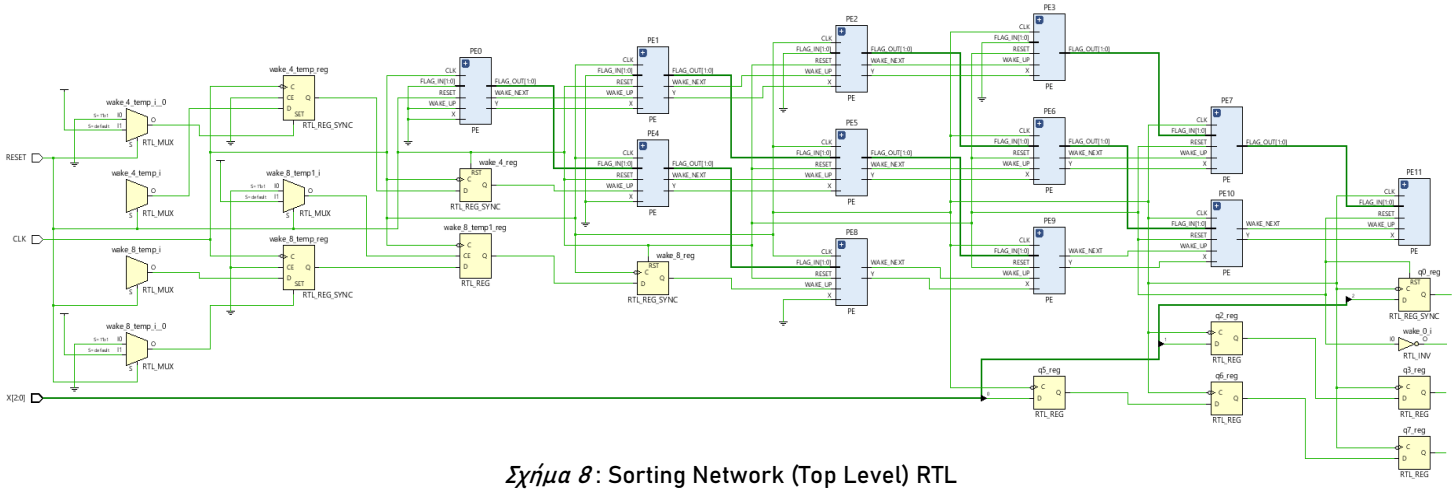
*Εικόνα 6*: Synthesized Design του Processing Element

## 4. Περιγραφή του Top Level του κυκλώματος

Το top level του κυκλώματος είναι structural και περιλαμβάνει $3 \times 4 = 12$ PEs τα οποία χρησιμοποιούνται για την ταξινόμηση 4 αριθμών των 3-bits. Στην είσοδο λαμβάνονται οι λέξεις των 3 bits (σήμα X[2:0]), το σήμα CLK το οποίο διαμοιράζεται σε όλα τα σύγχρονα στοιχεία του κυκλώματος καθώς και το RESET. Τα PE0, PE4 και PE8 λαμβάνουν τα X[2], X[1] και X[0] αντίστοιχα, με τις κατάλληλες καθυστερήσεις προκειμένου να γίνει το pipelining. Τα 3 αυτά cells «ξυπνούν» διαδοχικά (με καθυστέρηση ενός clock_cycle ) μόλις το RESET πάρει την τιμή 0. Τα PEs του 1ου column του mesh γυρνούν στην κατάσταση «CLOSED» (απενεργοποιούνται) με τη βοήθεια ενός εσωτερικού μετρητή (Counter) ο οποίος ξεκινάει να μετράει μόλις το σήμα WAKE_UP πάρει την τιμή 1. Ύστερα από 4 clock_cycles (όσα και τα διαδοχικά bits που λαμβάνονται στην είσοδο) το WAKE_UP παίρνει την τιμή 0 και το PE κλείνει, αδρανοποιώντας στη συνέχεια τα γειτονικά του, με αποτέλεσμα ο αλγόριθμος να ολοκληρώνεται επιτυχώς. Τα παραπάνω φαίνονται καθαρά στο *σχήμα 7*.



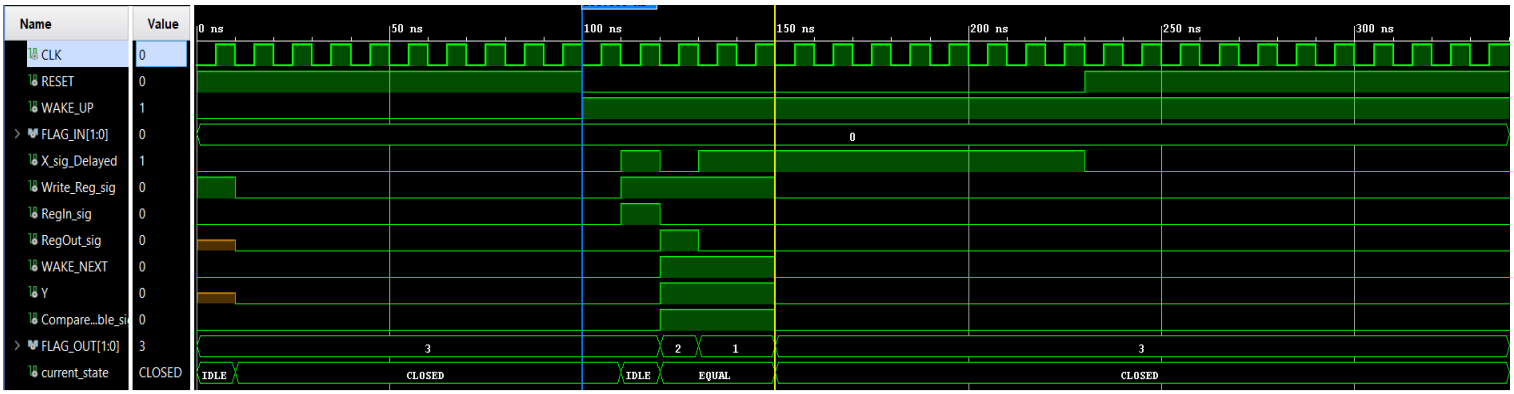

*Εικόνα 7*: Sorting Network (Top Level) Abstraction



*Σχήμα 8*: Sorting Network (Top Level) RTL

## 5. Επαλήθευση ορθής λειτουργίας του κυκλώματος (Simulations)

### 5.1. Processing Element Simulation




*Σχήμα 9*: Behavioral Simulation της λειτουργίας του PE (i)

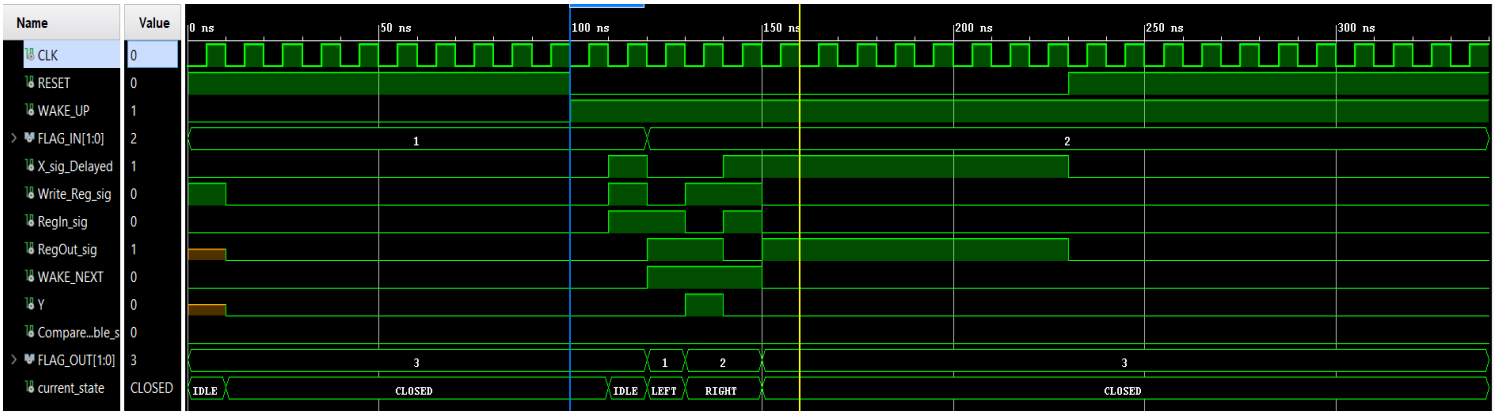
Στο *σχήμα 9* φαίνεται η λειτουργία ενός PE που βρίσκεται στο 1ο row και το σήμα FLAG_IN έχει πάντα την τιμή "00" που σημαίνει πως το PE βρίσκεται συνεχώς όσο λειτουργεί στην κατάσταση «EQUAL».

Δίνουμε ως input τα bits 1, 0, 1, 1 διαδοχικά και παρατηρούμε τα εξής : Όταν το RESET γίνεται μηδέν το σήμα WAKE_UP παίρνει την τιμή 1 και στο αμέσως επόμενο clock_cycle το PE μεταβαίνει στην κατάσταση IDLE. Εκεί ενεργοποιείται το σήμα Write_Reg με αποτέλεσμα να γράφεται το 1 στον Register. Στη συνέχεια το PE μεταβαίνει στην κατάσταση «EQUAL» όπου το σήμα Compare_Enable ενεργοποιείται και συγκρίνονται το 1 που είναι αποθηκευμένο στο Register (RegOut_sig) με το 0 που έρχεται στην είσοδο. Αποθηκεύεται το 0 (RegIn_Sig = 0) και περνάει στην έξοδο το 1 (Y =1), στα δύο επόμενα Clock_cycles ο Comparator συγκρίνει το 0 (αποθηκευμένο στον Register) με δύο διαδοχικά 1 που λαμβάνονται στην είσοδο και τα περνάει στην έξοδο. Μόλις σταματήσει να λαμβάνει bits στην είσοδο το PE μεταβαίνει στην κατάσταση «CLOSED» και το σήμα WAKE_NEXT απενεργοποιείται. Η μετάβαση στην κατάσταση «CLOSED» γίνεται με τη βοήθεια ενός Counter ο οποίος ενεργοποιείται όταν η κατάσταση είναι «IDLE» και σταματάει μόλις περάσουν 4 clock_cycles (όσα και τα bit που λαμβάνει το PE στην είσοδο).

```
50    stimulus: process
51    begin
52
53      RESET <= '1';
54      WAKE_UP <= '0';
55      FLAG_IN <= "00";
56      X <= '0';
57      wait for 10*clock_period;
58      RESET <= '0';
59      WAKE_UP <= '1';
60      X <= '1';
61      wait for 1*clock_period;
62      X <= '0';
63  --    FLAG_IN <= "10";
64      wait for 1*clock_period;
65      X <= '1';
66  --    FLAG_IN <= "01";
67      wait for 1*clock_period;
68      X <= '1';
69      wait for 10*clock_period;
70      RESET <= '1';
71      wait for 10*clock_period;
72
73      stop_the_clock <= true;
74      wait;
75    end process;
```

*Stimulus process του PE (σχήμα 9)*

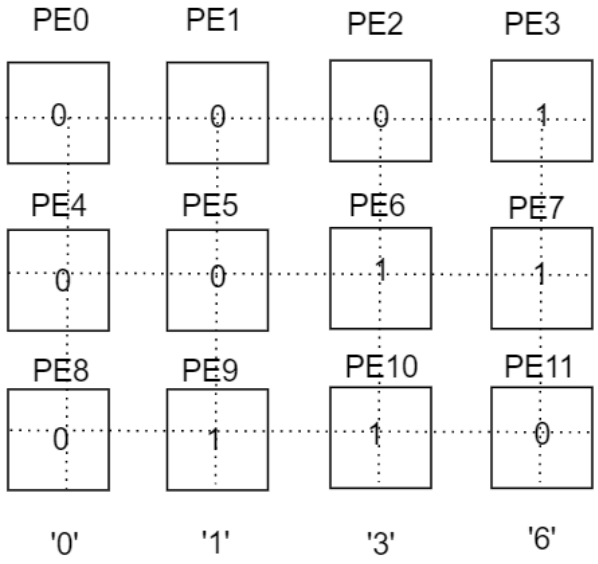
Στο *σχήμα 10* επιβεβαιώνεται η ορθή λειτουργία του PE με διαφορετικό stimulus process όπου δοκιμάζουμε την απόκριση του συστήματος σε διαφορετικές τιμές του σήματος FLAG_IN. Ως input δίνουμε την ακολουθία 1, 0, 0, 1. Το FLAG_IN είναι αρχικά "01" και στη συνέχεια γίνεται "10". Βλέπουμε ότι στην κατάσταση IDLE εγγράφεται στον Register το 1 και στη συνέχεια μεταβαίνει σωστά στην κατάσταση LEFT. Εκεί λαμβάνεται στην είσοδο το 0 και περνιέται στην έξοδο, ενώ διατηρείται αποθηκευμένο το 1. Στη συνέχεια εφόσον το σήμα FLAG_IN έχει γίνει "10" περνάει στην κατάσταση RIGHT όπου λαμβάνεται το 0 το οποίο αποθηκεύεται και περνιέται στην έξοδο η τιμή που ήταν αποθηκευμένη στον Register (το 1). Τέλος, παραμένοντας στην κατάσταση RIGHT, λαμβάνεται το 1 το οποίο αποθηκεύεται και στην έξοδο περνάει το 0. Τελικά όταν το PE κλείνει έχει αποθηκευμένη την τιμή 1.



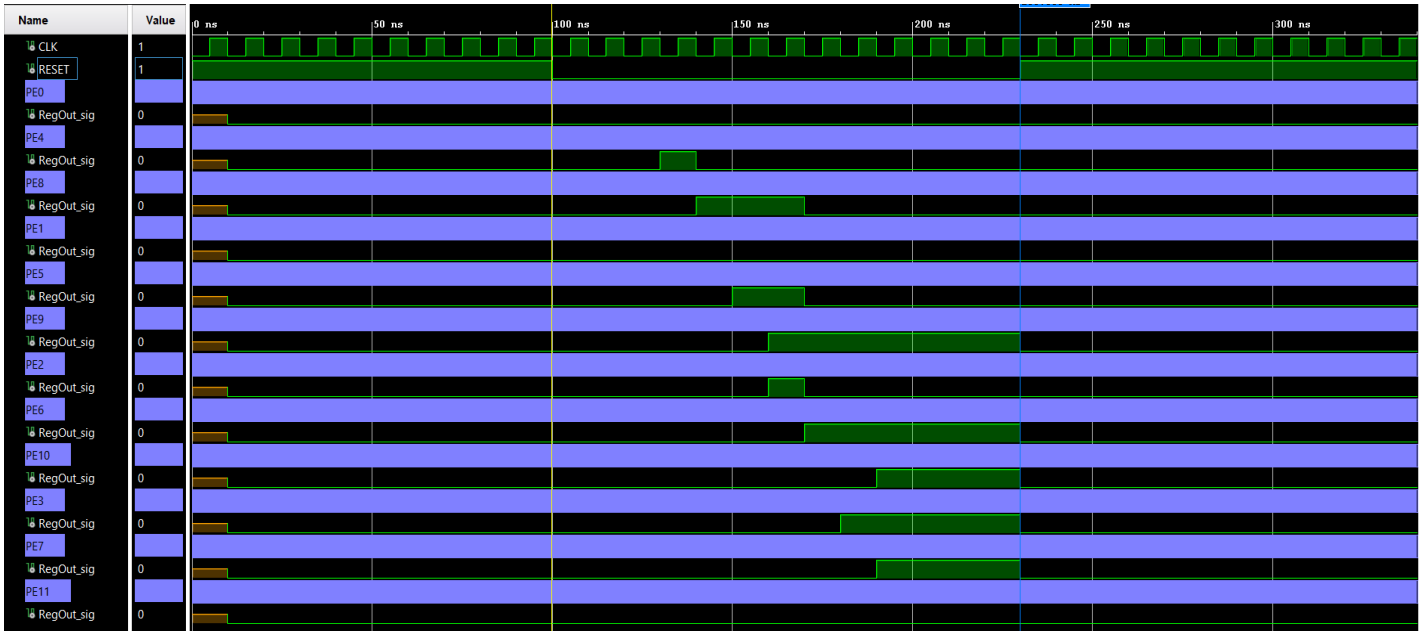

*Σχήμα 10*: Behavioral Simulation της λειτουργίας του PE(ii)

## 5.2. Top level (Sorting Network) Simulation

Στο σχήμα 11 βλέπουμε τα αποτελέσματα της λειτουργίας του κυκλώματος όταν στην είσοδο λαμβάνονται διαδοχικά οι αριθμοί 3, 1, 6, 0 (011, 001, 110, 000). Όταν όλα τα PEs έχουν ολοκληρώσει τη λειτουργία τους και λίγο πριν το RESET πάρει την τιμή 1, οι αριθμοί έχουν ταξινομηθεί με αύξουσα σειρά (*σχήμα 11 / σχήμα 12*) και η λειτουργία του αλγόριθμου της ενότητας 2 έχει υλοποιηθεί.

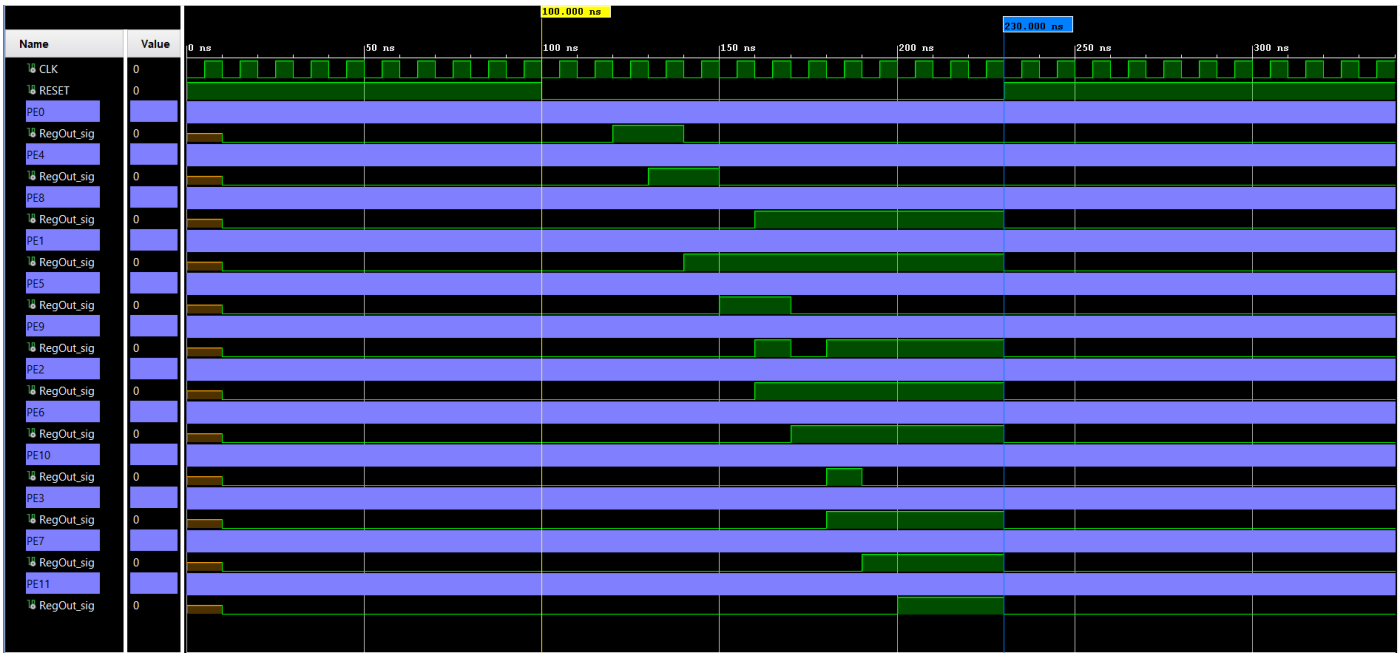



*Σχήμα 11*: Οι τιμές που έχουν αποθηκεύσει τα PEs του mesh μετά την ολοκλήρωση της 1ης φάσης της λειτουργίας του κυκλώματος όταν στην είσοδο δόθηκαν με τη σειρά οι αριθμοί 3,1,6,0.




*Σχήμα 12*: Behavioral Simulation της λειτουργίας του top level όταν στην είσοδο δόθηκαν με τη σειρά οι αριθμοί 3,1,6,0.

Στο *σχήμα 13* δίνεται άλλο ένα παράδειγμα της λειτουργίας, όπου ταξινομούνται οι αριθμοί 6,7,1,5.




*Σχήμα 13*: Behavioral Simulation της λειτουργίας του top level όταν στην είσοδο δόθηκαν με τη σειρά οι αριθμοί 6,7,1,5.

## 6. Future Work

Το κύκλωμα αυτό είναι σχεδιασμένο ώστε να ταξινομεί 4 αριθμούς των 3-bits. Στο top-level το κάθε PE έχει προγραμματιστεί ξεχωριστά κι έχει συνδεθεί με τα κατάλληλα σήματα προκειμένου η συστοιχία των 12 PEs να μπορεί να ταξινομεί σωστά τους αριθμούς. Ενδεχομένως με μια πιο προσεκτική σχεδίαση των δομικών στοιχείων το κύκλωμα να μπορεί να παραμετροποιηθεί ώστε να ταξινομεί περισσότερους αριθμούς με περισσότερα bits (πχ $12 \times 8$) χρησιμοποιώντας generic εντολές και for generate loops ώστε τα PEs να κάνουν populate το chip έχοντας την κατάλληλη διασύνδεση μεταξύ του. Επίσης θα μπορούσε να υλοποιηθεί και η $2^n$ φάση του αλγορίθμου, δηλαδή η εξαγωγή των αριθμών απ' το network με αύξουσα σειρά και η αποθήκευση τους σε κάποιο άλλο unit (memory).

## Αναφορές

F. THOMSON LEIGHTON, CHAPTER 1 - ARRAYS AND TREES, Editor(s): F. THOMSON LEIGHTON,

Introduction to Parallel Algorithms and Architectures,

Morgan Kaufmann, 1992, Pages 1-276, ISBN 9781483207728, https://doi.org/10.1016/B978-1-4832-0772-8.50005-4.

(https://www.sciencedirect.com/science/article/pii/B9781483207728500054)