ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση αλγορίθμου FFT σε γλώσσα VHDL

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Ι. Μούτσος

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2006

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση αλγορίθμου FFT σε γλώσσα VHDL

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Ι. Μούτσος

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Απριλίου 2006.

.....	.....	.....
Κ. Πεκμεστζή	Ν. Κοζύρης	Γ. Οικονομάκος
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

Αθήνα, Απρίλιος 2006

.....
Νικόλαος Ι. Μούτσος

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

Copyright © Νικόλαος Μούτσος 2006.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

στην οικογένειά μου

ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της διπλωματικής είναι η σχεδίαση και περιγραφή σε γλώσσα VHDL ενός κυκλώματος υπολογισμού του N-σημείων διακριτού μετασχηματισμού Fourier, με βάση τον αλγόριθμο του βάσης-2 γρήγορου μετασχηματισμού Fourier N-σημείων με αποδεκατισμό στο πεδίο της συχνότητας. Τα σημεία θα είναι μιγαδικοί αριθμοί με το πραγματικό και φανταστικό μέρος σε μορφή συμπληρώματος ως προς 2. Για τον σκοπό αυτό σχεδιάζονται και περιγράφονται σε γλώσσα VHDL βασικά κυκλώματα πράξεων με μιγαδικούς αριθμούς, όπως μιγαδικός αθροιστής, μιγαδικός αφαιρέτης και μιγαδικός πολλαπλασιαστής. Πιο αναλυτικά στο κεφάλαιο 1 περιγράφονται τα βασικά μιγαδικά αριθμητικά κυκλώματα που θα χρησιμοποιηθούν στην υλοποίηση, στο κεφάλαιο 2 παρουσιάζεται ο αλγόριθμος βάσης-2 γρήγορος μετασχηματισμός Fourier με αποδεκατισμό στο πεδίο της συχνότητας, στο κεφάλαιο 3 παρουσιάζεται η σχεδίαση του κυκλώματος του αλγορίθμου που προτείνεται και στο κεφάλαιο 4 μια εναλλακτική σχεδίαση με μικρότερο χρόνο καθυστέρησης. Στο παράρτημα παρατίθεται και όλος ο κώδικας σε VHDL που χρησιμοποιείται για την περιγραφή καθώς και ο κώδικας με τον οποίο ελέγχετε.

Λέξεις Κλειδιά: Γρήγορος μετασχηματισμός Fourier, FFT, μιγαδικός αθροιστής συνεχούς διοχεύτεσυης, μιγαδικός αφαιρέτης συνεχούς διοχεύτευσης, μιγαδικός πολλαπλασιαστής συνεχούς διοχέτευσης, μορφή συμπληρώματος ως προς 2, VHDL, ROM, RAM, καταχωρητής ολίσθησης, μετρητής

ABSTRACT

The scope of this thesis is the design and implementation of the N-Point Fast Fourier Transform (FFT), based on Radix-2 FFT with decimation in frequency algorithm, in VHDL. The points must have real and imaginary part in the two complement's form. For that reason basic complex arithmetic circuits are designed and implemented in VHDL, including a complex adder, a complex subtracter and a complex multiplier. Specifically in chapter 1 is illustrated the basic complex arithmetic circuits, that used at implementation, in chapter 2 the Radix-2 N-Point FFT with decimation in frequency algorithm is presented, in chapter 3 the design of FFT circuit that is proposed is shown and in chapter 4 an alternative design with reduced latency is proposed. In appendices all the VHDL code used for the implementation of the design, and testing is attached.

Keywords: Fast Fourier Transform, FFT, complex adder pipelined, complex subtracter pipelined, complex multiplier pipelined, two complement's form, VHDL, ROM, RAM, shift register, counter

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ 7

ABSTRACT 9

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 11

Πίνακας Σχημάτων 15

1 ΜΙΓΑΔΙΚΑ ΑΡΙΘΜΗΤΙΚΑ ΚΥΚΛΩΜΑΤΑ 18

1.1 Υλοποίηση Μιγαδικού Αθροιστή (Complex Adder) 18

1.1.1 Μιγαδικός Πλήρης Αθροιστής (Complex Full Adder ή CFA) 18

1.1.2 Μιγαδικός Αθροιστής Διάδοσης Κρατουμένου 20

1.1.3 Άθροιση αριθμών με μορφή συμπληρώματος ως προς 2 20

1.1.4 Μετατροπή του μιγαδικού αθροιστή διάδοσης κρατουμένου σε συνεχούς διοχέτευσης (PIPELINING) 22

1.2 Υλοποίηση Μιγαδικού Αφαιρέτη 24

1.2.1 Βασικό κύτταρο του μιγαδικού αφαιρέτη 24

1.2.2 Μιγαδικός Αφαιρέτης Διάδοσης Κρατουμένου 25

1.2.3 Αφαίρεση αριθμών σε μορφή συμπληρώματος ως προς 2 26

1.2.4 Μετατροπή του μιγαδικού αφαιρέτη διάδοσης κρατουμένου σε συνεχούς διοχέτευσης (PIPELINING) 27

1.3 Υλοποίηση Μιγαδικού Πολλαπλασιαστή 29

1.3.1 Εισαγωγή 29

1.3.2 1ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC1) 30

1.3.3 2ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC2) 31

1.3.4 3ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC3) 32

1.3.5 4ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC4) 33

1.3.6 Παράλληλος Μιγαδικός αθροιστής με αποθήκευση κρατουμένου 34

1.3.7 Μιγαδικός πολλαπλασιαστής συνεχούς διοχέτευσης με μιγαδικούς αθροιστές αποθήκευσης κρατουμένου 39

2 ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΔΙΑΚΡΙΤΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ FOURIER ΜΕ ΒΑΣΗ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΤΟΥ ΓΡΗΓΟΡΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ FOURIER 43

2.1 Διακριτός Μετασχηματισμός Fourier (DFT) 43

2.2 Βάσης-2 Γρήγορος Μετασχηματισμός Fourier με αποδεκατισμό στο πεδίο της συχνότητας 43

3 ΥΛΟΠΟΙΗΣΗ ΜΕ ΒΑΣΗ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΒΑΣΗΣ-2 ΓΡΗΓΟΡΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ FOURIER ΜΕ ΑΠΟΔΕΚΑΤΙΣΜΟ ΣΤΟ ΠΕΔΙΟ ΤΗΣ ΣΥΧΝΟΤΗΤΑΣ 47

3.1 Μονάδα Butterfly (Butterfly Unit) 47

3.1.1 Εισαγωγή 47

3.2 Αριθμητικά κυκλώματα μέσα στην μονάδα butterfly 49

3.2.1 Μιγαδικός Αθροιστής 49

3.2.2 Μιγαδικός Αφαιρέτης 49

3.2.3 Μιγαδικός Πολλαπλασιαστής 49

3.3 Μονάδα εξαγωγής των παραγόντων αναστροφής 51

3.4 Μονάδα Αναδιάταξης (Shuffling Unit) 52

3.5 Στοιχεία καθυστέρησης (Delay Elements) 53

3.6 Μετρητές (Counters) 54

3.7 Έλεγχος 55
check1.m 55
3.8 Υπολογισμός του σηματοθορυβικού λόγου (SNR) της υλοποίησης 60
3.9 Συμπεράσματα 61
4 ΜΙΑ ΕΝΑΛΛΑΚΤΙΚΗ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ FFT ΜΕ ΛΙΓΟΤΕΡΑ ΣΤΟΙΧΕΙΑ ΚΑΘΥΣΤΕΡΗΣΗΣ 63
4.1 Εισαγωγή 63
4.2 Συνδυασμός του μιγαδικού αφαιρέτη με τον μιγαδικό πολλαπλασιαστή 64
4.3 Τροποποιημένες μονάδες butterfly 65
4.3.1 1η μονάδα butterfly 65
4.3.2 Ενδιάμεσες μονάδες butterfly 68
4.3.3 Τελευταία μονάδα butterfly 70
4.4 Τροποποιημένη μονάδα αναδιάταξης 71
4.5 Απαλοιφή περιττών στοιχείων καθυστέρησης 72
4.6 Έλεγχος 73
check2.m 73
calc_fft_delay.m 74
4.7 Συμπεράσματα 75
4.8 Περαιτέρω βελτιστοποιήσεις 77
ΒΙΒΛΙΟΓΡΑΦΙΑ 78
ΠΑΡΑΡΤΗΜΑ 79
Α. Περιγραφή του μιγαδικού αθροιστή σε VHDL 81
dff.vhd 81
full_adder.vhd 82
Complex_full_adder.vhd 82
Complex_Adder.vhd 83
complex_adder_tb.vhd 86
Β. Περιγραφή του μιγαδικού αφαιρέτη σε VHDL 90
complex_subtracter.vhd 90
complex_subtracter_tb.vhd 93
C. Περιγραφή του μιγαδικού πολλαπλασιαστή σε VHDL 97
cmc1_cell.vhd 99
cmc2_cell.vhd 104
cmc3_cell.vhd 109
cmc4_cell.vhd 113
cfa1_cell.vhd 116
cfa2_cell.vhd 119
complex_multiplier.vhd 121
complex_multiplier_tb.vhd 129

D. Περιγραφή σε VHDL της αρχιτεκτονικής του βάσης-2 γρήγορου μετασχηματισμού Fourier με αποδεκατισμό στο πεδίο της συχνότητας *133*

D.1 Μιγαδικός αθροιστής **133**

D.2 Μιγαδικός αφαιρέτης **133**

D.3 Μιγαδικός πολλαπλασιαστής **133**
complex_multiplier.vhd 133
complex_multiplier_tb.vhd 141

D.4 Μιγαδικός καταχωρητής ολίσθησης **144**
Complex_shift_register.vhd 144
CSM_Tff.vhd 145
CSM_Counter.vhd 146
CSM_RAM.vhd 147
Complex_Shift_Mem.vhd 148

D.5 Μονάδα Butterfly **150**
tff.vhd 150
BU_addr_counter.vhd 151
BU_Rom.vhd 151
WN_Unit.vhd 153
butterfly_unit.vhd 156
Last_Butterfly_Unit.vhd 159
butterfly_unit_tb.vhd 161

D.6 Μονάδα αναδιάταξης **166**
SU_counter.vhd 166
SU_Mux.vhd 167
Shuffler.vhd 170
Shuffling_Unit.vhd 172
shuffling_unit_tb.vhd 174

D.7 Μονάδα βάσης-2 γρήγορου μετασχηματισμού Fourier **175**
FFT_Unit.vhd 175
FFT_Unit_TB.vhd 178

E. Αλλαγές στην περιγραφή σε VHDL του FFT με τον συνδυασμό μιγαδικού αφαιρέτη-πολλαπλασιαστή *182*

E.1 Μιγαδικός αθροιστής **182**
Complex_Adder2.vhd 182
Complex_Adder3.vhd 185
Complex_Adder4.vhd 188

E.2 Μιγαδικός αφαιρέτης **190**
Complex_Subtracter2.vhd 190
Complex_Subtracter3.vhd 194
Complex_Subtracter4.vhd 196

E.3 Μιγαδικός πολλαπλασιαστής **199**
complex_multiplier.vhd 199

E.4 Μιγαδικός καταχωρητής ολίσθησης **206**

E.5 Μονάδα Butterfly **206**
WN_Unit.vhd 206
First_Butterfly_Unit.vhd 210
butterfly_unit.vhd 215
Last_Butterfly_Unit.vhd 219

E.6 Μονάδα αναδιάταξης **221**
Mux.vhd 221
Shuffler.vhd 222

Shuffling_Unit.vhd 225

E.7 Μονάδα βάσης-2 γρήγορου μετασχηματισμού Fourier με συνδυασμό του μιγαδικού αφαιρέτη-πολλαπλασιαστή 227

FFT_Unit.vhd 227

FFT_Unit_TB.vhd 230

Πίνακας Σχημάτων

Σχήμα 1.1 Μορφή πλήρη αθροιστή 18
Σχήμα 1.2 Βασικό κύτταρο πλήρη μιγαδικού αθροιστή 19
Σχήμα 1.3 Συνεπτυγμένη μορφή κυττάρου του πλήρη μιγαδικού αθροιστή 19
Σχήμα 1.4 Μιγαδικός αθροιστής διάδοσης κρατουμένου 20
Σχήμα 1.5 Κύτταρο υπολογισμού του MSB του αθροίσματος μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2 21
Σχήμα 1.6 Εναλλακτικό κύτταρο υπολογισμού του MSB του αθροίσματος μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2 21
Σχήμα 1.7 Μιγαδικός αθροιστής διάδοσης κρατουμένου μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2 22
Σχήμα 1.8 Μετατροπή μιγαδικού αθροιστή διάδοσης κρατουμένου σε συνεχούς διοχέτευσης 22
Σχήμα 1.9 Μιγαδικός αθροιστής διάδοσης κρατουμένου συνεχούς διοχέτευσης 23
Σχήμα 1.10 Μορφή βασικού κυττάρου μιγαδικού αφαιρέτη 24
Σχήμα 1.11 Κύτταρο πλήρη μιγαδικού αφαιρέτη χρησιμοποιώντας έναν πλήρη μιγαδικό αθροιστή 25
Σχήμα 1.12 Συνεπτυγμένη μορφή κυττάρου πλήρη μιγαδικού αφαιρέτη χρησιμοποιώντας έναν πλήρη μιγαδικό αθροιστή 25
Σχήμα 1.13 Μορφή μιγαδικού αφαιρέτη διάδοσης κρατουμένου 25
Σχήμα 1.14 Μιγαδικός αφαιρέτης διάδοσης κρατουμένου 26
Σχήμα 1.15 Μιγαδικός αθροιστής διάδοσης κρατουμένου μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2 27
Σχήμα 1.16 Μετατροπή μιγαδικού αθροιστή διάδοσης κρατουμένου, μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2, σε συνεχούς διοχέτευσης 27
Σχήμα 1.17 Μιγαδικός αθροιστής διάδοσης κρατουμένου, μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2, συνεχούς διοχέτευσης 28
Σχήμα 1.18 Μορφή μιγαδικού πολλαπλασιαστή με αθροιστή αποθήκευσης κρατουμένου 29
Σχήμα 1.19 1ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 30
Σχήμα 1.20 Συνεπτυγμένη μορφή 1ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή 30
Σχήμα 1.21 2ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 31
Σχήμα 1.22 Συνεπτυγμένη μορφή 2ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή 31
Σχήμα 1.23 3ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 33
Σχήμα 1.24 Συνεπτυγμένη μορφή 3ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή 33
Σχήμα 1.25 4ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 34
Σχήμα 1.26 Συνεπτυγμένη μορφή 4ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή 34
Σχήμα 1.27 Πρώτο άθροισμα των κρατουμένων εξόδου των κυττάρων του μιγαδικού πολλαπλασιαστή 34
Σχήμα 1.28 Μιγαδικός πλήρης αθροιστής 1ου τύπου 35
Σχήμα 1.29 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 1ου τύπου 35
Σχήμα 1.30 Μιγαδικός πλήρης αθροιστής 2ου τύπου 36
Σχήμα 1.31 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 2ου τύπου 36
Σχήμα 1.32 Μιγαδικός αθροιστής με αποθήκευση κρατουμένου για την άθροιση των κρατουμένων εξόδου των κυττάρων του μιγαδικού πολλαπλασιαστή 37
Σχήμα 1.33 Μιγαδικός πλήρης αθροιστής 3ου τύπου 38
Σχήμα 1.34 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 3ου τύπου 38
Σχήμα 1.35 Μιγαδικός πλήρης αθροιστής 4ου τύπου 38
Σχήμα 1.36 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 4ου τύπου 38
Σχήμα 1.37 Μιγαδικός Πολλαπλασιαστής συνεχούς διοχέτευσης 39
Σχήμα 1.38 Παράσταση των D flip-flop που χρησιμοποιούνται στα κύτταρα του μιγαδικού πολλαπλασιαστή 40
Σχήμα 1.39 1ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 40
Σχήμα 1.40 2ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 40
Σχήμα 1.41 3ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 41
Σχήμα 1.42 4ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή 41
Σχήμα 1.43 1ος τύπος μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή 42
Σχήμα 1.44 2ος τύπος μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή 42
Σχήμα 2.1 Αναγωγή του υπολογισμού βάσης-2 γρήγορου μετασχηματισμού Fourier N σημείων στον υπολογισμό δύο ίδιων μετασχηματισμών Fourier των N/2 σημείων 44
Σχήμα 2.2 Διαδικασία butterfly 45
Σχήμα 2.3 Υπολογισμός του βάσης-2 γρήγορου μετασχηματισμού Fourier 8 σημείων 45
Σχήμα 2.4 Οι έξοδοι κάθε σταδίου butterfly με δείκτες εκφρασμένους σε δυαδική μορφή κατά τον υπολογισμό του βάσης-2 γρήγορου μετασχηματισμού Fourier 8 σημείων 46

Σχήμα 3.1 Μορφή υλοποίησης με βάση των αλγόριθμο βάσης-2 γρήγορου μετασχηματισμού Fourier με αποδεκατισμό στο πεδίο της συχνότητας 47
Σχήμα 3.2 Μορφή μονάδας butterfly 48
Σχήμα 3.3 Μιγαδικός πολλαπλασιαστής που θα χρησιμοποιηθεί μέσα στην μονάδα butterfly 50
Σχήμα 3.4 Μονάδα ROM μέσα στην μονάδα butterfly που τροφοδοτεί τον μιγαδικό πολλαπλασιαστή με τους παράγοντες αναστροφής (twiddle factors) 51
Σχήμα 3.5 Μορφή μονάδας αναδιάταξης 52
Σχήμα 3.6 Μονάδα αναδιάταξης 53
Σχήμα 3.7 Μιγαδικός καταχωρητής ολίσθησης 53
Σχήμα 3.8 Μνήμη RAM με 2 θύρες σαν καταχωρητής ολίσθησης 54
Σχήμα 3.9 Εσωτερικό μνήμης RAM 54
Σχήμα 3.10 Σύγχρονος Δυαδικός μετρητής 54
Σχήμα 3.11 Το φάσμα του μετασχηματισμού Fourier των δεδομένων εισόδου όπως το υπολόγισαν οι συναρτήσεις του MATLAB 57
Σχήμα 3.12 Το φάσμα του μετασχηματισμού Fourier των δεδομένων εξόδου της συγκεκριμένης υλοποίησης στο test bench 57
Σχήμα 3.13 Γραφική αναπαράσταση των δεδομένων εισόδου στην συγκεκριμένη υλοποίηση του FFT στο test bench 58
Σχήμα 3.14 Γραφική παράσταση του αντίστροφου μετασχηματισμού Fourier των δεδομένων εξόδου της συγκεκριμένης υλοποίησης στο test bench 58
Σχήμα 3.15 Γραφική παράσταση του αντίστροφου μετασχηματισμού Fourier που έγινε στα δεδομένα του γρήγορου μετασχηματισμού Fourier των δεδομένων εισόδου 59
Σχήμα 3.16 Γραφική παράσταση του μέτρου της διαφοράς των δύο υπολογισμών, του γρήγορου μετασχηματισμού Fourier των δεδομένων εισόδου, απ' την υλοποίηση και από το MATLAB 59
Σχήμα 3.17 Σηματοθορυβικός λόγος σε decibels της υλοποίησης του FFT, για διάφορες τιμές του μήκους σε bit των δεδομένων εισόδου, και του μήκους σε bit των παραγόντων αναστροφής 60
Σχήμα 3.18 Συνοπτική μορφή της αρχιτεκτονικής του FFT 61
Σχήμα 4.1 Μορφή μονάδας butterfly με συνδυασμό μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή 63
Σχήμα 4.2 Συνδυασμός μιγαδικού αφαιρέτη με μιγαδικό πολλαπλασιαστή, χρησιμοποιώντας λιγότερα στοιχεία καθυστέρησης, σε κάθε μονάδα butterfly 64
Σχήμα 4.3 Συνδυασμός μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή, στην 1η μονάδα butterfly της τροποποιημένης υλοποίησης 66
Σχήμα 4.4 Είσοδος και έξοδος δεδομένων απ' την μεριά του μιγαδικού αθροιστή στη 1η τροποποιημένη μονάδα butterfly 67
Σχήμα 4.5 Συνδυασμός μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή στις ενδιάμεσες τροποποιημένες μονάδες butterfly 68
Σχήμα 4.6 Είσοδος και έξοδος απ' την μεριά του μιγαδικού αθροιστή των ενδιάμεσων τροποποιημένων μονάδων butterfly 69
Σχήμα 4.7 Μορφή της τελευταίας μονάδας butterfly στην τροποποιημένη υλοποίηση 70
Σχήμα 4.8 Μονάδα αναδιάταξης της τροποποιημένης υλοποίησης 71
Σχήμα 4.9 Απαλοιφή περιττών στοιχείων καθυστέρησης 72
Σχήμα 4.10 Συνοπτική μορφή της τροποποιημένης αρχιτεκτονικής του FFT 75
Σχήμα A.1 Περιγραφή του μιγαδικού πλήρη αθροιστή (CFA) στο block diagram αρχείο complex_full_adder.bde 82
Σχήμα C.1 Η περιγραφή των 1ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc1_cell.bde 98
Σχήμα C.2 Η περιγραφή των 2ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc2_cell.bde 103
Σχήμα C.3 Η περιγραφή των 3ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc3_cell.bde 108
Σχήμα C.4 Η περιγραφή των 4ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc4_cell.bde 112
Σχήμα C.5 Περιγραφεί του 1ου τύπου μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή και έχει σχεδιαστεί στο block diagram αρχείο cfa1_cell.bde 116
Σχήμα C.6 Περιγραφή του 2ου τύπου μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή και έχει σχεδιαστεί στο block diagram αρχείο cfa2_cell.bde 119
Σχήμα D.1 Σχεδιάγραμμα μονάδας μνήμης RAM ως μονάδα καθυστέρησης που βρίσκετε στο αρχείο Complex_Shift_Mem.bde 148
Σχήμα D.2 Σχεδιάγραμμα της μονάδας butterfly που βρίσκετε στο αρχείο butterfly_unit.bde 155
Σχήμα D.3 Σχεδιάγραμμα της τελευταίας μονάδας butterfly που βρίσκετε στο αρχείο last_butterfly_unit.bde 159
Σχήμα D.4 Μονάδα αναδιάταξης χωρίς τον μετρητή που οδηγεί τους πολυπλέκτες για σωστό συγχρονισμό 169
Σχήμα D.5 Μονάδα αναδιάταξης όπως σχεδιάστηκε στο αρχείο Shuffling_Unit.bde 172

Σχήμα Ε.1 Σχεδιάγραμμα της πρώτης μονάδας butterfly, της τροποποιημένης αρχιτεκτονικής, που βρίσκετε στο αρχείο First_Butterfly_Unit.bde 209
Σχήμα Ε.2 Σχεδιάγραμμα της τροποποιημένης μονάδας butterfly, που χρησιμοποιείται στις ενδιάμεσες μονάδες butterfly, και βρίσκετε στο αρχείο butterfly_unit.bde 214
Σχήμα Ε.3 Σχεδιάγραμμα της τροποποιημένης μονάδας butterfly, που χρησιμοποιείται στην τελευταία μονάδας butterfly, και βρίσκεται στο αρχείο Last_Butterfly_Unit.vhd 219
Σχήμα Ε.4 Τροποποιημένη μονάδα αναδιάταξης χωρίς τον μετρητή που οδηγεί τους πολυπλέκτες για σωστό συγχρονισμό και βρίσκεται στο αρχείο Shuffler.bde 222
Σχήμα Ε.5 Τροποποιημένη μονάδα αναδιάταξης όπως σχεδιάστηκε στο αρχείο Shuffling_Unit.bde 225

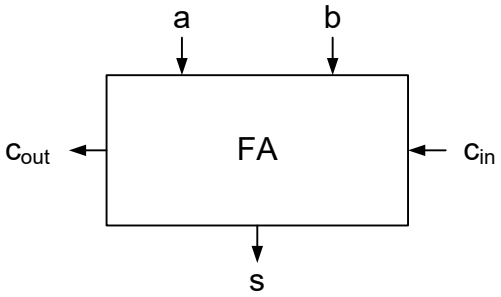
1 ΜΙΓΑΔΙΚΑ ΑΡΙΘΜΗΤΙΚΑ ΚΥΚΛΩΜΑΤΑ

Επειδή θα χρησιμοποιηθούν κατά κύριο λόγο μιγαδικοί αριθμοί, θα πρέπει να αναλυθούν τα κυκλώματα που υλοποιούν τις βασικές αριθμητικές πράξεις με αυτούς. Ακολουθεί η περιγραφή τους σε καθένα ξεχωριστά.

1.1 Υλοποίηση Μιγαδικού Αθροιστή (Complex Adder)

1.1.1 Μιγαδικός Πλήρης Αθροιστής (Complex Full Adder ή CFA)

Για την δημιουργία αριθμητικών κυκλωμάτων, τα οποία κάνουν πράξεις χρησιμοποιώντας ακεραίους αριθμούς, βασική δομική μονάδα αποτελεί ο πλήρης αθροιστής (Full Adder ή FA) που φαίνεται στο Σχήμα 1.1 όπου a και b τα bits των προστιθέμενων αριθμών , c_{in} και c_{out} τα κρατούμενα εισόδου και εξόδου αντίστοιχα και s το άθροισμα των δύο bits.



Σχήμα 1.1 Μορφή πλήρη αθροιστή

Ας υποθέσουμε αρχικά ότι οι μιγαδικοί αριθμοί που θέλουμε να προσθέσουμε έχουνε μη προσημασμένους αριθμούς σε πραγματικό και φανταστικό μέρος. Τότε αν X και Y οι δύο μιγαδικοί αριθμοί αυτοί θα είναι της μορφής:

$$X = X_r + jX_i \text{ όπου } X_r = \sum_{k=0}^{N-1} x_{rk} 2^k \text{ και } X_i = \sum_{k=0}^{N-1} x_{ik} 2^k$$

και

$$Y = Y_r + jY_i \text{ όπου } Y_r = \sum_{k=0}^{N-1} y_{rk} 2^k \text{ και } Y_i = \sum_{k=0}^{N-1} y_{ik} 2^k$$

Οπότε για το άθροισμά τους θα έχουμε:

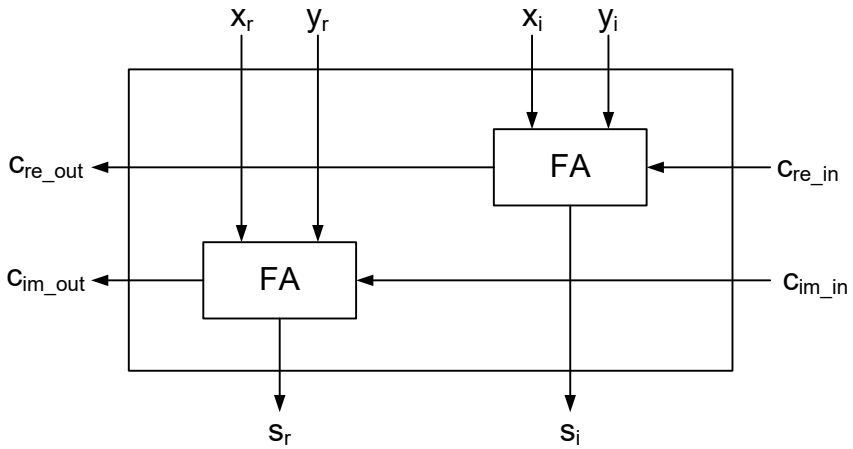
$$X + Y = X_r + Y_r + j(X_i + Y_i) \Leftrightarrow$$
$$X + Y = \sum_{k=0}^{N-1} (x_{rk} + y_{rk}) 2^k + j \sum_{k=0}^{N-1} (x_{ik} + y_{ik}) 2^k \Leftrightarrow$$
$$X + Y = \sum_{k=0}^{N-1} [(x_{rk} + y_{rk}) + j(x_{ik} + y_{ik})] 2^k \Leftrightarrow$$

$$X + Y = \sum_{k=0}^{N-1} (s_{rk} + js_{ik}) 2^k \Leftrightarrow$$

$$X + Y = \sum_{k=0}^{N-1} s_k 2^k = S$$

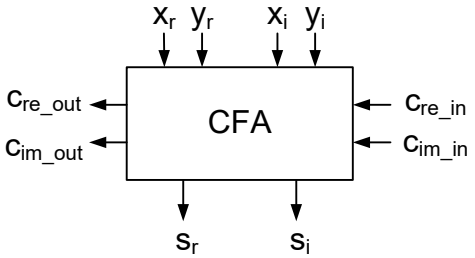
όπου $s_k = s_{rk} + js_{ik} = (x_{rk} + y_{rk}) + j(x_{ik} + y_{ik})$ για κάθε k=0…N-1.

Επομένως βλέπουμε ότι η πρόσθεση δύο μιγαδικών αριθμών προκύπτει από την άθροιση των ίδιας τάξης μιγαδικών bits και άρα η όλοι διαδικασία μπορεί να γίνει χρησιμοποιώντας το κύτταρο που φαίνεται στο Σχήμα 1.2 όπου x_r+jx_i και y_r+jy_i τα μιγαδικά bits που θα προστεθούν, $c_{re_in}+jc_{im_in}$ και $c_{re_out}+jc_{im_out}$ τα μιγαδικά κρατούμενα εισόδου και εξόδου αντίστοιχα και s_r+js_i το μιγαδικό άθροισμα των bits.



Σχήμα 1.2 Βασικό κύτταρο πλήρη μιγαδικού αθροιστή

Όπως είναι φανερό, το παραπάνω κύτταρο είναι ένας πλήρης μιγαδικός αθροιστής (Complex Full Adder ή CFA) ο οποίος φαίνεται, σε μια πιο συνεπτυγμένη μορφή, στο Σχήμα 1.3.

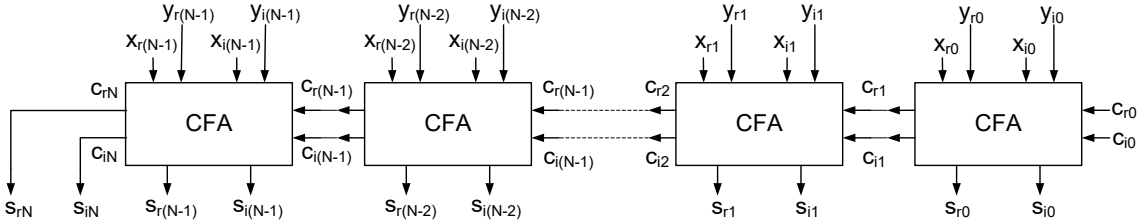


Σχήμα 1.3 Συνεπτυγμένη μορφή κυττάρου του πλήρη μιγαδικού αθροιστή

Ο παραπάνω πλήρης μιγαδικός αθροιστής θα αποτελέσει την βασική δομική μονάδα για την δημιουργία μιγαδικών αριθμητικών κυκλωμάτων.

1.1.2 Μιγαδικός Αθροιστής Διάδοσης Κρατουμένου

Με την υπόθεση ότι οι μιγαδικοί αριθμοί που θέλουμε να προσθέσουμε έχουνε μη προσημασμένους αριθμούς σε πραγματικό και φανταστικό μέρος, μπορούμε να συνδέσουμε πλήρης μιγαδικούς αθροιστές σε σειρά όπως φαίνεται στο Σχήμα 1.4.



Σχήμα 1.4 Μιγαδικός αθροιστής διάδοσης κρατουμένου

Αυτός είναι και ο πιο απλός παράλληλος μιγαδικός αθροιστής ο οποίος δέχεται παράλληλα τους μιγαδικούς αριθμούς X και Y, οι οποίοι θα προστεθούν, δίνοντας ως αποτέλεσμα τον μιγαδικό αριθμό S. Τα $c_k=c_{rk}+jc_{ik}$ όπου k=0...N+1 αποτελούν τα μιγαδικά κρατούμενα απ' τις προηγούμενες βαθμίδες. Φυσικά το κρατούμενο εισόδου c_0 πρέπει να είναι 0, δηλαδή $c_{r0}=c_{i0}=0$.

1.1.3 Άθροιση αριθμών με μορφή συμπληρώματος ως προς 2

Ας υποθέσουμε ότι έχουμε δύο μιγαδικούς αριθμούς X και Y τους οποίους θέλουμε να προσθέσουμε. Αν τους παραστήσουμε σε μορφή συμπληρώματος ως προς 2 τότε αυτοί θα έχουν την ακόλουθη μορφή:

$$X = X_r + jX_i \text{ όπου } X_r = -x_{r(N-1)}2^{N-1} + \sum_{k=0}^{N-2} x_{rk}2^k \text{ και } X_i = -x_{i(N-1)}2^{N-1} + \sum_{k=0}^{N-2} x_{ik}2^k$$

και

$$Y = Y_r + jY_i \text{ όπου } Y_r = -y_{r(N-1)}2^{N-1} + \sum_{k=0}^{N-2} y_{rk}2^k \text{ και } Y_i = -y_{i(N-1)}2^{N-1} + \sum_{k=0}^{N-2} y_{ik}2^k$$

Οπότε για το άθροισμα θα έχουμε:

$$X + Y = X_r + Y_r + j\left(X_i + Y_i\right) \Leftrightarrow$$

$$X + Y = -\left(x_{r(N-1)} + y_{r(N-1)}\right)2^{N-1} + \sum_{k=0}^{N-2}\left(x_{rk} + y_{rk}\right)2^k + j\left[-\left(x_{i(N-1)} + y_{i(N-1)}\right)2^{N-1} + \sum_{k=0}^{N-2}\left(x_{ik} + y_{ik}\right)2^k\right] \Leftrightarrow$$

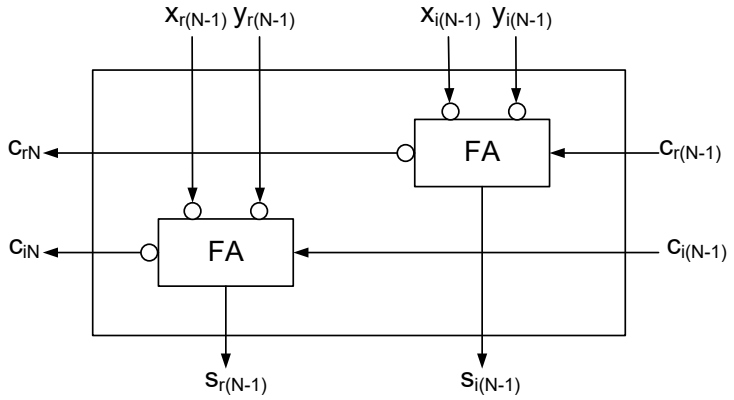
$$X + Y = -\left[\left(x_{r(N-1)} + y_{r(N-1)}\right) + j\left(x_{i(N-1)} + y_{i(N-1)}\right)\right]2^{N-1} + \sum_{k=0}^{N-2}\left[\left(x_{rk} + y_{rk}\right) + j\left(x_{ik} + y_{ik}\right)\right]2^k \Leftrightarrow$$

$$X + Y = -\left(s_{r(N-1)} + js_{i(N-1)}\right)2^{N-1} + \sum_{k=0}^{N-2}\left(s_{rk} + js_{ik}\right)2^k$$

$$X + Y = -s_{N-1}2^{N-1} + \sum_{k=0}^{N-2} s_k 2^k$$

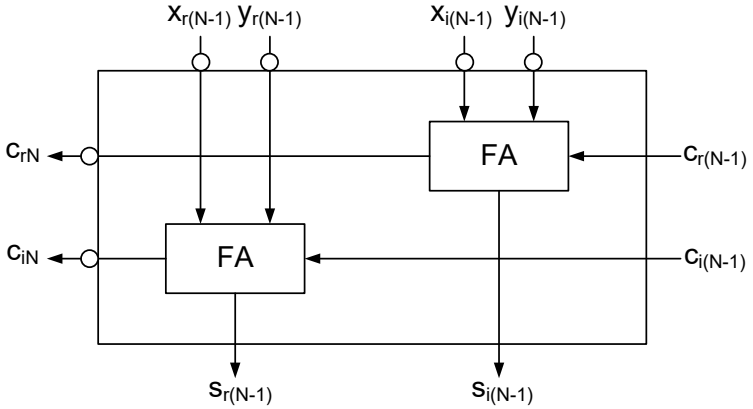
όπου $s_k = s_{rk} + js_{ik} = \left(x_{rk} + y_{rk}\right) + j\left(x_{ik} + y_{ik}\right)$ για κάθε k=0…N-1.

Επομένως παρατηρούμε ότι το μοναδικό πρόβλημα προκύπτει στα bits αρνητικής αξίας που είναι η πρόσθεση των $x_{r(N-1)}$ και $y_{r(N-1)}$ καθώς και η πρόσθεση των $x_{i(N-1)}$ και $y_{i(N-1)}$. Όπως γνωρίζουμε όμως, για να αθροίσουμε σε έναν πλήρη αθροιστή σήματα αρνητικής αξίας, το μόνο που πρέπει να κάνουμε είναι να προσθέσουμε σε αυτά αντιστροφείς. Τότε όμως ο συγκεκριμένως πλήρης μιγαδικός αθροιστής θα έχει πλήρης αθροιστές με τις δύο εισόδους τους αρνητικά bit, και επομένως θα πρέπει να θεωρήσουμε και τα κρατούμενα εξόδου τους αρνητικούς αριθμούς. Έτσι το κύτταρο της N-1 τάξης θα είναι όπως φαίνεται στο Σχήμα 1.5.



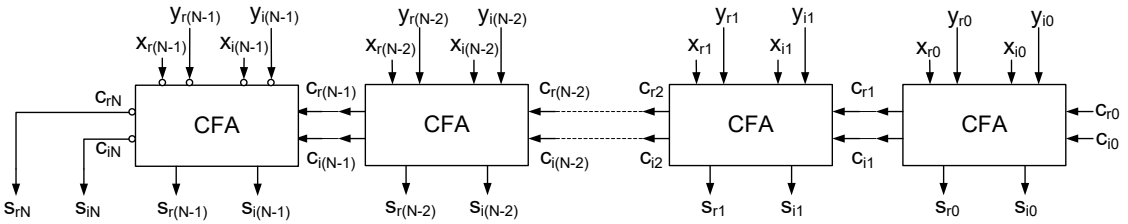
Σχήμα 1.5 Κύτταρο υπολογισμού του MSB του αθροίσματος μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2

Αν βγάλουμε τους αντιστροφείς έξω απ' το κύτταρο του μιγαδικού πλήρη αθροιστή θα πάρουμε το κύτταρο που φαίνεται στο Σχήμα 1.6.



Σχήμα 1.6 Εναλλακτικό κύτταρο υπολογισμού του MSB του αθροίσματος μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2

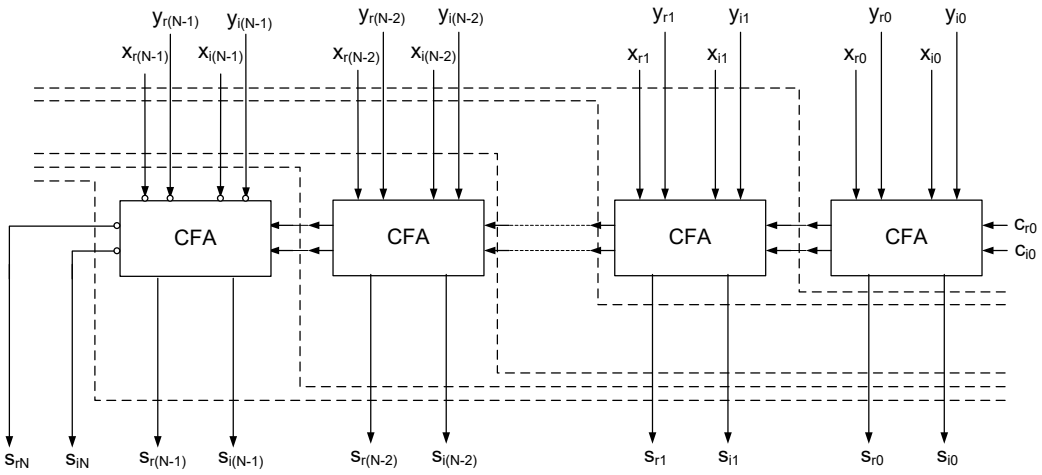
Επομένως ο ζητούμενος μιγαδικός αθροιστής διάδοσης κρατουμένου ο οποίος αθροίζει μιγαδικούς αριθμούς με εκφρασμένο πραγματικό και φανταστικό μέρος σε μορφή συμπληρώματος ως προς 2 φαίνεται στο Σχήμα 1.7.



Σχήμα 1.7 Μιγαδικός αθροιστής διάδοσης κρατουμένου μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2

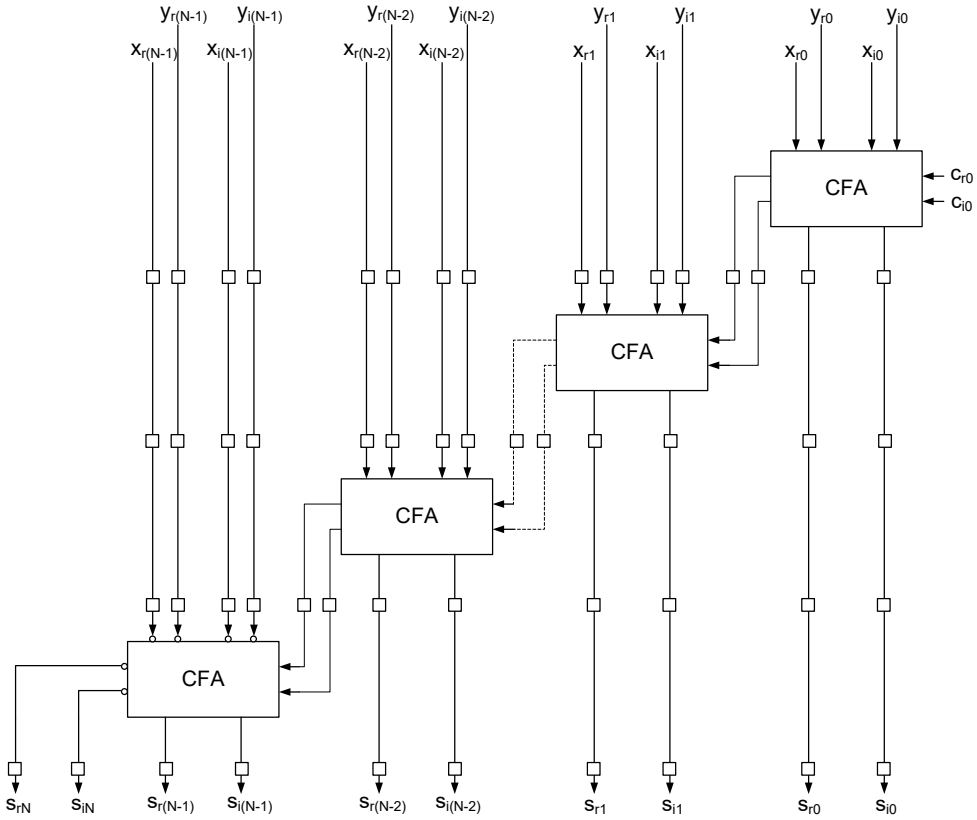
1.1.4 Μετατροπή του μιγαδικού αθροιστή διάδοσης κρατουμένου σε συνεχούς διοχέτευσης (PIPELINING)

Για να μετατρέψουμε το κύκλωμα σε συνεχούς διοχέτευσης, «κόβουμε» όλα τα σήματα της ίδιας φοράς με ευθείες ώστε να χωρίσουμε τις βαθμίδες που ενώνουν όπως φαίνεται στο Σχήμα 1.8.



Σχήμα 1.8 Μετατροπή μιγαδικού αθροιστή διάδοσης κρατουμένου σε συνεχούς διοχέτευσης

Επομένως αν βάλουμε στα σημεία τομής D flip-flops το κύκλωμα μετατρέπετε σε συνεχούς διοχέτευσης και θα έχει την μορφή που παρουσιάζεται στο Σχήμα 1.9.



Σχήμα 1.9 Μιγαδικός αθροιστής διάδοσης κρατουμένου συνεχούς διοχέτευσης

1.2 Υλοποίηση Μιγαδικού Αφαιρέτη

1.2.1 Βασικό κύτταρο του μιγαδικού αφαιρέτη

Ας υποθέσουμε αρχικά ότι οι μιγαδικοί αριθμοί που θέλουμε να αφαιρέσουμε έχουνε μη προσημασμένους αριθμούς σε πραγματικό και φανταστικό μέρος. Τότε αν X και Y οι δύο μιγαδικοί αριθμοί αυτοί θα είναι της μορφής:

$$X = X_r + jX_i \text{ όπου } X_r = \sum_{k=0}^{N-1} x_{rk} 2^k \text{ και } X_i = \sum_{k=0}^{N-1} x_{ik} 2^k$$

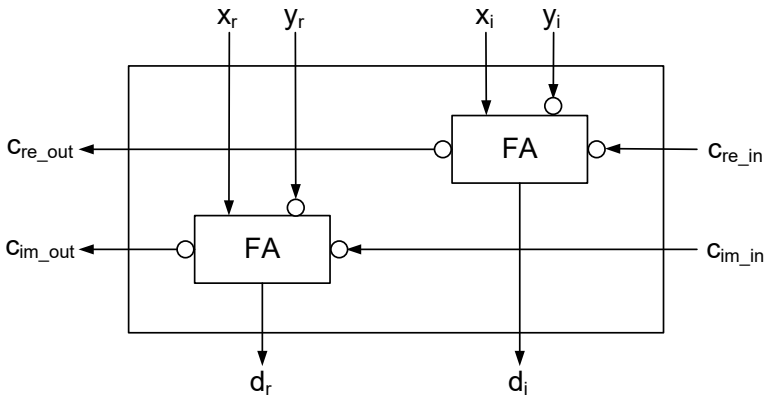
και

$$Y = Y_r + jY_i \text{ όπου } Y_r = \sum_{k=0}^{N-1} y_{rk} 2^k \text{ και } Y_i = \sum_{k=0}^{N-1} y_{ik} 2^k$$

Οπότε για την αφαίρεση θα έχουμε:

$$X - Y = X_r - Y_r + j(X_i - Y_i) \Leftrightarrow$$
$$X - Y = \sum_{k=0}^{N-1} (x_{rk} - y_{rk}) 2^k + j \sum_{k=0}^{N-1} (x_{ik} - y_{ik}) 2^k \Leftrightarrow$$
$$X - Y = \sum_{k=0}^{N-1} [(x_{rk} - y_{rk}) + j(x_{ik} - y_{ik})] 2^k \Leftrightarrow$$
$$X - Y = \sum_{k=0}^{N-1} (d_{rk} + jd_{ik}) 2^k \Leftrightarrow$$
$$X - Y = \sum_{k=0}^{N-1} d_k 2^k = D$$

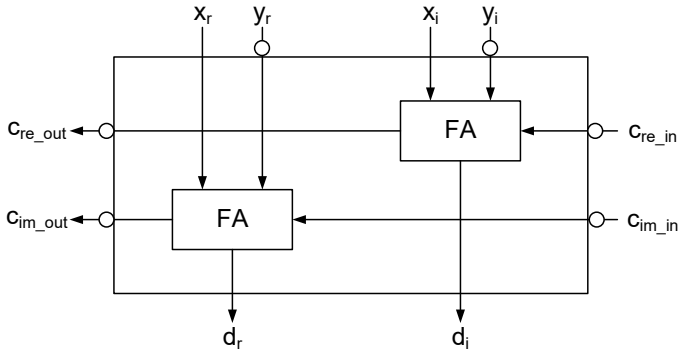
όπου $d_k = d_{rk} + jd_{ik} = (x_{rk} - y_{rk}) + j(x_{ik} - y_{ik})$ για κάθε k=0…N-1.



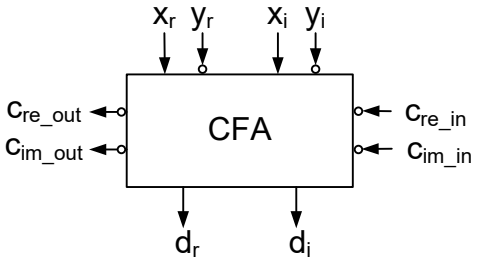
Σχήμα 1.10 Μορφή βασικού κυττάρου μιγαδικού αφαιρέτη

Βλέπουμε λοιπόν ότι η αφαίρεση δύο μιγαδικών αριθμών, με μη προσημασμένους αριθμούς σε πραγματικό και φανταστικό μέρος, εφαρμόζεται με την αφαίρεση των bit ίδιας τάξης σε πραγματικό και φανταστικό μέρος. Έτσι πρέπει να χρησιμοποιήσουμε δύο πλήρης αθροιστές, έναν για το πραγματικό και έναν για το φανταστικό μέρος, και να βάλουμε αντιστροφείς στα σήματα με αρνητικό πρόσημο. Τα σήματα αυτά για να γίνετε σωστά η πράξη πρέπει να είναι τα bit του αφαιρετέου, καθώς και τα bits των κρατουμένων εισόδου και εξόδου, για κάθε πλήρη αθροιστή όπως φαίνετε στο Σχήμα 1.10.

Το ίδιο αποτέλεσμα μπορούμε να έχουμε χρησιμοποιώντας τον μιγαδικό πλήρη αθροιστή όπως φαίνεται στο Σχήμα 1.11 και σε μια πιο συνεπτυγμένη μορφή στο Σχήμα 1.12.



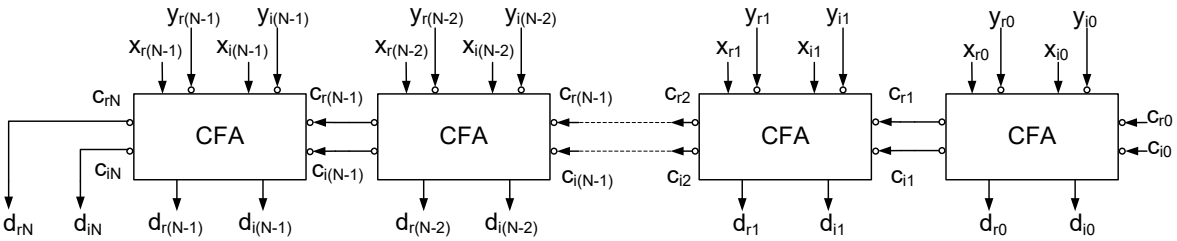
Σχήμα 1.11 Κύτταρο πλήρη μιγαδικού αφαιρέτη χρησιμοποιώντας έναν πλήρη μιγαδικό αθροιστή



Σχήμα 1.12 Συνεπτυγμένη μορφή κυττάρου πλήρη μιγαδικού αφαιρέτη χρησιμοποιώντας έναν πλήρη μιγαδικό αθροιστή

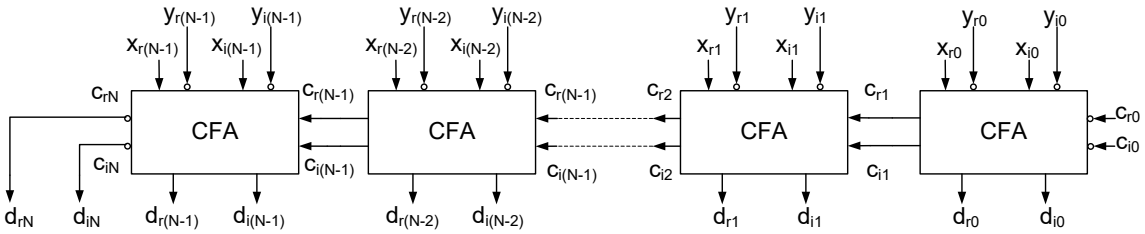
1.2.2 Μιγαδικός Αφαιρέτης Διάδοσης Κρατουμένου

Με την υπόθεση ότι οι μιγαδικοί αριθμοί που θέλουμε να προσθέσουμε έχουνε μη προσημασμένους αριθμούς σε πραγματικό και φανταστικό μέρος, μπορούμε να συνδέσουμε πλήρης μιγαδικούς αθροιστές σε σειρά όπως φαίνεται στο Σχήμα 1.13.



Σχήμα 1.13 Μορφή μιγαδικού αφαιρέτη διάδοσης κρατουμένου

Επειδή οι αντιστροφείς των κρατουμένων εισόδου-εξόδου αναιρούνται έχουμε τον μιγαδικό αφαιρέτη που φαίνεται στο Σχήμα 1.14.



Σχήμα 1.14 Μιγαδικός αφαιρέτης διάδοσης κρατουμένου

Αυτός είναι και ο πιο απλός παράλληλος μιγαδικός αφαιρέτης ο οποίος δέχεται παράλληλα τους μιγαδικούς αριθμούς X και Y με τους οποίους θα γίνει η αφαίρεση και δίνουν ως αποτέλεσμα τον μιγαδικό αριθμό D. Τα $c_k=c_{rk}+jc_{ik}$ όπου k=0...N+1 αποτελούν τα μιγαδικά κρατούμενα απ' τις προηγούμενες βαθμίδες και τα οποία έχουν αρνητικό πρόσημο. Φυσικά το κρατούμενο εισόδου c_0 πρέπει να είναι 0, δηλαδή $c_{r0}=c_{i0}=0$ αλλά μπορούμε και να παραλείψουμε τους αντιστροφείς στα κρατούμενα εισόδου και να θέσουμε το c_0 ίσο με 1+j δηλαδή $c_{r0}=c_{i0}=1$.

1.2.3 Αφαίρεση αριθμών σε μορφή συμπληρώματος ως προς 2

Ας υποθέσουμε ότι έχουμε δύο μιγαδικούς αριθμούς X και Y τους οποίους θέλουμε να προσθέσουμε. Αν τους παραστήσουμε σε μορφή συμπληρώματος ως προς 2 τότε αυτοί θα έχουν την ακόλουθη μορφή:

$$X = X_r + jX_i \text{ όπου } X_r = -x_{r(N-1)}2^{N-1} + \sum_{k=0}^{N-2} x_{rk}2^k \text{ και } X_i = -x_{i(N-1)}2^{N-1} + \sum_{k=0}^{N-2} x_{ik}2^k$$

και

$$Y = Y_r + jY_i \text{ όπου } Y_r = -y_{r(N-1)}2^{N-1} + \sum_{k=0}^{N-2} y_{rk}2^k \text{ και } Y_i = -y_{i(N-1)}2^{N-1} + \sum_{k=0}^{N-2} y_{ik}2^k$$

Οπότε για την αφαίρεση θα έχουμε:

$$X - Y = X_r - Y_r + j(X_i - Y_i) \Leftrightarrow$$

$$X - Y = X_r + \overline{Y}_r + 1 + j(X_i + \overline{Y}_i + 1) \Leftrightarrow$$

$$X - Y = -(x_{r(N-1)} + \overline{y}_{r(N-1)})2^{N-1} + \sum_{k=0}^{N-2}(x_{rk} + \overline{y}_{rk})2^k + 1 +$$

$$+ j\left[-(x_{i(N-1)} + \overline{y}_{i(N-1)})2^{N-1} + \sum_{k=0}^{N-2}(x_{ik} + \overline{y}_{ik})2^k + 1\right] \Leftrightarrow$$

$$X - Y = -\left[(x_{r(N-1)} + \overline{y}_{r(N-1)}) + j(x_{i(N-1)} + \overline{y}_{i(N-1)})\right]2^{N-1} +$$

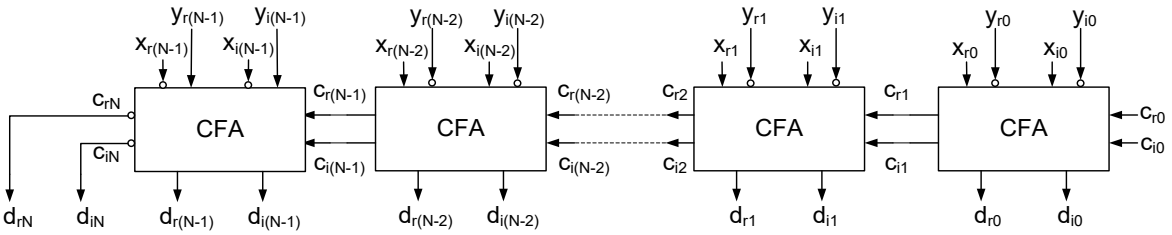
$$+ \sum_{k=0}^{N-2}\left[(x_{rk} + \overline{y}_{rk}) + j(x_{ik} + \overline{y}_{ik})\right]2^k + (1 + j) \Leftrightarrow$$

$$X - Y = -(d_{r(N-1)} + jd_{i(N-1)})2^{N-1} + \sum_{k=0}^{N-2}(d_{rk} + jd_{ik})2^k + (1 + j)$$

$$X - Y = -d_{N-1}2^{N-1} + \sum_{k=0}^{N-2} d_k 2^k + (1 + j) = D$$

όπου $d_k = d_{rk} + jd_{ik} = (x_{rk} + \overline{y}_{rk}) + j(x_{ik} + \overline{y}_{ik})$ για κάθε k=0…N-1.

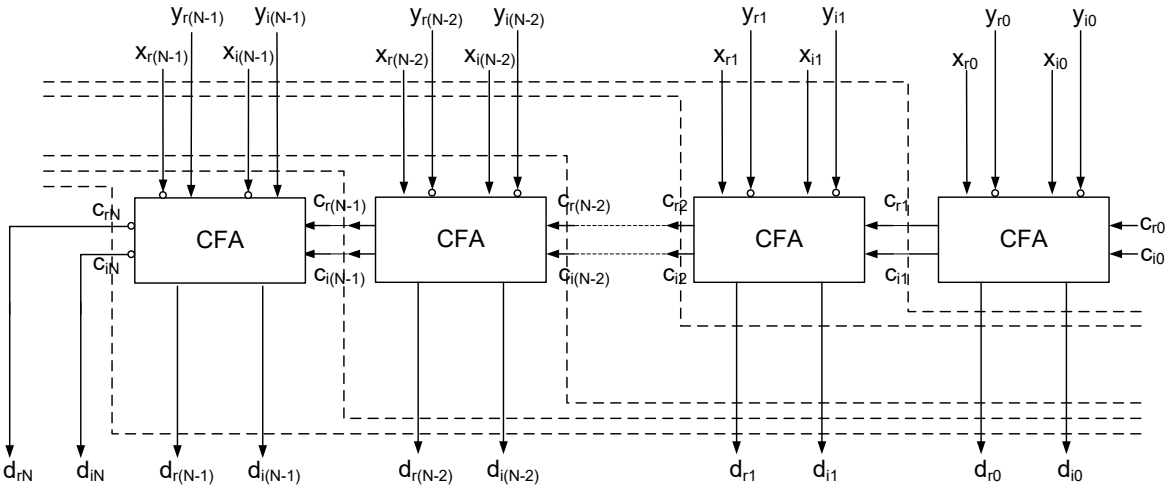
Επομένως μπορούμε να χρησιμοποιήσουμε τον μιγαδικός αθροιστής διάδοσης κρατουμένου ο οποίος αθροίζει μιγαδικούς αριθμούς με εκφρασμένο πραγματικό και φανταστικό μέρος σε μορφή συμπληρώματος ως προς 2, στον οποίο θα έχουμε ως είσοδο το συμπλήρωμα του αφαιρετέου και το κρατούμενο εισόδου c_0 πρέπει να έχει την τιμή 1+j, δηλαδή $c_{r0}=c_{i0}=1$, και ο ζητούμενος μιγαδικός αφαιρέτης θα έχει την μορφή που φαίνεται στο Σχήμα 1.15.



Σχήμα 1.15 Μιγαδικός αθροιστής διάδοσης κρατουμένου μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2

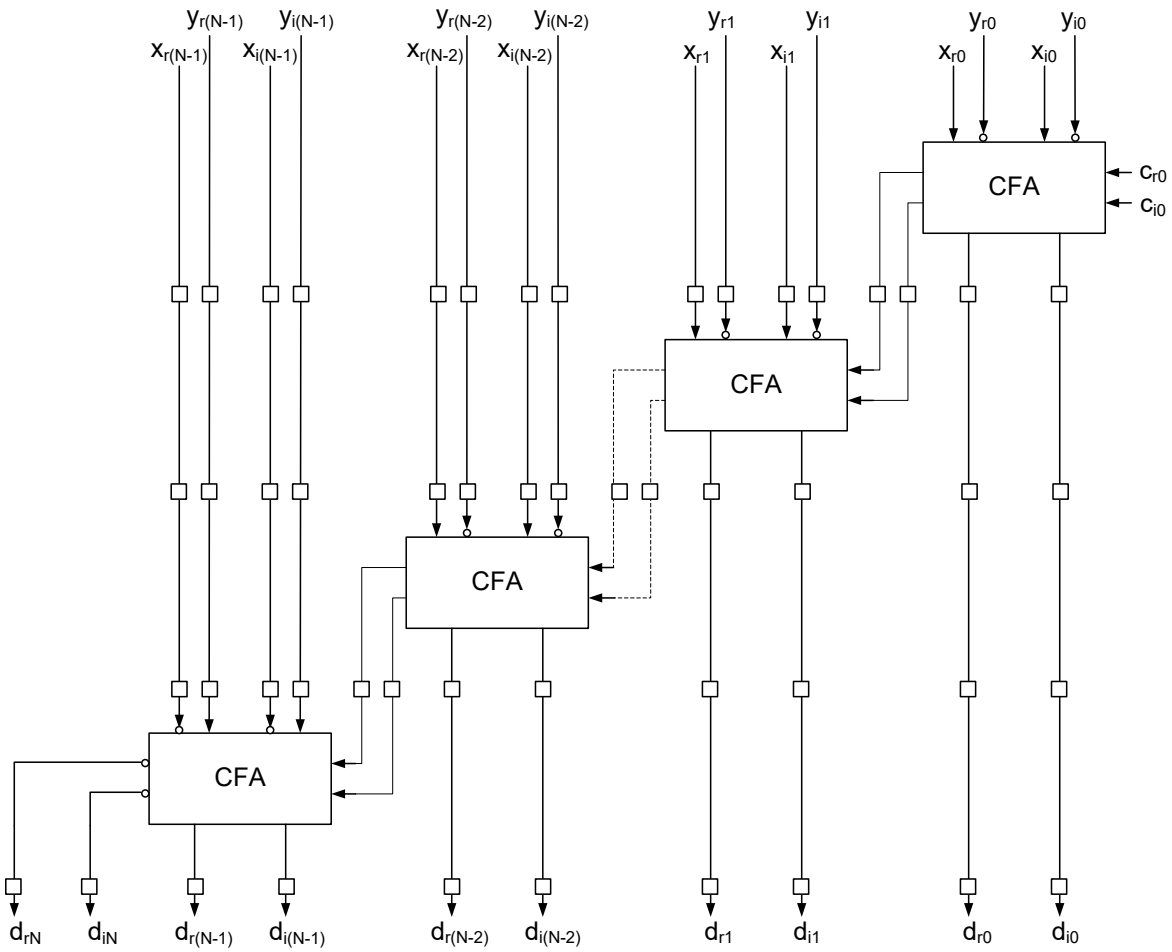
1.2.4 Μετατροπή του μιγαδικού αφαιρέτη διάδοσης κρατουμένου σε συνεχούς διοχέτευσης (PIPELINING)

Για να μετατρέψουμε το κύκλωμα σε συνεχούς διοχέτευσης, «κόβουμε» όλα τα σήματα της ίδιας φοράς με ευθείες ώστε να χωρίσουμε τις βαθμίδες που ενώνουν όπως φαίνεται στο Σχήμα 1.16.



Σχήμα 1.16 Μετατροπή μιγαδικού αθροιστή διάδοσης κρατουμένου, μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2, σε συνεχούς διοχέτευσης

Επομένως αν βάλουμε στα σημεία τομής D flip-flops το κύκλωμα μετατρέπετε σε συνεχούς διοχέτευσης και θα έχει την μορφή που παρουσιάζεται στο Σχήμα 1.17.



Σχήμα 1.17 Μιγαδικός αθροιστής διάδοσης κρατουμένου, μιγαδικών αριθμών σε μορφή συμπληρώματος ως προς 2, συνεχούς διοχέτευσης

1.3 Υλοποίηση Μιγαδικού Πολλαπλασιαστή

1.3.1 Εισαγωγή

Ας υποθέσουμε ότι έχουμε δύο μιγαδικούς αριθμούς X και Y τους οποίους θέλουμε να προσθέσουμε. Αν τους παραστήσουμε σε μορφή συμπληρώματος ως προς 2 τότε αυτοί θα έχουν την μορφή $X = X_r + jX_i$ όπου $X_r = -x_{r(NX-1)}2^{NX-1} + \sum_{k=0}^{NX-2} x_{rk}2^k$ και $X_i = -x_{i(NX-1)}2^{NX-1} + \sum_{k=0}^{NX-2} x_{ik}2^k$

και επομένως $X = -x_{(NX-1)}2^{NX-1} + \sum_{k=0}^{NX-2} x_k 2^k$ όπου $x_k=x_{rk}+jx_{ik}$ για κάθε k=0…NX-1.

Ομοίως $Y = Y_r + jY_i$ όπου $Y_r = -y_{r(NY-1)}2^{NY-1} + \sum_{k=0}^{NY-2} y_{rk}2^k$ και $Y_i = -y_{i(NY-1)}2^{NY-1} + \sum_{k=0}^{NY-2} y_{ik}2^k$ και

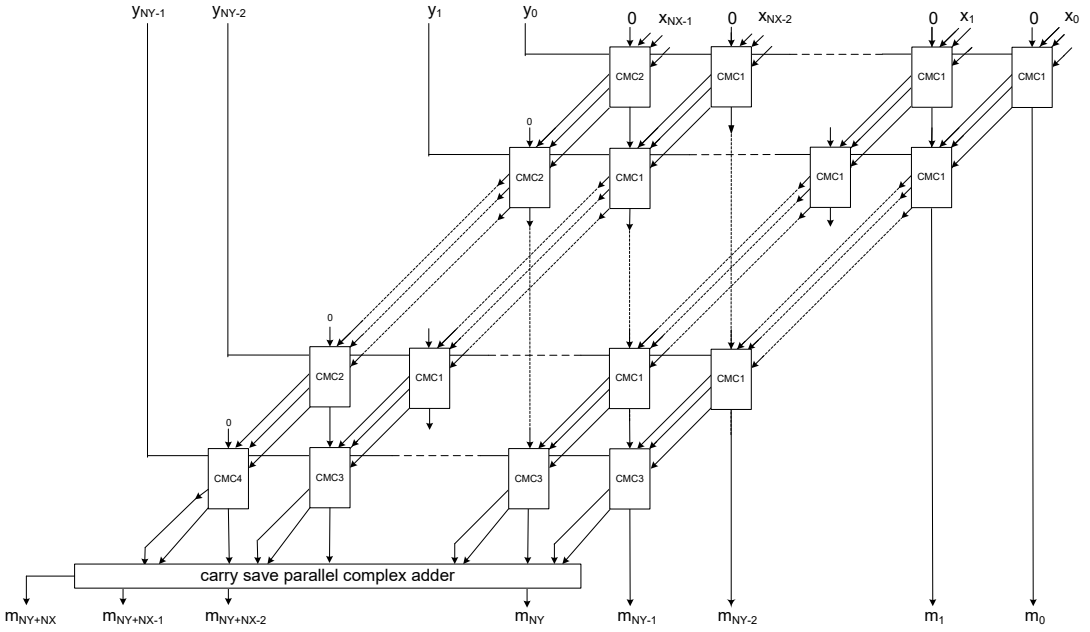
επομένως $Y = -y_{(NY-1)}2^{NY-1} + \sum_{k=0}^{NY-2} y_k 2^k$ όπου $y_k=y_{rk}+jy_{ik}$ για κάθε k=0…NY-1.

Χρησιμοποιώντας τις παραπάνω σχέσεις έχουμε για το γινόμενο των X και Y:

$$X \cdot Y = \left(-x_{(NX-1)}2^{NX-1} + \sum_{k=0}^{NX-2} x_k 2^k \right) \cdot \left(-y_{(NY-1)}2^{NY-1} + \sum_{m=0}^{NY-2} y_m 2^m \right) \Rightarrow$$

$$X \cdot Y = x_{(NX-1)}y_{(NY-1)}2^{NX+NY-2} - \sum_{m=0}^{NY-2} x_{(NX-1)}y_m 2^{m+NX-1} - \sum_{k=0}^{NX-2} y_{(NY-1)}x_k 2^{k+NY-1} + \sum_{k=0}^{NX-2}\sum_{m=0}^{NY-2} x_k y_m 2^{k+m}$$

Για να υπολογίσουμε το παραπάνω γινόμενο θα πρέπει να προσθέσουμε όλους τους όρους $x_k y_m$ δίνοντας προσοχή στα πρόσημα και στην τάξη μεγέθους που αντιπροσωπεύουν. Για τον σκοπό αυτό μπορεί να χρησιμοποιηθεί ένας μιγαδικός παράλληλος πολλαπλασιαστής με αθροιστή αποθήκευσης κρατουμένων ο οποίος θα έχει την μορφή που φαίνεται στο Σχήμα 1.18.



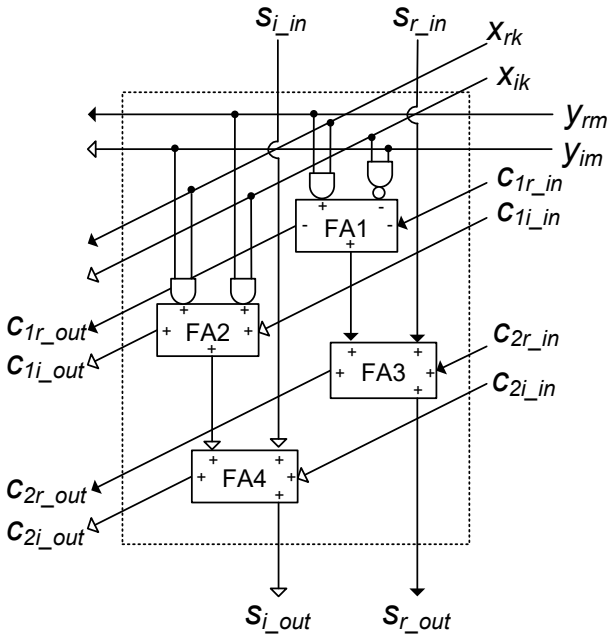
Σχήμα 1.18 Μορφή μιγαδικού πολλαπλασιαστή με αθροιστή αποθήκευσης κρατουμένου

Το πλέγμα των κυττάρων του μιγαδικού πολλαπλασιαστή (CMC) υπολογίζει και αθροίζει όλα τα μερικά γινόμενα με τα κρατούμενα κάθε τάξης να διαδίδονται στην επόμενη βαθμίδα, όπως γίνετε και στους αντίστοιχους πολλαπλασιαστές πραγματικών αριθμών. Στο τέλος ο παράλληλος αθροιστής αποθήκευσης κρατουμένου αθροίζει τα κρατούμενα που δεν έχουν προστεθεί με τα αντίστοιχα αθροίσματα μερικών γινόμενων κάθε τάξης.

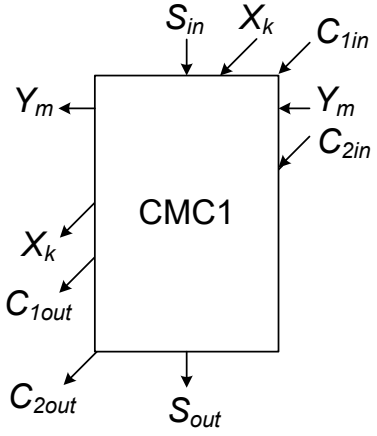
1.3.2 1ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC1)

Ο όρος $\sum_{k=0}^{NX-2}\sum_{m=0}^{NY-2} x_k y_m 2^{k+m}$ υπολογίζεται από αυτό το είδος κυττάρων.

Έχουμε $x_k y_m = (x_{rk} + jx_{ik})(y_{rm} + jy_{im}) = x_{rk}y_{rm} - x_{ik}y_{im} + j(x_{ik}y_{rm} + x_{rk}y_{im})$ επομένως κάθε κύτταρο θα έχει την μορφή που φαίνεται στο Σχήμα 1.19 όπου οι δύο πρώτοι πλήρης αθροιστές (FA1 και FA2) προσθέτουν τα μερικά γινόμενα του πραγματικού και φανταστικού μέρους, λαμβάνοντας υπόψη και τα κρατούμενα που διαδίδονται από αντίστοιχα αθροίσματα προηγούμενων τάξεων bit, και οι άλλοι δύο πλήρης αθροιστές (FA3 και FA4) προσθέτουν αυτά με τα αντίστοιχα του προηγούμενου επιπέδου.



Σχήμα 1.19 1ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



Σχήμα 1.20 Συνεπτυγμένη μορφή 1ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή

Πρέπει να προσέξουμε τα σήματα $x_{ik}y_{im}$ που έχουν αρνητικό πρόσημο και για αυτόν τον λόγο βάλαμε την πύλη NAND στον πρώτο πλήρη αθροιστή (FA1). Αυτή η αλλαγή όμως θα οδηγούσε την έξοδο του αθροίσματος του πλήρη αθροιστή σε αρνητικό πρόσημο, πράγμα που δεν θέλουμε. Για να μην συμβεί αυτό θα πρέπει να έχουμε και μια ακόμα απ' τις εισόδους του συγκεκριμένου πλήρη αθροιστή (FA1) αρνητική. Για αυτόν τον λόγο επιλέξαμε την είσοδο των πραγματικών κρατουμένων (c_{1r_in}) απ' την προηγούμενη βαθμίδα να έχουν αρνητικό πρόσημο, με αποτέλεσμα η αρνητική έξοδος να μετατοπιστεί στο κρατούμενο εξόδου του πλήρη αθροιστή (FA1) δηλαδή στο πραγματικό κρατούμενο εξόδου c_{1r_out}. Επομένως, σύμφωνα με την παραπάνω αλλαγή, θα έπρεπε κανονικά να βάλουμε αντιστροφείς στα συγκεκριμένα πραγματικά

κρατούμενα. Όμως αν αυτοί έμπαιναν, όπως φαίνεται και απ' την διασύνδεση των κυττάρων τύπου 1 (CMC1), θα αλληλοαναιρούνταν όλοι εκτός από αυτούς στις εισόδους c_{1r_in} της πρώτης γραμμής και τις εξόδους c_{1r_out} της τελευταίας. Τους τελευταίους θα τους λάβουμε υπόψη μας στην διασύνδεση με τα κύτταρα τύπου 3 (CMC3) ενώ για τους πρώτους θα τους απαλείψουμε θέτοντας απευθείας της εισόδους αυτές στο 1.

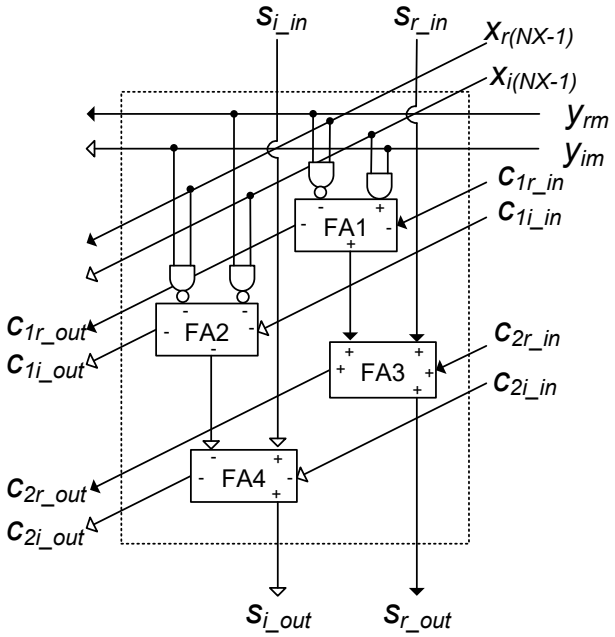
1.3.3 2ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC2)

Ο όρος $-\sum_{m=0}^{NY-2} x_{(NX-1)} y_m 2^{m+NX-1}$ υπολογίζεται από αυτό το είδος κυττάρων.

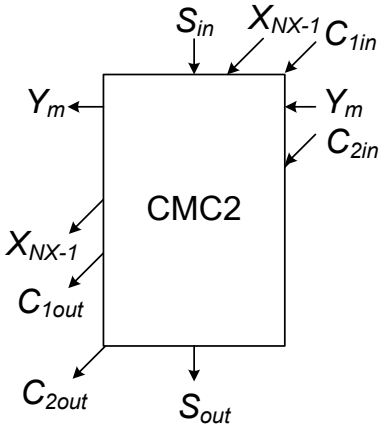
Έχουμε

$$-x_{(NX-1)}y_m = -\left(x_{r(NX-1)} + jx_{i(NX-1)}\right)\left(y_{rm} + jy_{im}\right) \Leftrightarrow$$
$$-x_{(NX-1)}y_m = \left(-x_{r(NX-1)}y_{rm} + x_{i(NX-1)}y_{im}\right) + j\left(-x_{i(NX-1)}y_{rm} - x_{r(NX-1)}y_{im}\right)$$

επομένως κάθε κύτταρο θα έχει την μορφή που φαίνεται στο Σχήμα 1.21.



Σχήμα 1.21 2ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



Σχήμα 1.22 Συνεπτυγμένη μορφή 2ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή

Ομοίως με προηγουμένως, στα γινόμενα που έχουν αρνητικό πρόσημο τοποθετήσαμε πύλες NAND πριν την είσοδό τους στους αθροιστές (FA1 και FA2).

Όσο αφορά των πρώτο πλήρη αθροιστή (FA1), όμοια με προηγουμένως, έχουμε την μια είσοδο του, απ 'την πύλη NAND, με αρνητικό πρόσημο πράγμα που οδηγεί την έξοδο του αθροίσματος σε αρνητικό πρόσημο, πράγμα που δεν θέλουμε. Για να μην συμβεί αυτό θα πρέπει να έχουμε και μια ακόμα απ' τις εισόδους του συγκεκριμένου πλήρη αθροιστή (FA1) αρνητική. Για αυτόν τον λόγο επιλέξαμε την είσοδο των πραγματικών κρατουμένων (c_{1r_in}) απ' την προηγούμενη βαθμίδα να έχουν αρνητικό πρόσημο, με αποτέλεσμα η αρνητική έξοδος να μετατοπιστεί στο κρατούμενο εξόδου του πλήρη αθροιστή (FA1) δηλαδή στο πραγματικό κρατούμενο εξόδου

c_{1r_out}. Επομένως, σύμφωνα με την παραπάνω αλλαγή, θα έπρεπε κανονικά να βάλουμε αντιστροφείς στα αντίστοιχα κρατούμενα. Όμως αν αυτοί έμπαιναν, όπως φαίνεται και απ' την διασύνδεση των κυττάρων τύπου 2 (CMC2), θα αλληλοαναιρούνταν όλοι εκτός από αυτόν τις εισόδου c_{1r_in} της πρώτης γραμμής και τις εξόδου c_{1r_out} της τελευταίας. Τον τελευταίο θα τον λάβουμε υπόψη μας στην διασύνδεση με το κύτταρο τύπου 4 (CMC4) ενώ για το πρώτο θα τον απαλείψουμε θέτοντας απευθείας την συγκεκριμένη είσοδο στο 1.

Όσο αφορά των δεύτερο πλήρη αθροιστή (FA2), έχουμε τις δύο εισόδους του με αρνητικό πρόσημο, απ΄τις πύλες NAND, πράγμα που οδηγεί την έξοδο c_{1i_out} να έχει αρνητικό πρόσημο, οπότε θα πρέπει και η είσοδος c_{1i_in} του επόμενου κυττάρου τύπου 2 (CMC2) να έχει αρνητικό πρόσημο. Για λόγους λοιπόν συμμετρίας, θεωρούμε όλες τις εισόδους του δεύτερου αθροιστή (FA2) με αρνητικό πρόσημο με αποτέλεσμα και όλες οι έξοδοί του να έχουν αρνητικό πρόσημο. Αυτό φυσικά έχει ως συνέπεια ο τέταρτος πλήρης αθροιστής (FA4) να έχει την μια είσοδό του με αρνητικό πρόσημο με αποτέλεσμα η έξοδος του αθροίσματός του να είναι και αυτή αρνητική. Αυτό όμως δεν θέλουμε να ισχύει και θεωρούμε και το κρατούμενο εισόδου του c_{2i_in} με αρνητικό πρόσημο. Επομένως θα έχει δύο εισόδους με αρνητικό πρόσημο οπότε και το κρατούμενο εξόδου c_{2i_out} θα είναι με αρνητικό πρόσημο. Επομένως, σύμφωνα με την παραπάνω αλλαγή, θα έπρεπε κανονικά να βάλουμε αντιστροφείς στα αντίστοιχα αρνητικά σήματα. Όμως αν αυτοί έμπαιναν, όπως φαίνεται και απ' την διασύνδεση των κυττάρων τύπου 2 (CMC2), θα αλληλοαναιρούνταν όλοι εκτός από αυτόν των εισόδων c_{1i_in} και c_{2i_in} της πρώτης γραμμής και των εξόδων c_{1i_out} και c_{2i_out} της τελευταίας. Τους τελευταίους θα τους λάβουμε υπόψη μας στην διασύνδεση με το κύτταρο τύπου 4 (CMC4) ενώ για τους πρώτους θα τους απαλείψουμε θέτοντας απευθείας τις συγκεκριμένες εισόδους στο 1.

1.3.4 3ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC3)

Ο όρος $-\sum_{k=0}^{NX-2} y_{(NY-1)} x_k 2^{k+NY-1}$ υπολογίζεται από αυτό το είδος κυττάρων.

Έχουμε

$$-y_{(NY-1)}x_k = -\left(y_{r(NY-1)} + jy_{i(NY-1)}\right)\left(x_{rk} + jx_{ik}\right) \Leftrightarrow$$
$$-y_{(NY-1)}x_k = \left(-y_{r(NY-1)}x_{rk} + y_{i(NY-1)}x_{ik}\right) + j\left(-y_{i(NY-1)}x_{rk} - y_{r(NY-1)}x_{ik}\right)$$

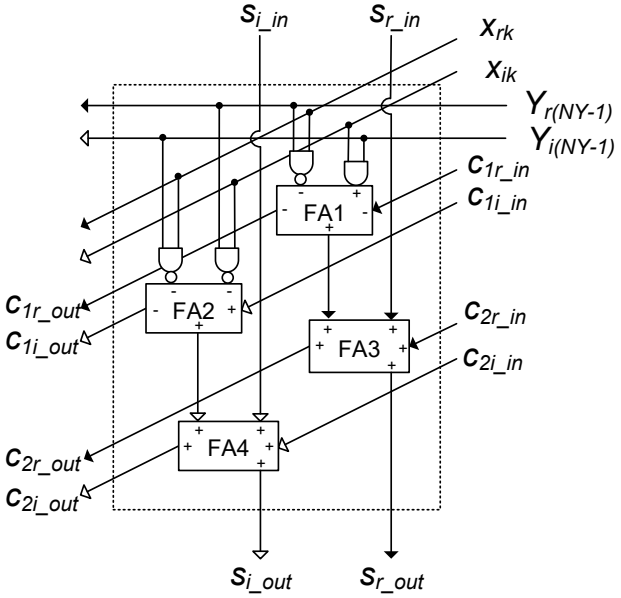
επομένως κάθε κύτταρο θα έχει την μορφή που φαίνεται στο Σχήμα 1.23.

Όπως κάναμε και στα προηγούμενα κύτταρα, στα γινόμενα με αρνητικό πρόσημο τοποθετήσαμε πύλες NAND στην είσοδο των πρώτων δύο πλήρων αθροιστών (FA1 και FA2) στους οποίους εισέρχονται και τα κρατούμενα, απ' τις προηγούμενες βαθμίδες (CMC1).

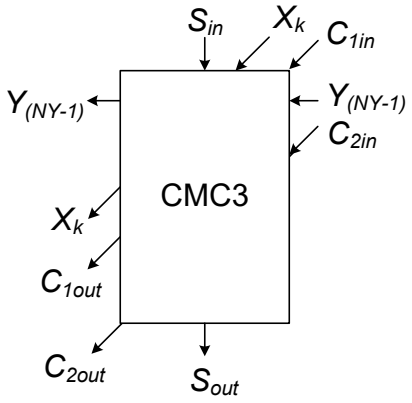
Έτσι για τον πρώτο πλήρη αθροιστή (FA1) έχουμε δύο εισόδους με αρνητικό πρόσημο, μια απ' την πύλη NAND και μια απ' το κρατούμενο (c_{1r_in}) από την τελευταία γραμμή των τύπου 1 κυττάρων (CMC1), τα οποία είναι με αρνητικό πρόσημο, οπότε το κρατούμενο εξόδου (c_{1r_out}) θα έχει και αυτό αρνητικό πρόσημο.

Στον δεύτερο πλήρη αθροιστή (FA2) έχουμε πάλι δύο εισόδους με αρνητικό πρόσημο, απ 'τις δύο πύλες NAND που υπολογίζουν τα μερικά γινόμενα, οπότε και το κρατούμενο εξόδου (c_{1i_out}) θα είναι με αρνητικό πρόσημο.

Οι αντιστροφείς που έπρεπε να μπουν στα κρατούμενα εισόδου (c_{1r_in}) του πρώτου πλήρη αθροιστή (FA1) αναιρούνται με αυτούς που έπρεπε να βάλουμε στα αντίστοιχα κρατούμενα εξόδου των κυττάρων τις προηγούμενης βαθμίδας (CMC1), τους οποίους για αυτόν τον λόγο δεν είχαμε τοποθετήσει προηγουμένως. Και τους αντιστροφείς που έπρεπε να μπουν στα αντίστοιχα κρατούμενα εξόδου θα τους λάβουμε υπόψη με την διασύνδεση με τον παράλληλο μιγαδικό αθροιστή με σώσιμο κρατουμένου ο οποίος και θα αθροίσει όλα τα κρατούμενα εξόδου που προκύπτουν.



Σχήμα 1.23 3ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



Σχήμα 1.24 Συνεπτυγμένη μορφή 3ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή

Παρατηρούμε ότι ο 3ος τύπος κυττάρου (CMC3) είναι ουσιαστικά ίδιος, σε υλοποίηση, με τον 2ο τύπο κυττάρου (CMC2). Το μόνο που αλλάζει είναι το πρόσημο, άρα και η σημασία, του κρατουμένου εξόδου c_{2i_out} των κυττάρων.

1.3.5 4ος Τύπος Κυττάρου του Μιγαδικού πολλαπλασιαστή (CMC4)

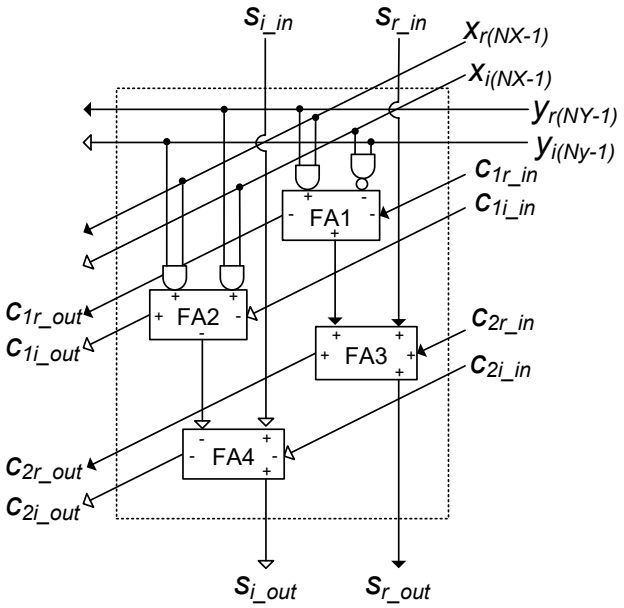
Ο όρος $x_{(NX-1)}y_{(NY-1)}2^{NX+NY-2}$ υπολογίζεται από αυτό το είδος κυττάρων.
Έχουμε

$$x_{(NX-1)}y_{(NY-1)} = \left(x_{r(NX-1)} + jx_{i(NX-1)}\right)\left(y_{r(NY-1)} + jy_{i(NY-1)}\right) \Leftrightarrow$$
$$x_{(NX-1)}y_{(NY-1)} = \left(x_{r(NX-1)}y_{r(NY-1)} - x_{i(NX-1)}y_{i(NY-1)}\right) + j\left(x_{i(NX-1)}y_{r(NY-1)} + x_{r(NX-1)}y_{i(NY-1)}\right)$$

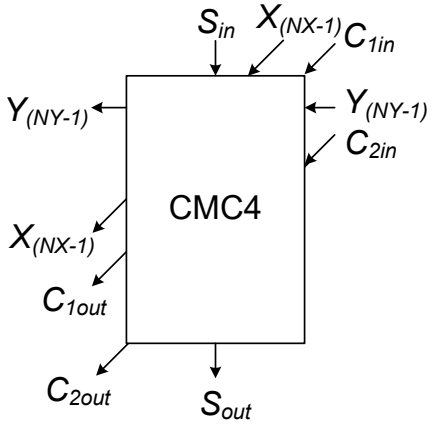
επομένως κάθε κύτταρο θα έχει την μορφή που φαίνεται στο Σχήμα 1.25.

Πάλι τοποθετήσαμε στο γινόμενο με αρνητική αξία $x_{i(NX-1)}y_{i(NY-1)}$ πύλη NAND πριν την είσοδό του στον πρώτο πλήρη αθροιστή (FA1). Λαμβάνοντας υπόψη ότι τα κρατούμενα που έρχονται στο κύτταρο τύπου 4 (CMC4) από τις προηγούμενες βαθμίδες, δηλαδή το τελευταίο κύτταρο τύπου 2 (CMC2) καθώς και τα πρόσημά τους προκύπτει ότι το μόνο σήμα εξόδου με αρνητικό πρόσημο είναι το φανταστικό κρατούμενο εξόδου c_{2i_out} του τέταρτου πλήρη αθροιστή (FA4). Ο αντιστροφέας στο κρατούμενο εισόδου c_{1r_in} του πρώτου πλήρη αθροιστή (FA1) αναιρείται με

αυτόν που θα έπρεπε να μπει στο αντίστοιχο κρατούμενο εξόδου του τελευταίου κυττάρου τύπου 2, το οποίο δεν είχαμε βάλει έχοντας το υπόψη. Τον αντιστροφέα που έπρεπε να μπει στο κρατούμενο εξόδου c_{2i_out} θα τον λάβουμε υπόψη με την διασύνδεση με τον παράλληλο μιγαδικό αθροιστή με σώσιμο κρατουμένου ο οποίος και θα αθροίσει όλα τα κρατούμενα εξόδου που προκύπτουν στην συγκεκριμένη γραμμή.



Σχήμα 1.25 4ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή

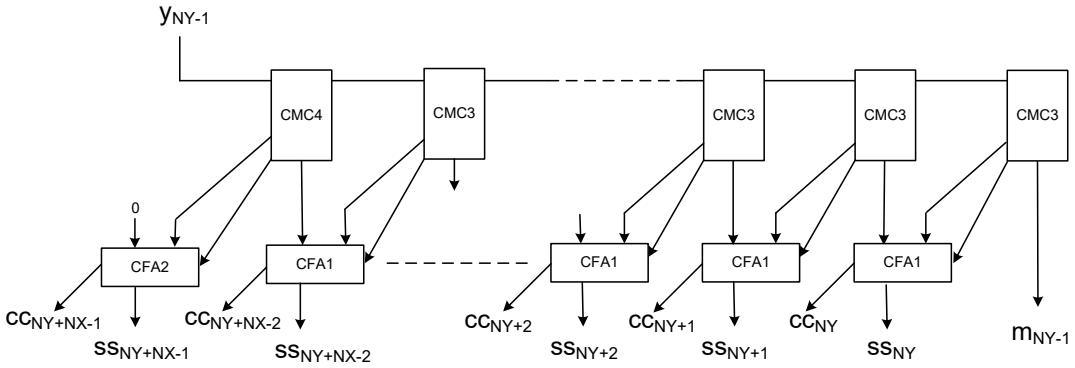


Σχήμα 1.26 Συνεπτυγμένη μορφή 4ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή

Παρατηρούμε ότι ο 4ος τύπος κυττάρου (CMC4) είναι ουσιαστικά ίδιος, σε υλοποίηση, με τον 1ο τύπο κυττάρου (CMC1). Το μόνο που αλλάζει είναι το πρόσημο, άρα και η σημασία, του κρατουμένου εξόδου c_{2i_out} των κυττάρων.

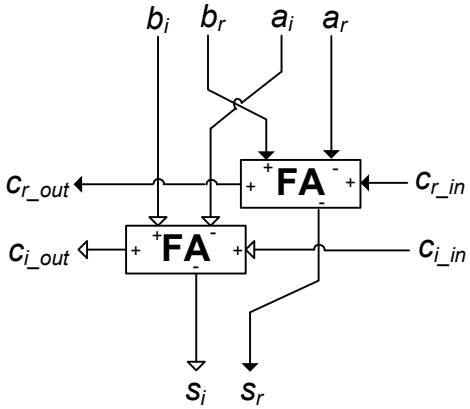
1.3.6 Παράλληλος Μιγαδικός αθροιστής με αποθήκευση κρατουμένου

Με τα παραπάνω κύτταρα έχουμε φτιάξει το βασικό πλέγμα με το οποίο υπολογίζεται το άθροισμα των μερικών γινομένων μέχρι και τη τάξη του 2^{NY-1}. Από εκεί και πάνω μας έχουν μείνει να αθροιστούν τα κρατούμενα που βγαίνουν απ' τα κύτταρα του μιγαδικού πολλαπλασιαστή τύπου 3 και 4 (CMC3 και CMC4). Για αρχή, αυτό μπορεί να γίνει με μια σειρά πλήρη μιγαδικών αθροιστών όπως φαίνεται στο Σχήμα 1.27.

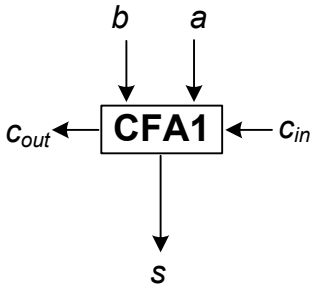


Σχήμα 1.27 Πρώτο άθροισμα των κρατουμένων εξόδου των κυττάρων του μιγαδικού πολλαπλασιαστή

Στους πλήρης μιγαδικούς αθροιστές 1ου τύπου αθροίζονται το μέχρι τώρα άθροισμα των μερικών γινομένων της αντίστοιχης τάξης με τα κρατούμενα εξόδου c_{1r_out} και c_{2i_out}, τα οποία έχουν αρνητικό πρόσημο, καθώς και με τα κρατούμενα εξόδου c_{2r_out} και c_{2i_out}, τα οποία έχουν θετικό πρόσημο, που έχουν δημιουργηθεί απ' τα αθροίσματα προηγούμενων τάξεων και ο οποίος φαίνεται αναλυτικά στο Σχήμα 1.28.



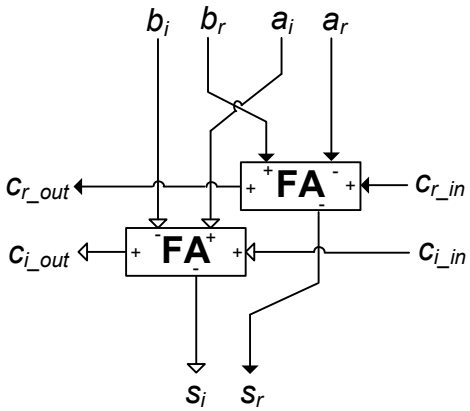
Σχήμα 1.28 Μιγαδικός πλήρης αθροιστής 1ου τύπου



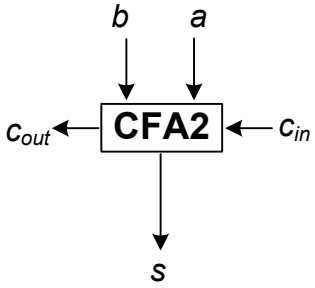
Σχήμα 1.29 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 1ου τύπου

Στα σήματα a εισέρχονται τα σήματα c_{1_out} απ' τα κύτταρα τύπου 3 του μιγαδικού πολλαπλασιαστή (CMC3) , τα οποία είναι και αυτά με αρνητικό πρόσημο, στα σήματα c_{in} τα αντίστοιχα κρατούμενα εξόδου c_{2_out}, τα οποία έχουν θετικό πρόσημο και στα σήματα b το άθροισμα των μερικών γινομένων της ίδιας τάξης, το οποίο πάντα είναι θετικό. Αφού σε καθέναν πλήρη αθροιστή έχουμε μόνο μια είσοδο με αρνητικό πρόσημο θα έχουμε και την έξοδο τους (s) με αρνητικό πρόσημο.

Τα κρατούμενα εξόδου του τύπου 4 κυττάρου του μιγαδικού πολλαπλασιαστή (CMC4) αθροίζονται στον τύπου 2 πλήρη μιγαδικό αθροιστή (CFA2). Εδώ έχουμε με αρνητικό πρόσημο το πραγματικό μέρος του κρατούμενου εξόδου c_{1_out}, δηλαδή το c_{1r_out}, καθώς και το μιγαδικό μέρος του κρατουμένου εξόδου c_{2_out}, δηλαδή το c_{2i_out}. Έτσι πάλι έχουμε στην είσοδο των πλήρων αθροιστών μόνο μια είσοδο με αρνητικό πρόσημο και άρα η έξοδός τους θα είναι αρνητική. Με την ίδια σύμβαση με προηγουμένως για τα σήματα εισόδου ο μιγαδικός πλήρης αθροιστής τύπου 2 (CFA2) φαίνεται στο Σχήμα 1.30.



Σχήμα 1.30 Μιγαδικός πλήρης αθροιστής 2ου τύπου



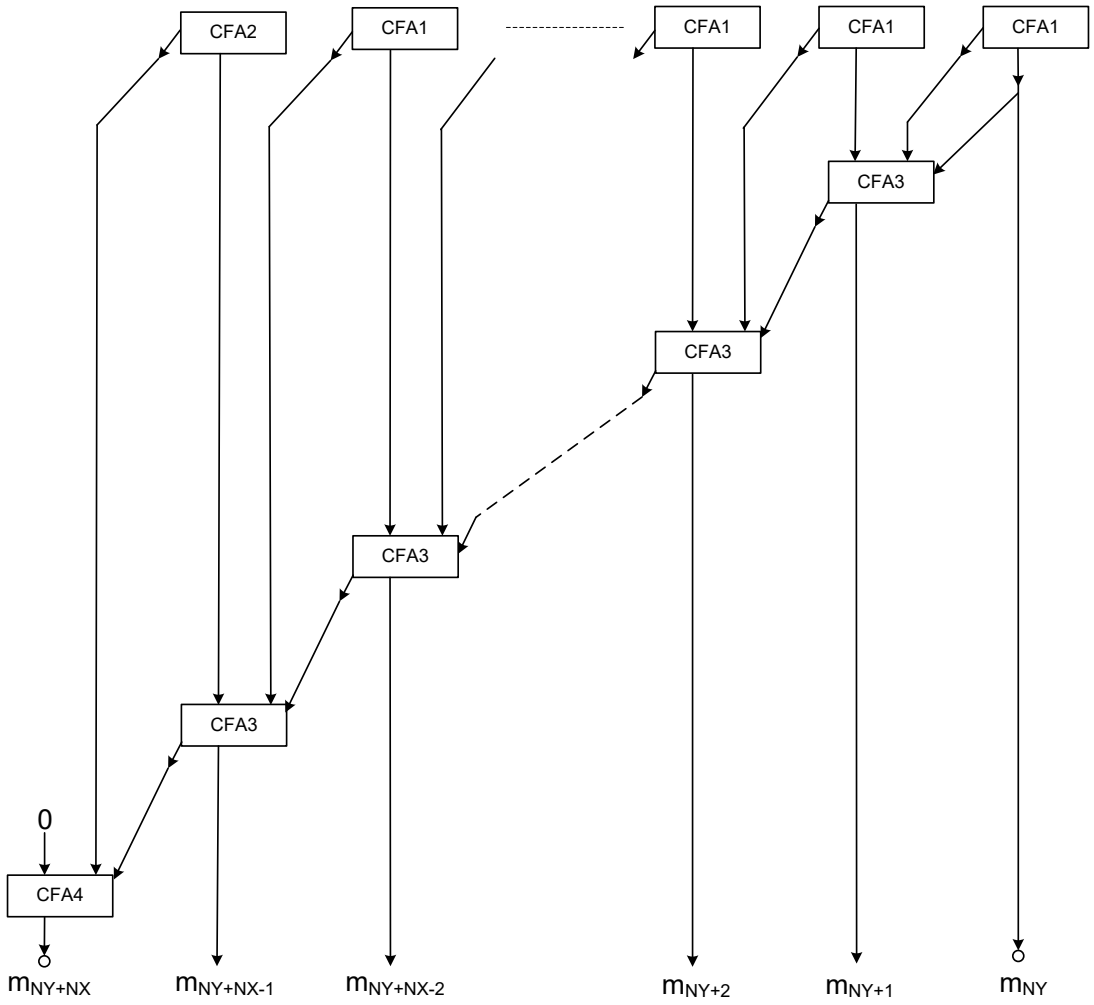
Σχήμα 1.31 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 2ου τύπου

Απ' τα παραπάνω γίνεται αντιληπτός ο λόγος που δεν βάλαμε αντιστροφείς στα σήματα εξόδου, με αρνητικό πρόσημο, των κυττάρων τύπου 3 και 4 του μιγαδικού πολλαπλασιαστή (CMC3 και CMC4), αφού αυτοί αναιρούνται με αυτούς που θα έπρεπε να βάλουμε στις αντίστοιχες εισόδους των πλήρη μιγαδικών αθροιστών. Αντίστοιχα για αυτούς τους αντιστροφείς που πρέπει να μπουν στα αθροίσματα εξόδου που έχουν αρνητικό πρόσημο και πάλι δεν τους τοποθετούμε διότι θα αναιρεθούν παρόμοια όταν τους λάβουμε υπόψη παρακάτω.

Μέχρι στιγμής λοιπόν έχουμε αθροίσει τις εξόδους όλων των τύπων κυττάρων του μιγαδικού πολλαπλασιαστή (CMC1, CMC2, CMC3 και CMC4) και μας μένει να αθροίσουμε τα μιγαδικά κρατούμενα των μιγαδικών πλήρων αθροιστών CC με τα αντίστοιχα αθροίσματα εξόδου SS αντίστοιχης τάξης και τα οποία έχουν όλα αρνητικό πρόσημο.

Ο δεξιότερος πλήρης μιγαδικός αθροιστής έχει στην έξοδό του το άθροισμα των μερικών γινομένων της τάξης 2^{NY} και το μόνο πρόβλημα είναι ότι αυτό έχει αρνητικό πρόσημο ενώ θα θέλαμε να έχει θετικό για να είναι σωστή η περιγραφή του γινομένου με αριθμούς σε μορφή συμπληρώματος ως προς 2. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε την ιδιότητα $-ss_{NY}2^{NY} = -ss_{NY}2^{NY+1} + ss_{NY}2^{NY}$ και άρα αρκεί να θεωρήσουμε το συγκεκριμένο άθροισμα μερικών γινομένων θετικό και να αφαιρέσουμε απ' το αντίστοιχο άθροισμα της επόμενης τάξης το ss_{NY}. Έτσι για να πάρουμε την έξοδο της τάξης 2^{NY} του γινομένου (m_{NY}) αρκεί να αντιστρέψουμε το σήμα ss_{NY} αφού δεν βάλαμε στην αντίστοιχη έξοδο του πλήρη μιγαδικού αθροιστή αντιστροφείς στα bits με αρνητικό πρόσημο.

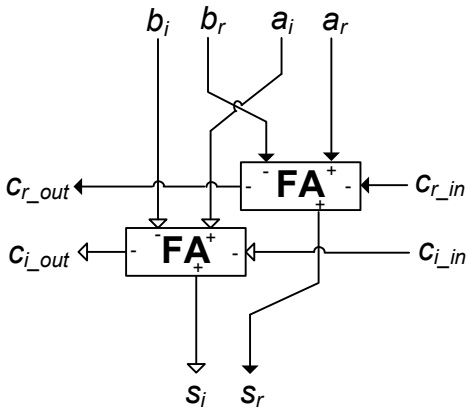
Επομένως το κύκλωμα που στην συνέχεια αθροίζει τα εναπομείναντα κρατούμενα και αθροίσματα έχει την μορφή που φαίνεται στο Σχήμα 1.32.



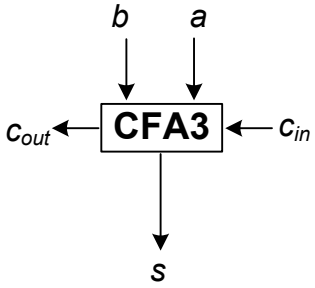
Σχήμα 1.32 Μιγαδικός αθροιστής με αποθήκευση κρατουμένου για την άθροιση των κρατουμένων εξόδου των κυττάρων του μιγαδικού πολλαπλασιαστή

Καθένας απ' τους πλήρης μιγαδικούς αθροιστές τύπου 3 (CFA3) έχει δύο εισόδους με αρνητικό πρόσημο με αποτέλεσμα τα bits με αρνητικό πρόσημο να διαδίδονται μέσο των κρατουμένων εξόδου ενώ το άθροισμα εξόδου αντίστοιχα να έχει θετικό πρόσημο. Αντίστοιχα στον πλήρη μιγαδικό αθροιστή τύπου 4 (CFA4) η μόνη είσοδος με αρνητικό πρόσημο είναι το μιγαδικό κρατούμενο εισόδου με αποτέλεσμα το αντίστοιχο άθροισμα εξόδου να έχει αρνητικό πρόσημο, πράγμα που είναι και επιθυμητό για την σωστή μορφή του γινομένου σε μορφή συμπληρώματος ως προς 2. Φυσικά όλοι οι αντιστροφείς που θα έπρεπε να τοποθετηθούν αναιρούνται εκτός απ' αυτόν που μας δίνει το MSB του γινομένου.

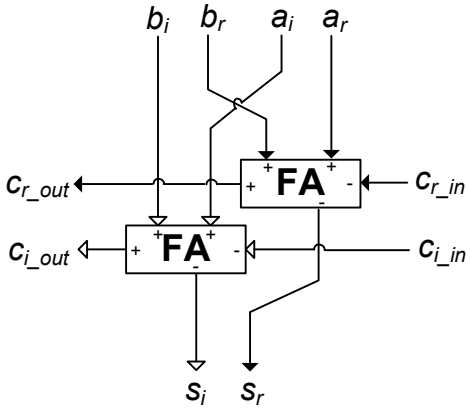
Λαμβάνοντας υπόψη τα πρόσημα των σημάτων οι μιγαδικοί πλήρης αθροιστές παρουσιάζονται με λεπτομέρεια στο Σχήμα 1.33 και στο Σχήμα 1.35.



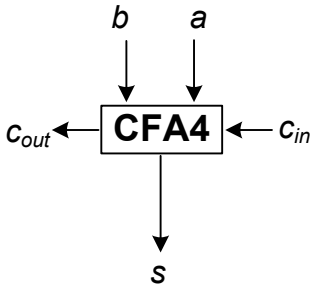
Σχήμα 1.33 Μιγαδικός πλήρης αθροιστής 3ου τύπου



Σχήμα 1.34 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 3ου τύπου



Σχήμα 1.35 Μιγαδικός πλήρης αθροιστής 4ου τύπου

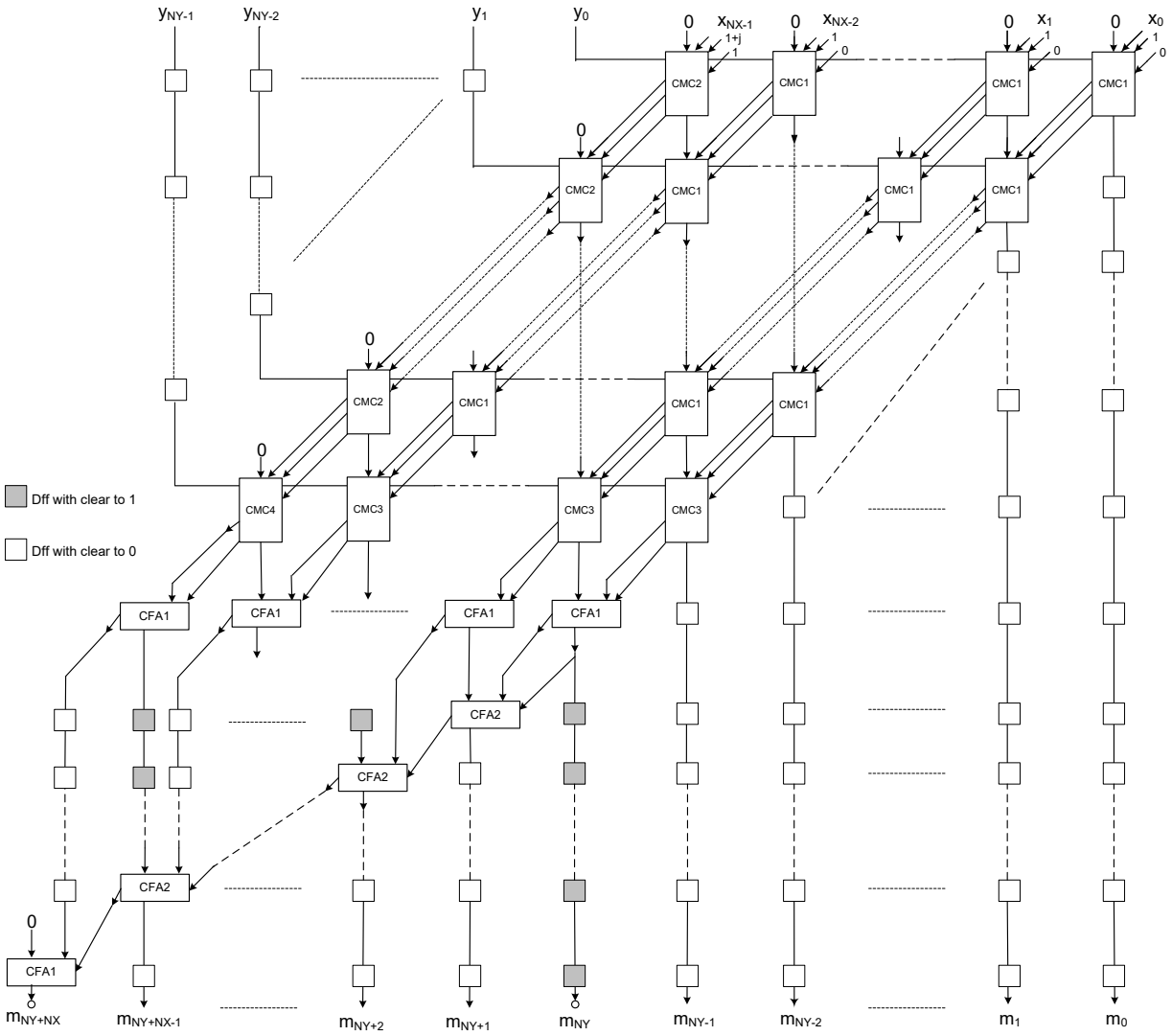


Σχήμα 1.36 Συνεπτυγμένη μορφή του μιγαδικού πλήρη αθροιστή 4ου τύπου

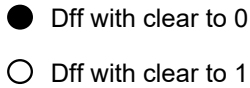
Φυσικά η κατηγοριοποίηση των πλήρη μιγαδικών αθροιστών έγινε όσον αφορά τα πρόσημα που αντιστοιχούν στα σήματα εισόδων και εξόδων κάθε τύπου, ώστε να εξηγηθεί η λειτουργία του κυκλώματος. Η διαφορά αυτή αντιστοιχεί μόνο στα πρόσημα και όχι στην υλοποίηση των πλήρη μιγαδικών αθροιστών που είναι σε όλα η ίδια.

1.3.7 Μιγαδικός πολλαπλασιαστής συνεχούς διοχέτευσης με μιγαδικούς αθροιστές αποθήκευσης κρατουμένου

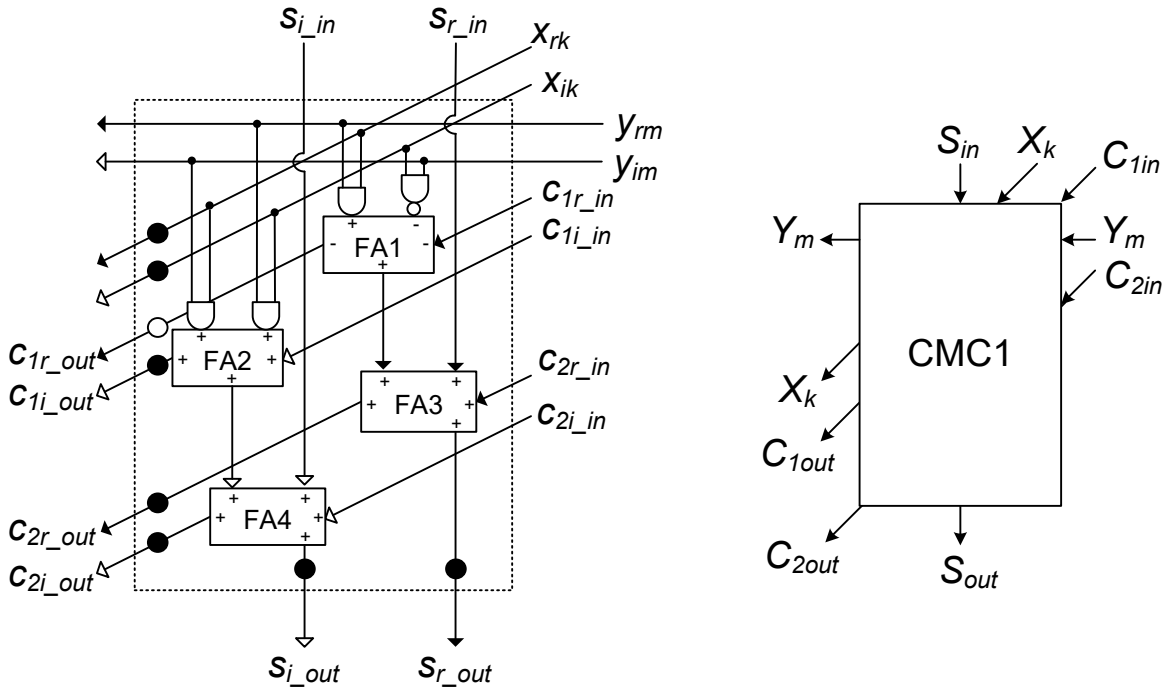
Με βάση τα όσα ειπώθηκαν στις προηγούμενες παραγράφους, ολόκληρος ο μιγαδικός πολλαπλασιαστής φαίνεται στο Σχήμα 1.37 σε μορφή συνεχούς διοχέτευσης όπου πρέπει να αναφερθεί ότι μέσα σε όλα τα κύτταρα του μιγαδικού πολλαπλασιαστή (CMC1,CMC2,CMC3 και CMC4) καθώς και σε όλα τα κύτταρα του μιγαδικού αθροιστή με αποθήκευση κρατουμένου (CFA1 και CFA2) έχουν τοποθετηθεί πριν τις εξόδους προς τις επόμενες βαθμίδες D flip-flop τα οποία φαίνονται αναλυτικά από το Σχήμα 1.38 έως και το Σχήμα 1.44. Το διαφορετικό χρώμα που έχουν κάποια flip-flop οφείλετε στο ότι αποθηκεύουν σήματα με αρνητικό πρόσημο. Έτσι αν θέλουμε σε κάποια αρχικοποίηση (CLEAR) το κύκλωμα να μην βγάζει μη μηδενικά αποτελέσματα θα πρέπει τα flip-flop αυτά να φτιαχτούν έτσι ώστε στη αρχικοποίηση (CLEAR) να θέτονται στο λογικό 1 και όχι στο 0.



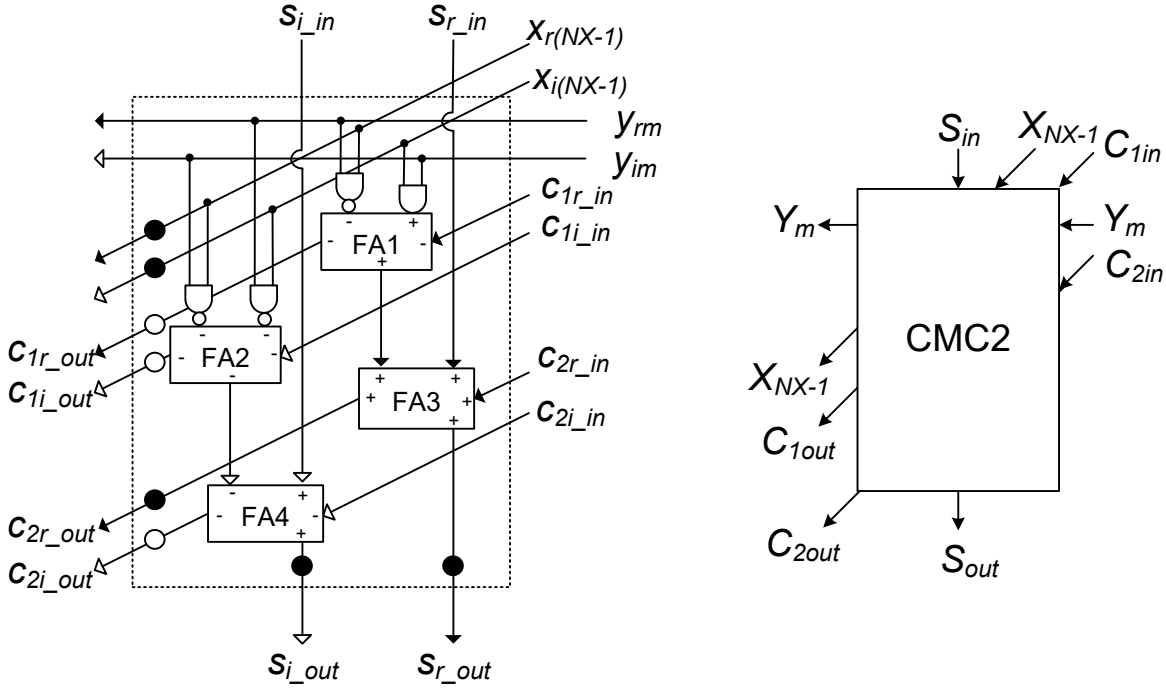
Σχήμα 1.37 Μιγαδικός Πολλαπλασιαστής συνεχούς διοχέτευσης



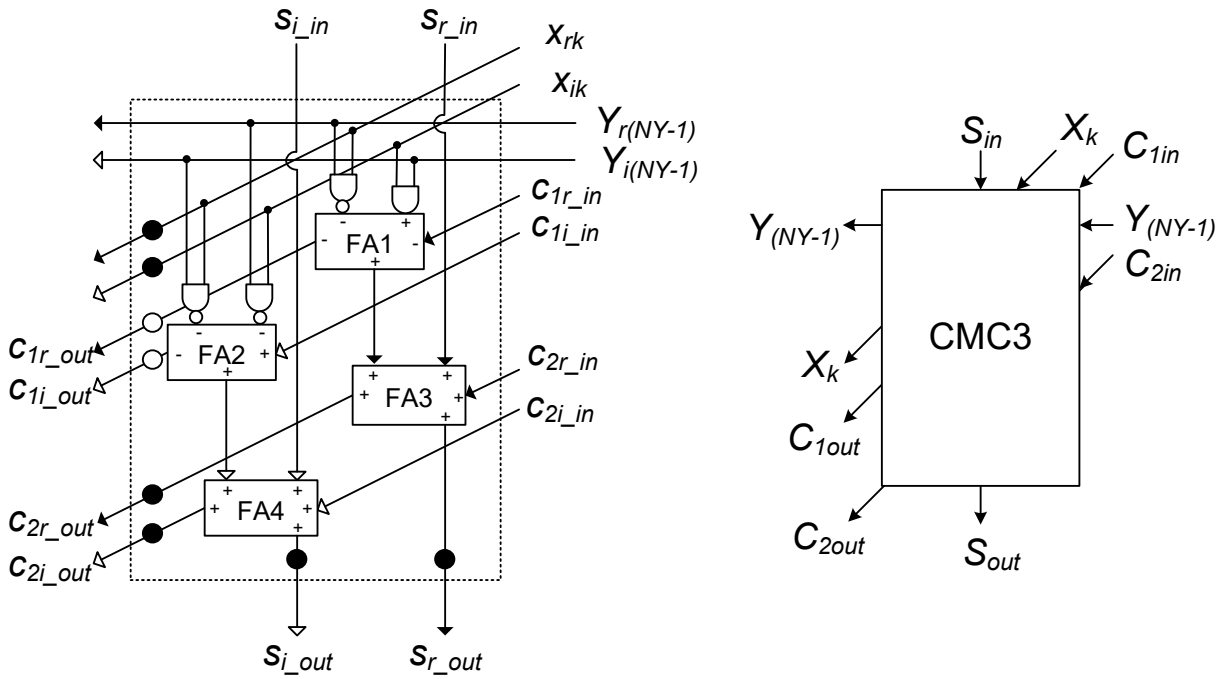
Σχήμα 1.38 Παράσταση των D flip-flop που χρησιμοποιούνται στα κύτταρα του μιγαδικού πολλαπλασιαστή



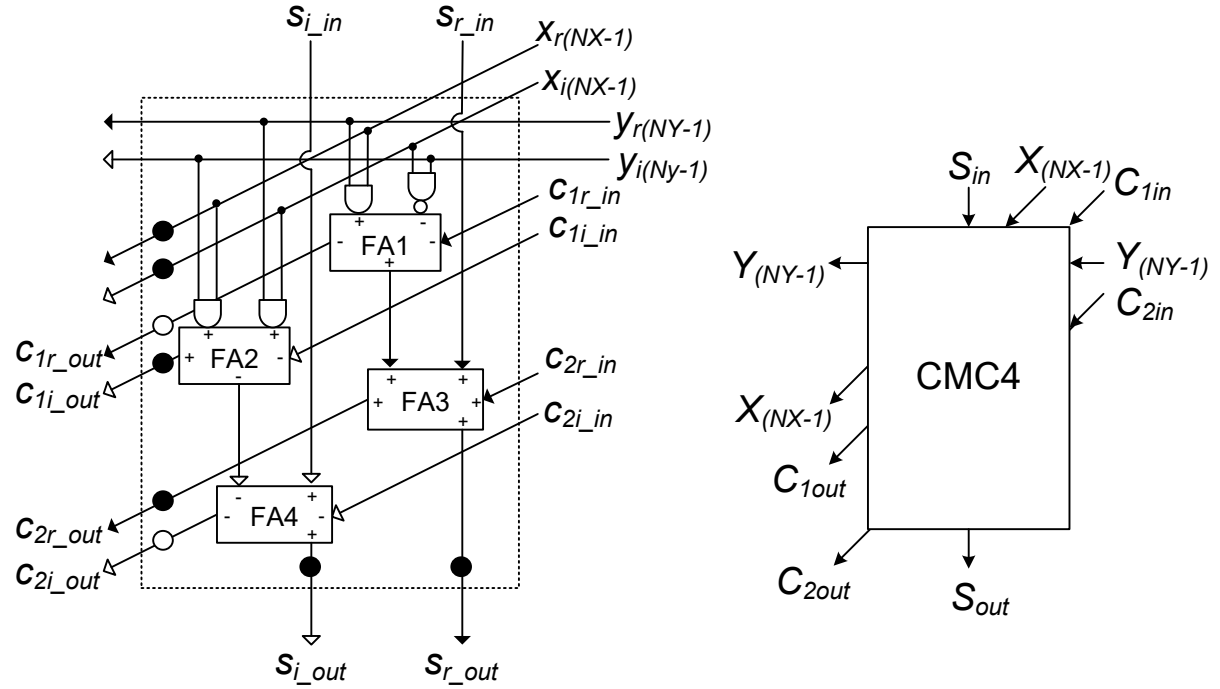
Σχήμα 1.39 1ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



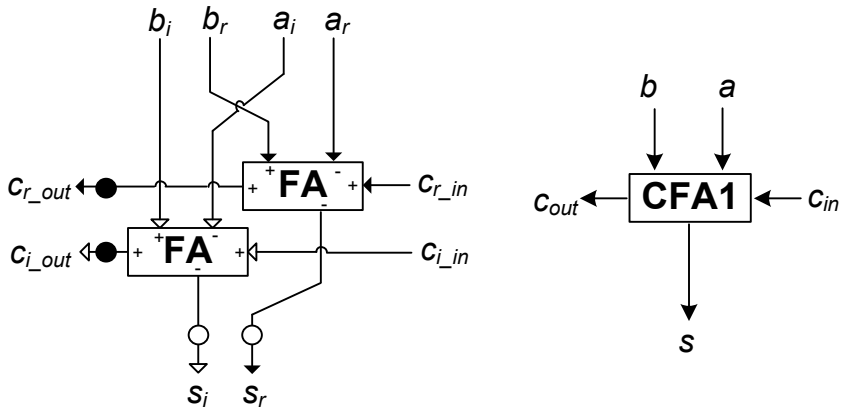
Σχήμα 1.40 2ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



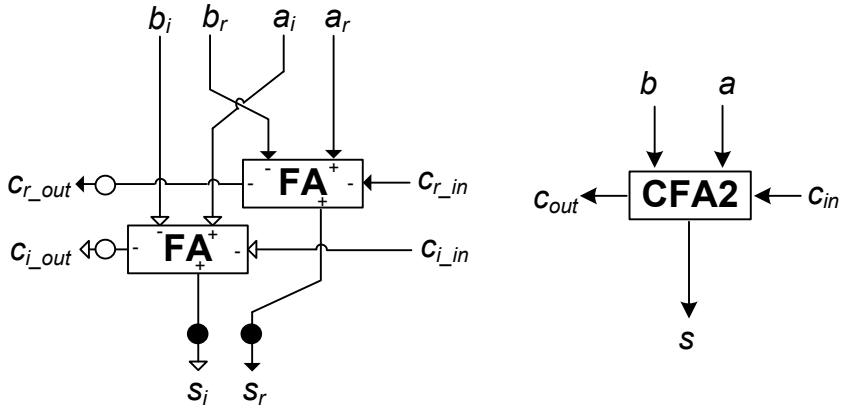
Σχήμα 1.41 3ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



Σχήμα 1.42 4ος τύπος κυττάρου του μιγαδικού πολλαπλασιαστή



Σχήμα 1.43 1ος τύπος μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή



Σχήμα 1.44 2ος τύπος μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή

2 ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΔΙΑΚΡΙΤΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ FOURIER ΜΕ ΒΑΣΗ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΤΟΥ ΓΡΗΓΟΡΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ FOURIER

2.1 Διακριτός Μετασχηματισμός Fourier (DFT)

Ο υπολογισμός του N σημείων διακριτού μετασχηματισμού Fourier ορίζεται ως ο υπολογισμός της πεπερασμένης ακολουθίας N μιγαδικών αριθμών $\{X(n)\}$ δοθέντος μιας πεπερασμένης ακολουθίας N μιγαδικών αριθμών $\{x(n)\}$ με βάση τον τύπο

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad 0 \le k \le N-1$$

όπου $W_N = e^{-j\frac{2\pi}{N}}$

Παρατηρούμε ότι για κάθε τιμή του k θα πρέπει να γίνουν N μιγαδικοί πολλαπλασιασμοί και N-1 μιγαδικές προσθέσεις. Δηλαδή για να υπολογιστεί ο διακριτός μετασχηματισμός Fourier θα πρέπει να γίνουν N^2 μιγαδικοί πολλαπλασιασμοί και N(N-1) μιγαδικές προσθέσεις.

2.2 Βάσης-2 Γρήγορος Μετασχηματισμός Fourier με αποδεκατισμό στο πεδίο της συχνότητας

(Radix-2 FFT with decimation in frequency algorithm)

Έστω ότι θέλουμε να υπολογίσουμε τον $N=2^v$ σημείων διακριτό μετασχηματισμό Fourier. Διασπάμε το άθροισμα του διακριτού μετασχηματισμού Fourier σε δύο αθροίσματα, ένα που περιέχει μόνο τους πρώτους N/2 μιγαδικούς αριθμούς της ακολουθίας εισόδου και σε ένα με του υπόλοιπους N/2 μιγαδικούς αριθμούς. Έτσι έχουμε:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{kn} + \sum_{n=\frac{N}{2}}^{N-1} x(n) W_N^{kn} = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{kn} + W_N^{k\frac{N}{2}} \sum_{n=0}^{\frac{N}{2}-1} x(n+\tfrac{N}{2}) W_N^{kn}$$

Όμως $W_N^{k\frac{N}{2}} = e^{-jk\pi} = (-1)^k$ και άρα μπορούμε να γράψουμε

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + (-1)^k x(n+\tfrac{N}{2}) \right] W_N^{kn}$$

Τώρα χωρίζουμε τα X(k) σε άρτια και περιττά δείγματα και έχουμε

$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x(n+\frac{N}{2}) \right] W_N^{2kn}$ όπου $k = 0,1,...,\frac{N}{2}-1$

και

$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) - x(n+\frac{N}{2}) \right] W_N^{(2k+1)n}$ όπου $k = 0,1,...,\frac{N}{2}-1$

και λαμβάνοντας υπόψη ότι $W_N^2 = W_{\frac{N}{2}}$ έχουμε

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x(n+\tfrac{N}{2}) \right] W_{\frac{N}{2}}^{kn} \quad \text{όπου} \quad k = 0,1,...,\frac{N}{2}-1$$

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left[\left[x(n) - x(n+\tfrac{N}{2}) \right] W_N^n \right] W_{\frac{N}{2}}^{kn} \quad \text{όπου} \quad k = 0,1,...,\frac{N}{2}-1$$

Αν τώρα ορίσουμε τις ακολουθίες $g_1(n)$ και $g_2(n)$ ως

$$g_1(n) = x(n) + x(n+\tfrac{N}{2})$$

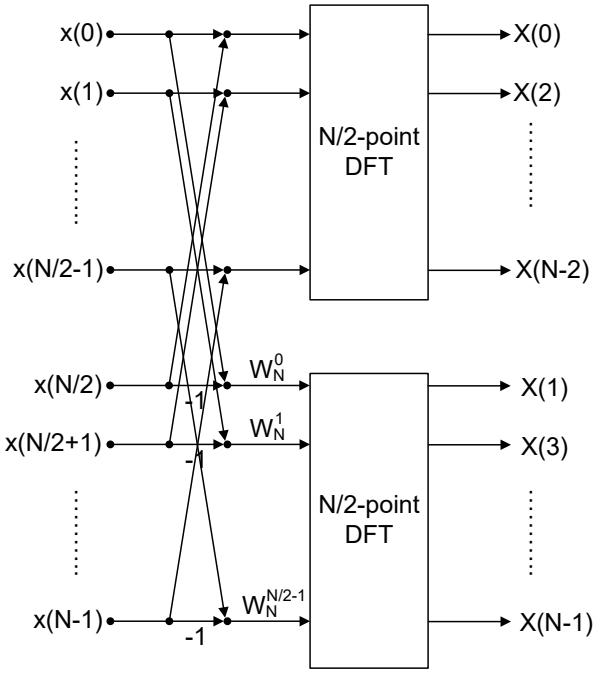
$$g_2(n) = \left[x(n) - x(n+\tfrac{N}{2}) \right] W_N^n \quad \text{όπου} \quad n = 0,1,...,\frac{N}{2}-1$$

τότε έχουμε ότι

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} g_1(n) W_{\frac{N}{2}}^{kn}$$

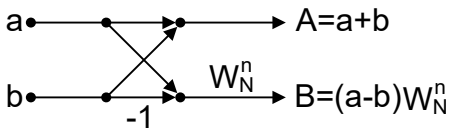
$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} g_2(n) W_{\frac{N}{2}}^{kn}$$

Επομένως ο υπολογισμός των ακολουθιών $g_1(n)$ και $g_2(n)$, με βάση τις παραπάνω σχέσεις, μας ανάγει το πρόβλημα του υπολογισμού του N σημείων διακριτού μετασχηματισμού Fourier στον υπολογισμό δύο N/2 σημείων διακριτών μετασχηματισμών Fourier με σημεία εισόδου τις ακολουθίες αυτές όπως φαίνεται στο Σχήμα 2.1.



Σχήμα 2.1 Αναγωγή του υπολογισμού βάσης-2 γρήγορου μετασχηματισμού Fourier N σημείων στον υπολογισμό δύο ίδιων μετασχηματισμών Fourier των N/2 σημείων

Παρατηρούμε ότι η βασική υπολογιστική διαδικασία γίνεται απ' την μονάδα που φαίνεται στο Σχήμα 2.2 και η οποία είναι γνωστή ως διαδικασία butterfly.

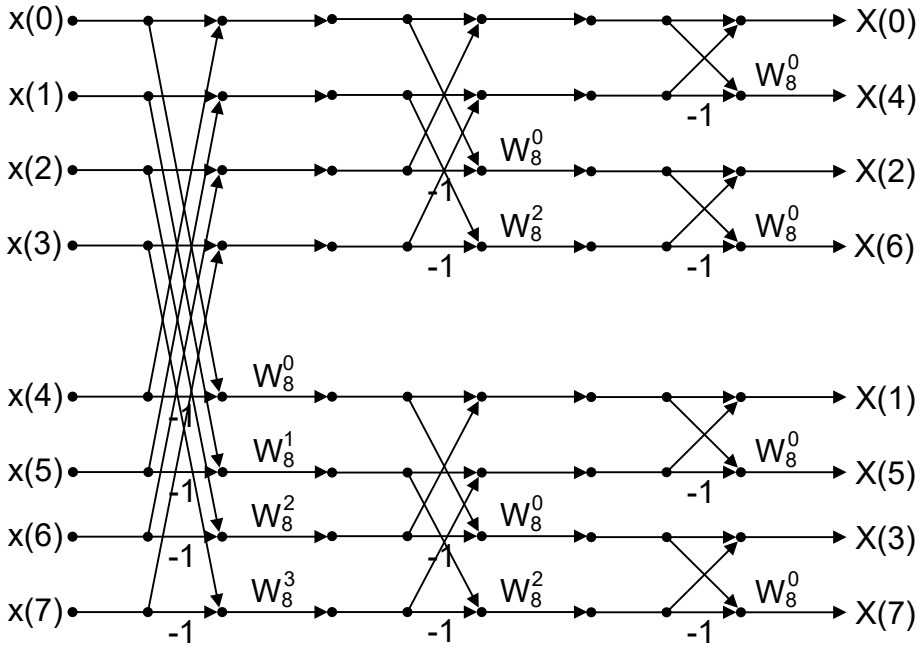


Σχήμα 2.2 Διαδικασία butterfly

Η υπολογιστική διαδικασία που περιγράψαμε παραπάνω μπορεί να επαναληφθεί για τον υπολογισμό των δύο N/2 σημείων διακριτών μετασχηματισμών Fourier, και χρησιμοποιώντας την επαναληπτικά, να οδηγηθούμε στον υπολογισμό του N σημείων διακριτού μετασχηματισμού Fourier που επιδιώκουμε.

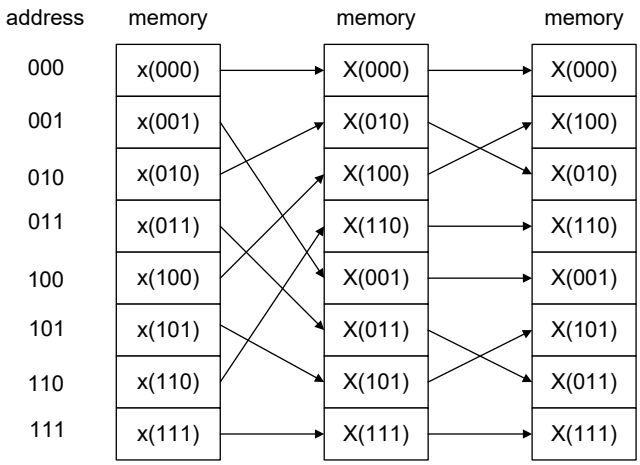
Η όλη διαδικασία περιέχει $\nu = \log_2 N$ στάδια διαχωρισμού που το καθένα εμπεριέχει N/2 διαδικασίες butterfly. Επομένως για τον υπολογισμό του N σημείων διακριτού μετασχηματισμού Fourier χρειάζονται $\frac{N}{2}\log_2 N$ μιγαδικοί πολλαπλασιασμοί και $N\log_2 N$ μιγαδικές προσθέσεις.

Για να έχουμε μια εποπτική εικόνα του αλγορίθμου που περιγράψαμε φαίνεται στο Σχήμα 2.3 ο υπολογισμός του 8 σημείων διακριτού μετασχηματισμού Fourier.



Σχήμα 2.3 Υπολογισμός του βάσης-2 γρήγορου μετασχηματισμού Fourier 8 σημείων

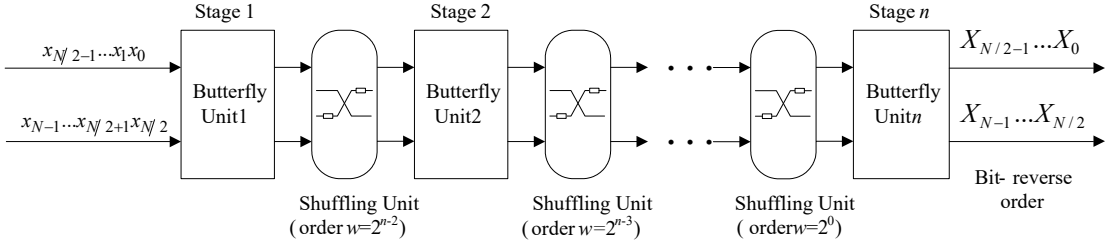
Μια και κάθε διαδικασία butterfly γίνετε σε ένα ζεύγος τιμών (a,b) που παράγει μια όμοια έξοδο (A,B) όπου μόνο αυτή μας χρειάζεται για τον παραπέρα υπολογισμό, μπορούμε να «αποθηκεύουμε» το (A,B) στην ίδια «θέση» με αυτήν του (a,b). Απ' το προηγούμενο σχήμα παρατηρούμε ότι τα δεδομένα εισέρχονται με την κανονική σειρά ενώ τα αποτελέσματα του διακριτού μετασχηματισμού Fourier βγαίνουν με αντεστραμμένη σειρά των bit (bit-reversed) διότι σε κάθε στάδιο διαχωρισμού των εξόδων των διαδικασιών butterfly σε άρτια και περιττή θέση έχουμε ένα είδος ανακατέματος των δειγμάτων. Αυτό μπορεί να κατανοηθεί παρατηρώντας το Σχήμα 2.4 όπου φαίνονται με δυαδική μορφή η «διεύθυνση» στην μνήμη και η σειρά των σημείων όπου είναι φανερή η αντιστροφή των bit στην σειρά των στοιχείων της εξόδου του διακριτού μετασχηματισμού Fourier.



Σχήμα 2.4 Οι έξοδοι κάθε σταδίου butterfly με δείκτες εκφρασμένους σε δυαδική μορφή κατά τον υπολογισμό του βάσης-2 γρήγορου μετασχηματισμού Fourier 8 σημείων

3 ΥΛΟΠΟΙΗΣΗ ΜΕ ΒΑΣΗ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΒΑΣΗΣ-2 ΓΡΗΓΟΡΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ FOURIER ΜΕ ΑΠΟΔΕΚΑΤΙΣΜΟ ΣΤΟ ΠΕΔΙΟ ΤΗΣ ΣΥΧΝΟΤΗΤΑΣ

Επομένως μπορούμε να χρησιμοποιήσουμε για κάθε στάδιο διαχωρισμού μια μονάδα butterfly που να εκτελεί την λειτουργία butterfly σειριακά για τα δεδομένα εισόδου στο αντίστοιχο στάδιο. Φυσικά πρέπει να γίνετε και η σωστή αναδιάταξη των στοιχείων εξόδου τα οποία θα εισαχθούν σαν δεδομένα στην επόμενη μονάδα butterfly. Αυτό μπορεί να γίνει όπως φαίνεται στο Σχήμα 3.1.



Σχήμα 3.1 Μορφή υλοποίησης με βάση των αλγόριθμο βάσης-2 γρήγορου μετασχηματισμού Fourier με αποδεκατισμό στο πεδίο της συχνότητας

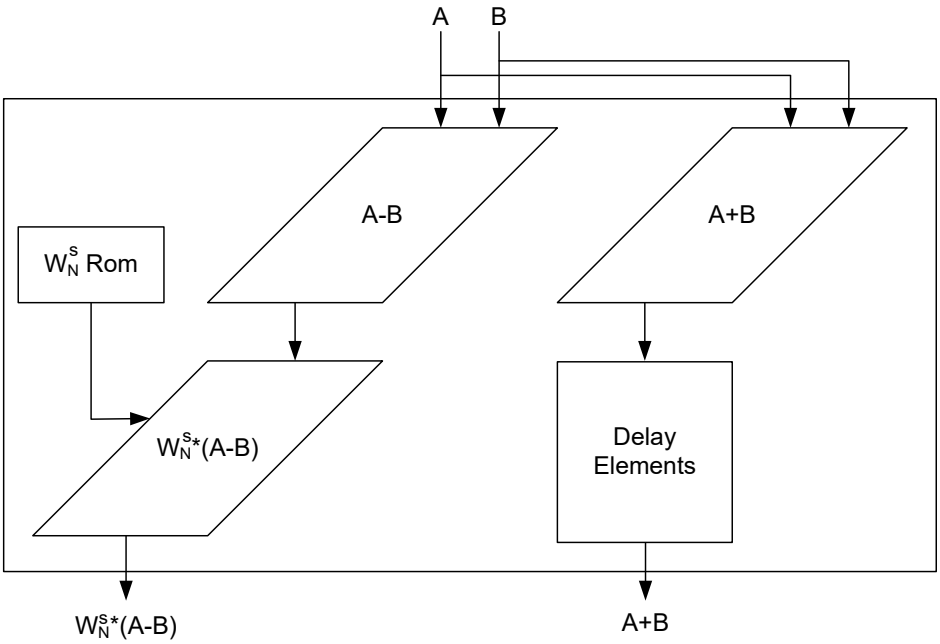
Κάθε μονάδα butterfly υλοποιεί την αντίστοιχη λειτουργία για κάθε στάδιο για όλα τα N δεδομένα εισόδου, ενώ η μονάδα αναδιάταξης (Shuffling Unit) υλοποιεί την αναδιάταξη των στοιχείων ώστε αυτά να εισάγονται με την σωστή σειρά στο επόμενο στάδιο. Με αυτόν τον τρόπο βλέπουμε ότι τα δεδομένα εισόδου μπαίνουν διαχωρισμένα στην μονάδα butterfly του πρώτου σταδίου και τα δεδομένα εξόδου βγαίνουν με αντεστραμμένη σειρά των bit (bit-reverse order) στην έξοδο της αρχιτεκτονικής.

3.1 Μονάδα Butterfly (Butterfly Unit)

3.1.1 Εισαγωγή

Η μονάδα butterfly όπως είδαμε θα πρέπει να υλοποιεί την διαδικασία butterfly και θα έχει την μορφή που φαίνεται στο Σχήμα 3.2.

Επειδή τα δεδομένα εξόδου μιας μονάδας butterfly είναι η είσοδος της επόμενης και για απλοποίηση της μονάδας θεωρούμε ότι τα δεδομένα εισόδου και εξόδου θα πρέπει να έχουν την ίδια ακρίβεια έστω N bit. Θα χρησιμοποιήσουμε για τα A και B αριθμούς σε μορφή συμπληρώματος ως προς 2 των N bit, για πραγματικό και φανταστικό μέρος. Τότε η έξοδος του αθροιστή και του αφαιρέτη θα αποτελείται από μιγαδικούς αριθμούς των N+1 bit, για πραγματικό και φανταστικό μέρος. Αντίστοιχα αν στην ROM έχουμε αποθηκεύσει τις δυνάμεις του W_N με ακρίβεια NW bit, πάλι με αριθμούς σε μορφή συμπληρώματος ως προς 2, για πραγματικό και φανταστικό μέρος, τότε η έξοδος του μιγαδικού πολλαπλασιαστή θα πρέπει να έχει N+NW+1 bit ώστε να μην χάσουμε σε ακρίβεια.



Σχήμα 3.2 Μορφή μονάδας butterfly

Όμως οι παράγοντες αναστροφής (twiddle factors) βρίσκονται στο διάστημα [-1,1] και επειδή θα είναι αποθηκευμένοι στην ROM με ακρίβεια NW bit μπορούμε να τους φανταστούμε ως ακέραιους της μορφής

$$W_N^n = 2^{NW-2}\left(\cos\theta + j\sin\theta\right)$$

Επομένως η έξοδος του μιγαδικού πολλαπλασιαστή θα είναι της μορφής

$$W_N^n \cdot \left(A-B\right) = 2^{NW-2}\left(\cos\theta + j\sin\theta\right)\left(d_{re} + jd_{im}\right) \Leftrightarrow$$
$$W_N^n \cdot \left(A-B\right) = 2^{NW-2}\left[\left(d_{re}\cos\theta - d_{im}\sin\theta\right) + j\left(d_{re}\sin\theta + d_{im}\cos\theta\right)\right]$$

όπου $d_{re}+jd_{im}$ η έξοδος του αφαιρέτη που όπως είπαμε τα d_{re} και d_{im} είναι των N+1 bit και επομένως $\left|d_{re}\right| \le 2^{N+1}$ και $\left|d_{im}\right| \le 2^{N+1}$. Άρα θα έχουμε για πραγματικό και φανταστικό μέρος αντίστοιχα

$$2^{NW-2}\left|d_{re}\cos\theta - d_{im}\sin\theta\right| \le 2^{NW-2}2^{N+1}\left|\cos\theta - \sin\theta\right| \le 2^{NW-2}2^{N+1}\sqrt{2}$$
$$2^{NW-2}\left|d_{re}\sin\theta + d_{im}\cos\theta\right| \le 2^{NW-2}2^{N+1}\left|\cos\theta + \sin\theta\right| \le 2^{NW-2}2^{N+1}\sqrt{2}$$

Θέλουμε η ακρίβεια να παραμείνει στα N bit και όπως είναι φανερό θα πρέπει, εκτός απ' την αποκοπή των τελευταίων NW-2 bits, να διαιρούμε σε κάθε μονάδα butterfly τα αποτελέσματα του πολλαπλασιαστή με $2\sqrt{2}$ ώστε να κρατιέται η ακρίβεια στα N bit. Για την απλοποίηση του κυκλώματος η διαίρεση θα γίνει με το 4 ώστε να έχουμε απλές ολισθήσεις. Αντίστοιχα στον αθροιστή για να βρίσκονται όλα τα δεδομένα στην ίδια τάξη θα πρέπει να γίνετε ανάλογη διαίρεση. Στην τελευταία μονάδα butterfly, όπου θα δειχθεί ότι δεν υπάρχει μιγαδικός πολλαπλασιαστής και τα αποτελέσματα του μιγαδικού αθροιστή και αφαιρέτη οδηγούνται στην έξοδο του κυκλώματος, για να διατηρούμε την ακρίβεια σταθερή σε N bit θα πρέπει να κάνουμε διαίρεση με το 2. Συνολικά λοιπόν τα αποτελέσματα στην έξοδο της υλοποίησης του FFT θα έχουν διαιρεθεί με $2 \cdot 4^{\log_2 N - 1} = \frac{N^2}{2}$.

3.2 Αριθμητικά κυκλώματα μέσα στην μονάδα butterfly

3.2.1 Μιγαδικός Αθροιστής

Τον μιγαδικό αθροιστή τον υλοποιούμε με τον μιγαδικό αθροιστή συνεχούς διοχέτευσης (pipelining) όπως ακριβώς τον περιγράψαμε στην παράγραφο 1.1.4. Τα δεδομένα εισόδου, μήκους N bit, εισάγονται στον μιγαδικό αθροιστή και στην έξοδο παίρνουμε το αποτέλεσμα της άθροισης το οποίο έχει μήκος N+1 bit. Από αυτό, εξαιτίας της διαίρεσης με το 4 που πρέπει να γίνει, δεν λαμβάνουμε υπόψη τα 2 LSB του αποτελέσματος. Φυσικά πρέπει στην έξοδο του κάθε butterfly η ακρίβεια πρέπει να παραμένει σταθερή στα N bit, οπότε μετά τα στοιχεία καθυστέρησης θα πρέπει να γίνει επέκταση προσήμου.

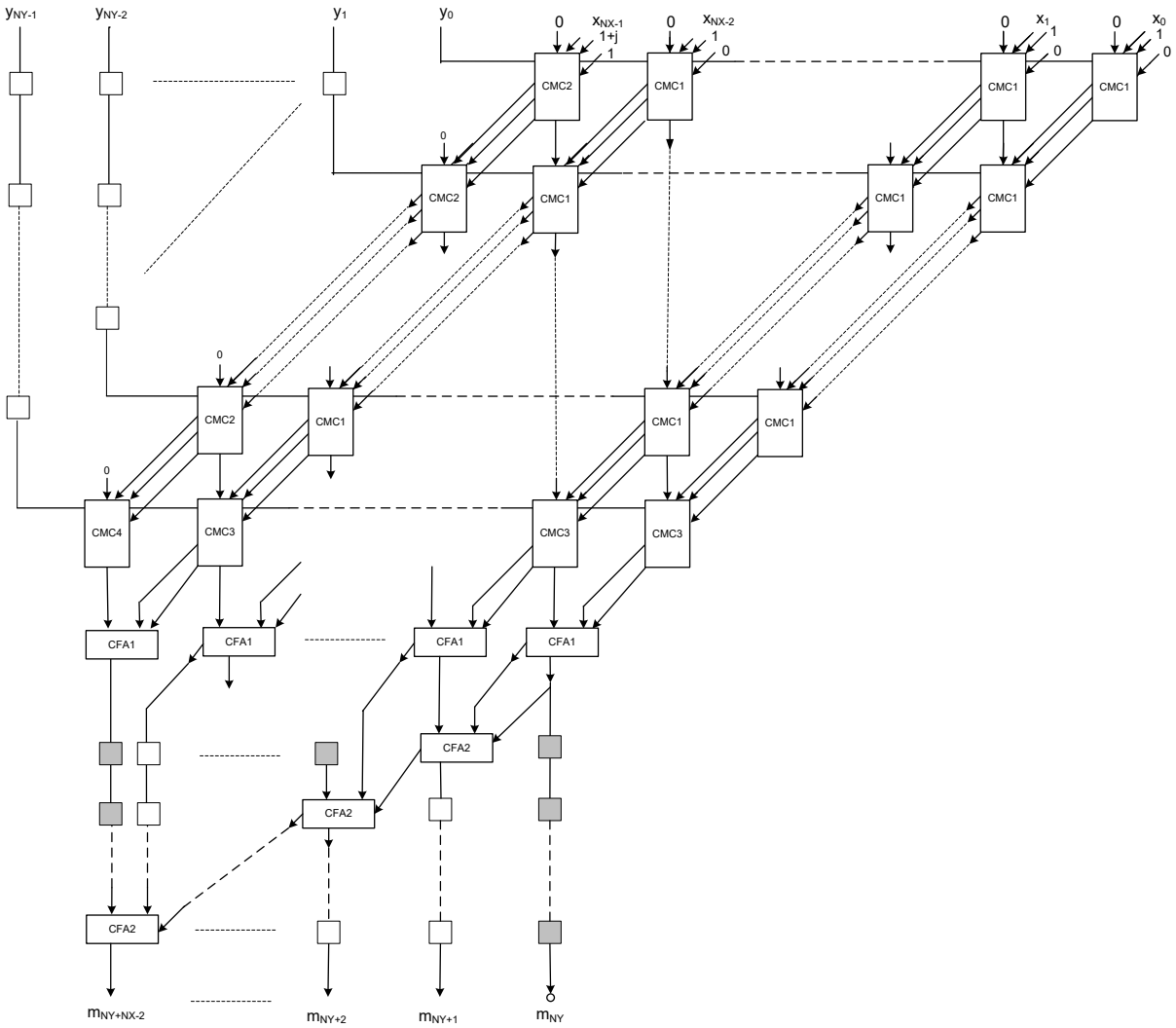
3.2.2 Μιγαδικός Αφαιρέτης

Τον μιγαδικό αφαιρέτη τον υλοποιούμε με τον μιγαδικό αφαιρέτη συνεχούς διοχέτευσης (pipelining) όπως ακριβώς τον περιγράψαμε στην παράγραφο 1.2.4. Τα δεδομένα εισόδου, μήκους N bit, εισάγονται στον μιγαδικό αφαιρέτη και στην έξοδο παίρνουμε το αποτέλεσμα της αφαίρεσης το οποίο έχει μήκος N+1 bit και το οποίο οδηγείται στον μιγαδικό πολλαπλασιαστή.

3.2.3 Μιγαδικός Πολλαπλασιαστής

Τον μιγαδικό πολλαπλασιαστή τον υλοποιούμε με βάση τον μιγαδικό πολλαπλασιαστή με σώσιμο κρατουμένου και συνεχούς διοχέτευσης που περιγράψαμε στην παράγραφο 1.3.7. Τα δεδομένα εισόδου είναι το αποτέλεσμα του μιγαδικού αφαιρέτη, το οποίο έχει μήκος N+1 bit, και η έξοδος την μνήμης ROM με τον αντίστοιχο παράγοντα αναστροφής (twiddle factor), ο οποίος έχει μήκος NW bit. Από όσα αναφέρθηκαν προηγουμένως είναι φανερό ότι στην έξοδο δεν μας χρειάζονται τα τελευταία NW bit και τα οποία θα αποκοπούν. Επίσης δεν μας χρειάζεται και τα δύο MSB της εξόδου του μιγαδικού πολλαπλασιαστή διότι οι παράγοντες αναστροφής (twiddle factors) που αποθηκεύτηκαν στην ROM είναι μεγαλύτεροι απ' τον μικρότερο αρνητικό αριθμό που μπορεί να παρασταθεί, σε πραγματικό και φανταστικό μέρος. Έτσι το γινόμενο δεν επηρεάζει ποτέ τα δύο MSB και άρα μπορούμε να απαλείψουμε τις δύο τελευταίες βαθμίδες του πολλαπλασιαστή.

Επομένως ο μιγαδικός πολλαπλασιαστής που θα χρησιμοποιήσουμε θα έχει την μορφή που φαίνεται στο Σχήμα 3.3.



Σχήμα 3.3 Μιγαδικός πολλαπλασιαστής που θα χρησιμοποιηθεί μέσα στην μονάδα butterfly

Τα δεδομένα απ' την έξοδο του μιγαδικού αφαιρέτη εισέρχονται στον μιγαδικό πολλαπλασιαστή απ' την μεριά των x και οι παράγοντες αναστροφής (twiddle factors) απ' την μνήμη ROM στην πλευρά των y. Φυσικά δεν υπάρχει πρόβλημα να βάζαμε τις εισόδους ανάποδα αλλά με αυτόν τον τρόπο έγινε η περιγραφή σε VHDL που παρατίθεται στο παράρτημα.

3.3 Μονάδα εξαγωγής των παραγόντων αναστροφής

Αν υπολογίζουμε τον NP σημείων FFT, τότε στη μονάδα butterfly που βρίσκετε στο S στάδιο, οι παράγοντες αναστροφής (twiddle factors) θα έχουν την μορφή

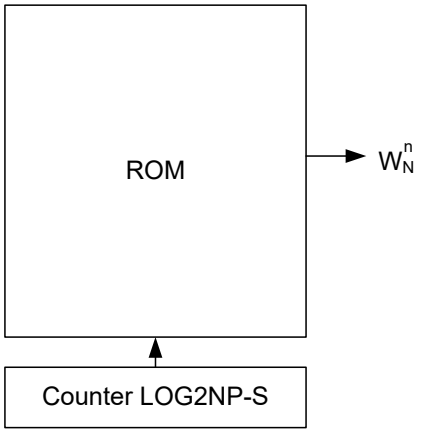
$$W_N^n = 2^{NW-2}\left(\cos\theta + j\sin\theta\right)$$

όπου n = 0...NP/2^S-1 και $\theta = -2^{S-1} \cdot n \cdot 2\pi / NP$.

Τους παράγοντες αναστροφής (twiddle factors) τους αποθηκεύουμε, ξεχωριστά για κάθε στάδιο, σε μια μνήμη ROM με μέγεθος NP/2^S και σε μορφή συμπληρώματος ως προς 2 για πραγματικό και φανταστικό μέρος. Την διεύθυνση απ' την οποία εξάγονται τα δεδομένα μπορούμε να την παίρνουμε, πολύ απλά, από έναν μετρητή ο οποίος θα παίρνει κατάλληλη αρχική τιμή ώστε να συγχρονίζετε κατάλληλα με τα δεδομένα εισόδου και να γίνονται οι σωστοί πολλαπλασιασμοί. Η καθυστέρηση αυτή είναι

(καθυστέρηση αφαιρέτη)+(S-1)(καθυστέρηση μονάδας butterfly)

Η καθυστέρηση αυτή δεν επηρεάζετε απ' τις μονάδες αναδιάταξης γιατί όπως θα δειχθεί αυτή είναι πολλαπλάσια του μεγέθους της ROM σε κάθε προηγούμενο στάδιο.



Σχήμα 3.4 Μονάδα ROM μέσα στην μονάδα butterfly που τροφοδοτεί τον μιγαδικό πολλαπλασιαστή με τους παράγοντες αναστροφής (twiddle factors)

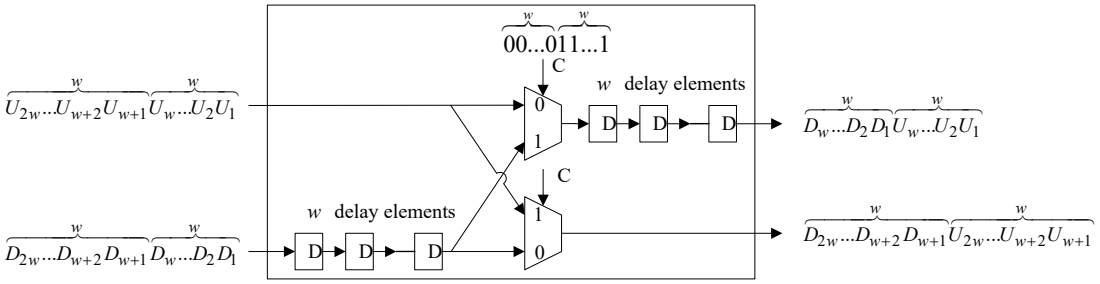
Επίσης είναι φανερό ότι στο τελευταίο στάδιο η μονάδα butterfly πολλαπλασιάζει μόνο με έναν παράγοντα αναστροφής (twiddle factor) και αυτός είναι η μονάδα. Επομένως στην μονάδα butterfly που βρίσκετε στο τελευταίο στάδιο δεν χρειάζεται να βάλουμε μιγαδικό πολλαπλασιαστή.

3.4 Μονάδα Αναδιάταξης (Shuffling Unit)

Όπως είδαμε, για τον υπολογισμό του γρήγορου μετασχηματισμού Fourier N σημείων (N-Point FFT), χωρίσαμε τα N δεδομένα εισόδου σε δύο ακολουθίες των N/2 σημείων και τα οποία τα εισάγουμε σειριακά στην πρώτη μονάδα butterfly. Από εκεί ο υπολογισμός συνεχίζετε με όμοιο τρόπο αλλά ξεχωριστά για τα N/2 δεδομένα εξόδου της πάνω βαθμίδας της μονάδας butterfly και ξεχωριστά για τα N/2 δεδομένα της κάτω. Θα πρέπει τα δεδομένα της πάνω βαθμίδας εξόδου να χωριστούν σε 2 ακολουθίες των N/4 σημείων η κάθε μία. Τα πρώτα N/4 σημεία θα πρέπει να οδηγηθούν στην πάνω είσοδο του επόμενου σταδίου butterfly και τα επόμενα N/4 σημεία στην κάτω. Κατόπιν η ίδια διαδικασία θα πρέπει να γίνει και για τα N/2 δεδομένα εξόδου του πρώτου σταδίου butterfly της κάτω βαθμίδας. Αυτό συνεχίζετε επαναληπτικά μέχρι το τελευταίο στάδιο butterfly όπου τα δεδομένα έχουν διασπαστεί σε ακολουθίες με δύο μόνο αριθμούς.

Πιο γενικά αν βρισκόμαστε στο s στάδιο μονάδας butterfly, στις εξόδους έχουμε ακολουθίες των $2^{\log N-s}$ σημείων και πρέπει η κάθε μια να χωριστεί σε ακολουθίες των $w=2^{\log N-s-1}$ σημείων ώστε να οδηγηθούν στο επόμενο στάδιο.

Η μονάδα που μπορεί να το κάνει αυτό φαίνεται στο Σχήμα 3.5.



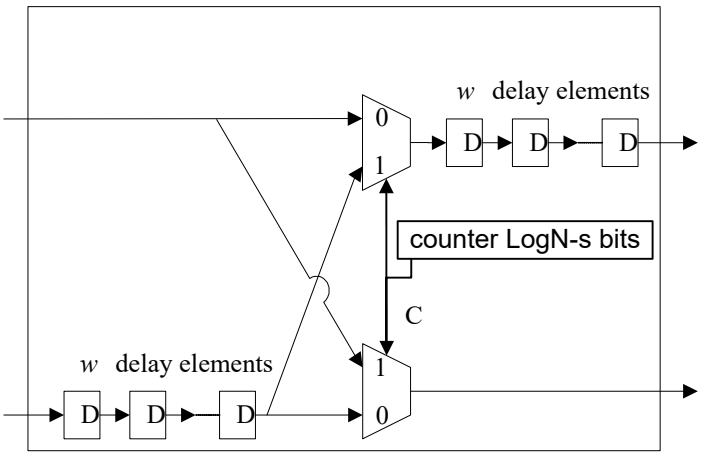
Σχήμα 3.5 Μορφή μονάδας αναδιάταξης

Το σήμα C είναι υπεύθυνο για την σωστή αναδιάταξη των δεδομένων εισόδου. Αρχικά παίρνει την τιμή '0' για w χτύπους του ρολογιού ώστε τα δεδομένα της προηγούμενης μονάδας butterfly, απ' την πάνω έξοδο, να καθυστερήσουν w χτύπους του ρολογιού και να οδηγηθούν στην επόμενη μονάδα butterfly. Κατόπιν το σήμα C παίρνει την τιμή '1' ώστε τα δεδομένα απ' την πάνω έξοδο της προηγούμενης μονάδας butterfly να οδηγηθούν στην κάτω είσοδο της επόμενης μονάδας και συγχρόνως τα δεδομένα απ' την κάτω έξοδο της προηγούμενης μονάδας, που έχουν καθυστερήσει ήδη w κύκλους, να οδηγούνται προς την πάνω είσοδο της επόμενης μονάδας butterfly.

Αν θέλουμε να υπολογίσουμε τον μετασχηματισμό FFT N σημείων και το στάδιο της επόμενης μονάδας butterfly απ' την μονάδα αναδιάταξης (Shuffling Unit) είναι το s, τότε το σήμα C το υλοποιούμε πολύ απλά με το MSB ενός μετρητή των logN-s bit των οποίο τον αρχικοποιούμε με τέτοιο τρόπο ώστε να ληφθεί υπόψη η καθυστέρηση των δεδομένων απ' τις προηγούμενες βαθμίδες και να γίνει σωστά ο συγχρονισμός η οποία είναι

$$(\text{s}-1)\cdot(\text{καθυστέρηση μονάδας butterfly}) \bmod 2^{\log N-s}.$$

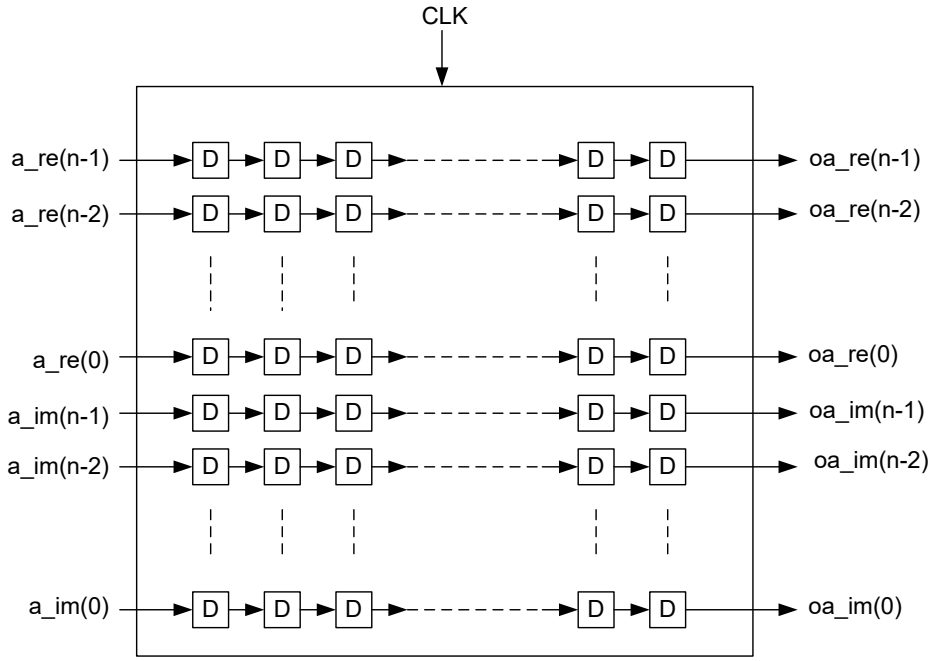
Η καθυστέρηση απ' τις προηγούμενες μονάδες αναδιάταξης δεν λαμβάνετε υπόψη διότι το w υποδιπλασιάζεται μετά από κάθε στάδιο.
Έτσι η μονάδα αναδιάταξης θα έχει την μορφή που φαίνεται στο Σχήμα 3.6.



Σχήμα 3.6 Μονάδα αναδιάταξης

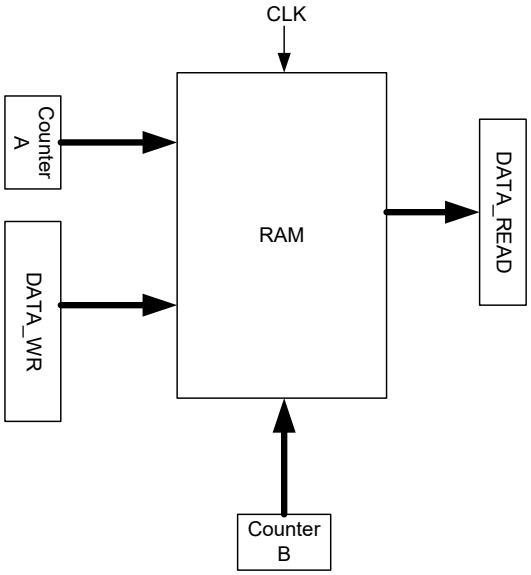
3.5 Στοιχεία καθυστέρησης (Delay Elements)

Τα στοιχεία καθυστέρησης, τα οποία υπάρχουν μέσα στην μονάδα butterfly καθώς και στην μονάδα αναδιάταξης, μπορούν να υλοποιηθούν με απλούς καταχωρητές ολίσθησης από D flip-flop, για πραγματικό και φανταστικό μέρος, όπως φαίνεται στο Σχήμα 3.7.

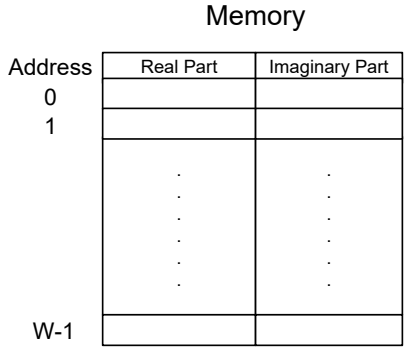


Σχήμα 3.7 Μιγαδικός καταχωρητής ολίσθησης

Επειδή αυτοί οι καταχωρητές ολίσθησης είναι αρκετά μεγάλη, ιδιαίτερα για τις πρώτες μονάδες αναδιάταξης (Shuffling Units), μπορεί να είναι προτιμότερο να υλοποιηθούν με ένα στοιχείο μνήμης RAM με 2 θύρες όπως φαίνεται στο Σχήμα 3.8.



Σχήμα 3.8 Μνήμη RAM με 2 θύρες σαν καταχωρητής ολίσθησης

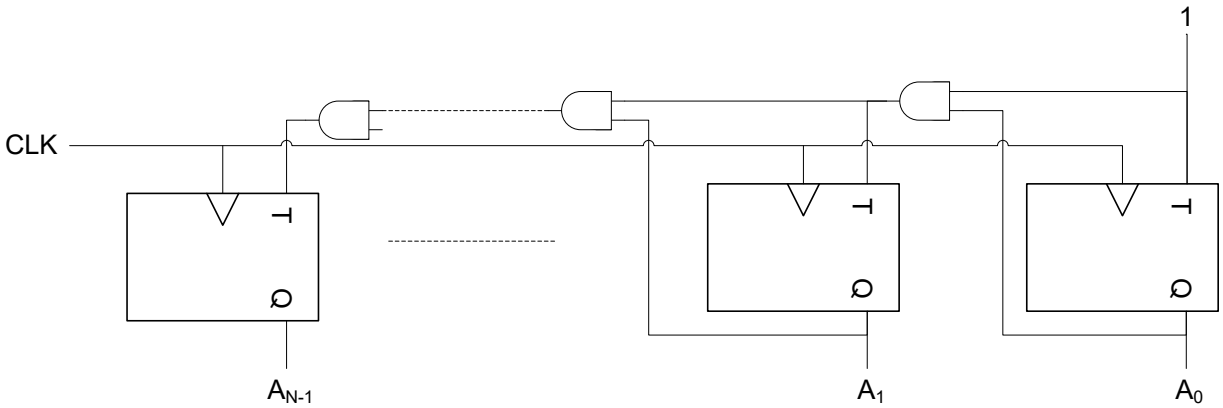


Σχήμα 3.9 Εσωτερικό μνήμης RAM

Ο μετρητής A (Counter A) δείχνει στην διεύθυνση της μνήμης που πρέπει το δεδομένο εισόδου (DATA_WR) να αποθηκευτεί. Αντίστοιχα ο μετρητής B (Counter B) δείχνει στην διεύθυνση της μνήμης απ' την οποία πρέπει να διαβάσουμε το δεδομένο εξόδου (DATA_READ). Πάντα ο μετρητής B (Counter B) δείχνει μία θέση μπροστά απ' τον μετρητή A (Counter A). Το μέγεθος της μνήμης είναι ίδιο με την καθυστέρηση που θέλουμε να επιτύχουμε. Υλοποίηση σε γλώσσα VHDL της μνήμης που περιγράφηκε υπάρχει στο παράρτημα.

3.6 Μετρητές (Counters)

Όλοι οι μετρητές που θα χρησιμοποιήσουμε έχουν την μορφή που φαίνεται στο Σχήμα 3.10. Η λειτουργία του είναι πολύ απλή. Απλά αλλάζει η κατάσταση μιας βαθμίδας μόνο αν όλες η προηγούμενες βαθμίδες είναι στο λογικό 1.



Σχήμα 3.10 Σύγχρονος Δυαδικός μετρητής

3.7 Έλεγχος

Με βάση όσα είπαμε στις προηγούμενες παραγράφους αυτού του κεφαλαίου, περιγράψαμε σε γλώσσα VHDL την προτεινόμενη υλοποίηση του βάσης-2 γρήγορου μετασχηματισμού Fourier, η οποία παρατίθεται στο παράρτημα D. Εκεί βρίσκεται και το αρχείο FFT_Unit_TB.vhd το οποίο περιέχει το Test Bench με το οποίο ελέγχουμε την περιγραφή.

Σε αυτό το Test Bench δημιουργείται μία μονάδα FFT η οποία υπολογίζει τον βάσης-2 γρήγορο μετασχηματισμό Fourier NP=2^5=32 σημείων τα οποία αποτελούνται από μιγαδικούς αριθμούς των N=13 bit σε πραγματικό και φανταστικό μέρος. Επίσης οι παράγοντες αναστροφής (twiddle factors) είναι αποθηκευμένοι στις μνήμες ROM σε μιγαδικούς αριθμούς των NW=12 bit σε πραγματικό και φανταστικό μέρος. Χρησιμοποιείται μια μονάδα FFT η οποία υπολογίζει σχετικά λίγα σημεία ώστε να είναι ευδιάκριτα τα διαγράμματα που θα παρουσιαστούν.

Τα δεδομένα εισόδου υπολογίζονται με βάση την συνάρτηση

$$f(t) = \left(2^{N-3} - 1\right) \cdot \cos\left(-2\pi t/3\right) + \left(2^{N-3} - 1\right) \cdot \cos\left(-2\pi t/11\right)$$

Με την εκτέλεση του test bench όλα τα δεδομένα εισόδου και εξόδου γράφονται στο αρχείο outputs.txt με την μορφή:

A_Re,A_Im,B_Re,B_Im,AO_Re,AO_Im,BO_Re,BO_Im

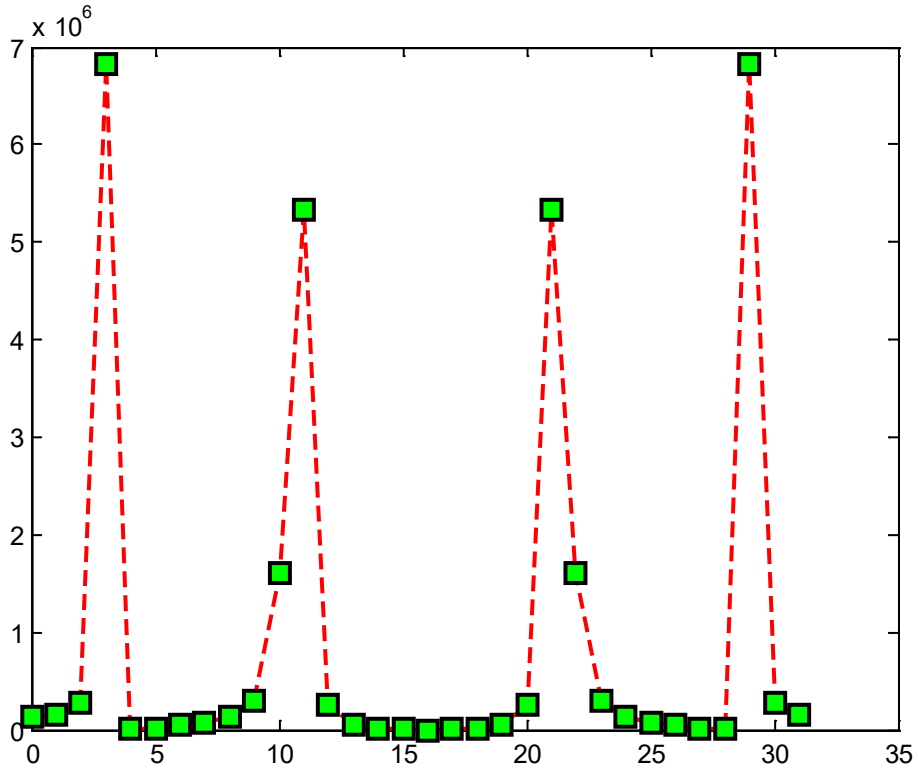
Χρησιμοποιώντας το αρχείο outputs.txt και εκτελώντας το script check1, που παρουσιάζετε παρακάτω, μπορούμε να εισάγουμε όλα τα δεδομένα στο MATLAB να κάνουμε πράξεις και να συγκρίνουμε τα αποτελέσματα.

check1.m

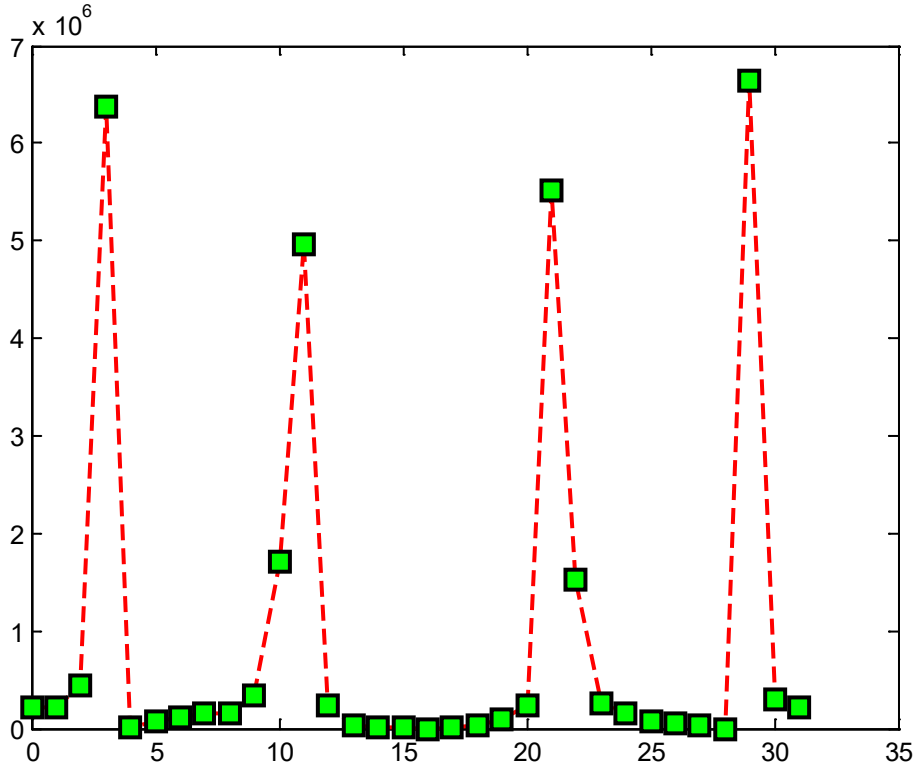
```
M = csvread('outputs.txt');
NW=12;
N=13;
LOG2NP = 5;
NP=2^LOG2NP;
Y1=[];
Y2=[];
X=[];
% put input numbers at array C in right order and calculate fft%
START=(LOG2NP-1)*(N+N+NW)+(N)+(2^(LOG2NP-1)-1)+1;
deigmata=1;
for i=START:NP/2:START+deigmata*NP/2-1
    A = M(i:i+NP/2-1,1)+j*M(i:i+NP/2-1,2);
    B = M(i:i+NP/2-1,3)+j*M(i:i+NP/2-1,4);
    C = cat(1,A,B);
    X = [X;C];
    Y1 = cat(1,Y1,fft(C,NP));
    AO = M(i:i+NP/2-1,5)+j*M(i:i+NP/2-1,6);
    BO = M(i:i+NP/2-1,7)+j*M(i:i+NP/2-1,8);
    for k=1:NP/2
        CO(2*k-1,1)=AO(k,1);
        CO(2*k,1)=BO(k,1);
    end
    Y2 = cat(1,Y2,bitrevorder(CO));
end
Y2=Y2*NP^2/2;
```

```
%Y1=round(Y1);
%Y1=fix(Y1);
%Y1=ceil(Y1);
%Y1=floor(Y1);

DY=Y1-Y2;

deigma=1;  % deigma <= deigmata
PY1 = Y1(1+(deigma-1)*NP:deigma*NP).* conj(Y1(1+(deigma-1)*NP:deigma*NP)) / NP;
PY2 = Y2(1+(deigma-1)*NP:deigma*NP).* conj(Y2(1+(deigma-1)*NP:deigma*NP)) / NP;
DPY = PY1-PY2;
f = (0:NP-1);

signal=Y1(1+(deigma-1)*NP:deigma*NP);
noise=Y2(1+(deigma-1)*NP:deigma*NP)-signal;
SNR=10*log10(sum(abs(signal).^2)/(sum(abs(noise).^2)))

figure(1);
set(1,'name','FFT Spectrum Calculated on Matlab');
plot(f,PY1(1:NP),'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(2);
set(2,'name','FFT Spectrum Calculated from implementaton');
plot(f,PY2(1:NP),'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(3);
set(3,'name','Input Data');
plot(f,real(X(1+(deigma-1)*NP:deigma*NP)),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(4);
set(4,'name','IFFT of FFT implementation output data ');
plot(f,real(ifft(Y2(1+(deigma-1)*NP:deigma*NP),NP)),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(5);
set(5,'name','IFFT of Matlab FFT calculation output data');
plot(f,real(ifft(Y1(1+(deigma-1)*NP:deigma*NP),NP)),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(6);
set(6,'name','Abs(Y1-Y2)');
plot(f,sqrt(abs(DY(1+(deigma-1)*NP:deigma*NP))),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
```

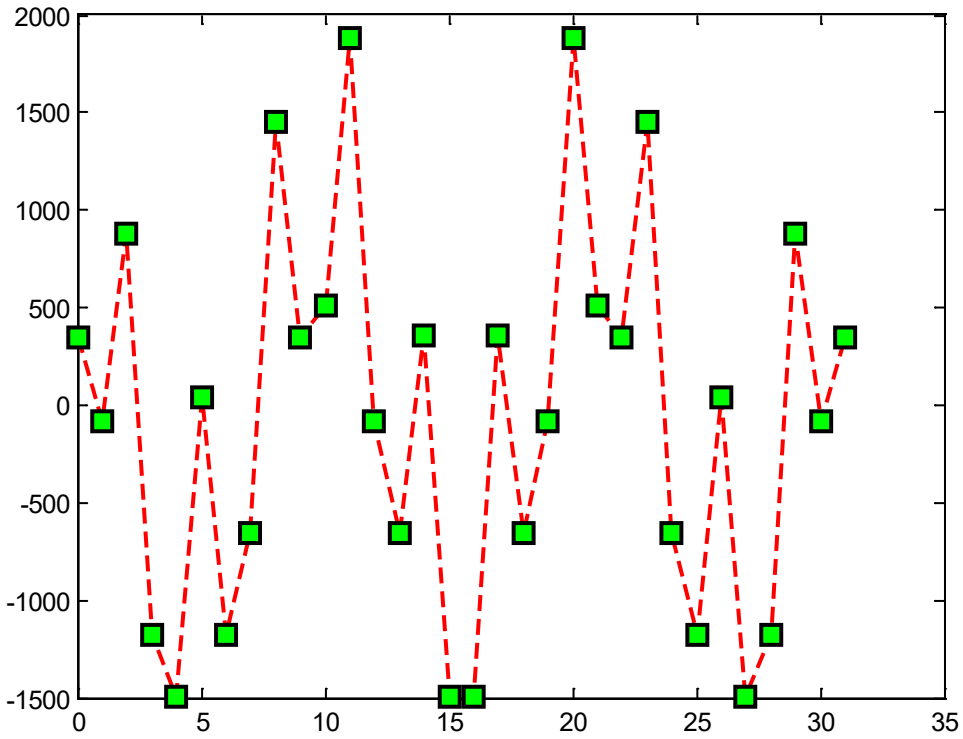
Με την εκτέλεση λοιπόν του script στο MATLAB παίρνουμε το Σχήμα 3.11 έως και το Σχήμα 3.16 απ' τα οποία παρατηρούμε την σωστή λειτουργία του κυκλώματος.



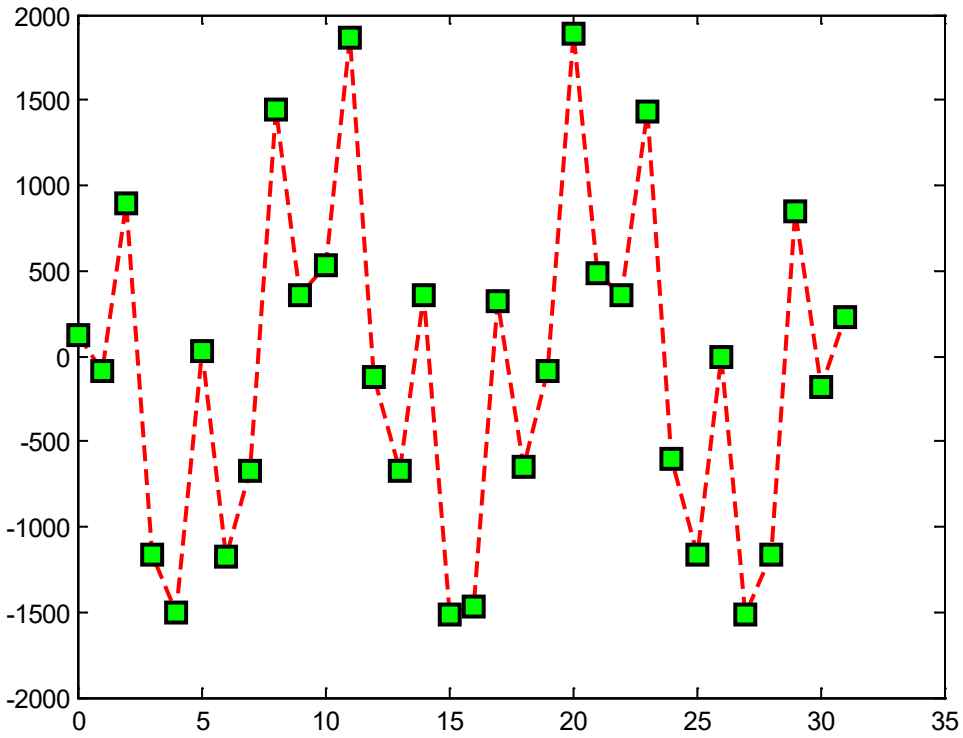
Σχήμα 3.11 Το φάσμα του μετασχηματισμού Fourier των δεδομένων εισόδου όπως το υπολόγισαν οι συναρτήσεις του MATLAB



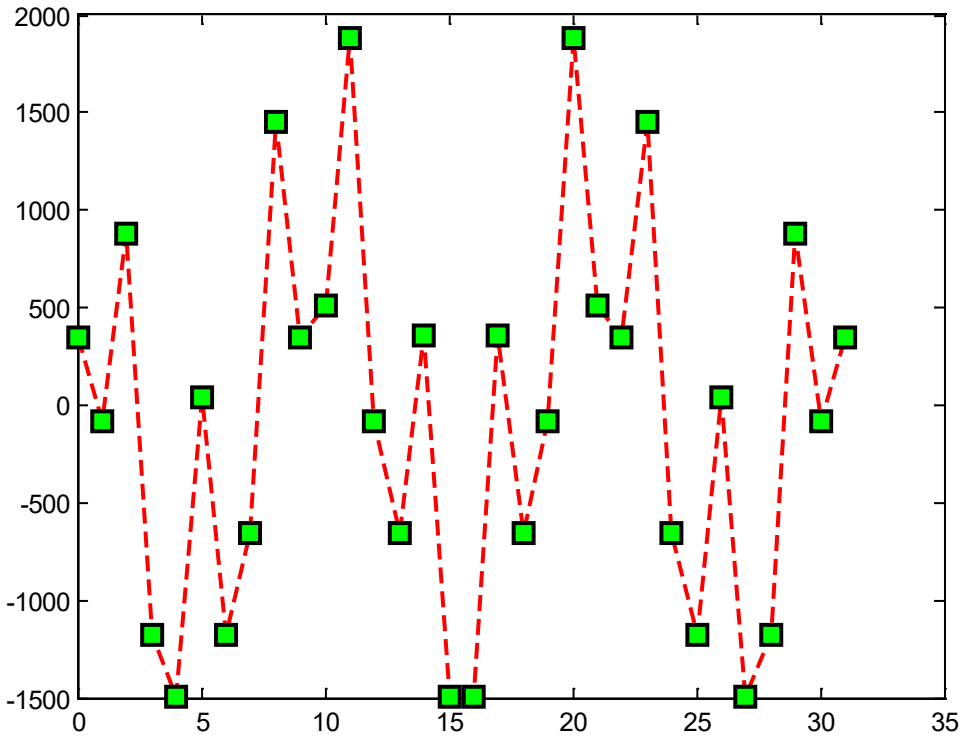
Σχήμα 3.12 Το φάσμα του μετασχηματισμού Fourier των δεδομένων εξόδου της συγκεκριμένης υλοποίησης στο test bench



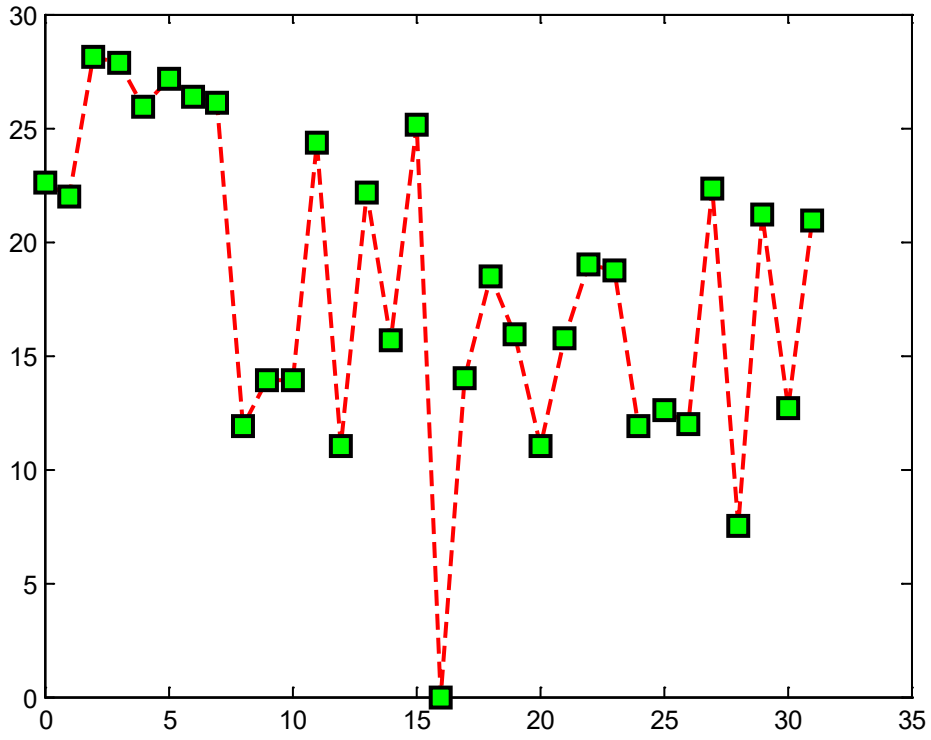
Σχήμα 3.13 Γραφική αναπαράσταση των δεδομένων εισόδου στην συγκεκριμένη υλοποίηση του FFT στο test bench



Σχήμα 3.14 Γραφική παράσταση του αντίστροφου μετασχηματισμού Fourier των δεδομένων εξόδου της συγκεκριμένης υλοποίησης στο test bench



Σχήμα 3.15 Γραφική παράσταση του αντίστροφου μετασχηματισμού Fourier που έγινε στα δεδομένα του γρήγορου μετασχηματισμού Fourier των δεδομένων εισόδου



Σχήμα 3.16 Γραφική παράσταση του μέτρου της διαφοράς των δύο υπολογισμών, του γρήγορου μετασχηματισμού Fourier των δεδομένων εισόδου, απ' την υλοποίηση και από το MATLAB

3.8 Υπολογισμός του σηματοθορυβικού λόγου (SNR) της υλοποίησης

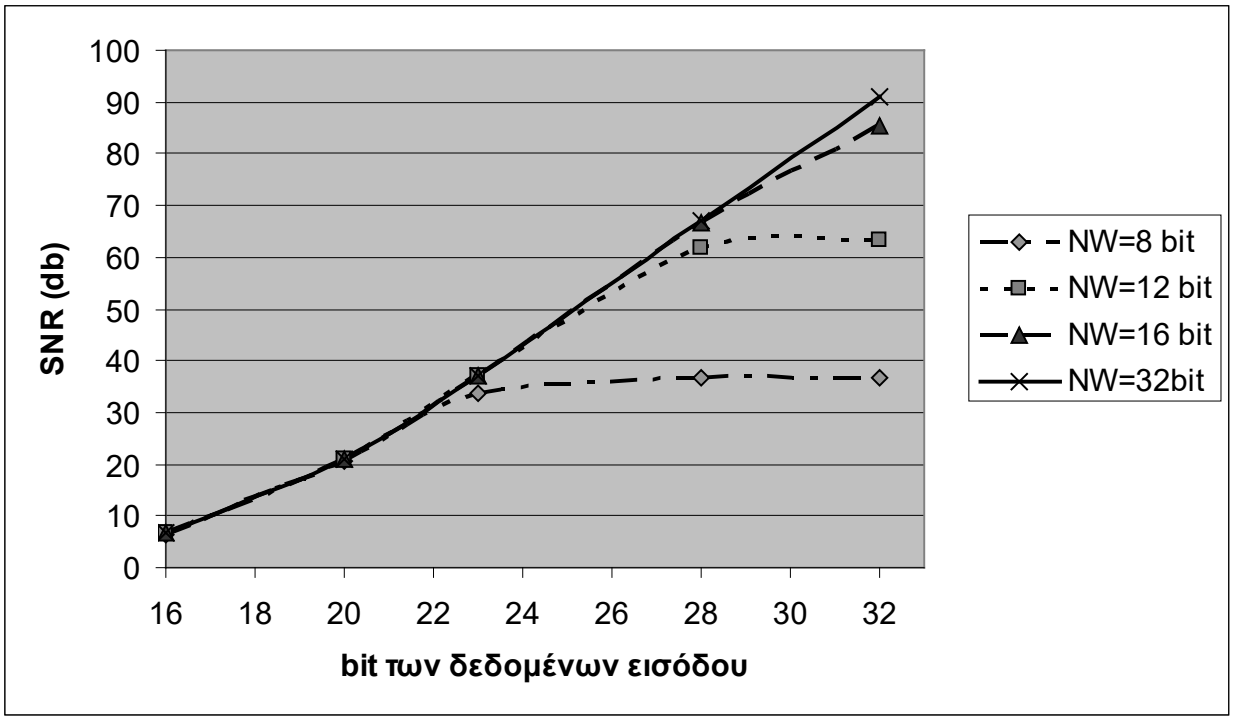
Στην αρχιτεκτονική που περιγράφηκε έχουμε τα ακόλουθα σφάλματα κατά τον υπολογισμό του γρήγορου μετασχηματισμού Fourier:

- σφάλματα εξαιτίας της αποκοπής των παραγόντων αναστροφής (twiddle factors)
- σφάλματα εξαιτίας της αποκοπής μετά των μιγαδικό πολλαπλασιασμό
- σφάλματα εξαιτίας της αποκοπής των εξόδων ώστε να γίνετε η διαίρεση με το 4 σε κάθε μονάδα butterfly ώστε να κρατιέται η ακρίβεια σταθερή.

Ο σηματοθορυβικός λόγος(SNR) ορίζεται ως ο λόγος $\frac{S}{N}$ όπου S είναι η ισχύς του σήματος και Ν είναι η ισχύς του θορύβου. Θόρυβο για την συγκεκριμένη υλοποίηση αποτελούν τα σφάλματα αποκοπής.

Μετρούμενος σε decibels παίρνει την μορφή $SNR(db) = 10\log_{10}\frac{S}{N}$

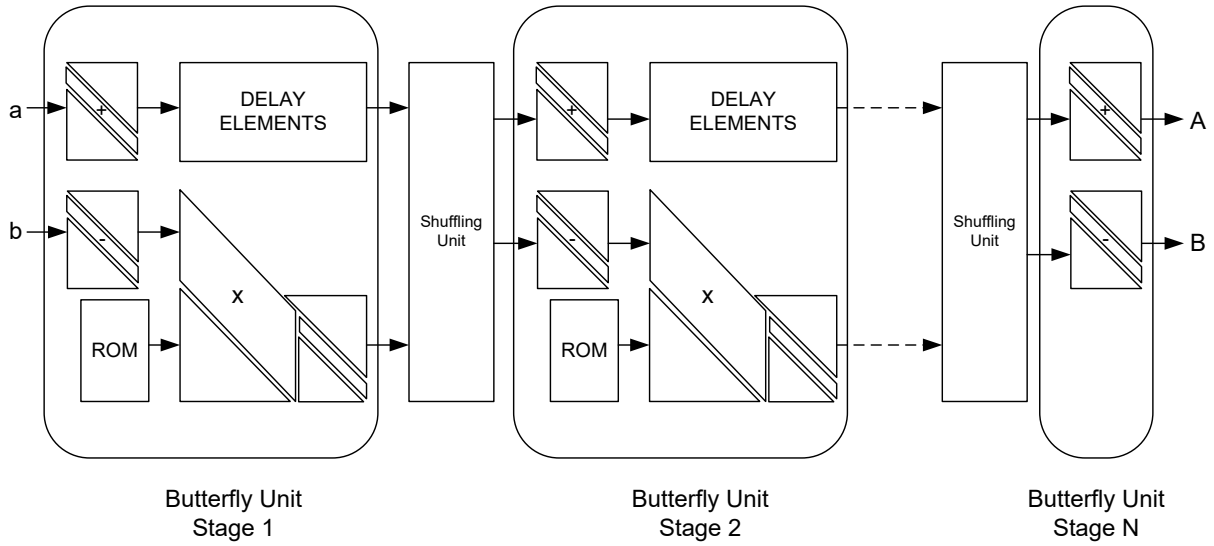
Με χρήση της συνάρτησης, του test bench και του script check1 που παρουσιάστηκαν στην παράγραφο 3.7, υπολογίσαμε τον σηματοθορυβικό λόγο (SNR) για διάφορες τιμές του μήκους σε bit των δεδομένων εισόδου (N bit) και του μήκους σε bit των παραγόντων αναστροφής (NW bit). Τα αποτελέσματα παρουσιάζονται στο Σχήμα 3.17.



Σχήμα 3.17 Σηματοθορυβικός λόγος σε decibels της υλοποίησης του FFT, για διάφορες τιμές του μήκους σε bit των δεδομένων εισόδου, και του μήκους σε bit των παραγόντων αναστροφής

3.9 Συμπεράσματα

Η αρχιτεκτονική που παρουσιάστηκε αποτελεί ένα αρκετά μεγάλο κύκλωμα το οποίο φαίνεται σε μια συνοπτική μορφή στο Σχήμα 3.18.



Σχήμα 3.18 Συνοπτική μορφή της αρχιτεκτονικής του FFT

Ο χρόνος καθυστέρησης κάθε μονάδας butterfly είναι

$$T_{butterfly} = 2N + NW$$

κύκλους ρολογιού, εκτός της τελευταίας μονάδας butterfly στην οποία λείπει ο μιγαδικός πολλαπλασιαστής και ο χρόνος καθυστέρησης είναι

$$T_{last_butterfly} = N$$

Ο χρόνος καθυστέρησης των μονάδων αναδιάταξης είναι

$$T_{Suffling_Units} = 2^{Log_2 NP-2} + 2^{Log_2 NP-3} + \ldots + 2^1 + 2^0 = \frac{\left(2^{Log_2 NP-1} - 1\right)}{(2-1)} = \frac{NP}{2} - 1$$

Επομένως ο χρόνος καθυστέρησης ολόκληρης της υλοποίησης του FFT είναι

$$T_{FFT} = \left(\log_2 NP - 1\right) \cdot \left(2N + NW\right) + N + NP/2$$

κύκλους ρολογιού, όπου N είναι το μήκος σε bit των μιγαδικών αριθμών εισόδου, σε φανταστικό και πραγματικό μέρος, NP είναι ο αριθμός τον σημείων του γρήγορου μετασχηματισμού Fourier που υπολογίζεται.

Εξαιτίας του μεγέθους του κυκλώματος περιμένουμε να έχουμε αρκετά μεγάλη κατανάλωση ισχύος. Με μερικές μετατροπές στα D flip-flops του κυκλώματος μπορούμε να το «κόψουμε», κρατώντας στο CLEAR κάποια D flip-flop, ώστε να χρησιμοποιείτε η ακρίβεια που θέλουμε ενώ

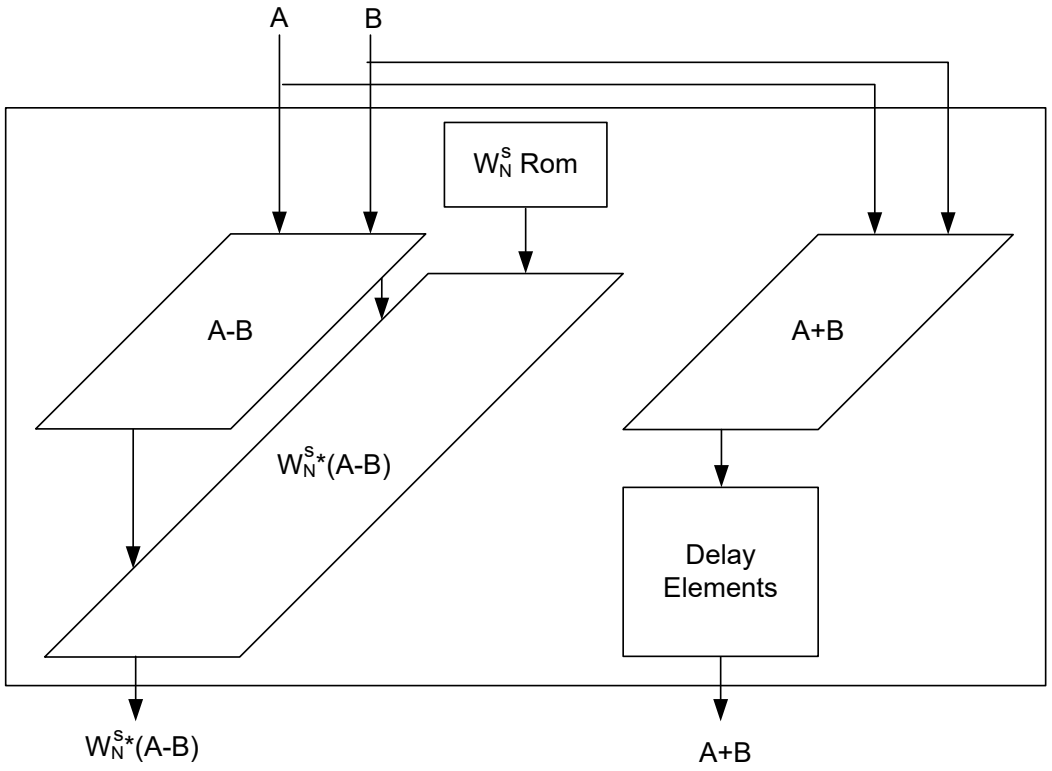
το υπόλοιπο κύκλωμα παραμένει ανενεργό. Έτσι θα μπορεί να γίνει η επιλογή της ακρίβειας του κυκλώματος, σε χρόνο λειτουργίας, με ανάλογη μείωση της κατανάλωσης ισχύος.

Όπως είδαμε στην παράγραφο 3.5 μπορούμε να υλοποιήσουμε τους μεγάλους μιγαδικούς καταχωρητές ολίσθησης με χρήση μνήμης RAM 2 θυρών. Με αυτόν τον τρόπο τα δεδομένα δεν ολισθαίνουν όλα σε κάθε κύκλο, αλλά παραμένουν ακίνητα μέσα στην μνήμη RAM και αλλάζουν μόνο τα δεδομένα προς ανάγνωση και εγγραφή. Με αυτόν τον τρόπο μπορεί να μειωθεί ακόμα η κατανάλωση ισχύος.

4 ΜΙΑ ΕΝΑΛΛΑΚΤΙΚΗ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ FFT ΜΕ ΛΙΓΟΤΕΡΑ ΣΤΟΙΧΕΙΑ ΚΑΘΥΣΤΕΡΗΣΗΣ

4.1 Εισαγωγή

Σε κάθε μονάδα butterfly παρατηρούμε ότι τα δεδομένα εξόδου του αφαιρέτη μπορούν να οδηγηθούν στον μιγαδικό πολλαπλασιαστή απευθείας σε μορφή απόκλισης του ενός bit (bit skew), ώστε να μην χρησιμοποιούμε στοιχεία καθυστέρησης ανάμεσά τους. Έχουμε την δυνατότητα λοιπόν μέσα σε κάθε μονάδα butterfly, να συνδυάσουμε τον μιγαδικό αφαιρέτη με τον μιγαδικό πολλαπλασιαστή και η μορφή του κυκλώματος παρουσιάζεται στο Σχήμα 4.1.



Σχήμα 4.1 Μορφή μονάδας butterfly με συνδυασμό μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή

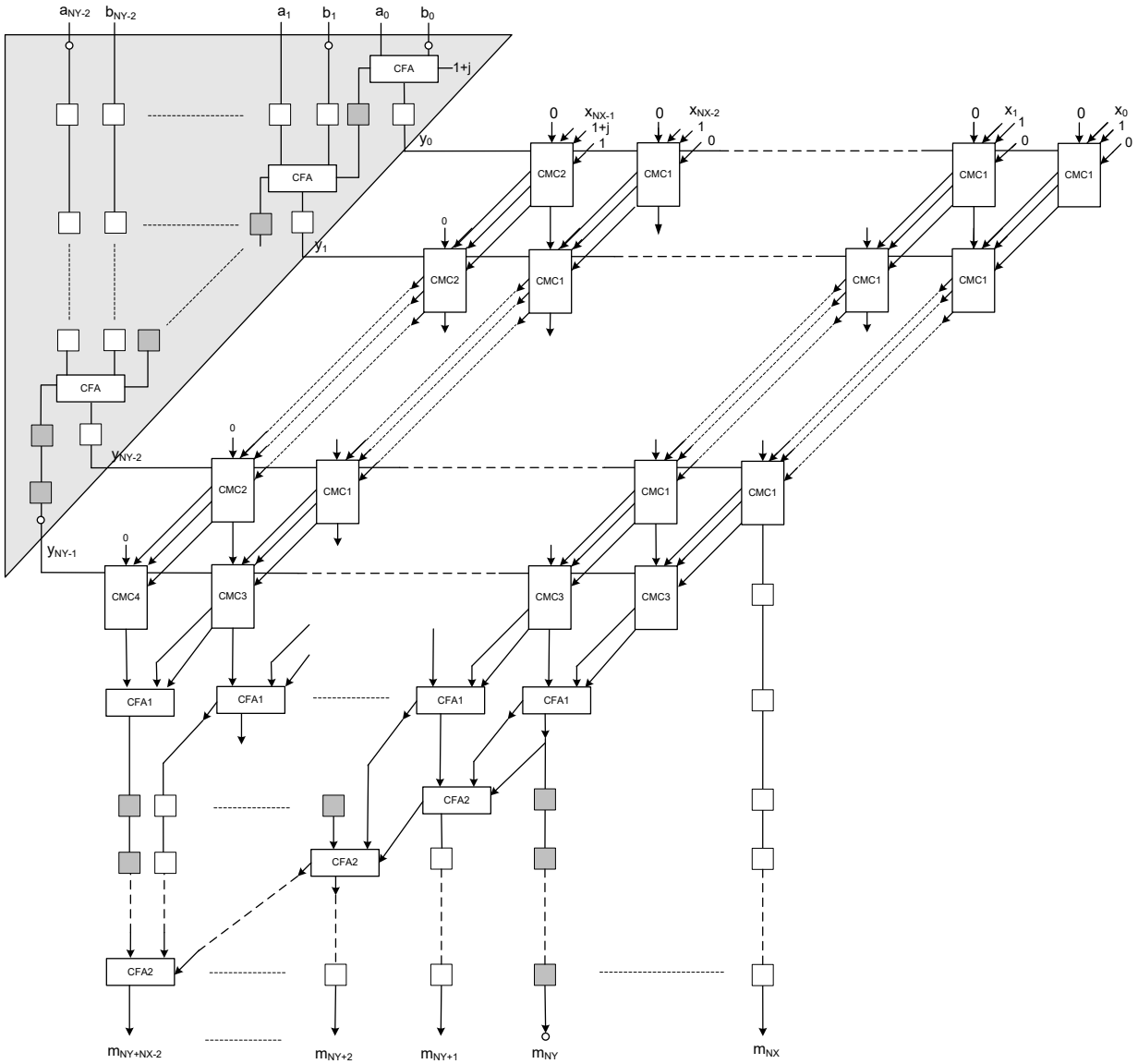
Έτσι εκεί που ήταν η είσοδος τον δεδομένων απ' την μνήμη ROM στον μιγαδικό πολλαπλασιαστή, τώρα είναι η είσοδος των αποτελεσμάτων του μιγαδικού αφαιρέτη και εκεί που ήταν η είσοδος των αποτελεσμάτων του μιγαδικού αφαιρέτη τώρα είναι η είσοδος τον δεδομένων απ' την μνήμη ROM στον μιγαδικό πολλαπλασιαστή.

Παρατηρούμε επίσης ότι κάθε μονάδα butterfly μπορεί να δεχθεί τις εισόδους της σε μορφή απόκλισης του ενός bit (bit skew). Επομένως όλες οι ενδιάμεσες μονάδες butterfly μπορούν να δέχονται τα δεδομένα σε μορφή απόκλισης ενός bit (bit skew) απ' την προηγούμενη βαθμίδα αλλά και να οδηγήσει τις εξόδους της, με την μορφή απόκλισης του ενός bit (bit skew), στην επόμενη μονάδα butterfly. Με αυτόν τον τρόπο απαλείφονται πολλά απ' τα στοιχεία καθυστέρησης που υπάρχουν ενδιάμεσα των μονάδων butterfly.

Επίσης λαμβάνουμε υπόψη το γεγονός ότι για τον υπολογισμό του N σημείων γρήγορου μετασχηματισμού Fourier, όταν το N είναι αρκετά μεγάλο, μπορούμε να απαλείψουμε τα στοιχεία καθυστέρησης (delay elements) μέσα στην μονάδα butterfly σε συνδυασμό με τα w στοιχεία καθυστέρησης της επόμενης μονάδας αναδιάταξης, όταν το w είναι ίσο ή μεγαλύτερο απ' την επιπλέον καθυστέρηση του πολλαπλασιαστή.

4.2 Συνδυασμός του μιγαδικού αφαιρέτη με τον μιγαδικό πολλαπλασιαστή

Αυτό μπορεί να γίνει όπως φαίνεται στο Σχήμα 4.2. Ο μιγαδικός αφαιρέτης συνεχούς διοχέτευσης έχει περιληφθεί μέσα σε γκρι τρίγωνο, ώστε να ξεχωρίζει απ' τον μιγαδικό πολλαπλασιαστή.



Σχήμα 4.2 Συνδυασμός μιγαδικού αφαιρέτη με μιγαδικό πολλαπλασιαστή, χρησιμοποιώντας λιγότερα στοιχεία καθυστέρησης, σε κάθε μονάδα butterfly

Με αυτόν τον τρόπο όχι μόνο γλιτώνουμε τα στοιχεία καθυστέρησης που υπήρχαν μεταξύ του μιγαδικού αφαιρέτη και του μιγαδικού πολλαπλασιαστή σε κάθε μονάδα butterfly, αλλά και τα στοιχεία καθυστέρησης που υπάρχουν μετά τον μιγαδικό αθροιστή κάθε μονάδας butterfly, τα οποία συγχρονίζονται με τα αποτελέσματα του μιγαδικού πολλαπλασιαστή.

4.3 Τροποποιημένες μονάδες butterfly

Εκμεταλλευόμενοι την μορφή απόκλισης του ενός bit (bit skew) στις εισόδους και εξόδους των αριθμητικών κυκλωμάτων μπορούμε, όπως είπαμε, να απαλείψουμε μεγάλο μέρος των στοιχείων καθυστέρησης που υπάρχουν ανάμεσα τους.

4.3.1 1η μονάδα butterfly

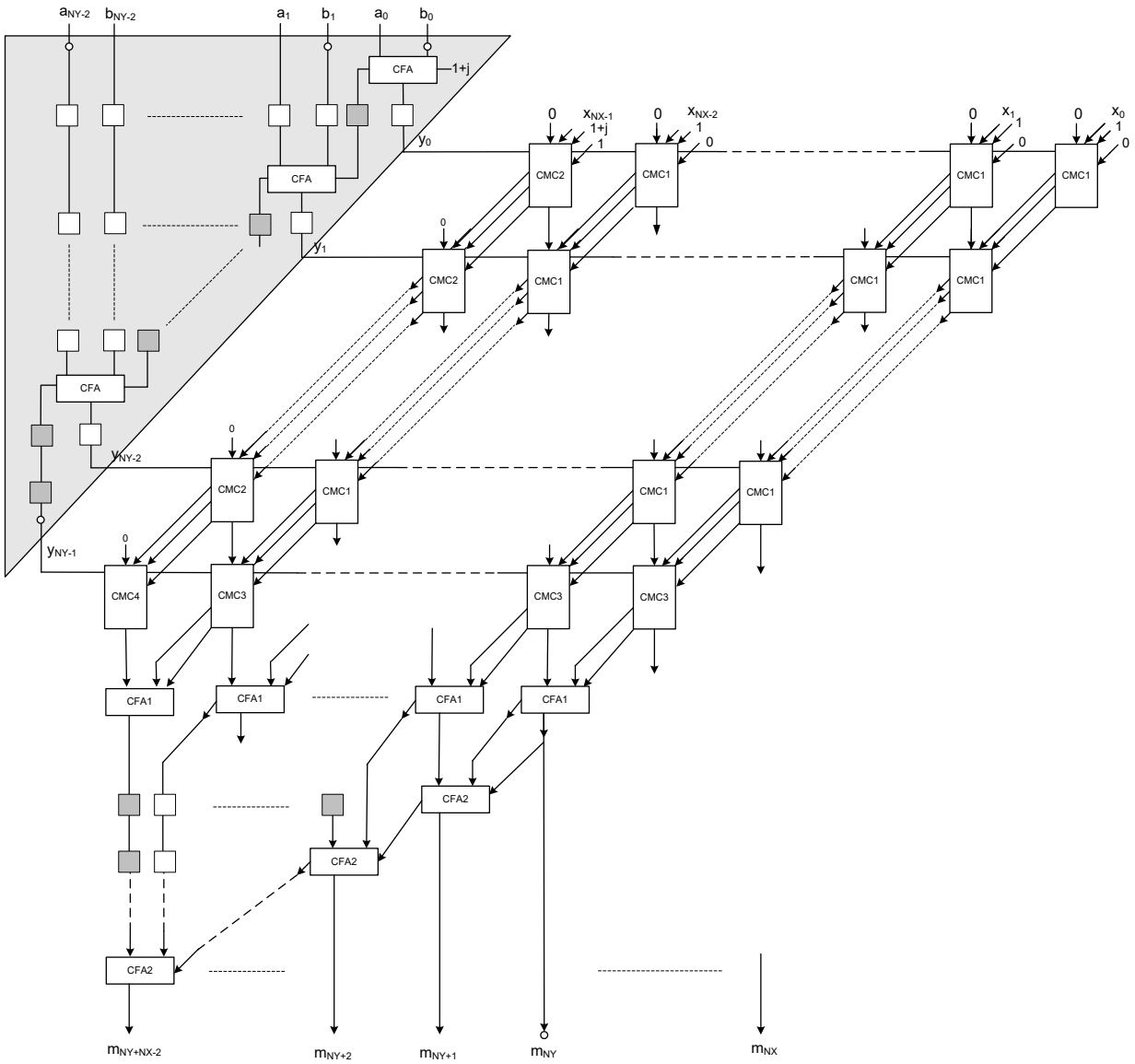
Στην πρώτη μονάδα butterfly τα δεδομένα στην είσοδο έρχονται σε παράλληλη μορφή και πρέπει να τοποθετήσουμε στοιχεία καθυστέρησης για τον συγχρονισμό. Αντίστοιχα στην έξοδο τα αποτελέσματα τα θέλουμε σε μορφή απόκλισης του ενός bit και άρα εκεί δεν βάζουμε στοιχεία καθυστέρησης. Επομένως ο συνδυασμός μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή στην πρώτη μονάδα butterfly φαίνεται στο Σχήμα 4.3.

Αν τα δεδομένα εισόδου στον FFT είναι με ακρίβεια N bit και οι παράγοντες αναστροφής (twiddle factors) έχουν αποθηκευτεί με ακρίβεια NW bit τότε η καθυστέρηση στον υπολογισμό του LSB απ' την πρώτη μονάδα butterfly, απ' την έξοδο του μιγαδικού πολλαπλασιαστή, θα είναι NW+2 κύκλοι ρολογιού.

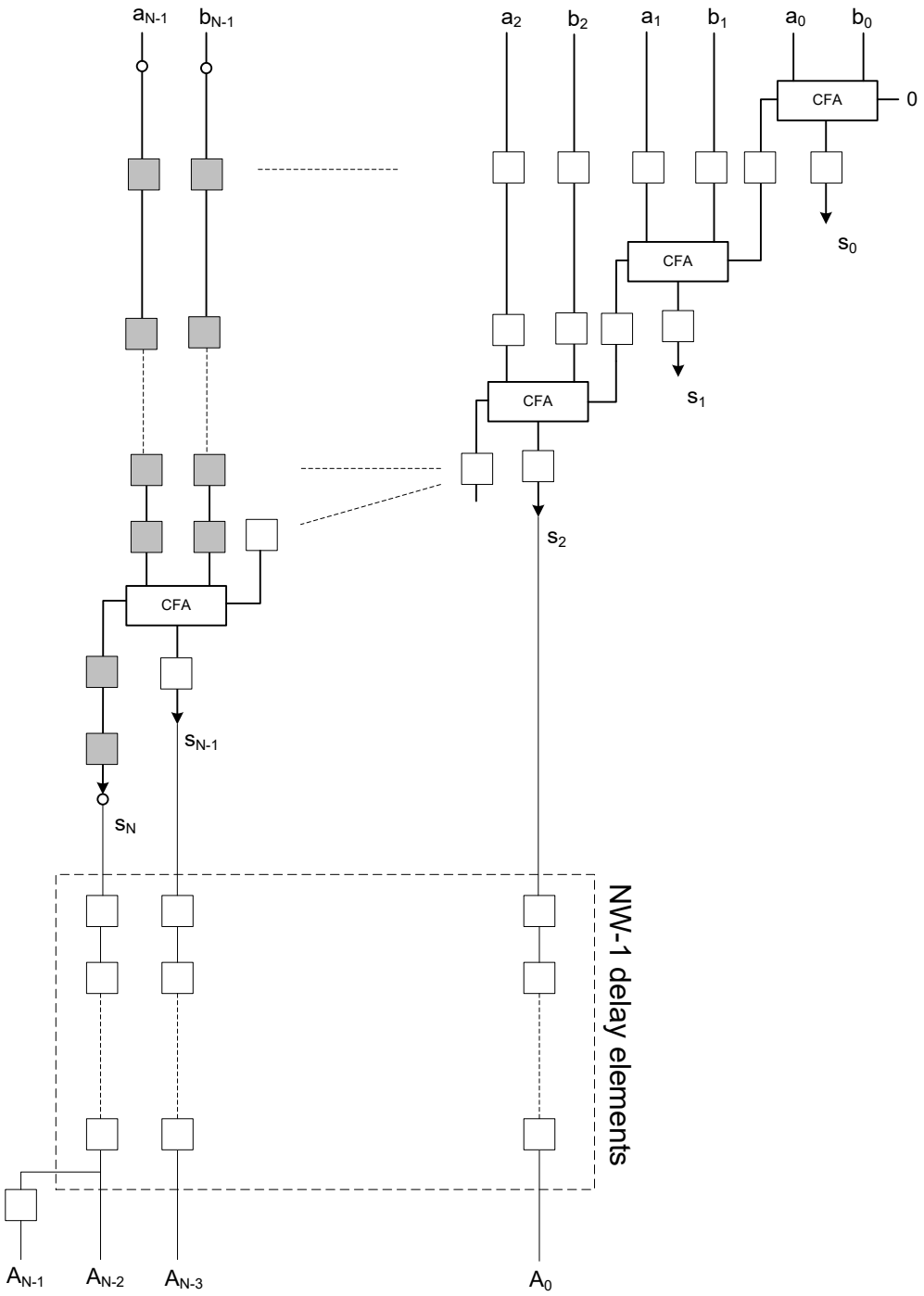
Απ' την άλλη μεριά, έχουμε τον μιγαδικό αθροιστή ο οποίος υπολογίζει το άθροισμα τον δεδομένων εισόδου. Ομοίως στην είσοδό του πρέπει να τοποθετηθούν στοιχεία καθυστέρησης, για τον απαραίτητο συγχρονισμό, αλλά όχι στην έξοδό του. Φυσικά για να οδηγηθεί η έξοδος του μιγαδικού αθροιστή στην επόμενη μονάδα butterfly σε μορφή απόκλισης του ενός bit θα πρέπει να βάλουμε ένα επιπλέον στοιχείο καθυστέρησης στην έξοδο του MSB. Ο μιγαδικός αθροιστής θα έχει την μορφή που φαίνεται στο Σχήμα 4.4.

Απ' την έξοδο του αθροιστή, εξαιτίας της διαίρεσης με το 4 σε κάθε μονάδα butterfly, παίρνουμε τα N-1 μεγαλύτερης αξίας bit. Έτσι τα 2 bit μικρότερης αξίας, παρόλο που υπολογίζονται, δεν οδηγούνται στην έξοδο της μονάδας butterfly. Επομένως η καθυστέρηση στον υπολογισμό του LSB απ' την πρώτη μονάδα butterfly, απ' την έξοδο του μιγαδικού αθροιστή, θα είναι 3 κύκλοι ρολογιού. Θα πρέπει όμως τα δεδομένα να συγχρονιστούν με τα δεδομένα εξόδου του μιγαδικού πολλαπλασιαστή και για τον λόγο αυτό πρέπει να βάλουμε NW-1 στοιχεία καθυστέρησης μετά την έξοδο του μιγαδικού αθροιστή.

Επίσης επειδή στην έξοδο της μονάδας butterfly πρέπει να έχουμε ακρίβεια N bit θα πρέπει, όπως έχουμε πει, να κάνουμε επέκταση προσήμου. Αυτό σε συνδυασμό με το ότι πρέπει τα δεδομένα να οδηγηθούν στην επόμενη βαθμίδα σε μορφή απόκλισης του ενός bit (bit skew) τοποθετεί και άλλο ένα στοιχείο καθυστέρησης στην επέκταση αυτή.



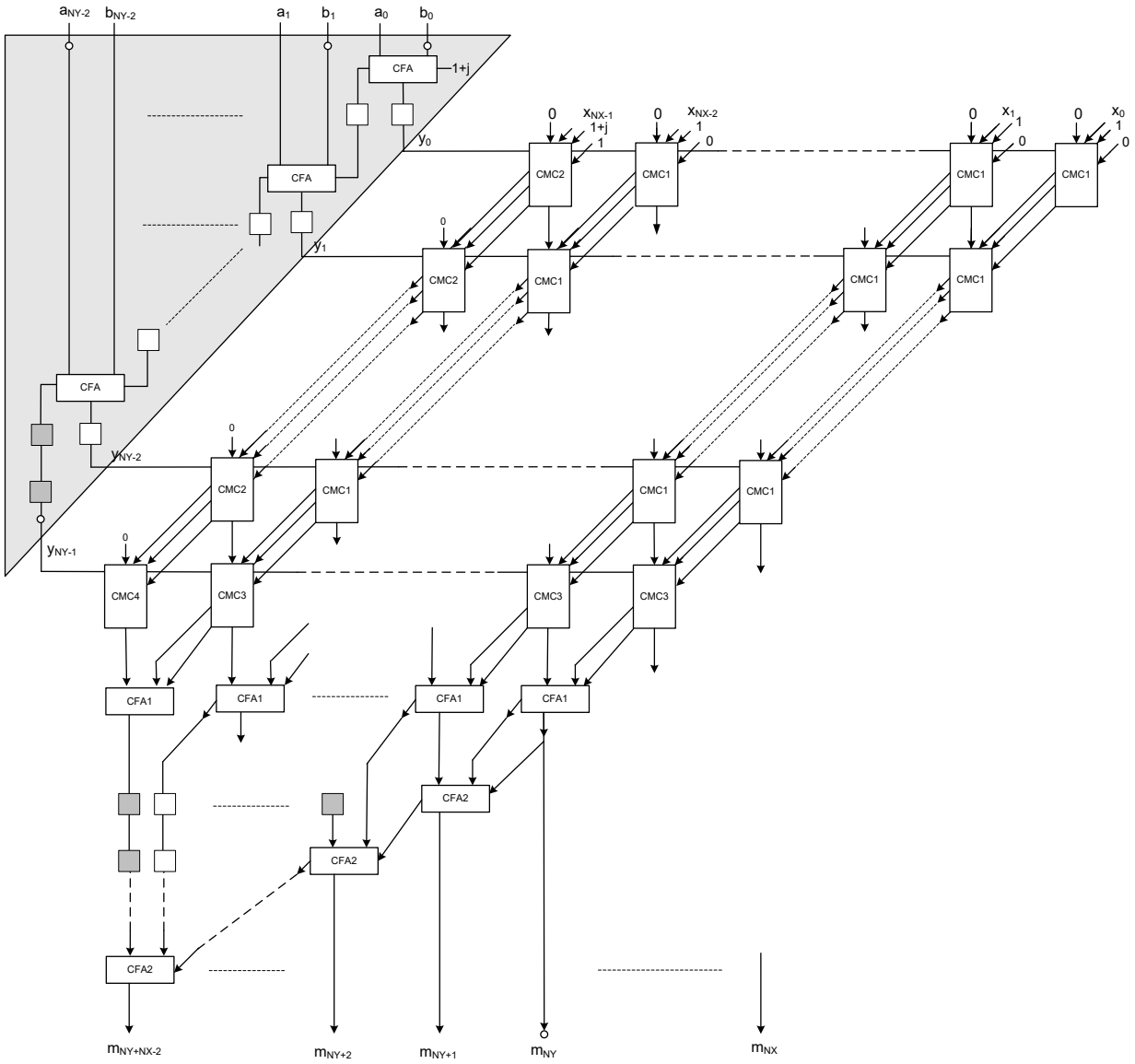
Σχήμα 4.3 Συνδυασμός μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή, στην 1η μονάδα butterfly της τροποποιημένης υλοποίησης



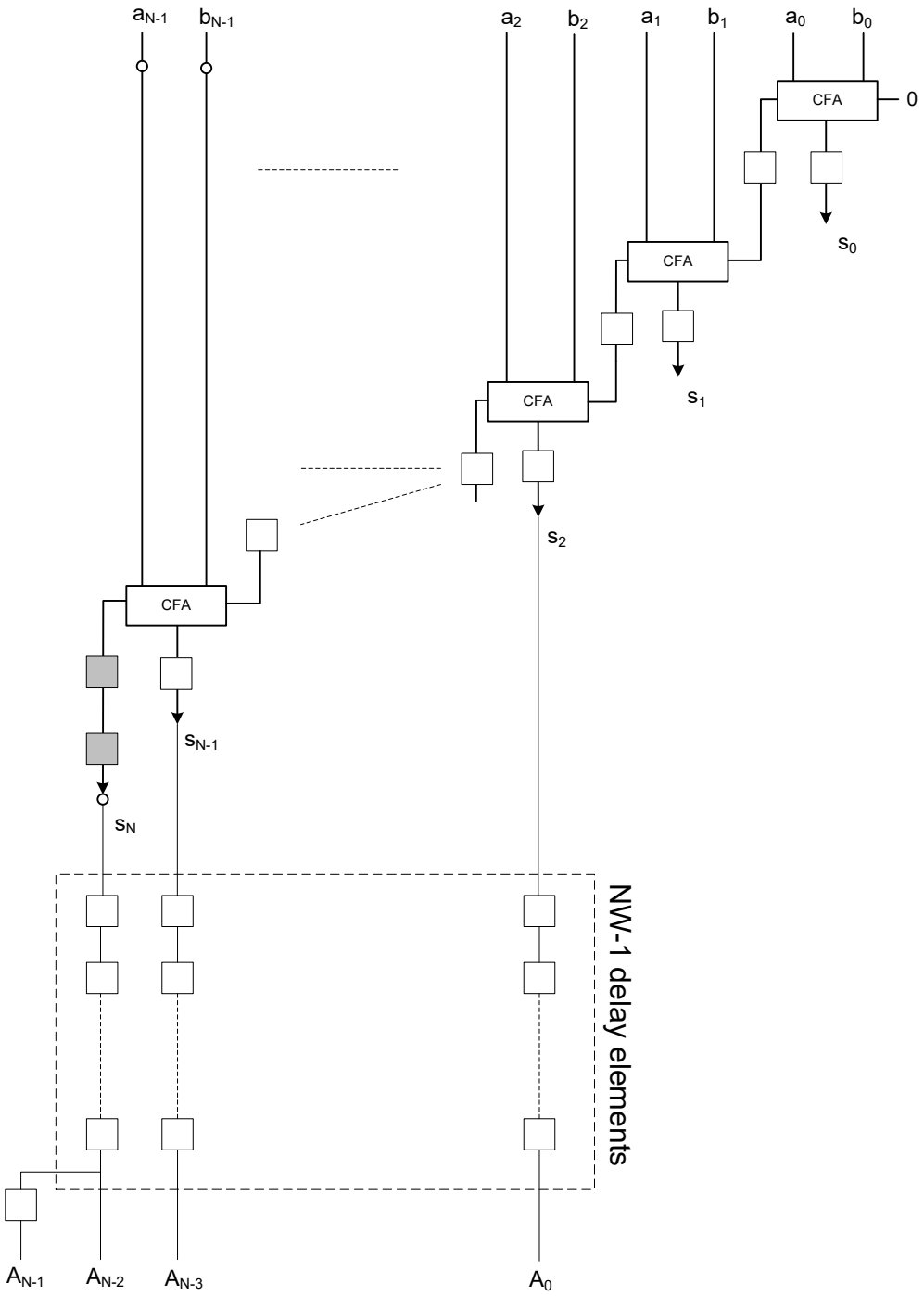
Σχήμα 4.4 Είσοδος και έξοδος δεδομένων απ' την μεριά του μιγαδικού αθροιστή στη 1η τροποποιημένη μονάδα butterfly

4.3.2 Ενδιάμεσες μονάδες butterfly

Η ίδια ανάλυση, όπως κάναμε για την πρώτη μονάδα butterfly, ισχύει και για τις ενδιάμεσες μονάδες. Τώρα όμως δεν χρειάζεται να βάλουμε στοιχεία καθυστέρησης ούτε στην είσοδο των αριθμητικών κυκλωμάτων, αφού τα δεδομένα μας έρχονται από την προηγούμενη μονάδα σε μορφή απόκλισης ενός bit (bit skew). Η μορφή του συνδυασμού του μιγαδικού αφαιρέτη και του μιγαδικού πολλαπλασιαστή παρουσιάζεται στο Σχήμα 4.5 και η έξοδος απ' την μεριά του μιγαδικού αθροιστή στο Σχήμα 4.6.



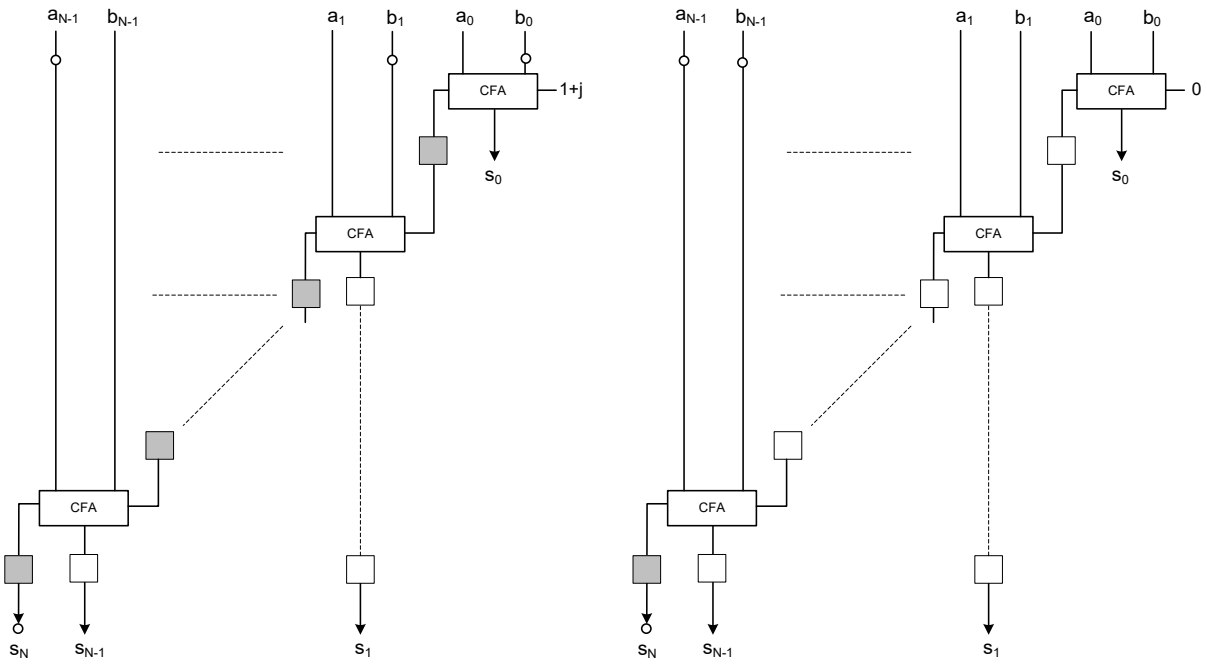
Σχήμα 4.5 Συνδυασμός μιγαδικού αφαιρέτη και μιγαδικού πολλαπλασιαστή στις ενδιάμεσες τροποποιημένες μονάδες butterfly



Σχήμα 4.6 Είσοδος και έξοδος απ' την μεριά του μιγαδικού αθροιστή των ενδιάμεσων τροποποιημένων μονάδων butterfly

4.3.3 Τελευταία μονάδα butterfly

Όπως έχουμε πει, στην τελευταία μονάδα butterfly όλοι οι πολλαπλασιασμοί γίνονται με το ένα, οπότε δεν υπάρχει λόγος τοποθέτησης μιγαδικού πολλαπλασιαστή. Τα δεδομένα εισόδου εισέρχονται στην μονάδα με την μορφή απόκλισης ενός bit (bit skew) και για αυτό δεν τοποθετούμε στοιχεία καθυστέρησης στην είσοδο. Στην έξοδο όμως πρέπει τα αποτελέσματα να οδηγηθούν σε παράλληλη μορφή και για τον λόγο αυτό πρέπει να τοποθετηθούν ανάλογα στοιχεία καθυστέρησης. Η μορφή της τελευταίας μονάδας butterfly φαίνεται στο Σχήμα 4.7.

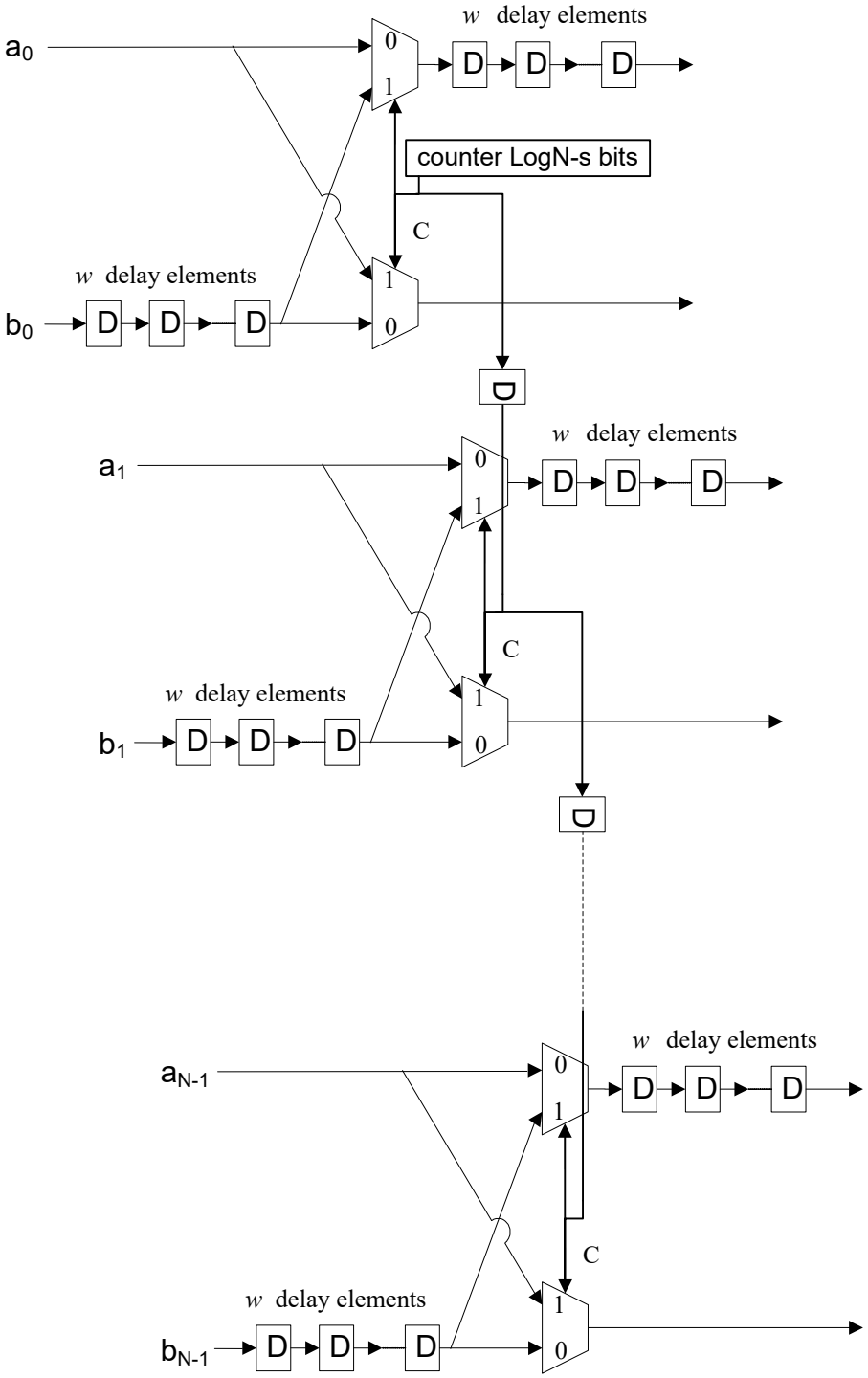


Σχήμα 4.7 Μορφή της τελευταίας μονάδας butterfly στην τροποποιημένη υλοποίηση

Όπως είναι φανερό η καθυστέρηση εξόδου του LSB και απ' τα δύο μέρη στην τελευταία μονάδα butterfly είναι ίση με N.

4.4 Τροποποιημένη μονάδα αναδιάταξης

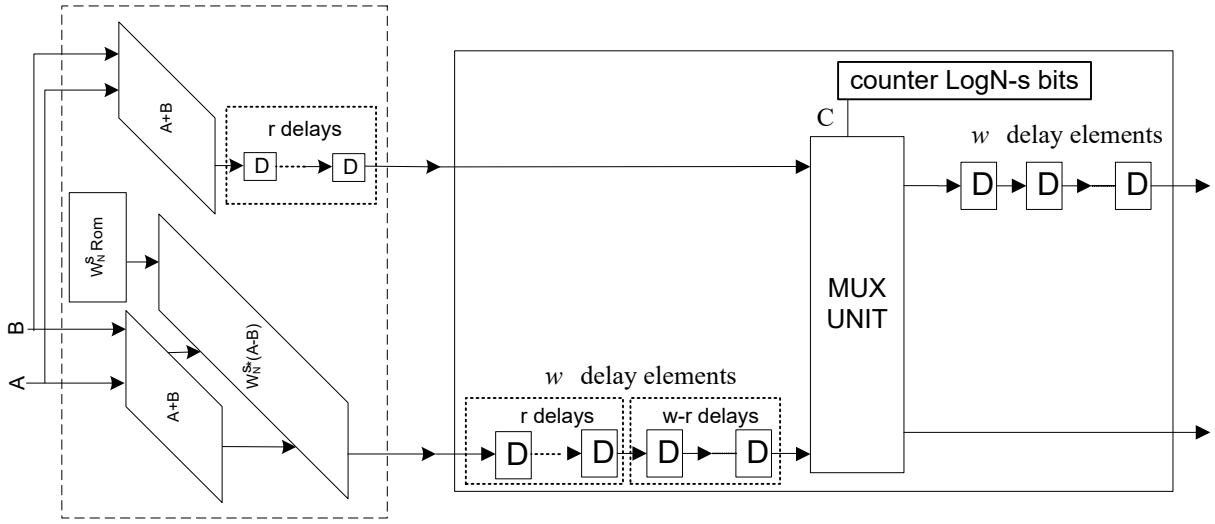
Παραπάνω τροποποιήσαμε τις μονάδες butterfly ώστε τα δεδομένα στις εισόδους και τις εξόδους να είναι σε μορφή απόκλισης ενός bit (bit skew). Επομένως θα πρέπει και η μονάδα butterfly να μετατραπεί ανάλογα ώστε να αναδιατάσσει τα δεδομένα με τον σωστό τρόπο. Αυτό μπορεί να γίνει όπως φαίνεται στο Σχήμα 4.8.



Σχήμα 4.8 Μονάδα αναδιάταξης της τροποποιημένης υλοποίησης

4.5 Απαλοιφή περιττών στοιχείων καθυστέρησης

Όταν έχουμε να υπολογίσουμε τον N σημείων γρήγορο μετασχηματισμό Fourier, και ειδικά όταν το N είναι αρκετά μεγάλο, στις πρώτες μονάδες αναδιάταξης έχουμε καταχωρητές ολίσθησης με αρκετά μεγάλο w. Όταν το w είναι ίσο η μεγαλύτερο απ' την επιπλέον καθυστέρηση που εισάγετε μέσα στη προηγούμενη μονάδα butterfly, ώστε να συγχρονιστούν τα δεδομένα εξόδου του μιγαδικού αθροιστή με τα δεδομένα εξόδου του μιγαδικού πολλαπλασιαστή, μπορούμε να τα απαλείψουμε όπως φαίνεται στο Σχήμα 4.9.



Σχήμα 4.9 Απαλοιφή περιττών στοιχείων καθυστέρησης

Όταν τα δεδομένα στην μνήμη ROM είναι αποθηκευμένα με ακρίβεια των NW bit τότε η επιπλέον καθυστέρηση r, που φαίνετε στο Σχήμα 4.9, είναι ίση με NW-1. Άρα όταν NW-1<w τότε μπορεί να γίνει η απαλοιφή. Με αυτό τον τρόπο μειώνουμε αρκετά το στοιχεία καθυστέρησης του κυκλώματος, ειδικά όταν ο αριθμός των σημείων N είναι μεγάλος.

4.6 Έλεγχος

Όλος ο κώδικας της περιγραφής σε VHDL παρατίθεται στο παράρτημα Ε. Ομοίως έχουμε φτιάξει test bench ουσιαστικά ίδιο με αυτό που περιγράψαμε στην παράγραφο 3.7. Το μόνο που άλλαξε είναι η προσαρμογή στην αλλαγή του χρόνου καθυστέρησης του κυκλώματος, αφού αυτός μίκρυνε. Επίσης στο κύκλωμα δεν άλλαξε κάτι ουσιαστικά, εκτός απ' τους χρόνους καθυστέρησης, και τα αποτελέσματα είναι ακριβώς τα ίδια με αυτά που παρουσιάστηκαν στην παράγραφο 3.7. Για να δούμε τα αποτελέσματα όμως θα πρέπει να χρησιμοποιήσουμε το script check2 το οποίο παρουσιάζετε παρακάτω που και σε αυτό το μόνο που έχει γίνει είναι η προσαρμογή στον νέο χρόνο καθυστέρησης, το οποίο χρησιμοποιεί και την συνάρτηση calc_fft_delay που δίνετε επίσης.

check2.m

```
M = csvread('outputs.txt');
NW=12;
N=13;
LOG2NP = 5;
NP=2^LOG2NP;
Y1=[];
Y2=[];
X=[];
% put input numbers at array C in right order and calculate fft%
START=calc_fft_delay(N,NW,LOG2NP)+1;
deigmata=1;
for i=START:NP/2:START+deigmata*NP/2-1
    A = M(i:i+NP/2-1,1)+j*M(i:i+NP/2-1,2);
    B = M(i:i+NP/2-1,3)+j*M(i:i+NP/2-1,4);
    C = cat(1,A,B);
    X = [X;C];
    Y1 = cat(1,Y1,fft(C,NP));
    AO = M(i:i+NP/2-1,5)+j*M(i:i+NP/2-1,6);
    BO = M(i:i+NP/2-1,7)+j*M(i:i+NP/2-1,8);
    for k=1:NP/2
        CO(2*k-1,1)=AO(k,1);
        CO(2*k,1)=BO(k,1);
    end
    Y2 = cat(1,Y2,bitrevorder(CO));
end
Y2=Y2*NP^2/2;

%Y1=round(Y1);
%Y1=fix(Y1);
%Y1=ceil(Y1);
%Y1=floor(Y1);

DY=Y1-Y2;

deigma=1;  % deigma <= deigmata
PY1 = Y1(1+(deigma-1)*NP:deigma*NP).* conj(Y1(1+(deigma-1)*NP:deigma*NP)) / NP;
PY2 = Y2(1+(deigma-1)*NP:deigma*NP).* conj(Y2(1+(deigma-1)*NP:deigma*NP)) / NP;
DPY = PY1-PY2;
f = (0:NP-1);

signal=Y1(1+(deigma-1)*NP:deigma*NP);
noise=Y2(1+(deigma-1)*NP:deigma*NP)-signal;
SNR=10*log10(sum(abs(signal).^2)/(sum(abs(noise).^2)))

figure(1);
set(1,'name','FFT Spectrum Calculated on Matlab');
plot(f,PY1(1:NP),'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
```

```
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(2);
set(2,'name','FFT Spectrum Calculated from implementaton');
plot(f,PY2(1:NP),'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(3);
set(3,'name','Input Data');
plot(f,real(X(1+(deigma-1)*NP:deigma*NP)),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(4);
set(4,'name','IFFT of FFT implementation output data ');
plot(f,real(ifft(Y2(1+(deigma-1)*NP:deigma*NP),NP)),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(5);
set(5,'name','IFFT of Matlab FFT calculation output data');
plot(f,real(ifft(Y1(1+(deigma-1)*NP:deigma*NP),NP)),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
figure(6);
set(6,'name','Abs(Y1-Y2)');
plot(f,sqrt(abs(DY(1+(deigma-1)*NP:deigma*NP))),'--rs',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10);
```

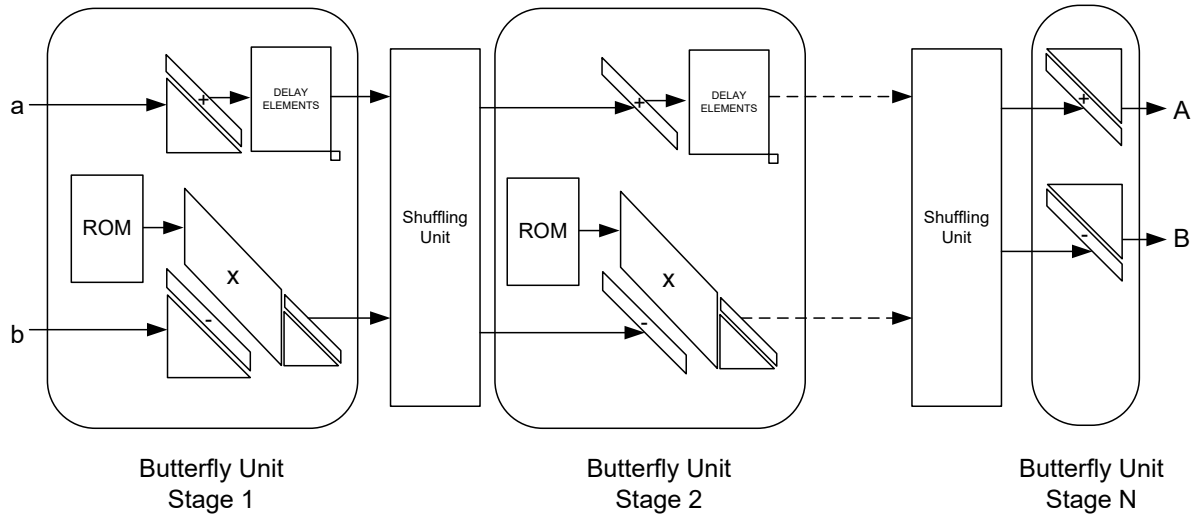
calc_fft_delay.m

```
function f = calc_fft_delay( N,NW, LOG2NP )
MULT_DELAY=NW-1;
BUTTERFLY_DELAY=NW+2;
W=2^(LOG2NP-2);
T=0;
S=2;
while S<=LOG2NP-1
     if W>=MULT_DELAY
          T=T+1;
    end
     W=W/2;
     S=S+1;
end
f = (LOG2NP-1)*BUTTERFLY_DELAY-T*MULT_DELAY+(N)+(2^(LOG2NP-1)-1);
```

Επίσης αφού δεν άλλαξε ουσιαστικά η αρχιτεκτονική του κυκλώματος και ο υπολογισμός του σηματοθορυβικού λόγου θα είναι ο ίδιος με αυτόν που παρουσιάστηκε στην παράγραφο 3.8.

4.7 Συμπεράσματα

Μια συνοπτική μορφή της τροποποιημένης αρχιτεκτονικής του FFT παρουσιάζεται στο Σχήμα 4.10.



Σχήμα 4.10 Συνοπτική μορφή της τροποποιημένης αρχιτεκτονικής του FFT

Ο χρόνος καθυστέρησης, για να πάρουμε το LSB του αποτελέσματος, σε κάθε μονάδα butterfly, μετά την τροποποίηση που κάναμε, είναι

$$T_{butterfly} = NW + 2$$

κύκλους ρολογιού, εκτός της τελευταίας μονάδας butterfly στην οποία λείπει ο μιγαδικός πολλαπλασιαστής και ο χρόνος καθυστέρησης, για να πάρουμε το αποτέλεσμα σε παράλληλη μορφή, είναι

$$T_{last_butterfly} = N$$

Ο χρόνος καθυστέρησης των μονάδων αναδιάταξης είναι ίδιος με προηγουμένως

$$T_{Suffling_Units} = \frac{NP}{2} - 1$$

Φυσικά κερδίζουμε και

$$T_{rem} = \sum_{S=1}^{Log_2 NP-1} f(S) \text{ όπου } f(S) = \begin{cases} NW-1, \alpha\nu & NW-1 < 2^{\log_2 NP-S-1} \\ 0 \quad , & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

κύκλους ρολογιού.

Επομένως ο χρόνος καθυστέρησης ολόκληρης της συγκεκριμένης υλοποίησης του FFT είναι

$$T_{FFT} = \left(\log_2 NP - 1\right) \cdot \left(NW + 2\right) + N + NP/2 - T_{rem}$$

κύκλους ρολογιού, όπου N είναι το μήκος σε bit των μιγαδικών αριθμών εισόδου, σε φανταστικό και πραγματικό μέρος, NP είναι ο αριθμός τον σημείων του γρήγορου μετασχηματισμού Fourier που υπολογίζουμε.

Με το κύκλωμα αυτό μπορεί να γλιτώσαμε αρκετά στοιχεία καθυστέρησης και μειώσαμε αρκετά τον χρόνο καθυστέρησης αλλά πλέον δεν είναι τόσο εύκολο το «κόψιμο» του κυκλώματος ώστε να μεταβάλουμε την ακρίβεια τον πράξεων προς όφελος της κατανάλωσης ισχύος σε χρόνο λειτουργίας. Για να γίνει αυτό πρέπει να σχεδιαστή επιπλέον ειδικό κύκλωμα.

Και σε αυτήν την υλοποίηση μπορούμε να χρησιμοποιήσουμε μνήμη RAM για την υλοποίηση των μεγάλων καταχωρητών ολίσθησης, όπως περιγράφηκε στην παράγραφο 3.5.

4.8 Περαιτέρω βελτιστοποιήσεις

Όπως είδαμε μπορούμε να υλοποιήσουμε τους μεγάλους σε μήκος μιγαδικούς καταχωρητές ολίσθησης με την χρήση μνήμης RAM 2 θυρών. Η μνήμη RAM 2 θυρών καταναλώνει αρκετή ισχύ εξαιτίας της συνεχόμενης ανάγνωσης και εγγραφής καθώς και το κύτταρό της είναι κατά 33% μεγαλύτερο απ' το αντίστοιχο της μνήμης RAM με 1 θύρα. Έτσι αν θέλουμε να μειώσουμε και άλλο την κατανάλωση ισχύος μπορούμε να αντικαταστήσουμε την μνήμη RAM 2 θυρών με δύο μνήμες RAM με 1 θύρα και να τοποθετήσουμε τα δεδομένα μέσα στην μνήμη έτσι ώστε όταν γίνετε ανάγνωση στην μία να γίνετε εγγραφή στην άλλη. Με τον τρόπο αυτό και μικραίνει η κατανάλωση ισχύος, αφού δεν έχουμε συνεχόμενη ανάγνωση και εγγραφή, αλλά και η επιφάνεια του κυκλώματος.

Επίσης είδαμε ότι σε κάθε μονάδα butterfly γίνετε διαίρεση με το 4 στα αποτελέσματα του μιγαδικού πολλαπλασιαστή, καθώς και του μιγαδικού αθροιστή, ώστε να διατηρείται η ακρίβεια σταθερή. Μπορούμε να βελτιώσουμε την ακρίβεια του κυκλώματος με το να αυξάνουμε την ακρίβεια κάθε μονάδας butterfly σε σχέση με την προηγούμενη. Έτσι μπορούμε να αυξάνουμε την ακρίβεια κατά 1 με αποτέλεσμα να κάνουμε αντίστοιχα διαίρεση με το 2, αντί διαίρεση με το 4 όπως κάναμε προηγουμένως, είτε να την αυξάνουμε κατά 2 ώστε να μην έχουμε καθόλου σφάλματα αποκοπής εξαιτίας αυτού του γεγονότος. Φυσικά αυτό θα οδηγήσει σε βελτίωση της ακρίβειας του κυκλώματος αλλά και σε ανάλογη αύξηση του μεγέθους του κυκλώματος.

ΒΙΒΛΙΟΓΡΑΦΙΑ

DSP Theory Books

1. Antoniou, A., (1993), Digital Filters : Analysis, Design and Applications, McGraw-Hill, USA.
2. Bellanger, M., (1984), Digital Processing Of Signals, Theory And Practice, John Wiley & Sons, UK
3. Brigham, E. O. and Cecchini, L. R., A Nomogram for Determining FFT System Dynamic Range, E-Systems Inc., USA.
4. Burrus, C. S. and Parks, T. W., (1985), DFT/FFT and Convolution Algorithms, Wiley Interscience, NY, USA.
5. Cooley, J.W. and Tukey, J.W., (1965), An Algorithm for the Machine Calculation of Complex Fourier Series, Mathematics of Computation, The American Mathematical Society, USA.
6. Defatta, D. J., Lucas, J. G. and Hodgkiss, W. S., (1988), Digital Signal Processing - A system Design Approach, John Wiley & Sons Inc., USA.
7. Ewers, G. and Marven, C. A Simple Approach to Digital Signal Processing, Texas Instruments Inc., ISBN : 0-904047-00-8.
8. Haddad, R. A., and Parsons, T. W., (1991), Digital Signal Processing - Theory, Applications and Hardware, Computer Sciences Press, USA.
9. Lyons R. G., (1999), Understanding Digital Signal Processing, Addison-Wesley, USA.
10. Marple, Digital Spectral Analysis with Applications, Prentice-Hall Inc., USA
11. Nussbaumer H., Fast Fourier Transform and Convolution Algorithms
12. Oppenheim, A. V. and Schafer, R. W., (1989), Discrete Time Signal Processing, Prentice-Hall Inc., USA.
13. Proakis, J. G., and Manolakis, D. G., (1992), Digital Signal Processing - Principles, Algorithms and Applications (2ed), Prentice-Hall, USA.
14. Stein, Jonathan (Y), Digital Signal Processing, A Computer Science Perspective, John Wiley and Sons, USA. (ISBN 0-471-29546-9)

VHDL

1. Κ. ΠΕΚΜΕΣΤΖΗ "Ψηφιακά Συστήματα VLSI" Έκδοση ΕΜΠ 1999
2. Douglas L. Perry, (2002), VHDL: Programming by Example 4th Edition, McGraw Hill
3. IEEE Standard VHDL (2000), Language Reference Manual, Draft
4. J. Armstrong, G. Frank (1995), Test Bench System Specification
5. J. R. Armstrong, F. G. Gray, (1993), Structured Logic Design with VHDL, Prentice Hall.
6. J. Armstrong & G. Gray (2000), VHDL Design – Representation and Synthesis, Prentice Hall.
7. J. Bhasker, 1995 A VHDL Primer Revised Edition, Prentice Hall.
8. Mark Zwolinski, Digital System Design With VHDL, 2nd Edition, ISBN 0-13-039985-X
9. Peter J. Ashenden, (1996), *The Designer's Guide to VHDL*, Morgan Kauffman, San Francisco.
10. Peter Pirsch, Architectures for Digital Signal Processing, John Wiley & Sons.
11. R. Lipsett, C. Schaefer and C. Ussery, (1989), VHDL Hardware Description and Design., Kluwer Academic Publishers.
12. Volnei A. Pedroni (2004), Circuit Design With VHDL, MIT Press, Hardcover, ISBN 0262162245
13. D. Smith, (1996), HDL Chip Design: A Practical Guide for Designing, Synthesizing, and Simulating ASICs and FPGAs using VHDL or Verilog, Doone Publications, Madison AL

Papers

1. K. Z. Pekmestzi, "Complex Number Multipliers". IEE Proceedings, Vol.136, Part E, No. 1, pp. 70-75, January 1989.

Matlab

1. The Math Works Inc., (1984-2004), Getting started with MATLAB
2. The Math Works Inc., (1984-2004), MATLAB Programming

ΠΑΡΑΡΤΗΜΑ

Α. Περιγραφή του μιγαδικού αθροιστή σε VHDL

Περιγραφή σε κώδικα VHDL του μιγαδικού αθροιστή συνεχούς διοχέτευσης με αποθήκευση κρατουμένου που περιγράφηκε στην παράγραφο 1.1.4. Τα αρχεία είναι τα εξής:

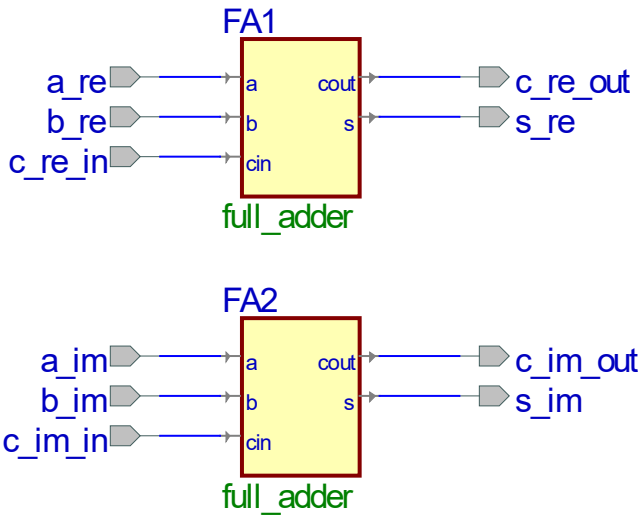
dff.vhd	Περιγραφή του D Flip-Flop που χρησιμοποιείτε στον μιγαδικό αθροιστή
full_adder.vhd	Περιγραφή του πλήρη αθροιστή
complex_full_adder.vhd	Περιγραφή του μιγαδικού πλήρη αθροιστή με βάση το block diagram που βρίσκεται στο αρχείο complex_full_adder.bde
complex_adder.vhd	Περιγραφή ολόκληρου του μιγαδικού αθροιστή
complex_adder_tb.vhd	Test bench που εκτελεί εξαντλητικό test με όλα τα δυνατά δεδομένα εισόδου ελέγχοντας την έξοδο του μιγαδικού αθροιστή

dff.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end dff;


architecture behave of dff is
begin
       process(clk,clear)
       begin
            if clear ='0' then
                 q<=CLEARQ;
            elsif clk'event and clk ='1' then
                 q<=d;
            end if ;
      end process;
end behave;
```

full_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity full_adder is
      port(
            a : in STD_LOGIC;
            b : in STD_LOGIC;
            cin : in STD_LOGIC;
            s : out STD_LOGIC;
            cout : out STD_LOGIC
          );
end full_adder;

architecture full_adder of full_adder is
begin
      s<=a xor b xor cin;
      cout<=((a or b) and cin) or (a and b);
end full_adder;
```



Σχήμα Α.1 Περιγραφή του μιγαδικού πλήρη αθροιστή (CFA) στο block diagram αρχείο complex_full_adder.bde

Complex_full_adder.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
```

```
  );
end Complex_full_adder;

architecture Complex_full_adder of Complex_full_adder is

---- Component declarations -----

component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

begin

----  Component instantiations  ----

FA1 : full_adder
  port map(
       a => a_re,
       b => b_re,
       cin => c_re_in,
       cout => c_re_out,
       s => s_re
  );

FA2 : full_adder
  port map(
       a => a_im,
       b => b_im,
       cin => c_im_in,
       cout => c_im_out,
       s => s_im
  );


end Complex_full_adder;
```

Complex_Adder.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Adder is
     generic   (
          N : POSITIVE := 4
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          s_im: out STD_LOGIC_VECTOR (N downto 0);
          s_re: out STD_LOGIC_VECTOR (N downto 0)
     );
end Complex_Adder;

architecture Complex_Adder_Structural of Complex_Adder is

component dff is
```

```
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CA_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

signal     a_im,a_re,b_im,b_re,s2_im,s2_re : CA_word_array(0 to N);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(0 to N);

begin
          -- CFA generation
CFAx:for i in 0 to N-1 generate
CFA:      Complex_full_adder port map(
                    a_im => a_im(i)(i),
                    a_re => a_re(i)(i),
                    b_im => b_im(i)(i),
                    b_re => b_re(i)(i),
                    c_im_in => c_im_in(i),
                    c_re_in => c_re_in(i),
                    c_im_out => c_im_out(i),
                    c_re_out => c_re_out(i),
                    s_im => s2_im(i)(i),
                    s_re => s2_re(i)(i)
               );
          end generate;

row1:for i in 0 to N-1 generate
col1:     for j in 0 to N-1 generate
                    -- input Dff generation
cond1:              if j>i and j<N-1 generate
gda_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_re(i)(j),
                              q => a_re(i+1)(j)
                         );
gda_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_im(i)(j),
                              q => a_im(i+1)(j)
                         );
```

```
gdb_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_re(i)(j),
                              q => b_re(i+1)(j)
                         );
gdb_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_im(i)(j),
                              q => b_im(i+1)(j)
                         );
                    end generate;
cond2:              if j=N-1 generate
gda_re1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_re(i)(j),
                              q => a_re(i+1)(j)
                         );
gda_im1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_im(i)(j),
                              q => a_im(i+1)(j)
                         );
gdb_re1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_re(i)(j),
                              q => b_re(i+1)(j)
                         );
gdb_im1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_im(i)(j),
                              q => b_im(i+1)(j)
                         );
                    end generate;
                    -- output Dff generation
cond3:              if j<=i generate
gds_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => s2_re(i)(j),
                              q => s2_re(i+1)(j)
                         );
gds_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => s2_im(i)(j),
                              q => s2_im(i+1)(j)
```

```
                         );
                    end generate;
                    -- carry Dff generation
cond4:              if i=j and i<N-1 generate
gdc_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => c_re_out(i),
                              q => c_re_in(i+1)
                         );
gdc_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => c_im_out(i),
                              q => c_im_in(i+1)
                         );
                    end generate;

               end generate;
          end generate;
          -- MSB carry Dff generation
gdc_re1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_re_out(N-1),
               q => c_re_out(N)
          );
gdc_im1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_im_out(N-1),
               q => c_im_out(N)
          );

-- input wires generation
     a_re(0) <= (not x_re(N-1))&x_re(N-2 downto 0);
     a_im(0) <= (not x_im(N-1))&x_im(N-2 downto 0);
     b_re(0) <= (not y_re(N-1))&y_re(N-2 downto 0);
     b_im(0) <= (not y_im(N-1))&y_im(N-2 downto 0);
     c_re_in(0) <= '0';
     c_im_in(0) <= '0';

-- output wires generation
     s_re <= (not c_re_out(N))&s2_re(N);
     s_im <= (not c_im_out(N))&s2_im(N);

end Complex_Adder_Structural;
```

complex_adder_tb.vhd

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

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;
```

```
entity complex_adder_tb is
     -- Generic declarations of the tested unit
          generic(
          N : POSITIVE := 4 );
end complex_adder_tb;

architecture TB_ARCHITECTURE of complex_adder_tb is

     file OUTPUT_FILE: TEXT open WRITE_MODE is "outputs.txt";

     -- Component declaration of the tested unit
     component Complex_Adder
          generic(
          N : INTEGER := 4 );
     port(
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          s_im: out STD_LOGIC_VECTOR (N downto 0);
          s_re: out STD_LOGIC_VECTOR (N downto 0) );
     end component;

     constant DELAY1 : time := 10 ns;


     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clk : std_logic;
     signal clear : std_logic := '0';
     signal x_im : std_logic_vector(N-1 downto 0);
     signal x_re : std_logic_vector(N-1 downto 0);
     signal y_im : std_logic_vector(N-1 downto 0);
     signal y_re : std_logic_vector(N-1 downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal s_im : std_logic_vector(N downto 0);
     signal s_re : std_logic_vector(N downto 0);

     -- Add your code here ...

begin

     -- Unit Under Test port map
     UUT : Complex_Adder
          generic map (
               N => N
          )

          port map (
               clk => clk,
               clear => clear,
               x_im => x_im,
               x_re => x_re,
               y_im => y_im,
               y_re => y_re,
               s_im => s_im,
               s_re => s_re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;
```

```
INPUT_Generation: process
variable ny_re : integer;
variable ny_im : integer;
variable nx_re : integer;
variable nx_im : integer;

begin
     x_re <= CONV_STD_LOGIC_VECTOR(0,N);
     x_im <= CONV_STD_LOGIC_VECTOR(0,N);
     y_re <= CONV_STD_LOGIC_VECTOR(0,N);
     y_im <= CONV_STD_LOGIC_VECTOR(0,N);
     clear <= '0';
     wait until (clk'event) and (clk='1');
     clear <= '1';

     for nx_re in -2**(N-1) to 2**(N-1)-1 loop
          for nx_im in -2**(N-1) to 2**(N-1)-1 loop
               for ny_re in -2**(N-1) to 2**(N-1)-1 loop
                    for ny_im in -2**(N-1) to 2**(N-1)-1 loop
                         x_re <= CONV_STD_LOGIC_VECTOR(nx_re,N);
                         x_im <= CONV_STD_LOGIC_VECTOR(nx_im,N);
                         y_re <= CONV_STD_LOGIC_VECTOR(ny_re,N);
                         y_im <= CONV_STD_LOGIC_VECTOR(ny_im,N);
                         wait until (clk'event) and (clk='1');
                    end loop;
               end loop;
          end loop;
     end loop;
     wait;

end process;

OUTPUT_Checking:process
     type XY_Matrix is array (NATURAL range <>) of std_logic_vector(0 to N-1);

     variable X_Re_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));
     variable X_Im_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));
     variable Y_Re_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));
     variable Y_Im_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));

     variable report_line : line;
     variable report_str : string(1 to 160);
     variable i,j,re_part, im_part : integer := 0;

begin

     wait until (clk'event) and (clk='1') and (clear='1');

     for i in 0 to N-1 loop
          X_Re_Matrix(i) := X_Re_Matrix(i+1);
          X_Im_Matrix(i) := X_Im_Matrix(i+1);
          Y_Re_Matrix(i) := Y_Re_Matrix(i+1);
          Y_Im_Matrix(i) := Y_Im_Matrix(i+1);
     end loop;
     X_Re_Matrix(N) := x_re;
     X_Im_Matrix(N) := x_im;
     Y_Re_Matrix(N) := y_re;
     Y_Im_Matrix(N) := y_im;

     re_part := conv_integer(X_Re_Matrix(0))+conv_integer(Y_Re_Matrix(0));
     im_part := conv_integer(X_Im_Matrix(0))+conv_integer(Y_Im_Matrix(0));

     report_line:=null;

     if ((conv_integer(s_re) /= re_part) or (conv_integer(s_im) /= im_part)) then

          write(report_line,string'(">>>>>>>>>>Wrong Result: "));
```

```
                write(report_line,conv_integer(X_Re_Matrix(0)));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(X_Im_Matrix(0)));
                write(report_line,string'(" + "));
                write(report_line,conv_integer(Y_Re_Matrix(0)));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(Y_Im_Matrix(0)));
                write(report_line,string'(" = "));
                write(report_line,conv_integer(s_re));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(s_im));

                write(report_line,string'("  Right: "));
                write(report_line,conv_integer(re_part));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(im_part));
                write(report_line,string'("       "));

                for i in report_line'range loop
                     report_str(i) := report_line(i);
                end loop;
                report report_str;

                writeline(OUTPUT_FILE,report_line);

           end if;

           if ( (conv_integer(X_Re_Matrix(0)) = 2**(N-1)-1) and
              (conv_integer(X_Im_Matrix(0)) = 2**(N-1)-1) and
                  (conv_integer(Y_Re_Matrix(0)) = 2**(N-1)-1) and
              (conv_integer(Y_Im_Matrix(0)) = 2**(N-1)-1) ) then
                write(report_line,string'("------->The End!!! "));
                writeline(OUTPUT_FILE,report_line);
                report "Test Finished!!!" severity FAILURE;
           end if;

     end process;


end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_complex_adder of complex_adder_tb is
     for TB_ARCHITECTURE
          for UUT : complex_adder
               use entity work.complex_adder(complex_adder_structural);
          end for;
     end for;
end TESTBENCH_FOR_complex_adder;
```

Β. Περιγραφή του μιγαδικού αφαιρέτη σε VHDL

Περιγραφή σε κώδικα VHDL του μιγαδικού αφαιρέτη συνεχούς διοχέτευσης με αποθήκευση κρατουμένου που περιγράφηκε στην παράγραφο 1.2.4. Τα αρχεία είναι που χρησιμοποιούμε για την περιγραφή είναι τα εξής:

dff.vhd	Περιγραφή του D Flip-Flop που χρησιμοποιείτε στον μιγαδικό αφαιρέτη
full_adder.vhd	Περιγραφή του πλήρη αθροιστή
complex_full_adder.vhd	Περιγραφή του μιγαδικού πλήρη αθροιστή με βάση το block diagram που βρίσκεται στο αρχείο complex_full_adder.bde
complex_subtracter.vhd	Περιγραφή ολόκληρου του μιγαδικού αφαιρέτη
complex_ subtracter _tb.vhd	Test bench που εκτελεί εξαντλητικό test με όλα τα δυνατά δεδομένα εισόδου ελέγχοντας την έξοδο του μιγαδικού αφαιρέτη

Από αυτά τα dff.vhd, full_adder.vhd και complex_full_adder.vhd είναι τα ίδια με αυτά που χρησιμοποιήθηκαν στη περιγραφή του μιγαδικού αθροιστή.

complex_subtracter.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Subtracter is
    generic    (
       N : POSITIVE := 4
    );
    port (
       clk: in STD_LOGIC;
       clear: in STD_LOGIC;
       x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       d_im: out STD_LOGIC_VECTOR (N downto 0);
       d_re: out STD_LOGIC_VECTOR (N downto 0)
    );
end Complex_Subtracter;

architecture Complex_Subtracter_Structural of Complex_Subtracter is

component dff is
    generic (
       CLEARQ : STD_LOGIC := '0'
    );
    port (
       clear: in STD_LOGIC;
       clk: in STD_LOGIC;
       d: in STD_LOGIC;
       q: out STD_LOGIC
    );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
```

```
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CS_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

signal  a_im,a_re,b_im,b_re,d2_im,d2_re : CS_word_array(0 to N);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(0 to N);

begin
       -- CFA generation
CFAx:   for i in 0 to N-1 generate
CFA:       Complex_full_adder port map(
               a_im => a_im(i)(i),
               a_re => a_re(i)(i),
               b_im => b_im(i)(i),
               b_re => b_re(i)(i),
               c_im_in => c_im_in(i),
               c_re_in => c_re_in(i),
               c_im_out => c_im_out(i),
               c_re_out => c_re_out(i),
               s_im => d2_im(i)(i),
               s_re => d2_re(i)(i)
         );
       end generate;

row1:   for i in 0 to N-1 generate
col1:     for j in 0 to N-1 generate
          -- input Dff generation
cond1:       if j>i and j<N-1 generate
gda_re0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => a_re(i)(j),
                 q => a_re(i+1)(j)
              );
gda_im0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => a_im(i)(j),
                 q => a_im(i+1)(j)
              );
gdb_re0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => b_re(i)(j),
                 q => b_re(i+1)(j)
              );
gdb_im0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => b_im(i)(j),
                 q => b_im(i+1)(j)
```

```
            );
        end generate;
cond2:      if j=N-1 generate
gda_re1:        dff generic map(
                CLEARQ => '1'
            ) port map(
                clear => clear,
                clk => clk,
                d => a_re(i)(j),
                q => a_re(i+1)(j)
            );
gda_im1:        dff generic map(
                CLEARQ => '1'
            ) port map(
                clear => clear,
                clk => clk,
                d => a_im(i)(j),
                q => a_im(i+1)(j)
            );
gdb_re1:        dff generic map(
                CLEARQ => '1'
            ) port map(
                clear => clear,
                clk => clk,
                d => b_re(i)(j),
                q => b_re(i+1)(j)
            );
gdb_im1:        dff generic map(
                CLEARQ => '1'
            ) port map(
                clear => clear,
                clk => clk,
                d => b_im(i)(j),
                q => b_im(i+1)(j)
            );
        end generate;
        -- output Dff generation
cond3:      if j<=i generate
gds_re0:        dff generic map(
                CLEARQ => '0'
            ) port map(
                clear => clear,
                clk => clk,
                d => d2_re(i)(j),
                q => d2_re(i+1)(j)
            );
gds_im0:        dff generic map(
                CLEARQ => '0'
            ) port map(
                clear => clear,
                clk => clk,
                d => d2_im(i)(j),
                q => d2_im(i+1)(j)
            );
        end generate;
        -- carry out-in Dff generation
cond4:      if i=j and i<N-1 generate
gds_re0:        dff generic map(
                CLEARQ => '0'
            ) port map(
                clear => clear,
                clk => clk,
                d => c_re_out(i),
                q => c_re_in(i+1)
            );
gds_im0:        dff generic map(
                CLEARQ => '0'
            ) port map(
```

```
                clear => clear,
                clk => clk,
                d => c_im_out(i),
                q => c_im_in(i+1)
            );
         end generate;

        end generate;
         end generate;
         -- carry out Dff generation
gds_re1:dff generic map(
        CLEARQ => '1'
      ) port map(
        clear => clear,
        clk => clk,
        d => c_re_out(N-1),
        q => c_re_out(N)
      );
gds_im1:dff generic map(
        CLEARQ => '1'
      ) port map(
        clear => clear,
        clk => clk,
        d => c_im_out(N-1),
        q => c_im_out(N)
      );

-- input wires generation
    a_re(0) <= (not x_re(N-1))&x_re(N-2 downto 0);
    a_im(0) <= (not x_im(N-1))&x_im(N-2 downto 0);
    b_re(0) <= y_re(N-1)&(not y_re(N-2 downto 0));
    b_im(0) <= y_im(N-1)&(not y_im(N-2 downto 0));
    c_re_in(0) <= '1';
    c_im_in(0) <= '1';

-- output wires generation
    d_re <= (not c_re_out(N))&d2_re(N);
    d_im <= (not c_im_out(N))&d2_im(N);

end Complex_Subtracter_Structural;
```

complex_subtracter_tb.vhd

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

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;

entity complex_subtracter_tb is
     -- Generic declarations of the tested unit
          generic(
          N : INTEGER := 4 );
end complex_subtracter_tb;

architecture TB_ARCHITECTURE of complex_subtracter_tb is

file OUTPUT_FILE: TEXT open WRITE_MODE is "outputs.txt";

     -- Component declaration of the tested unit
     component complex_subtracter
    generic    (
       N : POSITIVE := 4
    );
```

```
    port (
       clk: in STD_LOGIC;
       clear: in STD_LOGIC;
       x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       d_im: out STD_LOGIC_VECTOR (N downto 0);
       d_re: out STD_LOGIC_VECTOR (N downto 0) );
     end component;

     constant DELAY1 : time := 10 ns;

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal clk : std_logic;
    signal clear : std_logic := '0';
    signal x_im : std_logic_vector(N-1 downto 0);
    signal x_re : std_logic_vector(N-1 downto 0);
    signal y_im : std_logic_vector(N-1 downto 0);
    signal y_re : std_logic_vector(N-1 downto 0);
    -- Observed signals - signals mapped to the output ports of tested entity
    signal d_im : std_logic_vector(N downto 0);
    signal d_re : std_logic_vector(N downto 0);

     -- Add your code here ...

begin

     -- Unit Under Test port map
     UUT : complex_subtracter
          generic map (
               N => N
          )

          port map (
               clk => clk,
               clear => clear,
               x_im => x_im,
               x_re => x_re,
               y_im => y_im,
               y_re => y_re,
               d_im => d_im,
               d_re => d_re
          );

    -- Add your stimulus here ...
    CLK_Generation: process
    begin
       clk<='0';wait for DELAY1;
       clk<='1';wait for DELAY1;
    end process;

    INPUT_Generation: process
    variable ny_re : integer;
    variable ny_im : integer;
    variable nx_re : integer;
    variable nx_im : integer;

    begin
       x_re <= CONV_STD_LOGIC_VECTOR(0,N);
       x_im <= CONV_STD_LOGIC_VECTOR(0,N);
       y_re <= CONV_STD_LOGIC_VECTOR(0,N);
       y_im <= CONV_STD_LOGIC_VECTOR(0,N);
       clear <= '0';
       wait until (clk'event) and (clk='1');
       clear <= '1';

       for nx_re in -2**(N-1) to 2**(N-1)-1 loop
```

```
      for nx_im in -2**(N-1) to 2**(N-1)-1 loop
       for ny_re in -2**(N-1) to 2**(N-1)-1 loop
             for ny_im in -2**(N-1) to 2**(N-1)-1 loop
              x_re <= CONV_STD_LOGIC_VECTOR(nx_re,N);
              x_im <= CONV_STD_LOGIC_VECTOR(nx_im,N);
              y_re <= CONV_STD_LOGIC_VECTOR(ny_re,N);
              y_im <= CONV_STD_LOGIC_VECTOR(ny_im,N);
              wait until (clk'event) and (clk='1');
            end loop;
       end loop;
      end loop;
   end loop;
   wait;

end process;

OUTPUT_Checking:process
   type XY_Matrix is array (NATURAL range <>) of std_logic_vector(0 to N-1);

   variable X_Re_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));
   variable X_Im_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));
   variable Y_Re_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));
   variable Y_Im_Matrix : XY_Matrix (0 to N):=(others=>(others=>'0'));

   variable report_line : line;
   variable report_str : string(1 to 160);
   variable i,j,re_part, im_part : integer := 0;

begin

   wait until (clk'event) and (clk='1') and (clear='1');

   for i in 0 to N-1 loop
     X_Re_Matrix(i) := X_Re_Matrix(i+1);
     X_Im_Matrix(i) := X_Im_Matrix(i+1);
     Y_Re_Matrix(i) := Y_Re_Matrix(i+1);
     Y_Im_Matrix(i) := Y_Im_Matrix(i+1);
   end loop;
   X_Re_Matrix(N) := x_re;
   X_Im_Matrix(N) := x_im;
   Y_Re_Matrix(N) := y_re;
   Y_Im_Matrix(N) := y_im;

   re_part := conv_integer(X_Re_Matrix(0))-conv_integer(Y_Re_Matrix(0));
   im_part := conv_integer(X_Im_Matrix(0))-conv_integer(Y_Im_Matrix(0));

   report_line:=null;

   if ((conv_integer(d_re) /= re_part) or (conv_integer(d_im) /= im_part)) then

     write(report_line,string'(">>>>>>>>>>Wrong Result: "));

     write(report_line,conv_integer(X_Re_Matrix(0)));
     write(report_line,string'("+i"));
     write(report_line,conv_integer(X_Im_Matrix(0)));
     write(report_line,string'(" + "));
     write(report_line,conv_integer(Y_Re_Matrix(0)));
     write(report_line,string'("+i"));
     write(report_line,conv_integer(Y_Im_Matrix(0)));
     write(report_line,string'(" = "));
     write(report_line,conv_integer(d_re));
     write(report_line,string'("+i"));
     write(report_line,conv_integer(d_im));

     write(report_line,string'("  Right: "));
     write(report_line,conv_integer(re_part));
     write(report_line,string'("+i"));
     write(report_line,conv_integer(im_part));
```

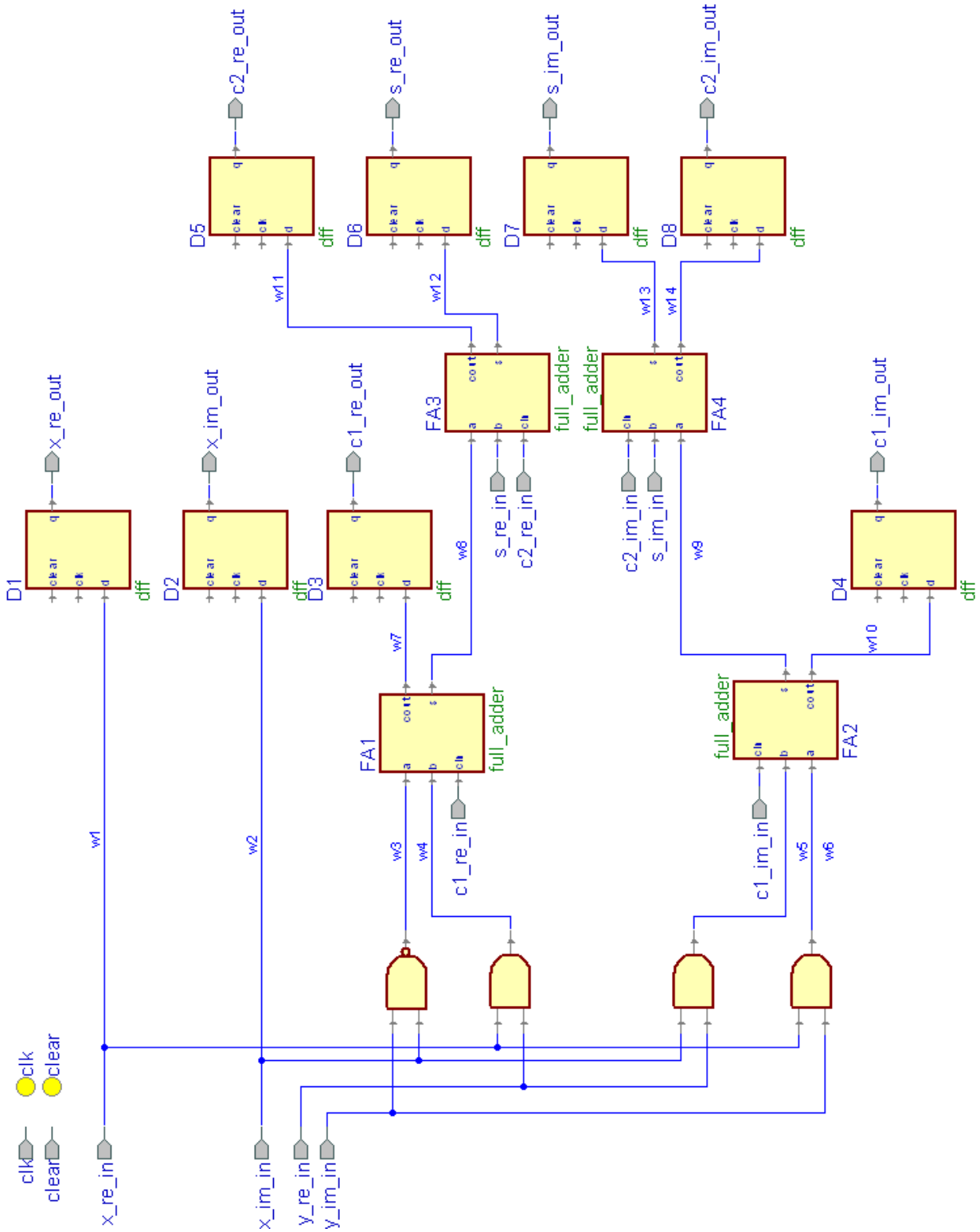
```
        write(report_line,string'("       "));

        for i in report_line'range loop
         report_str(i) := report_line(i);
        end loop;
        report report_str;

        writeline(OUTPUT_FILE,report_line);

      end if;

      if ( (conv_integer(X_Re_Matrix(0)) = 2**(N-1)-1) and
            (conv_integer(X_Im_Matrix(0)) = 2**(N-1)-1) and
         (conv_integer(Y_Re_Matrix(0)) = 2**(N-1)-1) and
            (conv_integer(Y_Im_Matrix(0)) = 2**(N-1)-1) ) then
        write(report_line,string'("------->The End!!! "));
        writeline(OUTPUT_FILE,report_line);
        report "Test Finished!!!" severity FAILURE;
      end if;

    end process;


end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_complex_subtracter of complex_subtracter_tb is
     for TB_ARCHITECTURE
          for UUT : complex_subtracter
               use entity work.complex_subtracter(complex_subtracter_structural);
          end for;
     end for;
end TESTBENCH_FOR_complex_subtracter;
```

C. Περιγραφή του μιγαδικού πολλαπλασιαστή σε VHDL

Περιγραφή σε κώδικα VHDL του μιγαδικού πολλαπλασιαστή συνεχούς διοχέτευσης χρησιμοποιώντας μιγαδικούς αθροιστές με αποθήκευση κρατουμένου που περιγράφηκε στην παράγραφο 1.3.7. Τα αρχεία που χρησιμοποιούμε για την περιγραφή είναι τα εξής:

dff.vhd	Περιγραφή του D Flip-Flop που χρησιμοποιείτε στον μιγαδικό αφαιρέτη
full_adder.vhd	Περιγραφή του πλήρη αθροιστή
cmc1_cell.vhd	Περιγραφή του 1ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή με βάση το block diagram που βρίσκεται στο αρχείο cmc1_cell.bde
cmc2_cell.vhd	Περιγραφή του 2ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή με βάση το block diagram που βρίσκεται στο αρχείο cmc2_cell.bde
cmc3_cell.vhd	Περιγραφή του 3ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή με βάση το block diagram που βρίσκεται στο αρχείο cmc3_cell.bde
cmc4_cell.vhd	Περιγραφή του 4ου τύπου κυττάρου του μιγαδικού πολλαπλασιαστή με βάση το block diagram που βρίσκεται στο αρχείο cmc4_cell.bde
cfa1_cell.vhd	Περιγραφή του 1ου τύπου κυττάρου μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικού πολλαπλασιαστή με βάση το block diagram που βρίσκεται στο αρχείο cfa1_cell.bde
cfa2_cell.vhd	Περιγραφή του 2ου τύπου κυττάρου μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικού πολλαπλασιαστή με βάση το block diagram που βρίσκεται στο αρχείο cfa2_cell.bde
complex_multiplier.vhd	Περιγραφή ολόκληρου του μιγαδικού πολλαπλασιαστή
complex_multiplier_tb.vhd	Test bench που εκτελεί εξαντλητικό test με όλα τα δυνατά δεδομένα εισόδου ελέγχοντας την έξοδο του μιγαδικού πολλαπλασιαστή

Από αυτά τα dff.vhd και full_adder.vhd είναι τα ίδια με αυτά που χρησιμοποιήθηκαν στη περιγραφή του μιγαδικού αθροιστή όπως και του μιγαδικού αφαιρέτη.



Σχήμα C.1 Η περιγραφή των 1ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc1_cell.bde

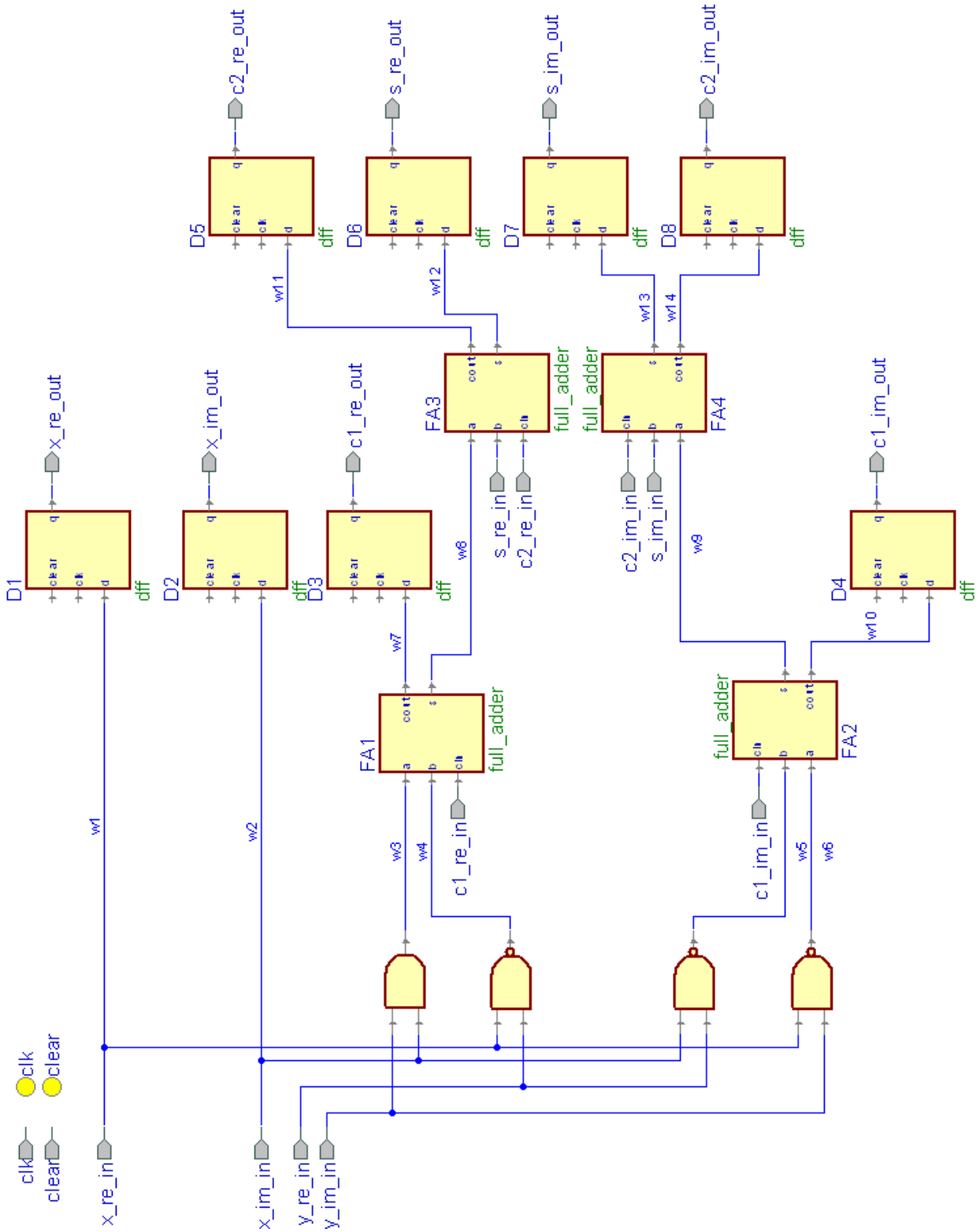
cmc1_cell.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity CMC1_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end CMC1_cell;

architecture CMC1_cell_structural of CMC1_cell is

---- Component declarations -----

component dff
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----

signal w1 : STD_LOGIC;
signal w10 : STD_LOGIC;
signal w11 : STD_LOGIC;
signal w12 : STD_LOGIC;
signal w13 : STD_LOGIC;
signal w14 : STD_LOGIC;
signal w2 : STD_LOGIC;
signal w3 : STD_LOGIC;
```

```
signal w4 : STD_LOGIC;
signal w5 : STD_LOGIC;
signal w6 : STD_LOGIC;
signal w7 : STD_LOGIC;
signal w8 : STD_LOGIC;
signal w9 : STD_LOGIC;

begin

----  Component instantiations  ----

D1 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w1,
       q => x_re_out
  );

D2 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w2,
       q => x_im_out
  );

D3 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w7,
       q => c1_re_out
  );

D4 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w10,
       q => c1_im_out
  );

D5 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w11,
       q => c2_re_out
  );

D6 : dff
  generic map (
```

```
        CLEARQ => '0'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w12,
        q => s_re_out
  );

D7 : dff
  generic map (
        CLEARQ => '0'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w13,
        q => s_im_out
  );

D8 : dff
  generic map (
        CLEARQ => '0'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w14,
        q => c2_im_out
  );

FA1 : full_adder
  port map(
        a => w3,
        b => w4,
        cin => c1_re_in,
        cout => w7,
        s => w8
  );

FA2 : full_adder
  port map(
        a => w6,
        b => w5,
        cin => c1_im_in,
        cout => w10,
        s => w9
  );

FA3 : full_adder
  port map(
        a => w8,
        b => s_re_in,
        cin => c2_re_in,
        cout => w11,
        s => w12
  );

FA4 : full_adder
  port map(
        a => w9,
        b => s_im_in,
        cin => c2_im_in,
        cout => w14,
        s => w13
  );

w3 <= not(w2 and y_im_in);
```

```
w6 <= y_im_in and w1;

w4 <= y_re_in and w1;

w5 <= y_re_in and w2;


---- Terminal assignment ----

    -- Inputs terminals
     w2 <= x_im_in;
     w1 <= x_re_in;


end CMC1_cell_structural;
```



Σχήμα C.2 Η περιγραφή των 2ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc2_cell.bde

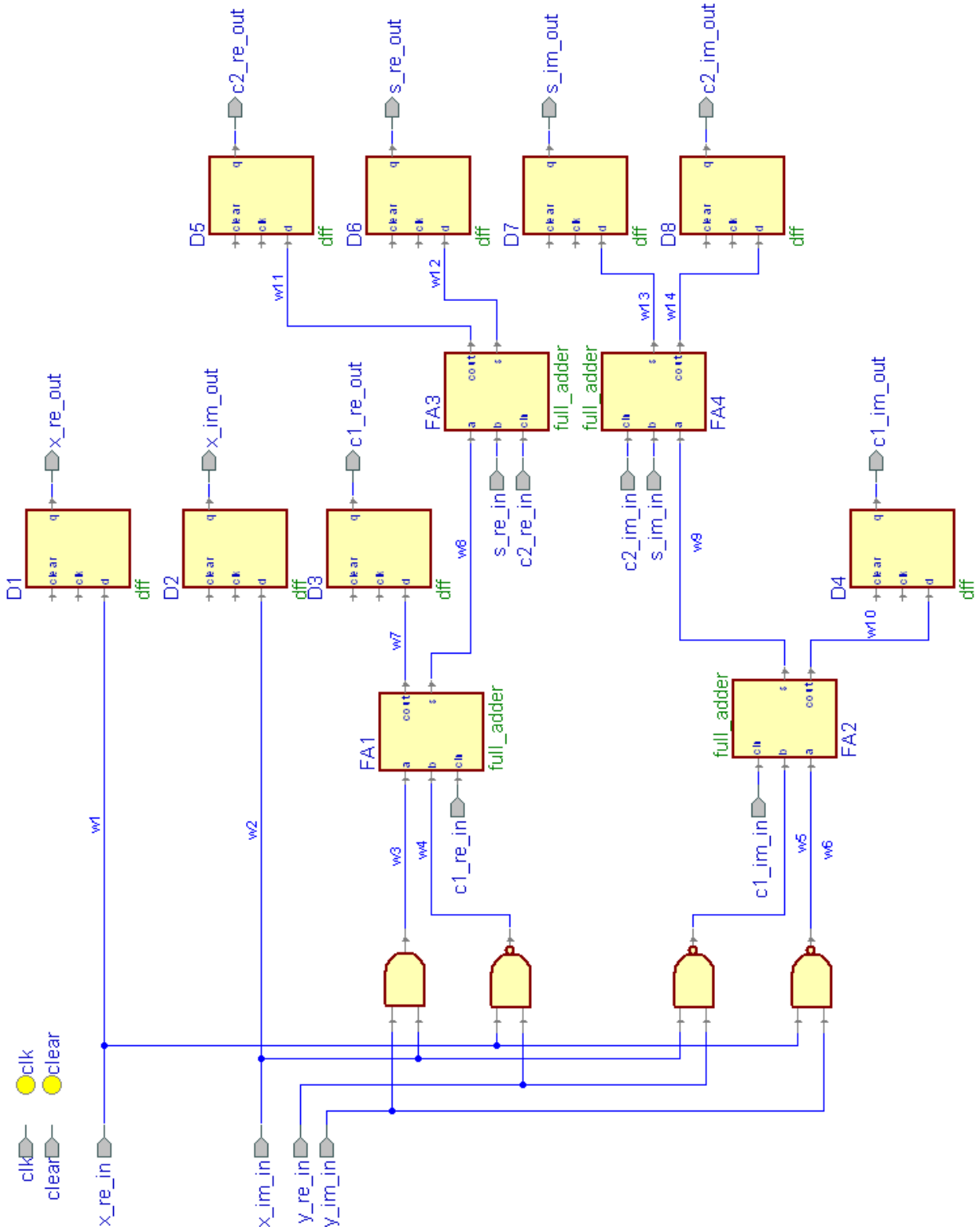
cmc2_cell.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity CMC2_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end CMC2_cell;

architecture CMC2_cell_structural of CMC2_cell is

---- Component declarations -----

component dff
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----

signal w1 : STD_LOGIC;
signal w10 : STD_LOGIC;
signal w11 : STD_LOGIC;
signal w12 : STD_LOGIC;
signal w13 : STD_LOGIC;
signal w14 : STD_LOGIC;
signal w2 : STD_LOGIC;
signal w3 : STD_LOGIC;
```

```
signal w4 : STD_LOGIC;
signal w5 : STD_LOGIC;
signal w6 : STD_LOGIC;
signal w7 : STD_LOGIC;
signal w8 : STD_LOGIC;
signal w9 : STD_LOGIC;

begin

----  Component instantiations  ----

D1 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w1,
       q => x_re_out
  );

D2 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w2,
       q => x_im_out
  );

D3 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w7,
       q => c1_re_out
  );

D4 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w10,
       q => c1_im_out
  );

D5 : dff
  generic map (
       CLEARQ => '0'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w11,
       q => c2_re_out
  );

D6 : dff
  generic map (
```

```
        CLEARQ => '0'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w12,
        q => s_re_out
  );

D7 : dff
  generic map (
        CLEARQ => '0'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w13,
        q => s_im_out
  );

D8 : dff
  generic map (
        CLEARQ => '1'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w14,
        q => c2_im_out
  );

FA1 : full_adder
  port map(
        a => w3,
        b => w4,
        cin => c1_re_in,
        cout => w7,
        s => w8
  );

FA2 : full_adder
  port map(
        a => w6,
        b => w5,
        cin => c1_im_in,
        cout => w10,
        s => w9
  );

FA3 : full_adder
  port map(
        a => w8,
        b => s_re_in,
        cin => c2_re_in,
        cout => w11,
        s => w12
  );

FA4 : full_adder
  port map(
        a => w9,
        b => s_im_in,
        cin => c2_im_in,
        cout => w14,
        s => w13
  );

w6 <= not(y_im_in and w1);
```

```
w5 <= not(y_re_in and w2);

w4 <= not(y_re_in and w1);

w3 <= w2 and y_im_in;


---- Terminal assignment ----

    -- Inputs terminals
     w2 <= x_im_in;
     w1 <= x_re_in;


end CMC2_cell_structural;
```



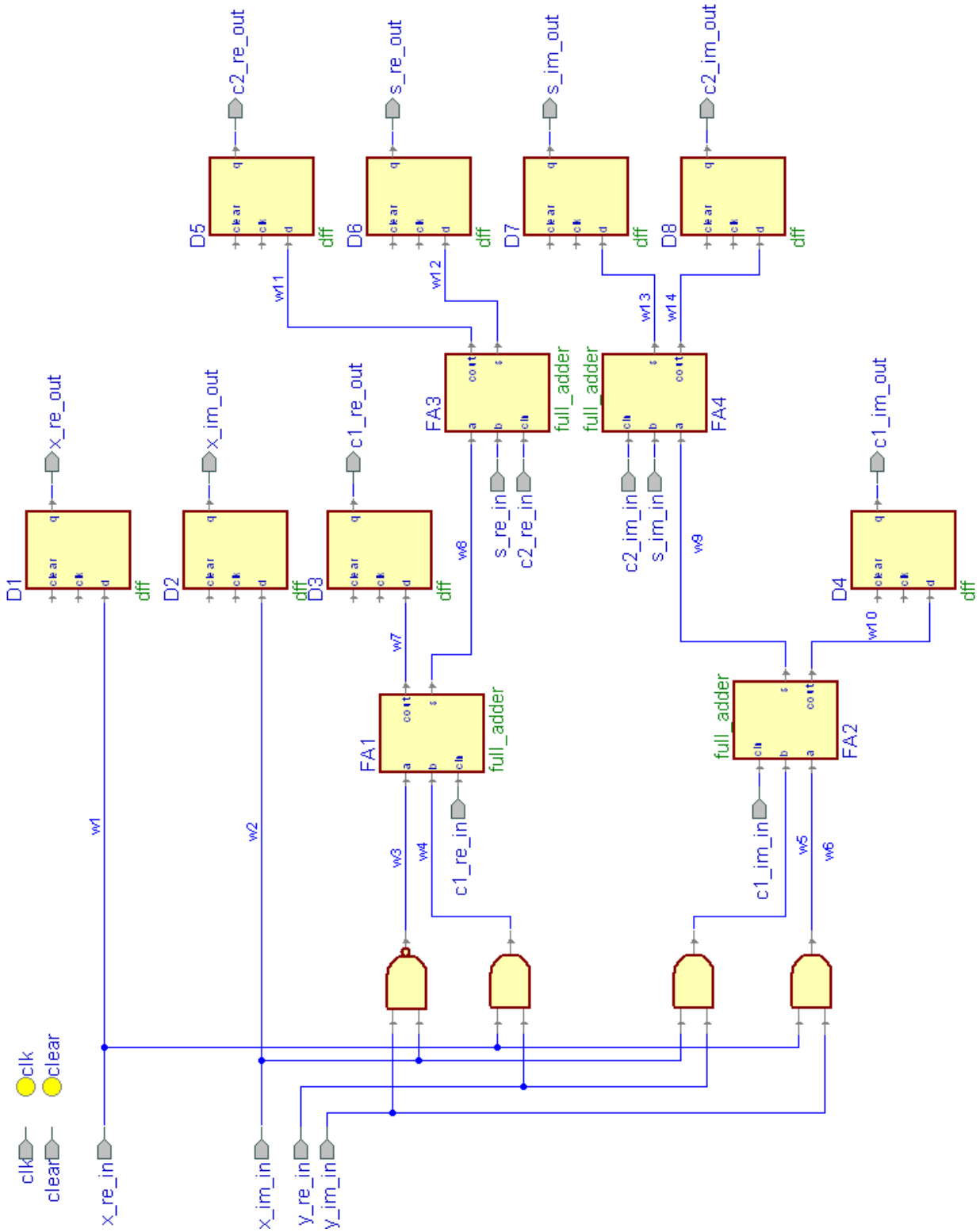
Σχήμα C.3 Η περιγραφή των 3ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc3_cell.bde

cmc3_cell.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity CMC3_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end CMC3_cell;

architecture CMC3_cell of CMC3_cell is

---- Component declarations -----

component dff
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----

signal w1 : STD_LOGIC;
signal w10 : STD_LOGIC;
signal w11 : STD_LOGIC;
signal w12 : STD_LOGIC;
signal w13 : STD_LOGIC;
signal w14 : STD_LOGIC;
signal w2 : STD_LOGIC;
signal w3 : STD_LOGIC;
```

```
signal w4 : STD_LOGIC;
signal w5 : STD_LOGIC;
signal w6 : STD_LOGIC;
signal w7 : STD_LOGIC;
signal w8 : STD_LOGIC;
signal w9 : STD_LOGIC;

begin

----  Component instantiations  ----

D1 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w1,
       q => x_re_out
  );

D2 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w2,
       q => x_im_out
  );

D3 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w7,
       q => c1_re_out
  );

D4 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w10,
       q => c1_im_out
  );

D5 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w11,
       q => c2_re_out
  );

D6 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w12,
       q => s_re_out
  );

D7 : dff
  port map(
       clear => clear,
```

```
       clk => clk,
       d => w13,
       q => s_im_out
  );

D8 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w14,
       q => c2_im_out
  );

FA1 : full_adder
  port map(
       a => w3,
       b => w4,
       cin => c1_re_in,
       cout => w7,
       s => w8
  );

FA2 : full_adder
  port map(
       a => w6,
       b => w5,
       cin => c1_im_in,
       cout => w10,
       s => w9
  );

FA3 : full_adder
  port map(
       a => w8,
       b => s_re_in,
       cin => c2_re_in,
       cout => w11,
       s => w12
  );

FA4 : full_adder
  port map(
       a => w9,
       b => s_im_in,
       cin => c2_im_in,
       cout => w14,
       s => w13
  );

w6 <= not(y_im_in and w1);

w5 <= not(y_re_in and w2);

w4 <= not(y_re_in and w1);

w3 <= w2 and y_im_in;


---- Terminal assignment ----

    -- Inputs terminals
     w2 <= x_im_in;
     w1 <= x_re_in;


end CMC3_cell;
```



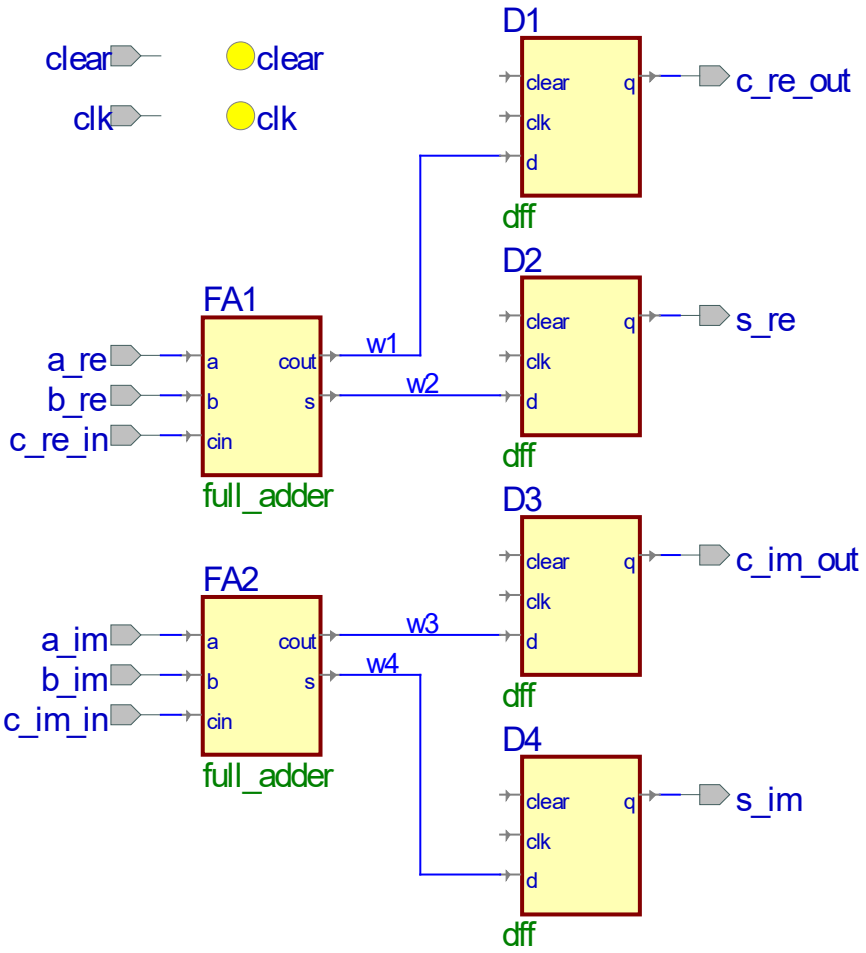
Σχήμα C.4 Η περιγραφή των 4ου τύπου κυττάρων του μιγαδικού πολλαπλασιαστή όπως σχεδιάστηκε στο block diagram αρχείο cmc4_cell.bde

cmc4_cell.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity CMC4_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end CMC4_cell;

architecture CMC4_cell of CMC4_cell is

---- Component declarations -----

component dff
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----

signal w1 : STD_LOGIC;
signal w10 : STD_LOGIC;
signal w11 : STD_LOGIC;
signal w12 : STD_LOGIC;
signal w13 : STD_LOGIC;
signal w14 : STD_LOGIC;
signal w2 : STD_LOGIC;
signal w3 : STD_LOGIC;
signal w4 : STD_LOGIC;
signal w5 : STD_LOGIC;
```

```
signal w6 : STD_LOGIC;
signal w7 : STD_LOGIC;
signal w8 : STD_LOGIC;
signal w9 : STD_LOGIC;

begin

----  Component instantiations  ----

D1 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w1,
       q => x_re_out
  );

D2 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w2,
       q => x_im_out
  );

D3 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w7,
       q => c1_re_out
  );

D4 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w10,
       q => c1_im_out
  );

D5 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w11,
       q => c2_re_out
  );

D6 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w12,
       q => s_re_out
  );

D7 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w13,
       q => s_im_out
  );

D8 : dff
  generic map (
```

```
        CLEARQ => '1'
  )
  port map(
        clear => clear,
        clk => clk,
        d => w14,
        q => c2_im_out
  );

FA1 : full_adder
  port map(
        a => w3,
        b => w4,
        cin => c1_re_in,
        cout => w7,
        s => w8
  );

FA2 : full_adder
  port map(
        a => w6,
        b => w5,
        cin => c1_im_in,
        cout => w10,
        s => w9
  );

FA3 : full_adder
  port map(
        a => w8,
        b => s_re_in,
        cin => c2_re_in,
        cout => w11,
        s => w12
  );

FA4 : full_adder
  port map(
        a => w9,
        b => s_im_in,
        cin => c2_im_in,
        cout => w14,
        s => w13
  );

w3 <= not(w2 and y_im_in);

w6 <= y_im_in and w1;

w4 <= y_re_in and w1;

w5 <= y_re_in and w2;


---- Terminal assignment ----

    -- Inputs terminals
     w2 <= x_im_in;
     w1 <= x_re_in;


end CMC4_cell;
```



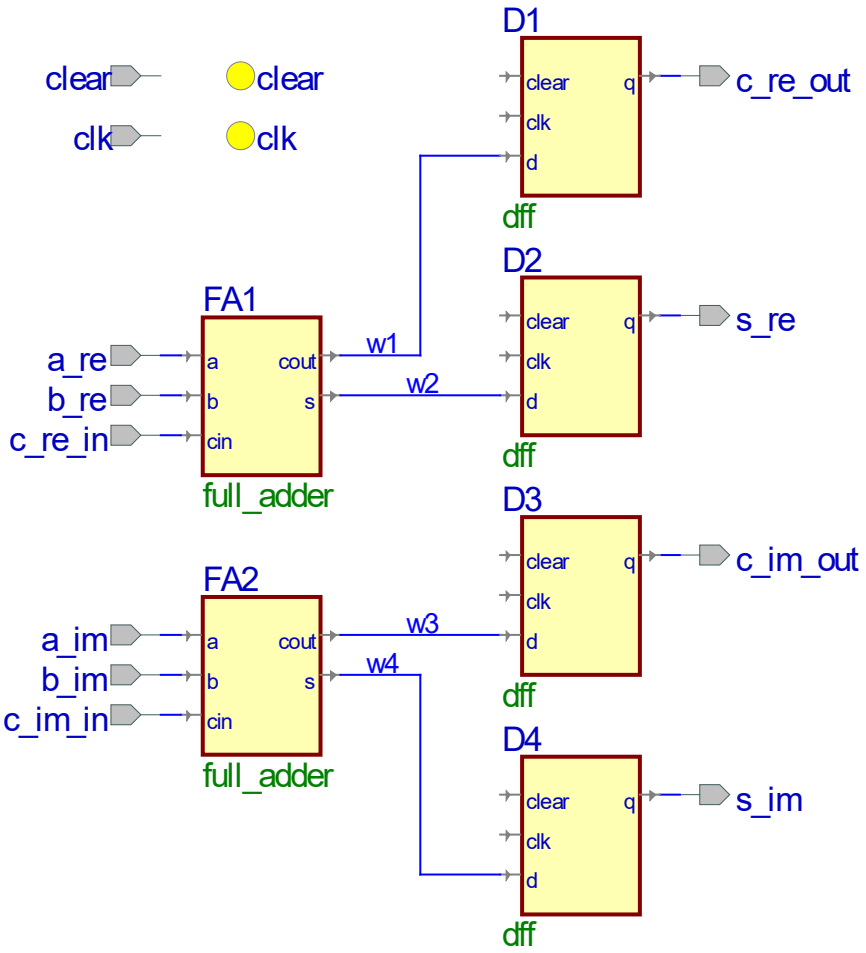
Σχήμα C.5 Περιγραφεί του 1ου τύπου μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή και έχει σχεδιαστεί στο block diagram αρχείο cfa1_cell.bde

cfa1_cell.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity CFA1_cell is
  port(
        a_im : in STD_LOGIC;
        a_re : in STD_LOGIC;
        b_im : in STD_LOGIC;
        b_re : in STD_LOGIC;
        c_im_in : in STD_LOGIC;
        c_re_in : in STD_LOGIC;
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        c_im_out : out STD_LOGIC;
        c_re_out : out STD_LOGIC;
        s_im : out STD_LOGIC;
        s_re : out STD_LOGIC
  );
end CFA1_cell;

architecture CFA1_cell_structural of CFA1_cell is

---- Component declarations -----

component dff
```

```
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----

signal w1 : STD_LOGIC;
signal w2 : STD_LOGIC;
signal w3 : STD_LOGIC;
signal w4 : STD_LOGIC;

begin

----  Component instantiations  ----

D1 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w1,
       q => c_re_out
  );

D2 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w2,
       q => s_re
  );

D3 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w3,
       q => c_im_out
  );

D4 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w4,
       q => s_im
  );

FA1 : full_adder
```

```
  port map(
        a => a_re,
        b => b_re,
        cin => c_re_in,
        cout => w1,
        s => w2
  );

FA2 : full_adder
  port map(
        a => a_im,
        b => b_im,
        cin => c_im_in,
        cout => w3,
        s => w4
  );


end CFA1_cell_structural;
```



Σχήμα C.6 Περιγραφή του 2ου τύπου μιγαδικού πλήρη αθροιστή που χρησιμοποιείται στον μιγαδικό πολλαπλασιαστή και έχει σχεδιαστεί στο block diagram αρχείο cfa2_cell.bde

cfa2_cell.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity CFA2_cell is
  port(
        a_im : in STD_LOGIC;
        a_re : in STD_LOGIC;
        b_im : in STD_LOGIC;
        b_re : in STD_LOGIC;
        c_im_in : in STD_LOGIC;
        c_re_in : in STD_LOGIC;
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        c_im_out : out STD_LOGIC;
        c_re_out : out STD_LOGIC;
        s_im : out STD_LOGIC;
        s_re : out STD_LOGIC
  );
end CFA2_cell;

architecture CFA2_cell_structural of CFA2_cell is

---- Component declarations -----

component dff
```

```
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component full_adder
  port (
       a : in STD_LOGIC;
       b : in STD_LOGIC;
       cin : in STD_LOGIC;
       cout : out STD_LOGIC;
       s : out STD_LOGIC
  );
end component;

---- Signal declarations used on the diagram ----

signal w1 : STD_LOGIC;
signal w2 : STD_LOGIC;
signal w3 : STD_LOGIC;
signal w4 : STD_LOGIC;

begin

----  Component instantiations  ----

D1 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w1,
       q => c_re_out
  );

D2 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w2,
       q => s_re
  );

D3 : dff
  generic map (
       CLEARQ => '1'
  )
  port map(
       clear => clear,
       clk => clk,
       d => w3,
       q => c_im_out
  );

D4 : dff
  port map(
       clear => clear,
       clk => clk,
       d => w4,
       q => s_im
  );

FA1 : full_adder
```

```
  port map(
        a => a_re,
        b => b_re,
        cin => c_re_in,
        cout => w1,
        s => w2
  );

FA2 : full_adder
  port map(
        a => a_im,
        b => b_im,
        cin => c_im_in,
        cout => w3,
        s => w4
  );


end CFA2_cell_structural;
```

complex_multiplier.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Multiplier is
     generic    (
           NX : POSITIVE := 4;
           NY : POSITIVE := 5
     );
     port (
           clk: in STD_LOGIC;
           clear: in STD_LOGIC;
           x_im: in STD_LOGIC_VECTOR (NX-1 downto 0);
           x_re: in STD_LOGIC_VECTOR (NX-1 downto 0);
           y_im: in STD_LOGIC_VECTOR (NY-1 downto 0);
           y_re: in STD_LOGIC_VECTOR (NY-1 downto 0);
           m_im: out STD_LOGIC_VECTOR (NX+NY downto 0);
           m_re: out STD_LOGIC_VECTOR (NX+NY downto 0)
     );
end Complex_Multiplier;

--}} End of automatically maintained section

architecture Complex_Multiplier_Structural of Complex_Multiplier is
subtype x_word is STD_LOGIC_VECTOR (NX-1 downto 0);
subtype dy_word is STD_LOGIC_VECTOR (NY-1 downto 0);
subtype CFA_word is STD_LOGIC_VECTOR (NX downto 0);
subtype dm_word is STD_LOGIC_VECTOR (NY+NX downto 0);
type x_word_array is array(natural range <>) of x_word;
type y_word_array is array(natural range <>) of dy_word;
type CFA_word_array is array(natural range <>) of CFA_word;
type dm_word_array is array(natural range <>) of dm_word;

signal     c1_im_in,c1_re_in,c2_im_in,c2_re_in,
           s_im_in,s_re_in,x_im_in,x_re_in,y_im_in,y_re_in,
           c1_im_out,c1_re_out,c2_im_out,c2_re_out,
           s_im_out,s_re_out,x_im_out,x_re_out : x_word_array(0 to NY-1);

signal  dy_re,dy_im : y_word_array(0 to NY-1);

signal     a_im,a_re,b_im,b_re,c_im_in,
           c_re_in,c_im_out,c_re_out,s_im,s_re : CFA_word_array(NY to NY+NX);

signal  dm_re,dm_im : dm_word_array(0 to NY+NX);


component dff is
```

```
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component CMC1_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end component;

component CMC2_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end component;

component CMC3_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
```

```
      s_re_in : in STD_LOGIC;
      x_im_in : in STD_LOGIC;
      x_re_in : in STD_LOGIC;
      y_im_in : in STD_LOGIC;
      y_re_in : in STD_LOGIC;
      c1_im_out : out STD_LOGIC;
      c1_re_out : out STD_LOGIC;
      c2_im_out : out STD_LOGIC;
      c2_re_out : out STD_LOGIC;
      s_im_out : out STD_LOGIC;
      s_re_out : out STD_LOGIC;
      x_im_out : out STD_LOGIC;
      x_re_out : out STD_LOGIC
  );
end component;

component CMC4_cell is
  port(
      c1_im_in : in STD_LOGIC;
      c1_re_in : in STD_LOGIC;
      c2_im_in : in STD_LOGIC;
      c2_re_in : in STD_LOGIC;
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      s_im_in : in STD_LOGIC;
      s_re_in : in STD_LOGIC;
      x_im_in : in STD_LOGIC;
      x_re_in : in STD_LOGIC;
      y_im_in : in STD_LOGIC;
      y_re_in : in STD_LOGIC;
      c1_im_out : out STD_LOGIC;
      c1_re_out : out STD_LOGIC;
      c2_im_out : out STD_LOGIC;
      c2_re_out : out STD_LOGIC;
      s_im_out : out STD_LOGIC;
      s_re_out : out STD_LOGIC;
      x_im_out : out STD_LOGIC;
      x_re_out : out STD_LOGIC
  );
end component;

component CFA1_cell is
  port(
      a_im : in STD_LOGIC;
      a_re : in STD_LOGIC;
      b_im : in STD_LOGIC;
      b_re : in STD_LOGIC;
      c_im_in : in STD_LOGIC;
      c_re_in : in STD_LOGIC;
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      c_im_out : out STD_LOGIC;
      c_re_out : out STD_LOGIC;
      s_im : out STD_LOGIC;
      s_re : out STD_LOGIC
  );
end component;

component CFA2_cell is
  port(
      a_im : in STD_LOGIC;
      a_re : in STD_LOGIC;
      b_im : in STD_LOGIC;
      b_re : in STD_LOGIC;
      c_im_in : in STD_LOGIC;
      c_re_in : in STD_LOGIC;
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      c_im_out : out STD_LOGIC;
      c_re_out : out STD_LOGIC;
```

```
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

begin
  -- <<enter your statements here>>

  -- cell generation
          -- CMC cell generation
row1:for i in 0 to NY-2 generate
               -- CMC1 cell generation
col1:      for j in NX-2 downto 0 generate
CMC1:           CMC1_cell port map (
                      c1_im_in => c1_im_in(i)(j),
                      c1_re_in => c1_re_in(i)(j),
                      c2_im_in => c2_im_in(i)(j),
                      c2_re_in => c2_re_in(i)(j),
                      clear => clear,
                      clk => clk,
                      s_im_in => s_im_in(i)(j),
                      s_re_in => s_re_in(i)(j),
                      x_im_in => x_im_in(i)(j),
                      x_re_in => x_re_in(i)(j),
                      y_im_in => y_im_in(i)(j),
                      y_re_in => y_re_in(i)(j),
                      c1_im_out => c1_im_out(i)(j),
                      c1_re_out => c1_re_out(i)(j),
                      c2_im_out => c2_im_out(i)(j),
                      c2_re_out => c2_re_out(i)(j),
                      s_im_out => s_im_out(i)(j),
                      s_re_out => s_re_out(i)(j),
                      x_im_out => x_im_out(i)(j),
                      x_re_out => x_re_out(i)(j)
                    );
               end generate;
               -- CMC2_cell generation ( for X MSB )
CMC2:      CMC2_cell port map (
                 c1_im_in => c1_im_in(i)(NX-1),
                 c1_re_in => c1_re_in(i)(NX-1),
                 c2_im_in => c2_im_in(i)(NX-1),
                 c2_re_in => c2_re_in(i)(NX-1),
                 clear => clear,
                 clk => clk,
                 s_im_in => s_im_in(i)(NX-1),
                 s_re_in => s_re_in(i)(NX-1),
                 x_im_in => x_im_in(i)(NX-1),
                 x_re_in => x_re_in(i)(NX-1),
                 y_im_in => y_im_in(i)(NX-1),
                 y_re_in => y_re_in(i)(NX-1),
                 c1_im_out => c1_im_out(i)(NX-1),
                 c1_re_out => c1_re_out(i)(NX-1),
                 c2_im_out => c2_im_out(i)(NX-1),
                 c2_re_out => c2_re_out(i)(NX-1),
                 s_im_out => s_im_out(i)(NX-1),
                 s_re_out => s_re_out(i)(NX-1),
                 x_im_out => x_im_out(i)(NX-1),
                 x_re_out => x_re_out(i)(NX-1)
               );
          end generate;

          -- CMC3_cell generation (for Y MSB)
col2:for j in NX-2 downto 0 generate
CMC3:      CMC3_cell port map (
                 c1_im_in => c1_im_in(NY-1)(j),
                 c1_re_in => c1_re_in(NY-1)(j),
                 c2_im_in => c2_im_in(NY-1)(j),
                 c2_re_in => c2_re_in(NY-1)(j),
                 clear => clear,
                 clk => clk,
```

```
                s_im_in => s_im_in(NY-1)(j),
                s_re_in => s_re_in(NY-1)(j),
                x_im_in => x_im_in(NY-1)(j),
                x_re_in => x_re_in(NY-1)(j),
                y_im_in => y_im_in(NY-1)(j),
                y_re_in => y_re_in(NY-1)(j),
                c1_im_out => c1_im_out(NY-1)(j),
                c1_re_out => c1_re_out(NY-1)(j),
                c2_im_out => c2_im_out(NY-1)(j),
                c2_re_out => c2_re_out(NY-1)(j),
                s_im_out => s_im_out(NY-1)(j),
                s_re_out => s_re_out(NY-1)(j),
                x_im_out => x_im_out(NY-1)(j),
                x_re_out => x_re_out(NY-1)(j)
              );
        end generate;

        -- CMC4_cell generation ( for X and Y MSB )
CMC4:CMC4_cell port map (
                c1_im_in => c1_im_in(NY-1)(NX-1),
                c1_re_in => c1_re_in(NY-1)(NX-1),
                c2_im_in => c2_im_in(NY-1)(NX-1),
                c2_re_in => c2_re_in(NY-1)(NX-1),
                clear => clear,
                clk => clk,
                s_im_in => s_im_in(NY-1)(NX-1),
                s_re_in => s_re_in(NY-1)(NX-1),
                x_im_in => x_im_in(NY-1)(NX-1),
                x_re_in => x_re_in(NY-1)(NX-1),
                y_im_in => y_im_in(NY-1)(NX-1),
                y_re_in => y_re_in(NY-1)(NX-1),
                c1_im_out => c1_im_out(NY-1)(NX-1),
                c1_re_out => c1_re_out(NY-1)(NX-1),
                c2_im_out => c2_im_out(NY-1)(NX-1),
                c2_re_out => c2_re_out(NY-1)(NX-1),
                s_im_out => s_im_out(NY-1)(NX-1),
                s_re_out => s_re_out(NY-1)(NX-1),
                x_im_out => x_im_out(NY-1)(NX-1),
                x_re_out => x_re_out(NY-1)(NX-1)
        );

        -- CFA_cell generation

        -- CFA1 type cell generation
col3:for j in NX-1 downto 0 generate
CFA2:     CFA1_cell port map (
                    a_im => a_im(NY)(j),
                    a_re => a_re(NY)(j),
                    b_im => b_im(NY)(j),
                    b_re => b_re(NY)(j),
                    c_im_in => c_im_in(NY)(j),
                    c_re_in => c_re_in(NY)(j),
                    clear => clear,
                    clk => clk,
                    c_im_out => c_im_out(NY)(j),
                    c_re_out => c_re_out(NY)(j),
                    s_im => s_im(NY)(j),
                    s_re => s_re(NY)(j)
              );
        end generate;
        -- Last CFA1 type cell generation who make the calculation of the MSB
LCFA2:    CFA1_cell port map (
                    a_im => a_im(NY+NX)(NX),
                    a_re => a_re(NY+NX)(NX),
                    b_im => '0',
                    b_re => '0',
                    c_im_in => c_im_in(NY+NX)(NX),
                    c_re_in => c_re_in(NY+NX)(NX),
                    clear => clear,
                    clk => clk,
```

```
                        c_im_out => c_im_out(NY+NX)(NX),
                        c_re_out => c_re_out(NY+NX)(NX),
                        s_im => s_im(NY+NX)(NX),
                        s_re => s_re(NY+NX)(NX)
            );
            -- CFA2 type cell generation
row4:for i in 1 to NX-1 generate
CFA1:       CFA2_cell port map (
                        a_im => a_im(NY+i)(i),
                        a_re => a_re(NY+i)(i),
                        b_im => b_im(NY+i)(i),
                        b_re => b_re(NY+i)(i),
                        c_im_in => c_im_in(NY+i)(i),
                        c_re_in => c_re_in(NY+i)(i),
                        clear => clear,
                        clk => clk,
                        c_im_out => c_im_out(NY+i)(i),
                        c_re_out => c_re_out(NY+i)(i),
                        s_im => s_im(NY+i)(i),
                        s_re => s_re(NY+i)(i)
                );
            end generate;

            -- CFA intermediate dff generation and connection
row5:for i in 1 to NX-1 generate
col5:       for j in NX downto i+1 generate
                        -- dff generation for CFA a inputs
gdCFAare:         dff generic map (
                              CLEARQ => '0'
                        ) port map (
                              clear => clear,
                              clk => clk,
                              d => a_re(NY+i)(j),
                              q => a_re(NY+i+1)(j)
                        );
gdCFAaim:         dff generic map (
                              CLEARQ => '0'
                        ) port map (
                              clear => clear,
                              clk => clk,
                              d => a_im(NY+i)(j),
                              q => a_im(NY+i+1)(j)
                        );
                        -- dff generation for CFA b inputs
cond1:                        if j<NX generate
gdCFAbre:               dff generic map (
                                    CLEARQ => '1'
                              ) port map (
                                    clear => clear,
                                    clk => clk,
                                    d => b_re(NY+i)(j),
                                    q => b_re(NY+i+1)(j)
                              );
gdCFAbim:               dff generic map (
                                    CLEARQ => '1'
                              ) port map (
                                    clear => clear,
                                    clk => clk,
                                    d => b_im(NY+i)(j),
                                    q => b_im(NY+i+1)(j)
                              );
                        end generate;
                  end generate;
            end generate;

            -- input X dff generation
row6:for i in 0 to NY-2 generate
col6:       for j in NY-1 downto i+1 generate
gdyre:                  dff generic map (
                              CLEARQ => '0'
```

```
                    ) port map (
                         clear => clear,
                         clk => clk,
                         d => dy_re(i)(j),
                         q => dy_re(i+1)(j)
                    );
gdyim:              dff generic map (
                         CLEARQ => '0'
                    ) port map (
                         clear => clear,
                         clk => clk,
                         d => dy_im(i)(j),
                         q => dy_im(i+1)(j)
                    );
               end generate;
          end generate;

          -- output m dff generation
row7:for i in 1 to NY+NX generate
col7:     for j in i-1 downto 0 generate
cond2:              if j /= NY generate
gdmre0:                  dff generic map (
                              CLEARQ => '0'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_re(i-1)(j),
                              q => dm_re(i)(j)
                         );
gdmim0:                  dff generic map (
                              CLEARQ => '0'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_im(i-1)(j),
                              q => dm_im(i)(j)
                         );
                    end generate;
cond3:              if j = NY generate
gdmre1:                  dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_re(i-1)(j),
                              q => dm_re(i)(j)
                         );
gdmim1:                  dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_im(i-1)(j),
                              q => dm_im(i)(j)
                         );
                    end generate;
               end generate;
          end generate;

  -- intermediate wires generation
          -- CMC intermediate wires generation
row8:for i in 1 to NY-1 generate
               c1_im_in(i) <= c1_im_out(i-1);
               c1_re_in(i) <= c1_re_out(i-1);
               c2_im_in(i) <= c2_im_out(i-1);
               c2_re_in(i) <= c2_re_out(i-1);
               x_im_in(i) <= x_im_out(i-1);
               x_re_in(i) <= x_re_out(i-1);
               s_im_in(i)(NX-1 downto 0) <= '0'&s_im_out(i-1)(NX-1 downto 1);
```

```
                s_re_in(i)(NX-1 downto 0) <= '0'&s_re_out(i-1)(NX-1 downto 1);

           end generate;
row9:for i in 0 to NY-1 generate
                y_re_in(i)(NX-2 downto 0) <= (others => y_re_in(i)(NX-1));
                y_im_in(i)(NX-2 downto 0) <= (others => y_im_in(i)(NX-1));
           end generate;

           -- input y CMC from dy dff
row10:     for i in 0 to NY-1 generate
                y_im_in(i)(NX-1) <= dy_im(i)(i);
                y_re_in(i)(NX-1) <= dy_re(i)(i);
           end generate;

           -- CFA intermediate wires generation
           --inputs CFA from CMC
           b_re(NY)(NX-1 downto 0) <= '0'&s_re_out(NY-1)(NX-1 downto 1);
           b_im(NY)(NX-1 downto 0) <= '0'&s_im_out(NY-1)(NX-1 downto 1);
           a_re(NY)(NX-1 downto 0) <= c1_re_out(NY-1)(NX-1 downto 0);
           a_im(NY)(NX-1 downto 0) <= c1_im_out(NY-1)(NX-1 downto 0);
           c_re_in(NY)(NX-1 downto 0) <= c2_re_out(NY-1)(NX-1 downto 0);
           c_im_in(NY)(NX-1 downto 0) <= c2_im_out(NY-1)(NX-1 downto 0);
           -- from CFA row to CFA_dff intermediate wires generation NEW!!!
           b_re(NY+1)(NX-1 downto 1) <= s_re(NY)(NX-1 downto 1);
           b_im(NY+1)(NX-1 downto 1) <= s_im(NY)(NX-1 downto 1);
           a_re(NY+1)(NX downto 1) <= c_re_out(NY)(NX-1 downto 0);
           a_im(NY+1)(NX downto 1) <= c_im_out(NY)(NX-1 downto 0);
           -- carry CFA
           c_im_in(NY+1)(1)<=s_im(NY)(0);
           c_re_in(NY+1)(1)<=s_re(NY)(0);
row11:     for i in 2 to NX generate
                c_im_in(NY+i)(i)<=c_im_out(NY+i-1)(i-1);
                c_re_in(NY+i)(i)<=c_re_out(NY+i-1)(i-1);
           end generate;


           -- output m dff intermediate wires generation
col8:for j in NY-1 downto 0 generate
                dm_re(j)(j) <= s_re_out(j)(0);
                dm_im(j)(j) <= s_im_out(j)(0);
           end generate;
col9:for j in NY+NX downto NY generate
                dm_re(j)(j) <= s_re(j)(j-NY);
                dm_im(j)(j) <= s_im(j)(j-NY);
            end generate;

  -- input connections
           c1_im_in(0)(NX-1) <= '1';
           c1_im_in(0)(NX-2 downto 0) <= (others => '0');
           c1_re_in(0) <= (others => '1');
           c2_im_in(0)(NX-1) <= '1';
           c2_im_in(0)(NX-2 downto 0) <= (others => '0');
           c2_re_in(0) <= (others => '0');
           s_im_in(0) <= (others => '0');
           s_re_in(0) <= (others => '0');
row12:     for i in 1 to NY-1 generate
                s_im_in(i)(NX-1) <= '0';
                s_re_in(i)(NX-1) <= '0';
           end generate;
           x_im_in(0) <= x_im;
           x_re_in(0) <= x_re;
           dy_re(0) <= y_re;
           dy_im(0) <= y_im;

  -- output connections
           m_re(NY+NX)<= not dm_re(NY+NX)(NY+NX);
           m_im(NY+NX)<= not dm_im(NY+NX)(NY+NX);
col10:     for j in NY+NX-1 downto NY+1 generate
                m_re(j)<=dm_re(NY+NX)(j);
                m_im(j)<=dm_im(NY+NX)(j);
```

```
          end generate;
          m_re(NY)<= not dm_re(NY+NX)(NY);
          m_im(NY)<= not dm_im(NY+NX)(NY);
col11:    for j in NY-1 downto 0 generate
               m_re(j)<=dm_re(NY+NX)(j);
               m_im(j)<=dm_im(NY+NX)(j);
          end generate;

end Complex_Multiplier_Structural;
```

complex_multiplier_tb.vhd

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

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;


entity complex_multiplier_tb is
     -- Generic declarations of the tested unit
          generic(
          NX : POSITIVE := 5;
          NY : POSITIVE := 4 );
end complex_multiplier_tb;

architecture TB_ARCHITECTURE of complex_multiplier_tb is

file OUTPUT_FILE: TEXT open WRITE_MODE is "outputs.txt";

     -- Component declaration of the tested unit
     component complex_multiplier
     generic    (
          NX : POSITIVE := 4;
          NY : POSITIVE := 5
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (NX-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (NX-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (NY-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (NY-1 downto 0);
          m_im: out STD_LOGIC_VECTOR (NX+NY downto 0);
          m_re: out STD_LOGIC_VECTOR (NX+NY downto 0)
     );
     end component;

     constant DELAY1 : time := 1 ns;

     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clk : std_logic;
     signal clear : std_logic := '0';
     signal x_im : std_logic_vector(NX-1 downto 0);
     signal x_re : std_logic_vector(NX-1 downto 0);
     signal y_im : std_logic_vector(NY-1 downto 0);
     signal y_re : std_logic_vector(NY-1 downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal m_im : std_logic_vector(NY+NX downto 0);
     signal m_re : std_logic_vector(NY+NX downto 0);

     -- Add your code here ...
begin

     -- Unit Under Test port map
```

```
     UUT : complex_multiplier
          generic map (
               NX => NX,
               NY => NY
          )

          port map (
               clk => clk,
               clear => clear,
               x_im => x_im,
               x_re => x_re,
               y_im => y_im,
               y_re => y_re,
               m_im => m_im,
               m_re => m_re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;

     INPUT_Generation: process
     variable nx_re : integer;
     variable nx_im : integer;
     variable ny_re : integer;
     variable ny_im : integer;

     begin
          x_re <= CONV_STD_LOGIC_VECTOR(0,NX);
          x_im <= CONV_STD_LOGIC_VECTOR(0,NX);
          y_re <= CONV_STD_LOGIC_VECTOR(0,NY);
          y_im <= CONV_STD_LOGIC_VECTOR(0,NY);
          clear <= '0';
          wait until (clk'event) and (clk='1');
          clear <= '1';

          for nx_re in -2**(NX-1) to 2**(NX-1)-1 loop
               for nx_im in -2**(NX-1) to 2**(NX-1)-1 loop
                    for ny_re in -2**(NY-1) to 2**(NY-1)-1 loop
                         for ny_im in -2**(NY-1) to 2**(NY-1)-1 loop
                              wait until (clk'event) and (clk='1');
                              x_re <= CONV_STD_LOGIC_VECTOR(nx_re,NX);
                              x_im <= CONV_STD_LOGIC_VECTOR(nx_im,NX);
                              y_re <= CONV_STD_LOGIC_VECTOR(ny_re,NY);
                              y_im <= CONV_STD_LOGIC_VECTOR(ny_im,NY);
                         end loop;
                    end loop;
               end loop;
          end loop;
          wait;

     end process;

     OUTPUT_Checking:process
          type X_Matrix is array (NATURAL range <>) of std_logic_vector(NX-1 downto
0);
          type Y_Matrix is array (NATURAL range <>) of std_logic_vector(NY-1 downto
0);

          variable X_Re_Matrix : X_Matrix (0 to NX+NY+1):=(others=>(others=>'0'));
          variable X_Im_Matrix : X_Matrix (0 to NX+NY+1):=(others=>(others=>'0'));
          variable Y_Re_Matrix : Y_Matrix (0 to NX+NY+1):=(others=>(others=>'0'));
          variable Y_Im_Matrix : Y_Matrix (0 to NX+NY+1):=(others=>(others=>'0'));

          variable report_line : line;
          variable report_str : string(1 to 160);
          variable i,j,re_part, im_part : integer := 0;
```

```
     begin

          wait until (clk'event) and (clk='1') and (clear='1');

          for i in 0 to NX+NY loop
               X_Re_Matrix(i) := X_Re_Matrix(i+1);
               X_Im_Matrix(i) := X_Im_Matrix(i+1);
               Y_Re_Matrix(i) := Y_Re_Matrix(i+1);
               Y_Im_Matrix(i) := Y_Im_Matrix(i+1);
          end loop;
          X_Re_Matrix(NX+NY+1) := x_re;
          X_Im_Matrix(NX+NY+1) := x_im;
          Y_Re_Matrix(NX+NY+1) := y_re;
          Y_Im_Matrix(NX+NY+1) := y_im;

          re_part := conv_integer(X_Re_Matrix(0))*conv_integer(Y_Re_Matrix(0))-
conv_integer(X_Im_Matrix(0))*conv_integer(Y_Im_Matrix(0));
          im_part :=
conv_integer(X_Im_Matrix(0))*conv_integer(Y_Re_Matrix(0))+conv_integer(X_Re_Matrix(0))
*conv_integer(Y_Im_Matrix(0));

          report_line:=null;

          if ((conv_integer(m_re) /= re_part) or (conv_integer(m_im) /= im_part)) then

               write(report_line,string'(">>>>>>>>>>Wrong Result: "));

               write(report_line,conv_integer(X_Re_Matrix(0)));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(X_Im_Matrix(0)));
               write(report_line,string'(" * "));
               write(report_line,conv_integer(Y_Re_Matrix(0)));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(Y_Im_Matrix(0)));
               write(report_line,string'(" = "));
               write(report_line,conv_integer(m_re));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(m_im));

               write(report_line,string'("  Right: "));
               write(report_line,conv_integer(re_part));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(im_part));
               write(report_line,string'("       "));

               for i in report_line'range loop
                    report_str(i) := report_line(i);
               end loop;
               report report_str;

               writeline(OUTPUT_FILE,report_line);

          end if;

          if ( (conv_integer(X_Re_Matrix(0)) = 2**(NX-1)-1) and
(conv_integer(X_Im_Matrix(0)) = 2**(NX-1)-1) and
               (conv_integer(Y_Re_Matrix(0)) = 2**(NY-1)-1) and
(conv_integer(Y_Im_Matrix(0)) = 2**(NY-1)-1) ) then
               write(report_line,string'("------->The End!!! "));
               writeline(OUTPUT_FILE,report_line);
               report "Test Finished!!!" severity FAILURE;
               wait;
          end if;

     end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_complex_multiplier of complex_multiplier_tb is
```

```
     for TB_ARCHITECTURE
          for UUT : complex_multiplier
               use entity work.complex_multiplier(complex_multiplier_structural);
          end for;
     end for;
end TESTBENCH_FOR_complex_multiplier;
```

D. Περιγραφή σε VHDL της αρχιτεκτονικής του βάσης-2 γρήγορου μετασχηματισμού Fourier με αποδεκατισμό στο πεδίο της συχνότητας

D.1 Μιγαδικός αθροιστής

Χρησιμοποιούμε την περιγραφή που παρουσιάζετε στο παράρτημα Α.

D.2 Μιγαδικός αφαιρέτης

Χρησιμοποιούμε την περιγραφή που παρουσιάζετε στο παράρτημα Β.

D.3 Μιγαδικός πολλαπλασιαστής

Όπως είδαμε στην παράγραφο 3.2.3 ο μιγαδικός πολλαπλασιαστής είναι αυτός που περιγράψαμε στο παράρτημα C αλλά με κάποιες τροποποιήσεις στον βασικό του κώδικα ο οποίος παρουσιάζετε ακολούθως.

complex_multiplier.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Multiplier is
     generic    (
          NX : POSITIVE := 4;     -- bits for X input
          NY : POSITIVE := 5    -- bits for Y input
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (NX-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (NX-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (NY-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (NY-1 downto 0);
          m_im: out STD_LOGIC_VECTOR (NX-2 downto 0);
          m_re: out STD_LOGIC_VECTOR (NX-2 downto 0)
     );
end Complex_Multiplier;

--}} End of automatically maintained section

architecture Complex_Multiplier_Structural of Complex_Multiplier is
subtype x_word is STD_LOGIC_VECTOR (NX-1 downto 0);
subtype dy_word is STD_LOGIC_VECTOR (NY-1 downto 0);
subtype CFA_word is STD_LOGIC_VECTOR (NX-1 downto 0);
subtype dm_word is STD_LOGIC_VECTOR (NY+NX-1 downto 0);

type x_word_array is array(natural range <>) of x_word;
type y_word_array is array(natural range <>) of dy_word;
type CFA_word_array is array(natural range <>) of CFA_word;
type dm_word_array is array(natural range <>) of dm_word;

signal     c1_im_in,c1_re_in,c2_im_in,c2_re_in,
```

```
          s_im_in,s_re_in,x_im_in,x_re_in,y_im_in,y_re_in,
          c1_im_out,c1_re_out,c2_im_out,c2_re_out,
          s_im_out,s_re_out,x_im_out,x_re_out : x_word_array(0 to NY-1);

signal  dy_re,dy_im : y_word_array(0 to NY-1);

signal    a_im,a_re,b_im,b_re,c_im_in,
          c_re_in,c_im_out,c_re_out,s_im,s_re : CFA_word_array(NY to NY+NX-1);

signal  dm_re,dm_im : dm_word_array(0 to NY+NX-1);


component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component CMC1_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end component;

component CMC2_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
```

```
        x_re_out : out STD_LOGIC
    );
end component;

component CMC3_cell is
    port(
        c1_im_in : in STD_LOGIC;
        c1_re_in : in STD_LOGIC;
        c2_im_in : in STD_LOGIC;
        c2_re_in : in STD_LOGIC;
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        s_im_in : in STD_LOGIC;
        s_re_in : in STD_LOGIC;
        x_im_in : in STD_LOGIC;
        x_re_in : in STD_LOGIC;
        y_im_in : in STD_LOGIC;
        y_re_in : in STD_LOGIC;
        c1_im_out : out STD_LOGIC;
        c1_re_out : out STD_LOGIC;
        c2_im_out : out STD_LOGIC;
        c2_re_out : out STD_LOGIC;
        s_im_out : out STD_LOGIC;
        s_re_out : out STD_LOGIC;
        x_im_out : out STD_LOGIC;
        x_re_out : out STD_LOGIC
    );
end component;

component CMC4_cell is
    port(
        c1_im_in : in STD_LOGIC;
        c1_re_in : in STD_LOGIC;
        c2_im_in : in STD_LOGIC;
        c2_re_in : in STD_LOGIC;
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        s_im_in : in STD_LOGIC;
        s_re_in : in STD_LOGIC;
        x_im_in : in STD_LOGIC;
        x_re_in : in STD_LOGIC;
        y_im_in : in STD_LOGIC;
        y_re_in : in STD_LOGIC;
        c1_im_out : out STD_LOGIC;
        c1_re_out : out STD_LOGIC;
        c2_im_out : out STD_LOGIC;
        c2_re_out : out STD_LOGIC;
        s_im_out : out STD_LOGIC;
        s_re_out : out STD_LOGIC;
        x_im_out : out STD_LOGIC;
        x_re_out : out STD_LOGIC
    );
end component;

component CFA1_cell is
    port(
        a_im : in STD_LOGIC;
        a_re : in STD_LOGIC;
        b_im : in STD_LOGIC;
        b_re : in STD_LOGIC;
        c_im_in : in STD_LOGIC;
        c_re_in : in STD_LOGIC;
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        c_im_out : out STD_LOGIC;
        c_re_out : out STD_LOGIC;
        s_im : out STD_LOGIC;
        s_re : out STD_LOGIC
    );
end component;
```

```
component CFA2_cell is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

begin
  -- <<enter your statements here>>

  -- cell generation
          -- CMC cell generation
row1:for i in 0 to NY-2 generate
                -- CMC1 cell generation
col1:      for j in NX-2 downto 0 generate
CMC1:           CMC1_cell port map (
                       c1_im_in => c1_im_in(i)(j),
                       c1_re_in => c1_re_in(i)(j),
                       c2_im_in => c2_im_in(i)(j),
                       c2_re_in => c2_re_in(i)(j),
                       clear => clear,
                       clk => clk,
                       s_im_in => s_im_in(i)(j),
                       s_re_in => s_re_in(i)(j),
                       x_im_in => x_im_in(i)(j),
                       x_re_in => x_re_in(i)(j),
                       y_im_in => y_im_in(i)(j),
                       y_re_in => y_re_in(i)(j),
                       c1_im_out => c1_im_out(i)(j),
                       c1_re_out => c1_re_out(i)(j),
                       c2_im_out => c2_im_out(i)(j),
                       c2_re_out => c2_re_out(i)(j),
                       s_im_out => s_im_out(i)(j),
                       s_re_out => s_re_out(i)(j),
                       x_im_out => x_im_out(i)(j),
                       x_re_out => x_re_out(i)(j)
                     );
                end generate;
                -- CMC2_cell generation ( for X MSB )
CMC2:      CMC2_cell port map (
                  c1_im_in => c1_im_in(i)(NX-1),
                  c1_re_in => c1_re_in(i)(NX-1),
                  c2_im_in => c2_im_in(i)(NX-1),
                  c2_re_in => c2_re_in(i)(NX-1),
                  clear => clear,
                  clk => clk,
                  s_im_in => s_im_in(i)(NX-1),
                  s_re_in => s_re_in(i)(NX-1),
                  x_im_in => x_im_in(i)(NX-1),
                  x_re_in => x_re_in(i)(NX-1),
                  y_im_in => y_im_in(i)(NX-1),
                  y_re_in => y_re_in(i)(NX-1),
                  c1_im_out => c1_im_out(i)(NX-1),
                  c1_re_out => c1_re_out(i)(NX-1),
                  c2_im_out => c2_im_out(i)(NX-1),
                  c2_re_out => c2_re_out(i)(NX-1),
                  s_im_out => s_im_out(i)(NX-1),
                  s_re_out => s_re_out(i)(NX-1),
                  x_im_out => x_im_out(i)(NX-1),
```

```
                x_re_out => x_re_out(i)(NX-1)
              );
          end generate;

          -- CMC3_cell generation (for Y MSB)
col2:for j in NX-2 downto 0 generate
CMC3:     CMC3_cell port map (
                c1_im_in => c1_im_in(NY-1)(j),
                c1_re_in => c1_re_in(NY-1)(j),
                c2_im_in => c2_im_in(NY-1)(j),
                c2_re_in => c2_re_in(NY-1)(j),
                clear => clear,
                clk => clk,
                s_im_in => s_im_in(NY-1)(j),
                s_re_in => s_re_in(NY-1)(j),
                x_im_in => x_im_in(NY-1)(j),
                x_re_in => x_re_in(NY-1)(j),
                y_im_in => y_im_in(NY-1)(j),
                y_re_in => y_re_in(NY-1)(j),
                c1_im_out => c1_im_out(NY-1)(j),
                c1_re_out => c1_re_out(NY-1)(j),
                c2_im_out => c2_im_out(NY-1)(j),
                c2_re_out => c2_re_out(NY-1)(j),
                s_im_out => s_im_out(NY-1)(j),
                s_re_out => s_re_out(NY-1)(j),
                x_im_out => x_im_out(NY-1)(j),
                x_re_out => x_re_out(NY-1)(j)
              );
          end generate;

          -- CMC4_cell generation ( for X and Y MSB )
CMC4:CMC4_cell port map (
                c1_im_in => c1_im_in(NY-1)(NX-1),
                c1_re_in => c1_re_in(NY-1)(NX-1),
                c2_im_in => c2_im_in(NY-1)(NX-1),
                c2_re_in => c2_re_in(NY-1)(NX-1),
                clear => clear,
                clk => clk,
                s_im_in => s_im_in(NY-1)(NX-1),
                s_re_in => s_re_in(NY-1)(NX-1),
                x_im_in => x_im_in(NY-1)(NX-1),
                x_re_in => x_re_in(NY-1)(NX-1),
                y_im_in => y_im_in(NY-1)(NX-1),
                y_re_in => y_re_in(NY-1)(NX-1),
                c1_im_out => c1_im_out(NY-1)(NX-1),
                c1_re_out => c1_re_out(NY-1)(NX-1),
                c2_im_out => c2_im_out(NY-1)(NX-1),
                c2_re_out => c2_re_out(NY-1)(NX-1),
                s_im_out => s_im_out(NY-1)(NX-1),
                s_re_out => s_re_out(NY-1)(NX-1),
                x_im_out => x_im_out(NY-1)(NX-1),
                x_re_out => x_re_out(NY-1)(NX-1)
          );

          -- CFA_cell generation
          -- CFA1 type cell generation
col3:for j in NX-1 downto 0 generate
CFA1:     CFA1_cell port map (
                    a_im => a_im(NY)(j),
                    a_re => a_re(NY)(j),
                    b_im => b_im(NY)(j),
                    b_re => b_re(NY)(j),
                    c_im_in => c_im_in(NY)(j),
                    c_re_in => c_re_in(NY)(j),
                    clear => clear,
                    clk => clk,
                    c_im_out => c_im_out(NY)(j),
                    c_re_out => c_re_out(NY)(j),
                    s_im => s_im(NY)(j),
                    s_re => s_re(NY)(j)
```

```
               );
          end generate;

          -- CFA2 type cell generation
row4:for i in 1 to NX-1 generate
CFA2:     CFA2_cell port map (
                    a_im => a_im(NY+i)(i),
                    a_re => a_re(NY+i)(i),
                    b_im => b_im(NY+i)(i),
                    b_re => b_re(NY+i)(i),
                    c_im_in => c_im_in(NY+i)(i),
                    c_re_in => c_re_in(NY+i)(i),
                    clear => clear,
                    clk => clk,
                    c_im_out => c_im_out(NY+i)(i),
                    c_re_out => c_re_out(NY+i)(i),
                    s_im => s_im(NY+i)(i),
                    s_re => s_re(NY+i)(i)
               );
          end generate;

          -- CFA intermediate dff generation and connection
row5:for i in 1 to NX-2 generate
col5:     for j in NX-1 downto i+1 generate
                    -- dff generation for CFA a inputs
gdCFAare:       dff generic map (
                         CLEARQ => '0'
                    ) port map (
                         clear => clear,
                         clk => clk,
                         d => a_re(NY+i)(j),
                         q => a_re(NY+i+1)(j)
                    );
gdCFAaim:       dff generic map (
                         CLEARQ => '0'
                    ) port map (
                         clear => clear,
                         clk => clk,
                         d => a_im(NY+i)(j),
                         q => a_im(NY+i+1)(j)
                    );
                    -- dff generation for CFA b inputs
cond1:                   if j<NX generate
gdCFAbre:           dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => b_re(NY+i)(j),
                              q => b_re(NY+i+1)(j)
                         );
gdCFAbim:           dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => b_im(NY+i)(j),
                              q => b_im(NY+i+1)(j)
                         );
                    end generate;
               end generate;
          end generate;

          -- input X dff generation
row6:for i in 0 to NY-2 generate
col6:     for j in NY-1 downto i+1 generate
gdyre:              dff generic map (
                         CLEARQ => '0'
                    ) port map (
                         clear => clear,
```

```
                         clk => clk,
                         d => dy_re(i)(j),
                         q => dy_re(i+1)(j)
                    );
gdyim:              dff generic map (
                         CLEARQ => '0'
                    ) port map (
                         clear => clear,
                         clk => clk,
                         d => dy_im(i)(j),
                         q => dy_im(i+1)(j)
                    );
               end generate;
          end generate;

          -- output m dff generation
row7:for i in 1 to NY+NX-1 generate
col7:     for j in i-1 downto 0 generate
cond2:              if j /= NY generate
gdmre0:                  dff generic map (
                              CLEARQ => '0'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_re(i-1)(j),
                              q => dm_re(i)(j)
                         );
gdmim0:                  dff generic map (
                              CLEARQ => '0'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_im(i-1)(j),
                              q => dm_im(i)(j)
                         );
                    end generate;
cond3:              if j = NY generate
gdmre1:                  dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_re(i-1)(j),
                              q => dm_re(i)(j)
                         );
gdmim1:                  dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_im(i-1)(j),
                              q => dm_im(i)(j)
                         );
                    end generate;
               end generate;
          end generate;

  -- intermediate wires generation
          -- CMC intermediate wires generation
row8:for i in 1 to NY-1 generate
               c1_im_in(i) <= c1_im_out(i-1);
               c1_re_in(i) <= c1_re_out(i-1);
               c2_im_in(i) <= c2_im_out(i-1);
               c2_re_in(i) <= c2_re_out(i-1);
               x_im_in(i) <= x_im_out(i-1);
               x_re_in(i) <= x_re_out(i-1);
               s_im_in(i)(NX-1 downto 0) <= '0'&s_im_out(i-1)(NX-1 downto 1);
               s_re_in(i)(NX-1 downto 0) <= '0'&s_re_out(i-1)(NX-1 downto 1);

          end generate;
```

```
row9:for i in 0 to NY-1 generate
                y_re_in(i)(NX-2 downto 0) <= (others => y_re_in(i)(NX-1));
                y_im_in(i)(NX-2 downto 0) <= (others => y_im_in(i)(NX-1));
           end generate;

           -- input y CMC from dy dff
row10:     for i in 0 to NY-1 generate
                y_im_in(i)(NX-1) <= dy_im(i)(i);
                y_re_in(i)(NX-1) <= dy_re(i)(i);
           end generate;

           -- CFA intermediate wires generation
           --inputs CFA from CMC
           b_re(NY)(NX-1 downto 0) <= '0'&s_re_out(NY-1)(NX-1 downto 1);
           b_im(NY)(NX-1 downto 0) <= '0'&s_im_out(NY-1)(NX-1 downto 1);
           a_re(NY)(NX-1 downto 0) <= c1_re_out(NY-1)(NX-1 downto 0);
           a_im(NY)(NX-1 downto 0) <= c1_im_out(NY-1)(NX-1 downto 0);
           c_re_in(NY)(NX-1 downto 0) <= c2_re_out(NY-1)(NX-1 downto 0);
           c_im_in(NY)(NX-1 downto 0) <= c2_im_out(NY-1)(NX-1 downto 0);
           -- from CFA row to CFA_dff intermediate wires generation
           b_re(NY+1)(NX-1 downto 1) <= s_re(NY)(NX-1 downto 1);
           b_im(NY+1)(NX-1 downto 1) <= s_im(NY)(NX-1 downto 1);
           a_re(NY+1)(NX-1 downto 1) <= c_re_out(NY)(NX-2 downto 0);
           a_im(NY+1)(NX-1 downto 1) <= c_im_out(NY)(NX-2 downto 0);
           -- carry CFA
           c_im_in(NY+1)(1)<=s_im(NY)(0);
           c_re_in(NY+1)(1)<=s_re(NY)(0);
row11:     for i in 2 to NX-1 generate
                c_im_in(NY+i)(i)<=c_im_out(NY+i-1)(i-1);
                c_re_in(NY+i)(i)<=c_re_out(NY+i-1)(i-1);
           end generate;

           -- output m dff intermediate wires generation
col8:for j in NY-1 downto 0 generate
                dm_re(j)(j) <= s_re_out(j)(0);
                dm_im(j)(j) <= s_im_out(j)(0);
           end generate;
col9:for j in NY+NX-1 downto NY generate
                dm_re(j)(j) <= s_re(j)(j-NY);
                dm_im(j)(j) <= s_im(j)(j-NY);
            end generate;

  -- input connections
           c1_im_in(0)(NX-1) <= '1';
           c1_im_in(0)(NX-2 downto 0) <= (others => '0');
           c1_re_in(0) <= (others => '1');
           c2_im_in(0)(NX-1) <= '1';
           c2_im_in(0)(NX-2 downto 0) <= (others => '0');
           c2_re_in(0) <= (others => '0');
           s_im_in(0) <= (others => '0');
           s_re_in(0) <= (others => '0');
row12:     for i in 1 to NY-1 generate
                s_im_in(i)(NX-1) <= '0';
                s_re_in(i)(NX-1) <= '0';
           end generate;
           x_im_in(0) <= x_im;
           x_re_in(0) <= x_re;
           dy_re(0) <= y_re;
           dy_im(0) <= y_im;

  -- output connections
col10:     for j in NY+NX-2 downto NY+1 generate
                m_re(j-NY)<= dm_re(NY+NX-2)(j);
                m_im(j-NY)<= dm_im(NY+NX-2)(j);
           end generate;
           m_re(0)<=not dm_re(NY+NX-2)(NY);
           m_im(0)<=not dm_im(NY+NX-2)(NY);

end Complex_Multiplier_Structural;
```

complex_multiplier_tb.vhd

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

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;


entity complex_multiplier_tb is
     -- Generic declarations of the tested unit
          generic(
          NX : POSITIVE := 5;
          NY : POSITIVE := 4 );
end complex_multiplier_tb;

architecture TB_ARCHITECTURE of complex_multiplier_tb is

file OUTPUT_FILE: TEXT open WRITE_MODE is "outputs.txt";

     -- Component declaration of the tested unit
     component complex_multiplier
     generic    (
          NX : POSITIVE := 4;
          NY : POSITIVE := 5
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (NX-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (NX-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (NY-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (NY-1 downto 0);
          m_im: out STD_LOGIC_VECTOR (NX-2 downto 0);
          m_re: out STD_LOGIC_VECTOR (NX-2 downto 0)
     );
     end component;

     constant DELAY1 : time := 1 ns;

     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clk : std_logic;
     signal clear : std_logic := '0';
     signal x_im : std_logic_vector(NX-1 downto 0);
     signal x_re : std_logic_vector(NX-1 downto 0);
     signal y_im : std_logic_vector(NY-1 downto 0);
     signal y_re : std_logic_vector(NY-1 downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal m_im : std_logic_vector(NX-2 downto 0);
     signal m_re : std_logic_vector(NX-2 downto 0);

     -- Add your code here ...
begin

     -- Unit Under Test port map
     UUT : complex_multiplier
          generic map (
               NX => NX,
               NY => NY
          )

          port map (
               clk => clk,
               clear => clear,
```

```
                x_im => x_im,
                x_re => x_re,
                y_im => y_im,
                y_re => y_re,
                m_im => m_im,
                m_re => m_re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;

     INPUT_Generation: process
     variable nx_re : integer;
     variable nx_im : integer;
     variable ny_re : integer;
     variable ny_im : integer;

     begin
          x_re <= CONV_STD_LOGIC_VECTOR(0,NX);
          x_im <= CONV_STD_LOGIC_VECTOR(0,NX);
          y_re <= CONV_STD_LOGIC_VECTOR(0,NY);
          y_im <= CONV_STD_LOGIC_VECTOR(0,NY);
          clear <= '0';
          wait until (clk'event) and (clk='1');
          clear <= '1';

          for nx_re in -2**(NX-1) to 2**(NX-1)-1 loop
               for nx_im in -2**(NX-1) to 2**(NX-1)-1 loop
                    for ny_re in -2**(NY-2) to 2**(NY-2) loop
                         for ny_im in -2**(NY-2) to 2**(NY-2) loop
                              wait until (clk'event) and (clk='1');
                              x_re <= CONV_STD_LOGIC_VECTOR(nx_re,NX);
                              x_im <= CONV_STD_LOGIC_VECTOR(nx_im,NX);
                              y_re <= CONV_STD_LOGIC_VECTOR(ny_re,NY);
                              y_im <= CONV_STD_LOGIC_VECTOR(ny_im,NY);
                         end loop;
                    end loop;
               end loop;
          end loop;
          wait;

     end process;

     OUTPUT_Checking:process
          type X_Matrix is array (NATURAL range <>) of std_logic_vector(NX-1 downto
0);
          type Y_Matrix is array (NATURAL range <>) of std_logic_vector(NY-1 downto
0);

          constant MULT_DELAY : integer := NX+NY-1;

          variable X_Re_Matrix : X_Matrix (0 to MULT_DELAY):=(others=>(others=>'0'));
          variable X_Im_Matrix : X_Matrix (0 to MULT_DELAY):=(others=>(others=>'0'));
          variable Y_Re_Matrix : Y_Matrix (0 to MULT_DELAY):=(others=>(others=>'0'));
          variable Y_Im_Matrix : Y_Matrix (0 to MULT_DELAY):=(others=>(others=>'0'));

          variable report_line : line;
          variable report_str : string(1 to 160);
          variable i,j,re_part, im_part : integer := 0;
          variable temp : std_logic_vector(NY+NX downto 0);

     begin

          wait until (clk'event) and (clk='1') and (clear='1');

          for i in 0 to MULT_DELAY-1 loop
```

```
                X_Re_Matrix(i) := X_Re_Matrix(i+1);
                X_Im_Matrix(i) := X_Im_Matrix(i+1);
                Y_Re_Matrix(i) := Y_Re_Matrix(i+1);
                Y_Im_Matrix(i) := Y_Im_Matrix(i+1);
           end loop;
           X_Re_Matrix(MULT_DELAY) := x_re;
           X_Im_Matrix(MULT_DELAY) := x_im;
           Y_Re_Matrix(MULT_DELAY) := y_re;
           Y_Im_Matrix(MULT_DELAY) := y_im;

           re_part := conv_integer(X_Re_Matrix(0))*conv_integer(Y_Re_Matrix(0))-
conv_integer(X_Im_Matrix(0))*conv_integer(Y_Im_Matrix(0));
           im_part :=
conv_integer(X_Im_Matrix(0))*conv_integer(Y_Re_Matrix(0))+conv_integer(X_Re_Matrix(0))
*conv_integer(Y_Im_Matrix(0));

           temp := conv_std_logic_vector(re_part,NX+NY+1);
           re_part := (conv_integer(temp(NY+NX-2 downto NY)));

           temp := conv_std_logic_vector(im_part,NX+NY+1);
           im_part := (conv_integer(temp(NY+NX-2 downto NY)));

           report_line:=null;

           if ((conv_integer(m_re) /= re_part) or (conv_integer(m_im) /= im_part)) then

                write(report_line,string'(">>>>>>>>>>Wrong Result: "));

                write(report_line,conv_integer(X_Re_Matrix(0)));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(X_Im_Matrix(0)));
                write(report_line,string'(" * "));
                write(report_line,conv_integer(Y_Re_Matrix(0)));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(Y_Im_Matrix(0)));
                write(report_line,string'(" = "));
                write(report_line,conv_integer(m_re));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(m_im));

                write(report_line,string'("  Right: "));
                write(report_line,conv_integer(re_part));
                write(report_line,string'("+i"));
                write(report_line,conv_integer(im_part));
                write(report_line,string'("       "));

                for i in report_line'range loop
                     report_str(i) := report_line(i);
                end loop;
                report report_str;

                writeline(OUTPUT_FILE,report_line);

           end if;

           if ( (conv_integer(X_Re_Matrix(0)) = 2**(NX-1)-1) and
(conv_integer(X_Im_Matrix(0)) = 2**(NX-1)-1) and
                (conv_integer(Y_Re_Matrix(0)) = 2**(NY-2)) and
(conv_integer(Y_Im_Matrix(0)) = 2**(NY-2)) ) then
                write(report_line,string'("------->The End!!! "));
                writeline(OUTPUT_FILE,report_line);
                report "Test Finished!!!" severity FAILURE;
                wait;
           end if;

     end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_complex_multiplier of complex_multiplier_tb is
```

```
     for TB_ARCHITECTURE
          for UUT : complex_multiplier
               use entity work.complex_multiplier(complex_multiplier_structural);
          end for;
     end for;
end TESTBENCH_FOR_complex_multiplier;
```

D.4 Μιγαδικός καταχωρητής ολίσθησης

Η μονάδα αυτή παρουσιάστηκε στη παράγραφο 3.5. Για την περιγραφή σε VHDL χρησιμοποιούμε τα αρχεία

dff.vhd	Περιγραφή του D flip-flop
Complex_shift_register.vhd	Περιγραφή του μιγαδικού καταχωρητή ολίσθησης

Complex_shift_register.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Shift_Register is
     generic   (
          N : Integer := 8;
          W : Integer := 2
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          a_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          a_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          oa_im: out STD_LOGIC_VECTOR (N-1 downto 0);
          oa_re: out STD_LOGIC_VECTOR (N-1 downto 0)
     );
end Complex_Shift_Register;

architecture Complex_Shift_Register_Structural of Complex_Shift_Register is
subtype word is STD_LOGIC_VECTOR (N-1 downto 0);
type word_array is array(natural range <>) of word;
signal wd_re_array, wd_im_array :  word_array(0 to W);


component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

begin
     -- dff generation

col: for j in 0 to W-1 generate
row:       for i in 0 to N-1 generate
SR_Dff_re:           dff  generic map (
                          CLEARQ=>'0'
                     )
                     port map (
                          clear => clear,
```

```
                        clk => clk,
                        d => wd_re_array(j)(i),
                        q => wd_re_array(j+1)(i)
                    );
SR_Dff_im:          dff  generic map (
                        CLEARQ=>'0'
                    )
                    port map (
                        clear => clear,
                        clk => clk,
                        d => wd_im_array(j)(i),
                        q => wd_im_array(j+1)(i)
                    );
                end generate;
            end generate;

    -- input wires generation
    wd_re_array(0) <= a_re;
    wd_im_array(0) <= a_im;


    -- output wires generation
    oa_re <= wd_re_array(W);
    oa_im <= wd_im_array(W);

end Complex_Shift_Register_Structural;
```

Μπορούμε, όπως είπαμε στην παράγραφο 3.5, να υλοποιήσουμε τα στοιχεία καθυστέρησης με την χρήση μνήμης RAM 2 θυρών. Μια περιγραφή που μπορούμε να χρησιμοποιήσουμε παρουσιάζετε παρακάτω και αποτελείτε απ' τα αρχεία

CSM_Tff.vhd	Περιγραφή των T Flip-Flop που χρησιμοποιούντε στους counters του μιγαδικού καταχωρητή ολίσθησης με χρήση μνήμης RAM 2 θυρών
CSM_Counter.vhd	Περιγραφή των μετρητών της μονάδας
CSM_RAM.vhd	Περιγραφή της μνήμης RAM 2 θυρών
Complex_Shift_Mem.vhd	Ανώτατο επίπεδο περιγραφής του μιγαδικού καταχωρητή ολίσθησης, με χρήση μνήμης RAM 2 θυρών, που παράγετε απ' το block diagram αρχείο Complex_Shift_Mem.bde

CSM_Tff.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity CSM_Tff is
      generic (
          CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
           T : in STD_LOGIC;
           clear : in STD_LOGIC;
           clk : in STD_LOGIC;
           Q : out STD_LOGIC
          );
end CSM_Tff;


architecture CSM_Tff_behav of CSM_Tff is
signal state : std_logic := CLEAR_BIT;
begin
       process(clk,clear,T)
       variable a,b : std_logic;
```

```
       begin
            if clear ='0' then
                 state <= CLEAR_BIT;
            elsif clk'event and clk ='1' then
                 if T='1' then
                      state <= not state;
                 end if;
            end if ;
      end process;
       Q <= state;
end CSM_Tff_behav;
```

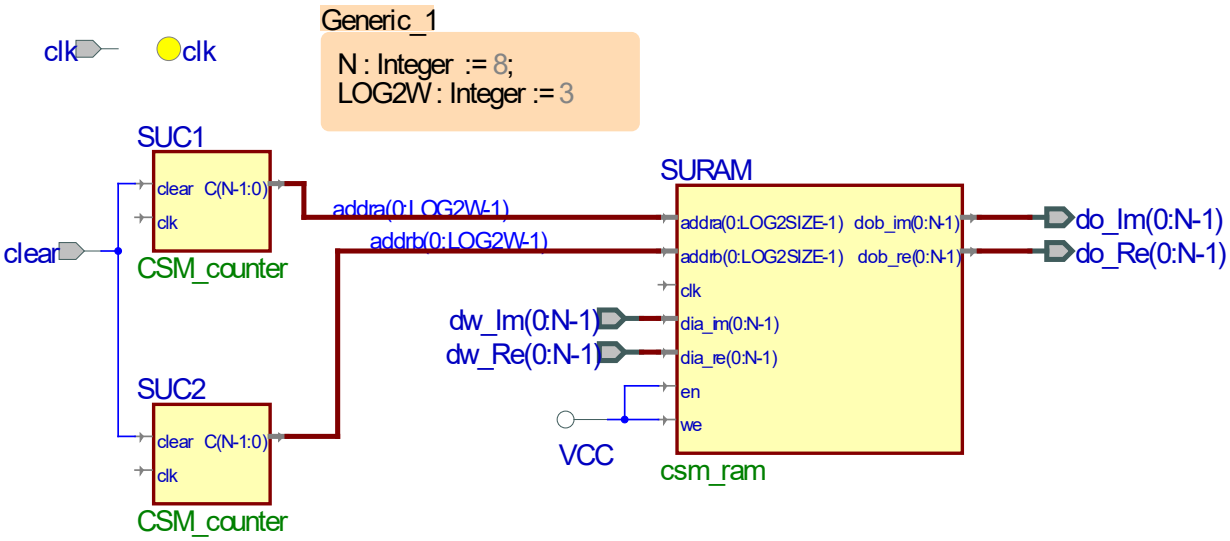
CSM_Counter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;

entity CSM_counter is
      generic (
          N : Integer := 5;
          START : Integer := 0
      );
      port(
           clk : in STD_LOGIC;
           clear : in STD_LOGIC;
           C : out STD_LOGIC_VECTOR(N-1 downto 0)
          );
end CSM_counter;

architecture CSM_counter_structural of CSM_counter is

component CSM_Tff is
      generic (
          CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
           T : in STD_LOGIC;
           clear : in STD_LOGIC;
           clk : in STD_LOGIC;
           Q : out STD_LOGIC
          );
end component;

signal TT,TQ : std_logic_vector(N-1 downto 0);
constant START_VECTOR : STD_LOGIC_VECTOR(N-1 downto 0) :=
CONV_STD_LOGIC_VECTOR(START,N);

begin

          TT(0) <= '1';
TffR:CSM_Tff  generic map (
                CLEAR_BIT => START_VECTOR(0) )
          port map (
                T=>TT(0),
                clear=>clear,
                clk=>clk,
                Q=>TQ(0)
          );
G1:       for i in 1 to N-1 generate
                     TT(i) <= TQ(i-1) AND TT(i-1);
Tffi:           CSM_Tff generic map (
                           CLEAR_BIT => START_VECTOR(i) )
                     port map (
                           T =>TT(i),
                           clear=>clear,
                           clk=>clk,
                           Q=>TQ(i)
```

```
                    );
          end generate;
          C<=TQ;

end CSM_counter_structural;
```

CSM_RAM.vhd

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

entity CSM_Ram is
     generic (
          N : Integer := 8;
          LOG2SIZE : INTEGER := 5
     );
     port (
          clk : in std_logic;
          en : in std_logic;
          we : in std_logic;
          addra : in std_logic_vector(0 to LOG2SIZE-1);
          addrb : in std_logic_vector(0 to LOG2SIZE-1);
          dia_im : in std_logic_vector(0 to N-1);
          dia_re : in std_logic_vector(0 to N-1);
          dob_im : out std_logic_vector(0 to N-1);
          dob_re : out std_logic_vector(0 to N-1)
     );
end CSM_Ram;

architecture behav of CSM_Ram is
type ram_type is array (0 to 2**LOG2SIZE-1) of std_logic_vector (0 to 2*N-1);
signal RAM : ram_type;
signal read_addrb : std_logic_vector(0 to LOG2SIZE-1);

begin
     process (clk)
     begin
          if (clk'event and clk = '1') then
               if (en = '1') then
                    if (we = '1') then
                         RAM(conv_integer(addra)) <= dia_re&dia_im;
                    end if;
                    read_addrb <= addrb;
               end if;
          end if;
     end process;
     dob_re <= RAM(conv_integer(read_addrb))(0 to N-1);
     dob_im <= RAM(conv_integer(read_addrb))(N to 2*N-1);
end behav;
```



Σχήμα D.1 Σχεδιάγραμμα μονάδας μνήμης RAM ως μονάδα καθυστέρησης που βρίσκετε στο αρχείο Complex_Shift_Mem.bde

Complex_Shift_Mem.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity Complex_Shift_Mem is
  generic(
        N : Integer  := 8;
LOG2W : Integer := 3
  );
  port(
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        dw_Im : in STD_LOGIC_VECTOR(0 to N-1);
        dw_Re : in STD_LOGIC_VECTOR(0 to N-1);
        do_Im : out STD_LOGIC_VECTOR(0 to N-1);
        do_Re : out STD_LOGIC_VECTOR(0 to N-1)
  );
end Complex_Shift_Mem;

architecture Complex_Shift_Mem of Complex_Shift_Mem is

---- Component declarations -----

component CSM_counter
  generic(
        N : INTEGER := 5;
        START : INTEGER := 0
  );
  port (
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        C : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component csm_ram
  generic(
        LOG2SIZE : INTEGER := 5;
        N : INTEGER := 8
  );
  port (
        addra : in STD_LOGIC_VECTOR(0 to LOG2SIZE-1);
        addrb : in STD_LOGIC_VECTOR(0 to LOG2SIZE-1);
```

```
       clk : in STD_LOGIC;
       dia_im : in STD_LOGIC_VECTOR(0 to N-1);
       dia_re : in STD_LOGIC_VECTOR(0 to N-1);
       en : in STD_LOGIC;
       we : in STD_LOGIC;
       dob_im : out STD_LOGIC_VECTOR(0 to N-1);
       dob_re : out STD_LOGIC_VECTOR(0 to N-1)
  );
end component;

----     Constants     -----
constant VCC_CONSTANT   : STD_LOGIC := '1';

---- Signal declarations used on the diagram ----

signal VCC : STD_LOGIC;
signal addra : STD_LOGIC_VECTOR (0 to LOG2W-1);
signal addrb : STD_LOGIC_VECTOR (0 to LOG2W-1);

begin

----  Component instantiations  ----

SUC1 : CSM_counter
  generic map (
       N => LOG2W,
       START => 0
  )
  port map(
       C => addra( 0 to LOG2W-1 ),
       clear => clear,
       clk => clk
  );

SUC2 : CSM_counter
  generic map (
       N => LOG2W,
       START => 1
  )
  port map(
       C => addrb( 0 to LOG2W-1 ),
       clear => clear,
       clk => clk
  );

SURAM : csm_ram
  generic map (
       LOG2SIZE => LOG2W,
       N => N
  )
  port map(
       addra => addra( 0 to LOG2W-1 ),
       addrb => addrb( 0 to LOG2W-1 ),
       clk => clk,
       dia_im => dw_Im( 0 to N-1 ),
       dia_re => dw_Re( 0 to N-1 ),
       dob_im => do_Im( 0 to N-1 ),
       dob_re => do_Re( 0 to N-1 ),
       en => VCC,
       we => VCC
  );


---- Power , ground assignment ----

VCC <= VCC_CONSTANT;

end Complex_Shift_Mem;
```

D.5 Μονάδα Butterfly

Η μονάδα butterfly χρησιμοποιεί για την περιγραφή της εκτός από τα παραπάνω και τα αρχεία:

tff.vhd	Περιγραφή T Flip-Flop
BU_addr_counter.vhd	Περιγραφή του μετρητή που χρησιμοποιείτε για την εισαγωγή της σωστής διεύθυνσης στην ROM με τους παράγοντες αναστροφής (twiddle factors) W_N
BU_Rom.vhd	Η μνήμη ROM με αποθηκευμένους τους παράγοντες αναστροφής (twiddle factors)W_N
WN_Unit.vhd	Περιγραφή της μονάδας που χρησιμοποιείτε για την εξαγωγή των παραγόντων αναστροφής (twiddle factors) W_N και την εισαγωγή τους στον μιγαδικό πολλαπλασιαστή
Batterfly_Unit.vhd	Περιγραφή της μονάδας Butterfly χρησιμοποιώντας το block diagram αρχείο Batterfly_Unit.bde
Last_Batterfly_Unit.vhd	Περιγραφή της μονάδας Butterfly του τελευταίου σταδίου, όπου ο πολλαπλασιασμός γίνετε μόνο με την μονάδα και ο πολλαπλασιαστής έχει παραλειφθεί χρησιμοποιώντας το block diagram αρχείο Last_Batterfly_Unit.bde
butterfly_unit_tb.vhd	Test bench το οποίο κάνει ελέγχει την σωστή λειτουργία της μονάδας buttefly

tff.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity tff is
      generic (
          CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
           T : in STD_LOGIC;
           clear : in STD_LOGIC;
           clk : in STD_LOGIC;
           Q : out STD_LOGIC
          );
end tff;

architecture tff_behav of tff is
signal state : std_logic;
begin
       process(clk,clear)
       variable a,b : std_logic;
       begin
            if clear ='0' then
                 state <= CLEAR_BIT;
            elsif clk'event and clk ='1' then
                 if T='1' then
                      state <= not state;
                 end if;
            end if ;
      end process;
       Q <= state;
end tff_behav;
```

BU_addr_counter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;

entity BU_addr_counter is
      generic (
          N : Integer := 5;       -- N bits counter
          START : Integer := 7   -- starting number for the counter
      );
      port(
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           addr : out INTEGER range 0 to 2**N-1
          );
end BU_addr_counter;

architecture BU_addr_counter of BU_addr_counter is

component tff is
      generic (
          CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
           T : in STD_LOGIC;
           clear : in STD_LOGIC;
           clk : in STD_LOGIC;
           Q : out STD_LOGIC
          );
end component;

signal TT,TQ : std_logic_vector(N-1 downto 0);
constant START_VECTOR : STD_LOGIC_VECTOR(N-1 downto 0) :=
CONV_STD_LOGIC_VECTOR(START,N);

begin
          TT(0) <= '1';
TffR:tff  generic map (
               CLEAR_BIT => START_VECTOR(0) )
          port map (
               TT(0),
               reset,
               clk,
               TQ(0)
          );
G1:       for i in N-1 downto 1 generate
                    TT(i) <= TQ(i-1) AND TT(i-1);
Tffi:               tff generic map (
                              CLEAR_BIT => START_VECTOR(i) )
                         port map (
                              T =>TT(i),
                              clear=>reset,
                              clk=>clk,
                              Q=>TQ(i)
                         );

          end generate;
          addr<=CONV_INTEGER(UNSIGNED(TQ));

end BU_addr_counter;
```

BU_Rom.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.math_real.all;
```

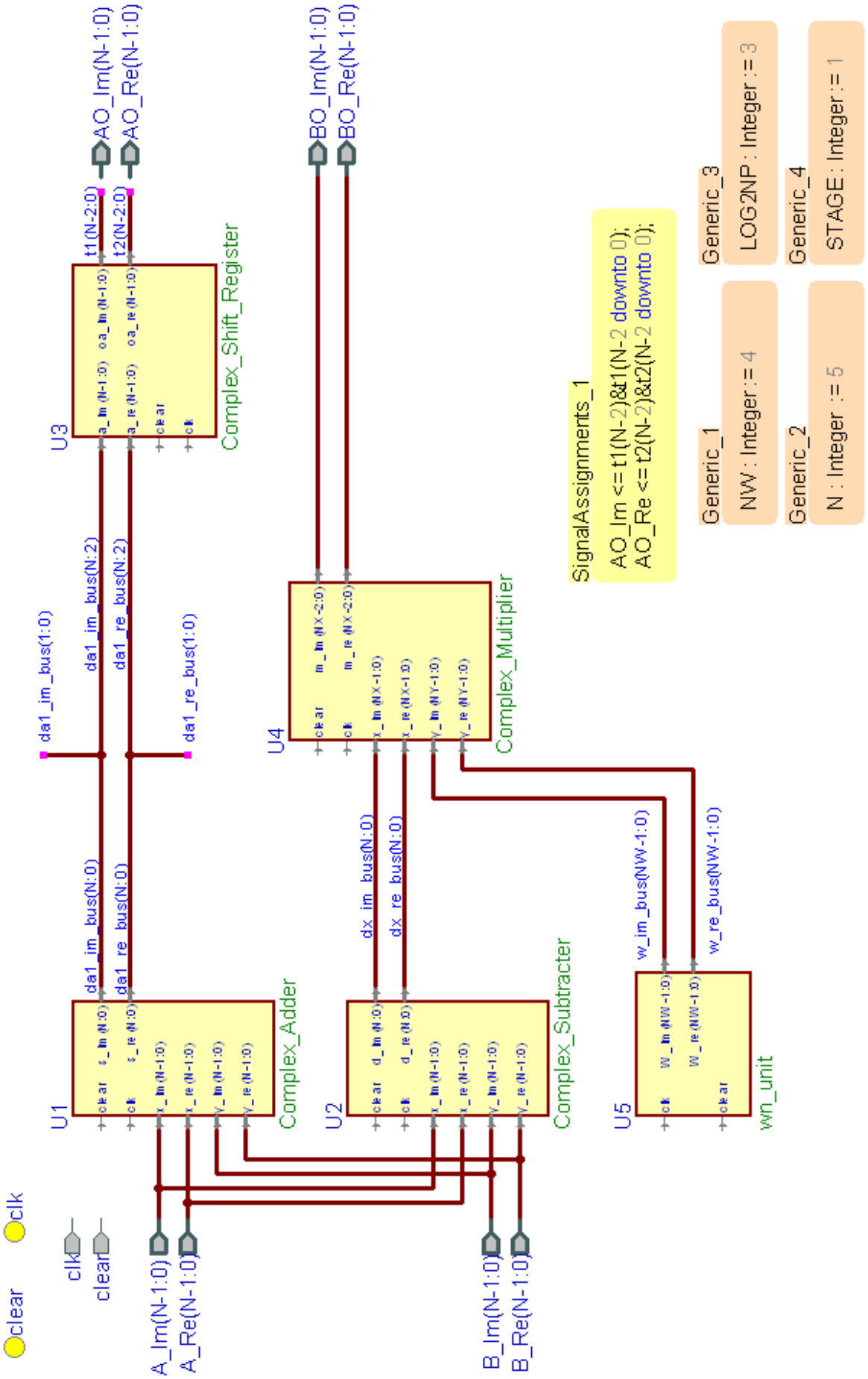
```
entity BU_Rom is
      generic (
          NW : Integer := 8;         -- NW bits for memory real and imaginary part
data
           LOG2NP : Integer := 3; -- n=log2(N) N-Point DFT
           STAGE : Integer := 2      -- the stage of butterfly
      );
      port(
           clk : in STD_LOGIC;
           Address : in  integer range 0 to (2**(LOG2NP-STAGE)-1);
           Data_Re : out STD_LOGIC_VECTOR(NW-1 downto 0);
           Data_Im : out STD_LOGIC_VECTOR(NW-1 downto 0)
          );

end BU_Rom;

architecture BU_Rom of BU_Rom is
TYPE rom_array is ARRAY(natural range <>) of STD_LOGIC_VECTOR (NW-1 downto 0);
constant memory_size : Integer := 2**(LOG2NP-STAGE);

function cos2c( X : integer ) return STD_LOGIC_VECTOR is
variable tcos : integer;
variable t : SIGNED(NW-1 downto 0);
begin
     tcos := integer( (2.0**(NW-2)-0.0)/1.0*COS( -real(x)*2.0*MATH_PI/real(2**LOG2NP)
) );
     t := CONV_SIGNED(tcos,NW);
     return CONV_STD_LOGIC_VECTOR(t,NW);
end function cos2c;

function calc_rom_re return rom_array is
variable mem : rom_array(0 to memory_size-1);

begin
L1:  for k in 0 to (memory_size-1) loop
          mem(k) := cos2c((2**(STAGE-1))*(k mod 2**(LOG2NP-STAGE)));
     end loop L1;
     return mem;

end function calc_rom_re;

function sin2c( X : integer ) return STD_LOGIC_VECTOR is
variable tsin : integer;
variable t : SIGNED(NW-1 downto 0);
begin
     tsin := integer( (2.0**(NW-2)-0.0)/1.0*SIN( -real(x)*2.0*MATH_PI/real(2**LOG2NP)
) );
     t := CONV_SIGNED(tsin,NW);
     return CONV_STD_LOGIC_VECTOR(t,NW);
end function sin2c;

function calc_rom_im return rom_array is
variable mem : rom_array(0 to memory_size-1);

begin
L1:  for k in 0 to memory_size-1 loop
          mem(k) := sin2c((2**(STAGE-1))*(k mod 2**(LOG2NP-STAGE)));
     end loop L1;
     return mem;

end function calc_rom_im;

constant memory_re : rom_array(0 to memory_size-1) := calc_rom_re;
constant memory_im : rom_array(0 to memory_size-1) := calc_rom_im;

begin
       process(clk) is
       begin
          if clk'event and clk='1' then
```

```
                Data_Re<=memory_re(Address);
                Data_Im<=memory_im(Address);
            end if;
      end process;
end BU_Rom;
```

WN_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity WN_Unit is
  generic(
        N : Integer := 8;
       NW : Integer := 8;
       LOG2NP : Integer := 3;  -- LOG2NP=log2(NP) NP=N-Point DFT;
       STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       W_im : out STD_LOGIC_VECTOR(NW-1 downto 0);
       W_re : out STD_LOGIC_VECTOR(NW-1 downto 0)
  );
end WN_Unit;

architecture WN_Unit of WN_Unit is

---- Component declarations -----

component BU_Tff is
      generic (
           CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
            T : in STD_LOGIC;
            clear : in STD_LOGIC;
            clk : in STD_LOGIC;
            Q : out STD_LOGIC
           );
end component;
component BU_addr_counter
  generic (
           N : Integer := 5;
           START : Integer := 7
  );
  port (
       clk : in STD_LOGIC;
       reset : in STD_LOGIC;
       addr : out INTEGER range 0 to 2**N-1
  );
end component;
component BU_Rom
  generic(
       NW : INTEGER := 8;
       LOG2NP : INTEGER := 3;
       STAGE : INTEGER := 1
  );
  port (
       Address : in INTEGER range 0 to 2**(LOG2NP-STAGE)-1;
       clk : in STD_LOGIC;
       Data_Im : out STD_LOGIC_VECTOR(0 to NW-1);
       Data_Re : out STD_LOGIC_VECTOR(0 to NW-1)
  );
end component;
```

```
---- Signal declarations used on the diagram ----

signal a : INTEGER :=0;

begin

----  Component instantiations  ----

U1 : BU_Rom
  generic map (
       NW => NW,
       LOG2NP => LOG2NP,
       STAGE => STAGE
  )
  port map(
       Address => a,
       Data_Im => W_im( NW-1 downto 0 ),
       Data_Re => W_re( NW-1 downto 0 ),
       clk => clk
  );

if1:  if LOG2NP-STAGE = 0 generate
           a<=0;
       end generate if1;
if2:  if LOG2NP-STAGE/=0 generate
U2 :   BU_addr_counter
          generic map (
               N => LOG2NP-STAGE,
               START => 1-N-(STAGE-1)*(N+N+NW)
          )
          port map(
                 addr => a,
                 clk => clk,
                 reset => clear
          );
      end generate if2;

end WN_Unit;
```



Σχήμα D.2 Σχεδιάγραμμα της μονάδας butterfly που βρίσκετε στο αρχείο butterfly_unit.bde

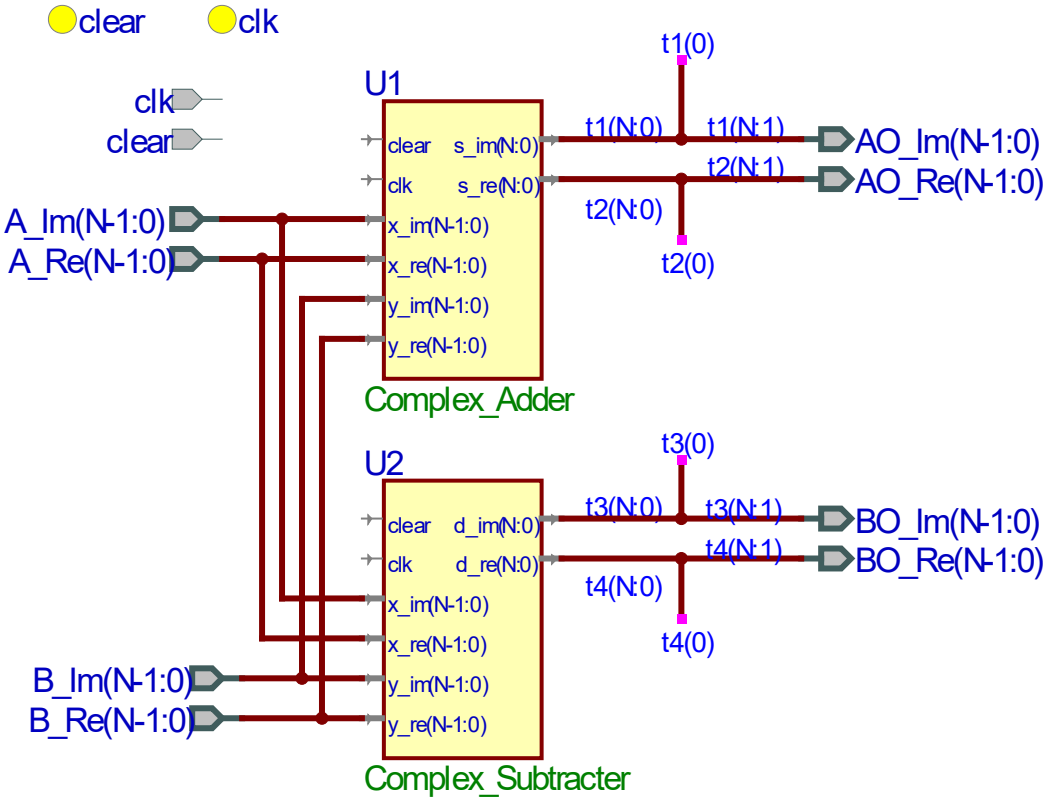
butterfly_unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;


-- other libraries declarations
library COMPLEX_ADDER;
library COMPLEX_MULTIPLIER;
library COMPLEX_SHIFT_REGISTER;
library COMPLEX_SUBTRACTER;

entity Butterfly_Unit is
  generic(
       NW : Integer := 4;
       N : Integer  := 5;
       LOG2NP : Integer := 3;
       STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Butterfly_Unit;

architecture Butterfly_Unit of Butterfly_Unit is

---- Component declarations -----

component WN_Unit
  generic(
       LOG2NP : INTEGER := 3;
       N : INTEGER := 8;
       NW : INTEGER := 8;
       STAGE : INTEGER := 1
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       W_im : out STD_LOGIC_VECTOR(NW-1 downto 0);
       W_re : out STD_LOGIC_VECTOR(NW-1 downto 0)
  );
end component;
component Complex_Adder
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       s_im : out STD_LOGIC_VECTOR(N downto 0);
       s_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;
component Complex_Multiplier
```

```
  generic(
       NX : POSITIVE := 4;
       NY : POSITIVE := 5
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(NX-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(NX-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(NY-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(NY-1 downto 0);
       m_im : out STD_LOGIC_VECTOR(NX-2 downto 0);
       m_re : out STD_LOGIC_VECTOR(NX-2 downto 0)
  );
end component;
component Complex_Shift_Register
  generic(
       N : INTEGER := 8;
       W : INTEGER := 2
  );
  port (
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       oa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       oa_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component Complex_Subtracter
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       d_im : out STD_LOGIC_VECTOR(N downto 0);
       d_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;

---- Signal declarations used on the diagram ----

signal da1_im_bus : STD_LOGIC_VECTOR (N downto 0);
signal da1_re_bus : STD_LOGIC_VECTOR (N downto 0);
signal dx_im_bus : STD_LOGIC_VECTOR (N downto 0);
signal dx_re_bus : STD_LOGIC_VECTOR (N downto 0);
signal t1 : STD_LOGIC_VECTOR (N-2 downto 0);
signal t2 : STD_LOGIC_VECTOR (N-2 downto 0);
signal w_im_bus : STD_LOGIC_VECTOR (NW-1 downto 0);
signal w_re_bus : STD_LOGIC_VECTOR (NW-1 downto 0);

---- Configuration specifications for declared components

for U1 : Complex_Adder use entity complex_adder.Complex_Adder;
for U2 : Complex_Subtracter use entity complex_subtracter.Complex_Subtracter;
for U3 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;
for U4 : Complex_Multiplier use entity complex_multiplier.Complex_Multiplier;

begin

---- User Signal Assignments ----
AO_Im <= t1(N-2)&t1(N-2 downto 0);
AO_Re <= t2(N-2)&t2(N-2 downto 0);
```

```
----  Component instantiations  ----

U1 : Complex_Adder
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       s_im => da1_im_bus( N downto 0 ),
       s_re => da1_re_bus( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U2 : Complex_Subtracter
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       d_im => dx_im_bus( N downto 0 ),
       d_re => dx_re_bus( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U3 : Complex_Shift_Register
  generic map (
       N => N-1,
       W => N+NW
  )
  port map(
       a_im => da1_im_bus( N downto 2 ),
       a_re => da1_re_bus( N downto 2 ),
       clear => clear,
       clk => clk,
       oa_im => t1( N-2 downto 0 ),
       oa_re => t2( N-2 downto 0 )
  );

U4 : Complex_Multiplier
  generic map (
       NX => N+1,
       NY => NW
  )
  port map(
       clear => clear,
       clk => clk,
       m_im => BO_Im( N-1 downto 0 ),
       m_re => BO_Re( N-1 downto 0 ),
       x_im => dx_im_bus( N downto 0 ),
       x_re => dx_re_bus( N downto 0 ),
       y_im => w_im_bus( NW-1 downto 0 ),
       y_re => w_re_bus( NW-1 downto 0 )
  );

U5 : WN_Unit
  generic map (
       LOG2NP => LOG2NP,
       N => N,
       NW => NW,
       STAGE => STAGE
  )
  port map(
```

```
        W_im => w_im_bus( NW-1 downto 0 ),
        W_re => w_re_bus( NW-1 downto 0 ),
        clear => clear,
        clk => clk
  );


end Butterfly_Unit;
```



Σχήμα D.3 Σχεδιάγραμμα της τελευταίας μονάδας butterfly που βρίσκετε στο αρχείο last_butterfly_unit.bde

Last_Butterfly_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;


-- other libraries declarations
library COMPLEX_ADDER;
library COMPLEX_SUBTRACTER;

entity Last_Butterfly_Unit is
  generic(
       N : Integer  := 5
  );
```

```
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Last_Butterfly_Unit;

architecture Last_Butterfly_Unit of Last_Butterfly_Unit is

---- Component declarations -----

component Complex_Adder
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       s_im : out STD_LOGIC_VECTOR(N downto 0);
       s_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;
component Complex_Subtracter
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       d_im : out STD_LOGIC_VECTOR(N downto 0);
       d_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;

---- Signal declarations used on the diagram ----

signal t1 : STD_LOGIC_VECTOR (N downto 0);
signal t2 : STD_LOGIC_VECTOR (N downto 0);
signal t3 : STD_LOGIC_VECTOR (N downto 0);
signal t4 : std_logic_vector (N downto 0);

---- Configuration specifications for declared components

for U1 : Complex_Adder use entity complex_adder.Complex_Adder;
for U2 : Complex_Subtracter use entity complex_subtracter.Complex_Subtracter;

begin

----  Component instantiations  ----

U1 : Complex_Adder
  generic map (
       N => N
  )
  port map(
```

```
       clear => clear,
       clk => clk,
       s_im => t1( N downto 0 ),
       s_re => t2( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U2 : Complex_Subtracter
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       d_im => t3( N downto 0 ),
       d_re => t4( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );


---- Terminal assignment ----

    -- Output\buffer terminals
     AO_Im <= t1( N downto 1 );
     AO_Re <= t2( N downto 1 );
     BO_Im <= t3( N downto 1 );
     BO_Re <= t4( N downto 1 );


end Last_Butterfly_Unit;
```

butterfly_unit_tb.vhd

```
library ieee,complex_adder,complex_multiplier,complex_subtracter;
use ieee.std_logic_1164.all;

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;
use IEEE.math_real.all;

entity butterfly_unit_tb is
     -- Generic declarations of the tested unit
          generic(
          NW : INTEGER := 4;
          N : INTEGER := 4;
          LOG2NP : INTEGER := 10;
          STAGE : INTEGER := 3 );
end butterfly_unit_tb;

architecture TB_ARCHITECTURE of butterfly_unit_tb is

file OUTPUT_FILE: TEXT open WRITE_MODE is "outputs.txt";

TYPE rom_array is ARRAY(natural range <>) of STD_LOGIC_VECTOR (NW-1 downto 0);
constant memory_size : Integer := 2**(LOG2NP-STAGE);

function cos2c( X : integer ) return STD_LOGIC_VECTOR is
variable tcos : integer;
variable t : SIGNED(0 to NW-1);
begin
```

```
     tcos := integer( (2.0**(NW-2)-0.0)/1.0*COS( -
real(x)*2.0*MATH_PI/real(2**LOG2NP) ) );
     t := CONV_SIGNED(tcos,NW);
     return CONV_STD_LOGIC_VECTOR(t,NW);
end function cos2c;

function calc_rom_re return rom_array is
variable mem : rom_array(0 to memory_size-1);

begin
L1:  for k in 0 to (memory_size-1) loop
          mem(k) := cos2c((2**(STAGE-1))*(k mod 2**(LOG2NP-STAGE)));
     end loop L1;
     return mem;

end function calc_rom_re;

function sin2c( X : integer ) return STD_LOGIC_VECTOR is
variable tsin : integer;
variable t : SIGNED(0 to NW-1);
begin
     tsin := integer( (2.0**(NW-2)-0.0)/1.0*SIN( -
real(x)*2.0*MATH_PI/real(2**LOG2NP) ) );
     t := CONV_SIGNED(tsin,NW);
     return CONV_STD_LOGIC_VECTOR(t,NW);
end function sin2c;

function calc_rom_im return rom_array is
variable mem : rom_array(0 to memory_size-1);

begin
L1:  for k in 0 to memory_size-1 loop
          mem(k) := sin2c((2**(STAGE-1))*(k mod 2**(LOG2NP-STAGE)));
     end loop L1;
     return mem;

end function calc_rom_im;

constant memory_re : rom_array(0 to memory_size-1) := calc_rom_re;
constant memory_im : rom_array(0 to memory_size-1) := calc_rom_im;

     -- Component declaration of the tested unit
     component butterfly_unit
          generic(
          NW : INTEGER := 8;
          N : INTEGER := 8;
          LOG2NP : INTEGER := 3;
          STAGE : INTEGER := 1 );
     port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0) );
     end component;

     constant DELAY1 : time := 1 ns;

     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clear : std_logic;
     signal clk : std_logic;
     signal A_Im : std_logic_vector(N-1 downto 0);
     signal A_Re : std_logic_vector(N-1 downto 0);
     signal B_Im : std_logic_vector(N-1 downto 0);
```

```
     signal B_Re : std_logic_vector(N-1 downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal AO_Im : std_logic_vector(N-1 downto 0);
     signal AO_Re : std_logic_vector(N-1 downto 0);
     signal BO_Im : std_logic_vector(N-1 downto 0);
     signal BO_Re : std_logic_vector(N-1 downto 0);

     -- Add your code here ...
begin

     -- Unit Under Test port map
     UUT : butterfly_unit
          generic map (
               NW => NW,
               N => N,
               LOG2NP => LOG2NP,
               STAGE => STAGE
          )

          port map (
               clear => clear,
               clk => clk,
               A_Im => A_Im,
               A_Re => A_Re,
               B_Im => B_Im,
               B_Re => B_Re,
               AO_Im => AO_Im,
               AO_Re => AO_Re,
               BO_Im => BO_Im,
               BO_Re => BO_Re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;

     INPUT_Generation: process
     variable nA_Re : integer;
     variable nA_Im : integer;
     variable nB_Re : integer;
     variable nB_Im : integer;

     begin
          A_Re <= CONV_STD_LOGIC_VECTOR(0,N);
          A_Im <= CONV_STD_LOGIC_VECTOR(0,N);
          B_Re <= CONV_STD_LOGIC_VECTOR(0,N);
          B_Im <= CONV_STD_LOGIC_VECTOR(0,N);
          clear <= '0';
          wait until (clk'event) and (clk='1');
          clear <= '1';

          for nA_Re in -2**(N-1) to 2**(N-1)-1 loop
               for nA_Im in -2**(N-1) to 2**(N-1)-1 loop
                    for nB_Re in -2**(N-1) to 2**(N-1)-1 loop
                         for nB_Im in -2**(N-1) to 2**(N-1)-1 loop
-- uncheck the folowing loop if you want to do ecsosting test for all coeficients
--                         for i in 0 to 2**(LOG2NP-STAGE)-1 loop
                              A_Re <= CONV_STD_LOGIC_VECTOR(nA_Re,N);
                              A_Im <= CONV_STD_LOGIC_VECTOR(nA_Im,N);
                              B_Re <= CONV_STD_LOGIC_VECTOR(nB_Re,N);
                              B_Im <= CONV_STD_LOGIC_VECTOR(nB_Im,N);
                              wait until (clk'event) and (clk='1');
--                         end loop;
                         end loop;
                    end loop;
               end loop;
          end loop;
```

```
          wait;

     end process;

     OUTPUT_Checking:process
          type Matrix is array (NATURAL range <>) of std_logic_vector(N-1 downto 0);
          type Int_Matrix is array (NATURAL range <>) of Integer;

          constant BUTTERFLY_DELAY : integer := N+N+NW;

          variable A_Re_Matrix : Matrix (0 to
BUTTERFLY_DELAY):=(others=>(others=>'0'));
          variable A_Im_Matrix : Matrix (0 to
BUTTERFLY_DELAY):=(others=>(others=>'0'));
          variable B_Re_Matrix : Matrix (0 to
BUTTERFLY_DELAY):=(others=>(others=>'0'));
          variable B_Im_Matrix : Matrix (0 to
BUTTERFLY_DELAY):=(others=>(others=>'0'));
          variable Addr_Matrix : Int_Matrix (0 to BUTTERFLY_DELAY):=(others=>0);


          variable report_line : line;
          variable report_str : string(1 to 160);
          variable i,j : integer := 0;
          variable AO_re_part, AO_im_part, BO_re_part, BO_im_part : integer := 0;
          variable t1_re,t1_im,t2_re,t2_im : integer :=0;
          variable t1 : std_logic_vector(N downto 0);
          variable t2 : std_logic_vector(N-1 downto 0);
          variable t3 : std_logic_vector(N+NW downto 0);
          variable cur_addr : integer := 1-N-(STAGE-1)*BUTTERFLY_DELAY+N-2;


     begin

          wait until (clk'event) and (clk='1') and (clear='1');
          cur_addr := (cur_addr+1) mod 2**(LOG2NP-STAGE);
          for i in 0 to BUTTERFLY_DELAY-1 loop
               A_Re_Matrix(i) := A_Re_Matrix(i+1);
               A_Im_Matrix(i) := A_Im_Matrix(i+1);
               B_Re_Matrix(i) := B_Re_Matrix(i+1);
               B_Im_Matrix(i) := B_Im_Matrix(i+1);
               Addr_Matrix(i) := Addr_Matrix(i+1);
          end loop;
          A_Re_Matrix(BUTTERFLY_DELAY) := A_Re;
          A_Im_Matrix(BUTTERFLY_DELAY) := A_Im;
          B_Re_Matrix(BUTTERFLY_DELAY) := B_Re;
          B_Im_Matrix(BUTTERFLY_DELAY) := B_Im;
          Addr_Matrix(BUTTERFLY_DELAY) := cur_addr;

          t1_re := conv_integer(A_Re_Matrix(0))+conv_integer(B_Re_Matrix(0));
          t1 := conv_std_logic_vector(t1_re,N+1);
          t1_re := conv_integer(t1(N)&t1(N downto 2));

          t1_im := (conv_integer(A_Im_Matrix(0))+conv_integer(B_Im_Matrix(0)));
          t1 := conv_std_logic_vector(t1_im,N+1);
          t1_im := conv_integer(t1(N)&t1(N downto 2));

          t2_re := (conv_integer(A_Re_Matrix(0))-conv_integer(B_Re_Matrix(0)));
          t2_im := (conv_integer(A_Im_Matrix(0))-conv_integer(B_Im_Matrix(0)));

          AO_re_part := t1_re;
          AO_im_part := t1_im;

          BO_re_part := t2_re*conv_integer(memory_re(Addr_Matrix(0)))-
t2_im*conv_integer(memory_im(Addr_Matrix(0)));
          t3 := conv_std_logic_vector(BO_re_part,N+NW+1);
          BO_re_part := conv_integer(t3(N+NW downto NW));
```

```
          BO_im_part :=
t2_re*conv_integer(memory_im(Addr_Matrix(0)))+t2_im*conv_integer(memory_re(Addr_Matr
ix(0)));
          t3 := conv_std_logic_vector(BO_im_part,N+NW+1);
          BO_im_part := conv_integer(t3(N+NW downto NW));

          report_line:=null;

          if ((conv_integer(AO_Re) /= AO_re_part) or (conv_integer(AO_Im) /=
AO_im_part)) then

               write(report_line,string'(">>>>>>>>>>Wrong Result AO: "));

               write(report_line,conv_integer(A_Re_Matrix(0)));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(A_Im_Matrix(0)));
               write(report_line,string'(" + "));
               write(report_line,conv_integer(B_Re_Matrix(0)));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(B_Im_Matrix(0)));
               write(report_line,string'(" = "));
               write(report_line,conv_integer(AO_re));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(AO_im));

               write(report_line,string'("  Right: "));
               write(report_line,conv_integer(AO_re_part));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(AO_im_part));
               write(report_line,string'("                              "));

               for i in report_line'range loop
                    report_str(i) := report_line(i);
               end loop;
               report report_str;

               writeline(OUTPUT_FILE,report_line);

          end if;

          if ((conv_integer(BO_Re) /= BO_re_part) or (conv_integer(BO_Im) /=
BO_im_part)) then

               write(report_line,string'(">>>>>>>>>>Wrong Result BO: A="));

               write(report_line,conv_integer(A_Re_Matrix(0)));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(A_Im_Matrix(0)));
               write(report_line,string'(" , B="));
               write(report_line,conv_integer(B_Re_Matrix(0)));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(B_Im_Matrix(0)));
               write(report_line,string'(" => BO="));
               write(report_line,conv_integer(BO_re));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(BO_im));

               write(report_line,string'("  Right: "));
               write(report_line,conv_integer(BO_re_part));
               write(report_line,string'("+i"));
               write(report_line,conv_integer(BO_im_part));
               write(report_line,string'(" addr: "));
               write(report_line,conv_integer(Addr_Matrix(0)));
               write(report_line,string'("      "));

               for i in report_line'range loop
                    report_str(i) := report_line(i);
               end loop;
               report report_str;
```

```
                writeline(OUTPUT_FILE,report_line);

           end if;

           if ( (conv_integer(A_Re_Matrix(0)) = 2**(N-1)-1) and
(conv_integer(A_Im_Matrix(0)) = 2**(N-1)-1) and
                 (conv_integer(B_Re_Matrix(0)) = 2**(N-1)-1) and
(conv_integer(B_Im_Matrix(0)) = 2**(N-1)-1) ) then
                write(report_line,string'("------->The End!!! "));
                writeline(OUTPUT_FILE,report_line);
                report "Test Finished!!!" severity FAILURE;
                wait;
           end if;

end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_butterfly_unit of butterfly_unit_tb is
     for TB_ARCHITECTURE
          for UUT : butterfly_unit
               use entity work.butterfly_unit(butterfly_unit);
          end for;
     end for;
end TESTBENCH_FOR_butterfly_unit;
```

D.6 Μονάδα αναδιάταξης

Η μονάδα αναδιάταξης χρησιμοποιεί για την περιγραφή της τα αρχεία:

tff.vhd	Περιγραφή T Flip-Flop της μονάδας αναδιάταξης
SU_counter.vhd	Περιγραφή του μετρητή που χρησιμοποιείτε για να συγχρονίζει τον πολυπλέκτη
SU_Mux	Περιγραφή του πολυπλέκτη της μονάδας αναδιάταξης
Shuffler.vhd	Περιγραφή της μονάδας αναδιάταξης χωρίς τον μετρητή για τον συγχρονισμό, χρησιμοποιώντας το block diagram αρχείο Shuffler.bde
Shuffling_Unit.vhd	Περιγραφή της μονάδας αναδιάταξης με το κύκλωμα του μετρητή για τον απαραίτητο συγχρονισμό, που δημιουργείται χρησιμοποιώντας το block diagram αρχείο Shuffling_Unit.bde
shuffling_unit_tb.vhd	Test bench με το οποίο μπορούμε να ελέγξουμε την σωστή λειτουργία της μονάδας αναδιάταξης

SU_counter.vhd

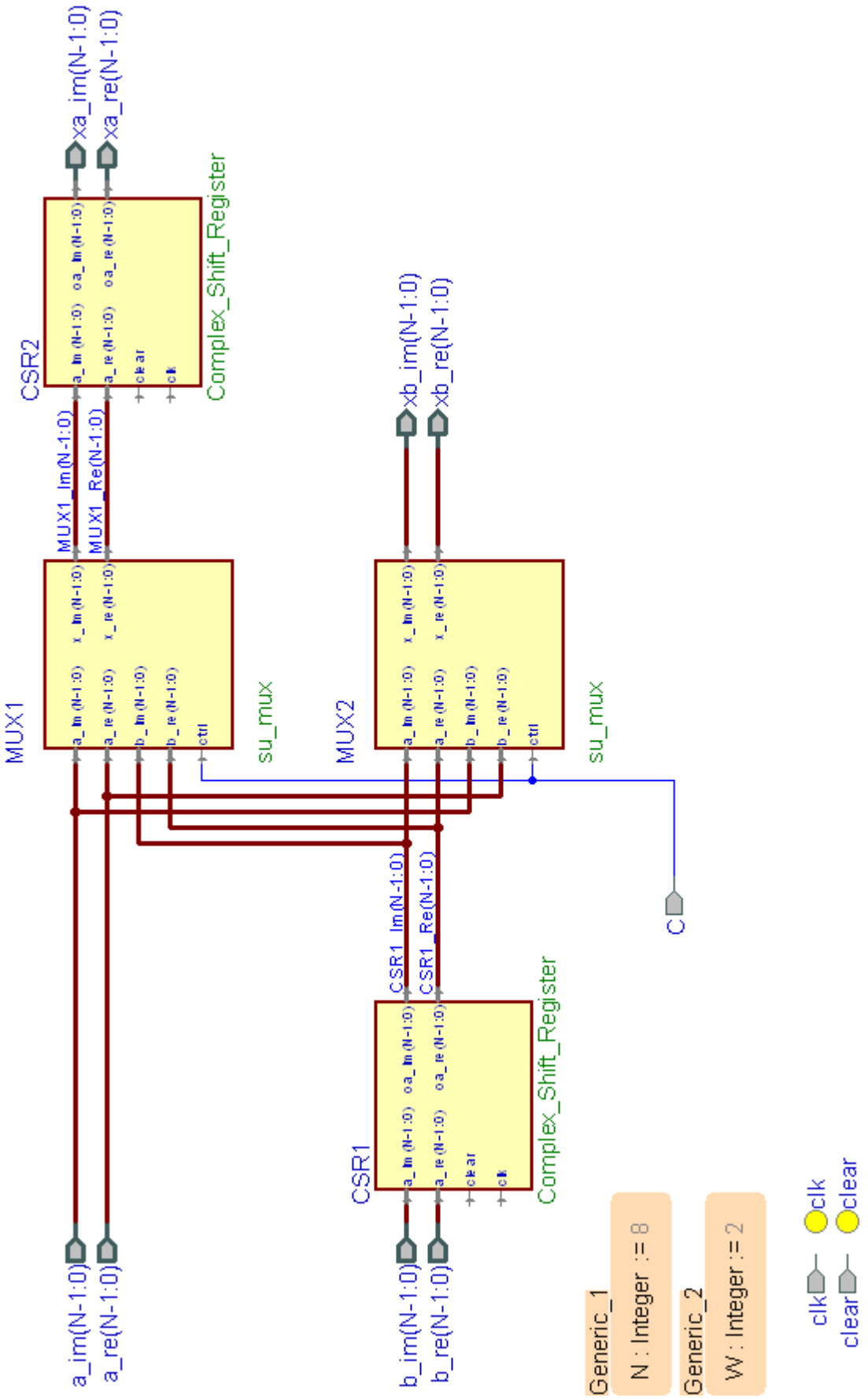
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;

entity SU_counter is
      generic (
           N : Integer := 5;
           START : Integer := 0
      );
      port(
            clk : in STD_LOGIC;
            clear : in STD_LOGIC;
            C : out STD_LOGIC_VECTOR(N-1 downto 0)
           );
end SU_counter;
```

```
architecture SU_counter of SU_counter is

component tff is
      generic (
          CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
           T : in STD_LOGIC;
           clear : in STD_LOGIC;
           clk : in STD_LOGIC;
           Q : out STD_LOGIC
          );
end component;

signal TT,TQ : std_logic_vector(N-1 downto 0);
constant START_VECTOR : STD_LOGIC_VECTOR(N-1 downto 0) :=
CONV_STD_LOGIC_VECTOR(START,N);

begin

          TT(0) <= '1';
TffR:tff  generic map (
               CLEAR_BIT => START_VECTOR(0) )
          port map (
               T=>TT(0),
               clear=>clear,
               clk=>clk,
               Q=>TQ(0)
          );
G1:       for i in 1 to N-1 generate
                    TT(i) <= TQ(i-1) AND TT(i-1);
Tffi:          tff generic map (
                         CLEAR_BIT => START_VECTOR(i) )
                    port map (
                         T =>TT(i),
                         clear=>clear,
                         clk=>clk,
                         Q=>TQ(i)
                    );
          end generate;
          C<=TQ;

end SU_counter;
```

SU_Mux.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;


entity SU_Mux is
      generic   (
          N : Integer := 8
      );
      port(
           a_re : in STD_LOGIC_VECTOR (N-1 downto 0);
           a_im : in STD_LOGIC_VECTOR (N-1 downto 0);
           b_re : in STD_LOGIC_VECTOR (N-1 downto 0);
           b_im : in STD_LOGIC_VECTOR (N-1 downto 0);
           ctrl : in STD_LOGIC;
           x_re : out STD_LOGIC_VECTOR (N-1 downto 0);
           x_im : out STD_LOGIC_VECTOR (N-1 downto 0)
          );
end SU_Mux;


architecture SU_Mux of SU_Mux is
```

```
begin
     process (a_re,a_im,b_re,b_im,ctrl)
     begin
I1:              if ctrl='0' then
                       x_re<=a_re;
                       x_im<=a_im;
                  elsif ctrl='1' then
                       x_re<=b_re;
                       x_im<=b_im;
                  end if I1;
     end process;
end SU_Mux;
```

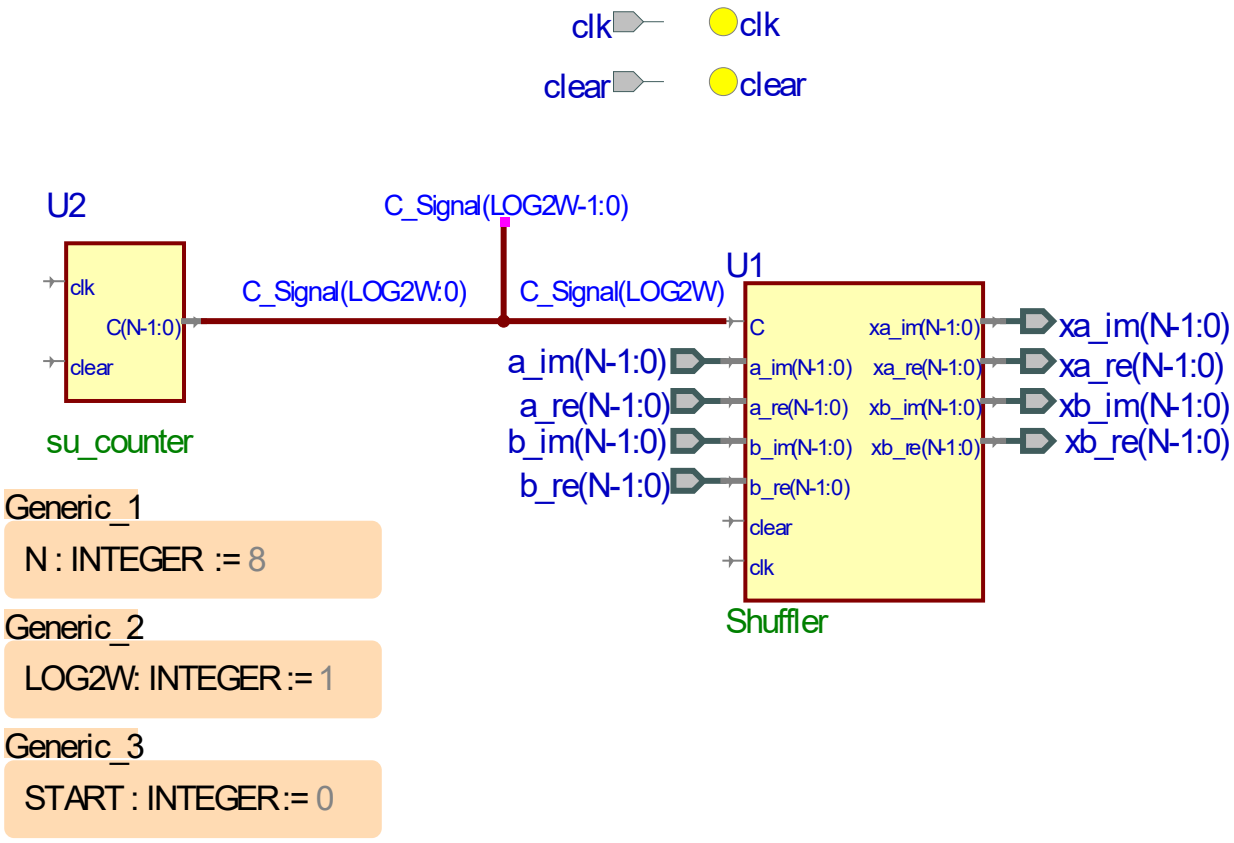


Σχήμα D.4 Μονάδα αναδιάταξης χωρίς τον μετρητή που οδηγεί τους πολυπλέκτες για σωστό συγχρονισμό

Shuffler.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;

-- other libraries declarations
library COMPLEX_SHIFT_REGISTER;

entity Shuffler is
  generic(
       N : Integer  := 8;
       W : Integer := 2
  );
  port(
       C : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Shuffler;

architecture Shuffler of Shuffler is

---- Component declarations -----

component SU_Mux
  generic(
       N : INTEGER := 8
  );
  port (
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       ctrl : in STD_LOGIC;
       x_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component Complex_Shift_Register
  generic(
       N : INTEGER := 8;
       W : INTEGER := 2
  );
  port (
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       oa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       oa_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

---- Signal declarations used on the diagram ----

signal CSR1_Im : STD_LOGIC_VECTOR (N-1 downto 0);
signal CSR1_Re : STD_LOGIC_VECTOR (N-1 downto 0);
signal MUX1_Im : STD_LOGIC_VECTOR (N-1 downto 0);
signal MUX1_Re : STD_LOGIC_VECTOR (N-1 downto 0);
```

```
---- Configuration specifications for declared components

for CSR1 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;
for CSR2 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;

begin

----  Component instantiations  ----

CSR1 : Complex_Shift_Register
  generic map (
       N => N,
       W => W
  )
  port map(
       a_im => b_im( N-1 downto 0 ),
       a_re => b_re( N-1 downto 0 ),
       clear => clear,
       clk => clk,
       oa_im => CSR1_Im( N-1 downto 0 ),
       oa_re => CSR1_Re( N-1 downto 0 )
  );

CSR2 : Complex_Shift_Register
  generic map (
       N => N,
       W => W
  )
  port map(
       a_im => MUX1_Im( N-1 downto 0 ),
       a_re => MUX1_Re( N-1 downto 0 ),
       clear => clear,
       clk => clk,
       oa_im => xa_im( N-1 downto 0 ),
       oa_re => xa_re( N-1 downto 0 )
  );

MUX1 : SU_Mux
  generic map (
       N => N
  )
  port map(
       a_im => a_im( N-1 downto 0 ),
       a_re => a_re( N-1 downto 0 ),
       b_im => CSR1_Im( N-1 downto 0 ),
       b_re => CSR1_Re( N-1 downto 0 ),
       ctrl => C,
       x_im => MUX1_Im( N-1 downto 0 ),
       x_re => MUX1_Re( N-1 downto 0 )
  );

MUX2 : SU_Mux
  generic map (
       N => N
  )
  port map(
       a_im => CSR1_Im( N-1 downto 0 ),
       a_re => CSR1_Re( N-1 downto 0 ),
       b_im => a_im( N-1 downto 0 ),
       b_re => a_re( N-1 downto 0 ),
       ctrl => C,
       x_im => xb_im( N-1 downto 0 ),
       x_re => xb_re( N-1 downto 0 )
  );

end Shuffler;
```



Σχήμα D.5 Μονάδα αναδιάταξης όπως σχεδιάστηκε στο αρχείο Shuffling_Unit.bde

Shuffling_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity Shuffling_Unit is
  generic(
       N : INTEGER  := 8;
       LOG2W: INTEGER := 1;
       START : INTEGER := 0
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Shuffling_Unit;

architecture Shuffling_Unit of Shuffling_Unit is

---- Component declarations -----

component Shuffler
  generic(
       N : INTEGER := 8;
```

```
        W : INTEGER := 2
    );
    port (
        C : in STD_LOGIC;
        a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
        a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
        b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
        b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
        xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
        xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
        xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
    );
end component;
component su_counter
    generic(
        N : INTEGER := 5;
        START : INTEGER := 0
    );
    port (
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        C : out STD_LOGIC_VECTOR(N-1 downto 0)
    );
end component;

---- Signal declarations used on the diagram ----

signal C_Signal : STD_LOGIC_VECTOR (LOG2W downto 0);

begin

----  Component instantiations  ----

U1 : Shuffler
    generic map (
        N => N,
        W => 2**LOG2W
    )
    port map(
        C => C_Signal(LOG2W),
        a_im => a_im( N-1 downto 0 ),
        a_re => a_re( N-1 downto 0 ),
        b_im => b_im( N-1 downto 0 ),
        b_re => b_re( N-1 downto 0 ),
        clear => clear,
        clk => clk,
        xa_im => xa_im( N-1 downto 0 ),
        xa_re => xa_re( N-1 downto 0 ),
        xb_im => xb_im( N-1 downto 0 ),
        xb_re => xb_re( N-1 downto 0 )
    );

U2 : su_counter
    generic map (
        N => LOG2W+1,
        START => START
    )
    port map(
        C => C_Signal( LOG2W downto 0 ),
        clear => clear,
        clk => clk
    );


end Shuffling_Unit;
```

shuffling_unit_tb.vhd

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

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;

entity shuffling_unit_tb is
     -- Generic declarations of the tested unit
          generic(
          N : INTEGER := 8;
          LOG2W : INTEGER := 2;
          START : INTEGER := 0 );
end shuffling_unit_tb;

architecture TB_ARCHITECTURE of shuffling_unit_tb is
     -- Component declaration of the tested unit
     component shuffling_unit
          generic(
          N : INTEGER := 8;
          LOG2W : INTEGER := 1;
          START : INTEGER := 0 );
     port(
          clear : in std_logic;
          clk : in std_logic;
          a_im : in std_logic_vector((N-1) downto 0);
          a_re : in std_logic_vector((N-1) downto 0);
          b_im : in std_logic_vector((N-1) downto 0);
          b_re : in std_logic_vector((N-1) downto 0);
          xa_im : out std_logic_vector((N-1) downto 0);
          xa_re : out std_logic_vector((N-1) downto 0);
          xb_im : out std_logic_vector((N-1) downto 0);
          xb_re : out std_logic_vector((N-1) downto 0) );
     end component;

     constant DELAY1 : time := 1 ns;

     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clear : std_logic;
     signal clk : std_logic;
     signal a_im : std_logic_vector((N-1) downto 0);
     signal a_re : std_logic_vector((N-1) downto 0);
     signal b_im : std_logic_vector((N-1) downto 0);
     signal b_re : std_logic_vector((N-1) downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal xa_im : std_logic_vector((N-1) downto 0);
     signal xa_re : std_logic_vector((N-1) downto 0);
     signal xb_im : std_logic_vector((N-1) downto 0);
     signal xb_re : std_logic_vector((N-1) downto 0);

     -- Add your code here ...

begin

     -- Unit Under Test port map
     UUT : shuffling_unit
          generic map (
               N => N,
               LOG2W => LOG2W,
               START => START
          )

          port map (
               clear => clear,
               clk => clk,
               a_im => a_im,
```

```
                a_re => a_re,
                b_im => b_im,
                b_re => b_re,
                xa_im => xa_im,
                xa_re => xa_re,
                xb_im => xb_im,
                xb_re => xb_re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;

     INPUT_Generation: process

     begin
          a_re <= CONV_STD_LOGIC_VECTOR(2,N);
          a_im <= CONV_STD_LOGIC_VECTOR(1,N);
          b_re <= CONV_STD_LOGIC_VECTOR(4,N);
          b_im <= CONV_STD_LOGIC_VECTOR(3,N);
          clear <= '0';
          wait until (clk'event) and (clk='1');
          clear <= '1';

          for i in 1 to 100 loop
               wait until (clk'event) and (clk='1');
               a_re <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(a_re)+4,N);
               a_im <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(a_im)+4,N);
               b_re <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(b_re)+4,N);
               b_im <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(b_im)+4,N);
          end loop;
          wait;

     end process;
end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_shuffling_unit of shuffling_unit_tb is
     for TB_ARCHITECTURE
          for UUT : shuffling_unit
               use entity work.shuffling_unit(shuffling_unit);
          end for;
     end for;
end TESTBENCH_FOR_shuffling_unit;
```

D.7 Μονάδα βάσης-2 γρήγορου μετασχηματισμού Fourier

Η μονάδα αυτή αποτελεί το ανώτατο επίπεδο της περιγραφής και είναι αυτή που ουσιαστικά δημιουργεί το κύκλωμα του FFT και αποτελείται απ' τα αρχεία

FFT_Unit.vhd	Περιγραφή της μονάδας FFT
FFT_Unit_TB.vhd	Test Bench με το οποίο ελέγχουμε την μονάδα FFT

FFT_Unit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

library BUTTERFLY_UNIT;
library SHUFFLING_UNIT;
```

```
entity FFT_Unit is
      generic(
          NW : Integer := 8;         -- bits for twiddle factors
          N : Integer  := 16;      -- bits for inputs and outputs
          LOG2NP : Integer := 3      -- NP-Points FFT
      );
      port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
      );
end FFT_Unit;

architecture FFT_Unit of FFT_Unit is

component Butterfly_Unit is
  generic(
          NW : Integer := 8;
          N : Integer  := 8;
          LOG2NP : Integer := 3;
          STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

component Last_Butterfly_Unit is
  generic(
       N : Integer  := 5
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

component Shuffling_Unit is
  generic(
       N : INTEGER  := 8;
        LOG2W: INTEGER := 1;
        START : INTEGER := 0
  );
  port(
       clear : in STD_LOGIC;
```

```
       clk : in STD_LOGIC;
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

subtype num_word is STD_LOGIC_VECTOR (N-1 downto 0);
type num_word_array is array(natural range <>) of num_word;
signal Up_Re_in,Up_Im_in,Down_Re_in,Down_Im_in,
        Up_Re_out,Up_Im_out,Down_Re_out,Down_Im_out : num_word_array(1 to LOG2NP);

begin

          --Batterfly Units generation
BUgi:for i in 1 to LOG2NP-1 generate
BU:             Butterfly_Unit
                generic map (
                     NW => NW,
                     N => N,
                     LOG2NP => LOG2NP,
                     STAGE => i
                )
                port map(
                     clear => clear,
                     clk => clk,
                     A_Im => Up_Im_in(i),
                     A_Re => Up_Re_in(i),
                     B_Im => Down_Im_in(i),
                     B_Re => Down_Re_in(i),
                     AO_Im => Up_Im_out(i),
                     AO_Re => Up_Re_out(i),
                     BO_Im => Down_Im_out(i),
                     BO_Re => Down_Re_out(i)
                );
          end generate BUgi;
LastBU:   Last_Butterfly_Unit
                generic map (
                     N => N
                )
                port map(
                     clear => clear,
                     clk => clk,
                     A_Im => Up_Im_in(LOG2NP),
                     A_Re => Up_Re_in(LOG2NP),
                     B_Im => Down_Im_in(LOG2NP),
                     B_Re => Down_Re_in(LOG2NP),
                     AO_Im => Up_Im_out(LOG2NP),
                     AO_Re => Up_Re_out(LOG2NP),
                     BO_Im => Down_Im_out(LOG2NP),
                     BO_Re => Down_Re_out(LOG2NP)
                );

          -- Shuffling untis generation
SUgi:for i in 2 to LOG2NP generate
SU:             Shuffling_Unit
                generic map (
                     N => N,
                     LOG2W => LOG2NP-i,
                     START => -(i-1)*(N+N+NW)
                )

                port map (
                     clear => clear,
                     clk => clk,
```

```
                    a_im => Up_Im_out(i-1),
                    a_re => Up_Re_out(i-1),
                    b_im => Down_Im_out(i-1),
                    b_re => Down_Re_out(i-1),
                    xa_im => Up_Im_in(i),
                    xa_re => Up_Re_in(i),
                    xb_im => Down_Im_in(i),
                    xb_re => Down_Re_in(i)
               );
          end generate SUgi;

          -- input signals
          Up_Im_in(1) <= A_Im;
          Up_Re_in(1) <= A_Re ;
          Down_Im_in(1) <= B_Im;
          Down_Re_in(1) <= B_Re;

          -- output signals
          AO_Im <= Up_Im_out(LOG2NP);
          AO_Re <= Up_Re_out(LOG2NP);
          BO_Im <= Down_Im_out(LOG2NP);
          BO_Re <= Down_Re_out(LOG2NP);

end FFT_Unit;
```

FFT_Unit_TB.vhd

```
library ieee,butterfly_unit,shuffling_unit;
use ieee.std_logic_1164.all;

     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;
use IEEE.math_real.all;

entity fft_unit_tb is
     -- Generic declarations of the tested unit
          generic(
          NW : INTEGER := 12;
          N : INTEGER := 13;
          LOG2NP : INTEGER := 5 );
end fft_unit_tb;

architecture TB_ARCHITECTURE of fft_unit_tb is
     file OUTPUT_FILE : TEXT open WRITE_MODE is "outputs.txt";

     -- Component declaration of the tested unit
     component fft_unit
          generic(
          NW : INTEGER := 8;
          N : INTEGER := 16;
          LOG2NP : INTEGER := 3 );
     port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0) );
     end component;

     constant DELAY1 : time := 1 ns;
```

```
     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clk : std_logic;
     signal clear : std_logic;
     signal A_Im : std_logic_vector((N-1) downto 0);
     signal A_Re : std_logic_vector((N-1) downto 0);
     signal B_Im : std_logic_vector((N-1) downto 0);
     signal B_Re : std_logic_vector((N-1) downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal AO_Im : std_logic_vector((N-1) downto 0);
     signal AO_Re : std_logic_vector((N-1) downto 0);
     signal BO_Im : std_logic_vector((N-1) downto 0);
     signal BO_Re : std_logic_vector((N-1) downto 0);

     -- Add your code here ...

function calc_f( t : positive ) return integer is
variable f : real;
begin
     f := real(2**(N-3)-1)*COS( -real(t)*2.0*MATH_PI/3.0 )+real(2**(N-3)-1)*COS( -
real(t)*2.0*MATH_PI/11.0 );
     return integer(f);
end function calc_f;

begin

     -- Unit Under Test port map
     UUT : fft_unit
          generic map (
               NW => NW,
               N => N,
               LOG2NP => LOG2NP
          )

          port map (
               clk => clk,
               clear => clear,
               A_Im => A_Im,
               A_Re => A_Re,
               B_Im => B_Im,
               B_Re => B_Re,
               AO_Im => AO_Im,
               AO_Re => AO_Re,
               BO_Im => BO_Im,
               BO_Re => BO_Re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;

     INPUT_Generation: process
     variable seed1 : integer := 11111111;
     variable seed2 : integer := 22222222;
     variable rnd_num : real;
     variable rnd_num_v1 : std_logic_vector(N-1 downto 0);
     variable rnd_num_v2 : std_logic_vector(N-1 downto 0);
     variable rnd_num_v3 : std_logic_vector(N-1 downto 0);
     variable rnd_num_v4 : std_logic_vector(N-1 downto 0);
     variable t,ft1,ft2 : integer :=0;

     begin
          A_Re <= CONV_STD_LOGIC_VECTOR(0,N);
          A_Im <= CONV_STD_LOGIC_VECTOR(0,N);
          B_Re <= CONV_STD_LOGIC_VECTOR(0,N);
          B_Im <= CONV_STD_LOGIC_VECTOR(0,N);
          clear <= '0';
```

```
          wait until (clk'event) and (clk='1');
          clear <= '1';

          while 1>0 loop
               t:=t+1;
               ft1 := integer(calc_f(t));
               ft2 := integer(calc_f(t+2**(LOG2NP-1)));
               UNIFORM(seed1,seed2,rnd_num);
               rnd_num_v1 :=
conv_std_logic_vector(integer(real(ft1)*(1.0+0.00*rnd_num*2.0**(0))),N);
               A_Re <= rnd_num_v1;
               UNIFORM(seed1,seed2,rnd_num);
               rnd_num_v2 :=
conv_std_logic_vector(integer(real(0)*(1.0+0.00*rnd_num*2.0**(0))),N);
               A_Im <= rnd_num_v2;
               UNIFORM(seed1,seed2,rnd_num);
               rnd_num_v3 :=
conv_std_logic_vector(integer(real(ft2)*(1.0+0.00*rnd_num*2.0**(0))),N);
               B_Re <= rnd_num_v3;
               UNIFORM(seed1,seed2,rnd_num);
               rnd_num_v4 :=
conv_std_logic_vector(integer(real(0)*(1.0+0.00*rnd_num*2.0**(0))),N);
               B_Im <= rnd_num_v4;
               wait until (clk'event) and (clk='1');
          end loop;
     end process;

     OUTPUT_Checking:process
          type N_Matrix is array (NATURAL range <>) of std_logic_vector(N-1 downto
0);

          constant FFT_DELAY : integer := (LOG2NP-1)*(N+N+NW)+(N)+(2**(LOG2NP-1)-1);

          variable A_Re_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));
          variable A_Im_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));
          variable B_Re_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));
          variable B_Im_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));

          variable report_line : line;
          variable report_str : string(1 to 160);
          variable i,j,re_part, im_part,temp_part : integer := 0;


     begin

          wait until (clk'event) and (clk='1') and (clear='1');

          for i in 0 to FFT_DELAY-1 loop
               A_Re_Matrix(i) := A_Re_Matrix(i+1);
               A_Im_Matrix(i) := A_Im_Matrix(i+1);
               B_Re_Matrix(i) := B_Re_Matrix(i+1);
               B_Im_Matrix(i) := B_Im_Matrix(i+1);
          end loop;
          A_Re_Matrix(FFT_DELAY) := A_Re;
          A_Im_Matrix(FFT_DELAY) := A_Im;
          B_Re_Matrix(FFT_DELAY) := B_Re;
          B_Im_Matrix(FFT_DELAY) := B_Im;

          report_line:=null;

          write(report_line,conv_integer(A_Re_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(A_Im_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(B_Re_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(B_Im_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(AO_Re));
          write(report_line,string'(","));
```

```
          write(report_line,conv_integer(AO_Im));
          write(report_line,string'(","));
          write(report_line,conv_integer(BO_Re));
          write(report_line,string'(","));
          write(report_line,conv_integer(BO_Im));
          write(report_line,string'("       "));

          for i in report_line'range loop
               report_str(i) := report_line(i);
          end loop;
          report report_str;

          writeline(OUTPUT_FILE,report_line);
     end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_fft_unit of fft_unit_tb is
     for TB_ARCHITECTURE
          for UUT : fft_unit
               use entity work.fft_unit(fft_unit);
          end for;
     end for;
end TESTBENCH_FOR_fft_unit;
```

Ε. Αλλαγές στην περιγραφή σε VHDL του FFT με τον συνδυασμό μιγαδικού αφαιρέτη-πολλαπλασιαστή

Ε.1 Μιγαδικός αθροιστής

Έχουμε τρεις βασικές περιγραφές για τον μιγαδικό αθροιστή

Complex_Adder2.vhd	Μιγαδικός αθροιστής με στοιχεία καθυστέρησης μόνο στην είσοδο, ο οποίος χρησιμοποιείται στην πρώτη μονάδα butterfly
Complex_Adder3.vhd	Μιγαδικός αθροιστής χωρίς στοιχεία καθυστέρησης, ο οποίος χρησιμοποιείται στις ενδιάμεσες μονάδες butterfly
Complex_Adder4.vhd	Μιγαδικός αθροιστής με στοιχεία καθυστέρησης μόνο στην έξοδο, ο οποίος χρησιμοποιείται στην τελευταία μονάδα butterfly

Complex_Adder2.vhd

```
-- Complex Adder with delays in input and output is in bit skew form

library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Adder2 is
     generic    (
          N : POSITIVE := 4
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          s_im: out STD_LOGIC_VECTOR (N downto 0);
          s_re: out STD_LOGIC_VECTOR (N downto 0)
     );
end Complex_Adder2;

architecture Complex_Adder2_Structural of Complex_Adder2 is

component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
```

```
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CA_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

signal     a_im,a_re,b_im,b_re : CA_word_array(0 to N);
signal s2_im,s2_re : STD_LOGIC_VECTOR(N-1 downto 0);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(0 to N);
signal carry_delay_re,carry_delay_im : STD_LOGIC;

begin
          -- CFA generation
CFAx:for i in 0 to N-1 generate
CFA:      Complex_full_adder port map(
                    a_im => a_im(i)(i),
                    a_re => a_re(i)(i),
                    b_im => b_im(i)(i),
                    b_re => b_re(i)(i),
                    c_im_in => c_im_in(i),
                    c_re_in => c_re_in(i),
                    c_im_out => c_im_out(i),
                    c_re_out => c_re_out(i),
                    s_im => s2_im(i),
                    s_re => s2_re(i)
               );
          end generate;

row1:for i in 0 to N-1 generate
col1:     for j in 0 to N-1 generate
                    -- input Dff generation
cond1:              if j>i and j<N-1 generate
gda_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_re(i)(j),
                              q => a_re(i+1)(j)
                         );
gda_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_im(i)(j),
                              q => a_im(i+1)(j)
                         );
gdb_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_re(i)(j),
                              q => b_re(i+1)(j)
                         );
gdb_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_im(i)(j),
                              q => b_im(i+1)(j)
                         );
                    end generate;
cond2:              if j=N-1 generate
gda_re1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
```

```
                              clear => clear,
                              clk => clk,
                              d => a_re(i)(j),
                              q => a_re(i+1)(j)
                         );
gda_im1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => a_im(i)(j),
                              q => a_im(i+1)(j)
                         );
gdb_re1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_re(i)(j),
                              q => b_re(i+1)(j)
                         );
gdb_im1:            dff generic map(
                              CLEARQ => '1'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => b_im(i)(j),
                              q => b_im(i+1)(j)
                         );
                    end generate;
                    -- output Dff generation
cond3:              if j=i generate
gds_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => s2_re(j),
                              q => s_re(j)
                         );
gds_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => s2_im(j),
                              q => s_im(j)
                         );
                    end generate;
                    -- carry Dff generation
cond4:              if i=j and i<N-1 generate
gdc_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => c_re_out(i),
                              q => c_re_in(i+1)
                         );
gdc_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => c_im_out(i),
                              q => c_im_in(i+1)
                         );
                    end generate;

               end generate;
```

```
          end generate;
          -- MSB carry Dff generation
gdc_re1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_re_out(N-1),
               q => c_re_out(N)
          );
gdc_im1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_im_out(N-1),
               q => c_im_out(N)
          );

--carry out delay for 1 clock cycle to syhronize with next unit
gds_re2:dff generic map(
         CLEARQ => '1'
       ) port map(
         clear => clear,
         clk => clk,
         d => c_re_out(N),
         q => carry_delay_re
       );
gds_im2:dff generic map(
         CLEARQ => '1'
       ) port map(
         clear => clear,
         clk => clk,
         d => c_im_out(N),
         q => carry_delay_im
       );

-- input wires generation
     a_re(0) <= (not x_re(N-1))&x_re(N-2 downto 0);
     a_im(0) <= (not x_im(N-1))&x_im(N-2 downto 0);
     b_re(0) <= (not y_re(N-1))&y_re(N-2 downto 0);
     b_im(0) <= (not y_im(N-1))&y_im(N-2 downto 0);
     c_re_in(0) <= '0';
     c_im_in(0) <= '0';

-- output wires generation
      s_re(N) <= not carry_delay_re;
      s_im(N) <= not carry_delay_im;

end Complex_Adder2_Structural;
```

Complex_Adder3.vhd

```
-- Comlplex Adder with input and output in bit skew form

library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Adder3 is
     generic    (
          N : POSITIVE := 4
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
```

```
          y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          s_im: out STD_LOGIC_VECTOR (N downto 0);
          s_re: out STD_LOGIC_VECTOR (N downto 0)
     );
end Complex_Adder3;

architecture Complex_Adder3_Structural of Complex_Adder3 is

component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CA_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

--signal   a_im,a_re,b_im,b_re,s2_im,s2_re : CA_word_array(0 to N);
signal     a_im,a_re,b_im,b_re,s2_im,s2_re : STD_LOGIC_VECTOR(N-1 downto 0);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(N downto 0);
signal carry_delay_re,carry_delay_im : STD_LOGIC;

begin
          -- CFA generation
CFAx:for i in 0 to N-1 generate
CFA:      Complex_full_adder port map(
                    a_im => a_im(i),
                    a_re => a_re(i),
                    b_im => b_im(i),
                    b_re => b_re(i),
                    c_im_in => c_im_in(i),
                    c_re_in => c_re_in(i),
                    c_im_out => c_im_out(i),
                    c_re_out => c_re_out(i),
                    s_im => s2_im(i),
                    s_re => s2_re(i)
               );
          end generate;

col1:for j in 0 to N-1 generate
               -- output Dff generation
gds_re0:  dff generic map(
                    CLEARQ => '0'
               ) port map(
                    clear => clear,
                    clk => clk,
                    d => s2_re(j),
                    q => s_re(j)
               );
gds_im0:  dff generic map(
```

```
                    CLEARQ => '0'
                ) port map(
                    clear => clear,
                    clk => clk,
                    d => s2_im(j),
                    q => s_im(j)
                );

                -- carry Dff generation
gdc_re0:   dff generic map(
                    CLEARQ => '0'
                ) port map(
                    clear => clear,
                    clk => clk,
                    d => c_re_out(j),
                    q => c_re_in(j+1)
                );
gdc_im0:   dff generic map(
                    CLEARQ => '0'
                ) port map(
                    clear => clear,
                    clk => clk,
                    d => c_im_out(j),
                    q => c_im_in(j+1)
                );
           end generate;

           -- MSB carry Dff generation
gdc_re1:dff generic map(
                CLEARQ => '1'
           ) port map(
                clear => clear,
                clk => clk,
                d => c_re_out(N-1),
                q => c_re_out(N)
           );
gdc_im1:dff generic map(
                CLEARQ => '1'
           ) port map(
                clear => clear,
                clk => clk,
                d => c_im_out(N-1),
                q => c_im_out(N)
           );

--carry out delay for 1 clock cycle to syhronize with next unit
gds_re2:dff generic map(
          CLEARQ => '1'
        ) port map(
          clear => clear,
          clk => clk,
          d => c_re_out(N),
          q => carry_delay_re
        );
gds_im2:dff generic map(
          CLEARQ => '1'
        ) port map(
          clear => clear,
          clk => clk,
          d => c_im_out(N),
          q => carry_delay_im
        );

-- input wires generation
     a_re <= (not x_re(N-1))&x_re(N-2 downto 0);
     a_im <= (not x_im(N-1))&x_im(N-2 downto 0);
     b_re <= (not y_re(N-1))&y_re(N-2 downto 0);
     b_im <= (not y_im(N-1))&y_im(N-2 downto 0);
     c_re_in(0) <= '0';
     c_im_in(0) <= '0';
```

```
-- output MSB
     s_re(N) <= not carry_delay_re;
     s_im(N) <= not carry_delay_im;

end Complex_Adder3_Structural;
```

Complex_Adder4.vhd

```
-- Complex Adder with output delays and input in bit skew form

library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Adder4 is
     generic    (
          N : POSITIVE := 4
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          s_im: out STD_LOGIC_VECTOR (N downto 0);
          s_re: out STD_LOGIC_VECTOR (N downto 0)
     );
end Complex_Adder4;

architecture Complex_Adder4_Structural of Complex_Adder4 is

component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CA_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

signal     a_im,a_re,b_im,b_re,s2_im,s2_re : CA_word_array(0 to N);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(0 to N);

begin
          -- CFA generation
CFAx:for i in 0 to N-1 generate
CFA:       Complex_full_adder port map(
```

```
                    a_im => a_im(i)(i),
                    a_re => a_re(i)(i),
                    b_im => b_im(i)(i),
                    b_re => b_re(i)(i),
                    c_im_in => c_im_in(i),
                    c_re_in => c_re_in(i),
                    c_im_out => c_im_out(i),
                    c_re_out => c_re_out(i),
                    s_im => s2_im(i)(i),
                    s_re => s2_re(i)(i)
               );
          end generate;

row1:for i in 0 to N-1 generate
col1:     for j in 0 to N-1 generate
                    -- input Dff generation none

                    -- output Dff generation
cond3:              if j<=i generate
gds_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => s2_re(i)(j),
                              q => s2_re(i+1)(j)
                         );
gds_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => s2_im(i)(j),
                              q => s2_im(i+1)(j)
                         );
                    end generate;
                    -- carry Dff generation
cond4:              if i=j and i<N-1 generate
gdc_re0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => c_re_out(i),
                              q => c_re_in(i+1)
                         );
gdc_im0:            dff generic map(
                              CLEARQ => '0'
                         ) port map(
                              clear => clear,
                              clk => clk,
                              d => c_im_out(i),
                              q => c_im_in(i+1)
                         );
                    end generate;

               end generate;
          end generate;
          -- MSB carry Dff generation
gdc_re1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_re_out(N-1),
               q => c_re_out(N)
          );
gdc_im1:dff generic map(
               CLEARQ => '1'
          ) port map(
```

```
                clear => clear,
                clk => clk,
                d => c_im_out(N-1),
                q => c_im_out(N)
            );

-- input wires generation
     a_re(N-1)(N-1) <= not x_re(N-1);
     a_im(N-1)(N-1) <= not x_im(N-1);
     b_re(N-1)(N-1) <= not y_re(N-1);
     b_im(N-1)(N-1) <= not y_im(N-1);
inp:for j in N-2 downto 0 generate
           a_re(j)(j) <= x_re(j);
           a_im(j)(j) <= x_im(j);
           b_re(j)(j) <= y_re(j);
           b_im(j)(j) <= y_im(j);
     end generate;
     c_re_in(0) <= '0';
     c_im_in(0) <= '0';

-- output wires generation
     s_re <= (not c_re_out(N))&s2_re(N);
     s_im <= (not c_im_out(N))&s2_im(N);

end Complex_Adder4_Structural;
```

E.2 Μιγαδικός αφαιρέτης

Έχουμε τρεις βασικές περιγραφές για τον μιγαδικό αφαιρέτη

Complex_Subtracter2.vhd	Μιγαδικός αφαιρέτης με στοιχεία καθυστέρησης μόνο στην είσοδο, ο οποίος χρησιμοποιείται στην πρώτη μονάδα butterfly
Complex_ Subtracter3.vhd	Μιγαδικός αφαιρέτης χωρίς στοιχεία καθυστέρησης, ο οποίος χρησιμοποιείται στις ενδιάμεσες μονάδες butterfly
Complex_ Subtracter4.vhd	Μιγαδικός αφαιρέτης με στοιχεία καθυστέρησης μόνο στην έξοδο, ο οποίος χρησιμοποιείται στην τελευταία μονάδα butterfly

Complex_Subtracter2.vhd

```
-- Complex Subtracter with delays in input and output is in bit skew form

library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Subtracter2 is
    generic    (
       N : POSITIVE := 4
    );
    port (
       clk: in STD_LOGIC;
       clear: in STD_LOGIC;
       x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       d_im: out STD_LOGIC_VECTOR (N downto 0);
       d_re: out STD_LOGIC_VECTOR (N downto 0)
    );
end Complex_Subtracter2;

architecture Complex_Subtracter2_Structural of Complex_Subtracter2 is
```

```
component dff is
    generic (
       CLEARQ : STD_LOGIC := '0'
    );
    port (
       clear: in STD_LOGIC;
       clk: in STD_LOGIC;
       d: in STD_LOGIC;
       q: out STD_LOGIC
    );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CS_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

signal  a_im,a_re,b_im,b_re,d2_im,d2_re : CS_word_array(0 to N);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(0 to N);
signal carry_delay_re,carry_delay_im : STD_LOGIC;

begin
       -- CFA generation
CFAx:   for i in 0 to N-1 generate
CFA:       Complex_full_adder port map(
               a_im => a_im(i)(i),
               a_re => a_re(i)(i),
               b_im => b_im(i)(i),
               b_re => b_re(i)(i),
               c_im_in => c_im_in(i),
               c_re_in => c_re_in(i),
               c_im_out => c_im_out(i),
               c_re_out => c_re_out(i),
               s_im => d2_im(i)(i),
               s_re => d2_re(i)(i)
         );
       end generate;

row1:   for i in 0 to N-1 generate
col1:     for j in 0 to N-1 generate
          -- input Dff generation
cond1:       if j>i and j<N-1 generate
gda_re0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => a_re(i)(j),
                 q => a_re(i+1)(j)
              );
gda_im0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => a_im(i)(j),
                 q => a_im(i+1)(j)
              );
```

```
gdb_re0:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => b_re(i)(j),
                 q => b_re(i+1)(j)
              );
gdb_im0:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => b_im(i)(j),
                 q => b_im(i+1)(j)
              );
           end generate;
cond2:        if j=N-1 generate
gda_re1:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => a_re(i)(j),
                 q => a_re(i+1)(j)
              );
gda_im1:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => a_im(i)(j),
                 q => a_im(i+1)(j)
              );
gdb_re1:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => b_re(i)(j),
                 q => b_re(i+1)(j)
              );
gdb_im1:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => b_im(i)(j),
                 q => b_im(i+1)(j)
              );
           end generate;
           -- output Dff generation
cond3:     if j=i generate
gds_re0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => d2_re(i)(j),
                 q => d2_re(i+1)(j)
              );
gds_im0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => d2_im(i)(j),
                 q => d2_im(i+1)(j)
              );
           end generate;
```

```
          -- carry out-in Dff generation
cond4:       if i=j and i<N-1 generate
gds_re0:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => c_re_out(i),
                 q => c_re_in(i+1)
              );
gds_im0:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => c_im_out(i),
                 q => c_im_in(i+1)
              );
           end generate;

         end generate;
          end generate;
          -- carry out Dff generation
gds_re1:dff generic map(
          CLEARQ => '1'
       ) port map(
          clear => clear,
          clk => clk,
          d => c_re_out(N-1),
          q => c_re_out(N)
       );
gds_im1:dff generic map(
          CLEARQ => '1'
       ) port map(
          clear => clear,
          clk => clk,
          d => c_im_out(N-1),
          q => c_im_out(N)
       );

--carry out delay for 1 clock cycle to syhronize with complex multiplier
gds_re2:dff generic map(
          CLEARQ => '1'
       ) port map(
          clear => clear,
          clk => clk,
          d => c_re_out(N),
          q => carry_delay_re
       );
gds_im2:dff generic map(
          CLEARQ => '1'
       ) port map(
          clear => clear,
          clk => clk,
          d => c_im_out(N),
          q => carry_delay_im
       );

-- input wires generation
    a_re(0) <= (not x_re(N-1))&x_re(N-2 downto 0);
    a_im(0) <= (not x_im(N-1))&x_im(N-2 downto 0);
    b_re(0) <= y_re(N-1)&(not y_re(N-2 downto 0));
    b_im(0) <= y_im(N-1)&(not y_im(N-2 downto 0));
    c_re_in(0) <= '1';
    c_im_in(0) <= '1';

-- output wires generation
      d_re(N) <= not carry_delay_re;
      d_im(N) <= not carry_delay_im;
col2:for j in N-1 downto 0 generate
```

```
          d_re(j)<=d2_re(j+1)(j);
          d_im(j)<=d2_im(j+1)(j);
     end generate;


end Complex_Subtracter2_Structural;
```

Complex_Subtracter3.vhd

```
-- Comlplex Subtracter with input and output in bit skew form

library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Subtracter3 is
     generic    (
          N : POSITIVE := 4
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
          d_im: out STD_LOGIC_VECTOR (N downto 0);
          d_re: out STD_LOGIC_VECTOR (N downto 0)
     );
end Complex_Subtracter3;

architecture Complex_Subtracter3_Structural of Complex_Subtracter3 is

component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
     );
end component;

component Complex_full_adder is
  port(
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

signal     a_im,a_re,b_im,b_re,d2_im,d2_re : STD_LOGIC_VECTOR(N-1 downto 0);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(N downto 0);
signal carry_delay_re,carry_delay_im : STD_LOGIC;
begin
          -- CFA generation
CFAx:for i in 0 to N-1 generate
CFA:       Complex_full_adder port map(
                    a_im => a_im(i),
                    a_re => a_re(i),
```

```
                    b_im => b_im(i),
                    b_re => b_re(i),
                    c_im_in => c_im_in(i),
                    c_re_in => c_re_in(i),
                    c_im_out => c_im_out(i),
                    c_re_out => c_re_out(i),
                    s_im => d2_im(i),
                    s_re => d2_re(i)
               );
          end generate;

col1:for j in 0 to N-1 generate
               -- output Dff generation
gds_re0:  dff generic map(
                    CLEARQ => '0'
               ) port map(
                    clear => clear,
                    clk => clk,
                    d => d2_re(j),
                    q => d_re(j)
               );
gds_im0:  dff generic map(
                    CLEARQ => '0'
               ) port map(
                    clear => clear,
                    clk => clk,
                    d => d2_im(j),
                    q => d_im(j)
               );

               -- carry Dff generation
gdc_re0:  dff generic map(
                    CLEARQ => '1'
               ) port map(
                    clear => clear,
                    clk => clk,
                    d => c_re_out(j),
                    q => c_re_in(j+1)
               );
gdc_im0:  dff generic map(
                    CLEARQ => '1'
               ) port map(
                    clear => clear,
                    clk => clk,
                    d => c_im_out(j),
                    q => c_im_in(j+1)
               );
          end generate;

          -- MSB carry Dff generation
gdc_re1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_re_out(N-1),
               q => c_re_out(N)
          );
gdc_im1:dff generic map(
               CLEARQ => '1'
          ) port map(
               clear => clear,
               clk => clk,
               d => c_im_out(N-1),
               q => c_im_out(N)
          );

--carry out delay for 1 clock cycle to syhronize with complex multiplier
gds_re2:dff generic map(
         CLEARQ => '1'
```

```
       ) port map(
         clear => clear,
         clk => clk,
         d => c_re_out(N),
         q => carry_delay_re
       );
gds_im2:dff generic map(
         CLEARQ => '1'
       ) port map(
         clear => clear,
         clk => clk,
         d => c_im_out(N),
         q => carry_delay_im
       );

-- input wires generation
     a_re <= (not x_re(N-1))&x_re(N-2 downto 0);
     a_im <= (not x_im(N-1))&x_im(N-2 downto 0);
     b_re <= y_re(N-1)&(not y_re(N-2 downto 0));
     b_im <= y_im(N-1)&(not y_im(N-2 downto 0));
     c_re_in(0) <= '1';
     c_im_in(0) <= '1';

-- output MSB
     d_re(N) <= not carry_delay_re;
     d_im(N) <= not carry_delay_im;

end Complex_Subtracter3_Structural;
```

Complex_Subtracter4.vhd

```
-- Complex Subtracter with output delays and input in bit skew form

library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Subtracter4 is
    generic    (
       N : POSITIVE := 4
    );
    port (
       clk: in STD_LOGIC;
       clear: in STD_LOGIC;
       x_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       x_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_im: in STD_LOGIC_VECTOR (N-1 downto 0);
       y_re: in STD_LOGIC_VECTOR (N-1 downto 0);
       d_im: out STD_LOGIC_VECTOR (N downto 0);
       d_re: out STD_LOGIC_VECTOR (N downto 0)
    );
end Complex_Subtracter4;

architecture Complex_Subtracter4_Structural of Complex_Subtracter4 is

component dff is
    generic (
       CLEARQ : STD_LOGIC := '0'
    );
    port (
       clear: in STD_LOGIC;
       clk: in STD_LOGIC;
       d: in STD_LOGIC;
       q: out STD_LOGIC
    );
end component;

component Complex_full_adder is
  port(
```

```
       a_im : in STD_LOGIC;
       a_re : in STD_LOGIC;
       b_im : in STD_LOGIC;
       b_re : in STD_LOGIC;
       c_im_in : in STD_LOGIC;
       c_re_in : in STD_LOGIC;
       c_im_out : out STD_LOGIC;
       c_re_out : out STD_LOGIC;
       s_im : out STD_LOGIC;
       s_re : out STD_LOGIC
  );
end component;

type CS_word_array is array(natural range <>) of STD_LOGIC_VECTOR (N-1 downto 0);

signal  a_im,a_re,b_im,b_re,d2_im,d2_re : CS_word_array(0 to N);
signal c_im_in,c_re_in,c_im_out,c_re_out : STD_LOGIC_VECTOR(0 to N);

begin
       -- CFA generation
CFAx:   for i in 0 to N-1 generate
CFA:       Complex_full_adder port map(
               a_im => a_im(i)(i),
               a_re => a_re(i)(i),
               b_im => b_im(i)(i),
               b_re => b_re(i)(i),
               c_im_in => c_im_in(i),
               c_re_in => c_re_in(i),
               c_im_out => c_im_out(i),
               c_re_out => c_re_out(i),
               s_im => d2_im(i)(i),
               s_re => d2_re(i)(i)
         );
       end generate;

row1:   for i in 0 to N-1 generate
col1:     for j in 0 to N-1 generate
            -- input Dff generation none

          -- output Dff generation
cond3:       if j<=i generate
gds_re0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => d2_re(i)(j),
                 q => d2_re(i+1)(j)
              );
gds_im0:         dff generic map(
                 CLEARQ => '0'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => d2_im(i)(j),
                 q => d2_im(i+1)(j)
              );
          end generate;
          -- carry out-in Dff generation
cond4:       if i=j and i<N-1 generate
gds_re0:         dff generic map(
                 CLEARQ => '1'
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => c_re_out(i),
                 q => c_re_in(i+1)
              );
gds_im0:         dff generic map(
                 CLEARQ => '1'
```

```
              ) port map(
                 clear => clear,
                 clk => clk,
                 d => c_im_out(i),
                 q => c_im_in(i+1)
              );
           end generate;

         end generate;
          end generate;
          -- carry out Dff generation
gds_re1:dff generic map(
         CLEARQ => '1'
       ) port map(
         clear => clear,
         clk => clk,
         d => c_re_out(N-1),
         q => c_re_out(N)
       );
gds_im1:dff generic map(
         CLEARQ => '1'
       ) port map(
         clear => clear,
         clk => clk,
         d => c_im_out(N-1),
         q => c_im_out(N)
       );

-- input wires generation
     a_re(N-1)(N-1) <= not x_re(N-1);
     a_im(N-1)(N-1) <= not x_im(N-1);
     b_re(N-1)(N-1) <= y_re(N-1);
     b_im(N-1)(N-1) <= y_im(N-1);
inp:for j in N-2 downto 0 generate
          a_re(j)(j) <= x_re(j);
          a_im(j)(j) <= x_im(j);
          b_re(j)(j) <= not y_re(j);
          b_im(j)(j) <= not y_im(j);
     end generate;
    c_re_in(0) <= '1';
    c_im_in(0) <= '1';

-- output wires generation
    d_re <= (not c_re_out(N))&d2_re(N);
    d_im <= (not c_im_out(N))&d2_im(N);

end Complex_Subtracter4_Structural;
```

E.3 Μιγαδικός πολλαπλασιαστής

Ο μιγαδικός πολλαπλασιαστής είναι αυτός που περιγράψαμε στο παράρτημα C αλλά με κάποιες τροποποιήσεις στον βασικό του κώδικα ο οποίος παρουσιάζετε ακολούθως και χρησιμοποιείται ο ίδιος σε κάθε μονάδα butterfly εκτός φυσικά απ' την τελευταία που δεν υπάρχει.

complex_multiplier.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Complex_Multiplier is
     generic    (
          NX : POSITIVE := 4;
          NY : POSITIVE := 5
     );
     port (
          clk: in STD_LOGIC;
          clear: in STD_LOGIC;
          x_im: in STD_LOGIC_VECTOR (NX-1 downto 0);
          x_re: in STD_LOGIC_VECTOR (NX-1 downto 0);
          y_im: in STD_LOGIC_VECTOR (NY-1 downto 0);
          y_re: in STD_LOGIC_VECTOR (NY-1 downto 0);
          m_im: out STD_LOGIC_VECTOR (NY-2 downto 0);
          m_re: out STD_LOGIC_VECTOR (NY-2 downto 0)
     );
end Complex_Multiplier;

--}} End of automatically maintained section

architecture Complex_Multiplier_Structural of Complex_Multiplier is
subtype x_word is STD_LOGIC_VECTOR (NX-1 downto 0);
subtype dy_word is STD_LOGIC_VECTOR (NY-1 downto 0);
subtype CFA_word is STD_LOGIC_VECTOR (NX-1 downto 0);
subtype dm_word is STD_LOGIC_VECTOR (NY+NX-1 downto 0);

type x_word_array is array(natural range <>) of x_word;
type y_word_array is array(natural range <>) of dy_word;
type CFA_word_array is array(natural range <>) of CFA_word;
type dm_word_array is array(natural range <>) of dm_word;

signal     c1_im_in,c1_re_in,c2_im_in,c2_re_in,
           s_im_in,s_re_in,x_im_in,x_re_in,y_im_in,y_re_in,
           c1_im_out,c1_re_out,c2_im_out,c2_re_out,
           s_im_out,s_re_out,x_im_out,x_re_out : x_word_array(0 to NY-1);

signal  dy_re,dy_im : y_word_array(0 to NY-1);

signal     a_im,a_re,b_im,b_re,c_im_in,
           c_re_in,c_im_out,c_re_out,s_im,s_re : CFA_word_array(NY to NY+NX-1);

signal  dm_re,dm_im : dm_word_array(0 to NY+NX-1);


component dff is
     generic (
          CLEARQ : STD_LOGIC := '0'
     );
     port (
          clear: in STD_LOGIC;
          clk: in STD_LOGIC;
          d: in STD_LOGIC;
          q: out STD_LOGIC
```

```
    );
end component;

component CMC1_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end component;

component CMC2_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
       c2_re_out : out STD_LOGIC;
       s_im_out : out STD_LOGIC;
       s_re_out : out STD_LOGIC;
       x_im_out : out STD_LOGIC;
       x_re_out : out STD_LOGIC
  );
end component;

component CMC3_cell is
  port(
       c1_im_in : in STD_LOGIC;
       c1_re_in : in STD_LOGIC;
       c2_im_in : in STD_LOGIC;
       c2_re_in : in STD_LOGIC;
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       s_im_in : in STD_LOGIC;
       s_re_in : in STD_LOGIC;
       x_im_in : in STD_LOGIC;
       x_re_in : in STD_LOGIC;
       y_im_in : in STD_LOGIC;
       y_re_in : in STD_LOGIC;
       c1_im_out : out STD_LOGIC;
       c1_re_out : out STD_LOGIC;
       c2_im_out : out STD_LOGIC;
```

```
      c2_re_out : out STD_LOGIC;
      s_im_out : out STD_LOGIC;
      s_re_out : out STD_LOGIC;
      x_im_out : out STD_LOGIC;
      x_re_out : out STD_LOGIC
  );
end component;

component CMC4_cell is
  port(
      c1_im_in : in STD_LOGIC;
      c1_re_in : in STD_LOGIC;
      c2_im_in : in STD_LOGIC;
      c2_re_in : in STD_LOGIC;
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      s_im_in : in STD_LOGIC;
      s_re_in : in STD_LOGIC;
      x_im_in : in STD_LOGIC;
      x_re_in : in STD_LOGIC;
      y_im_in : in STD_LOGIC;
      y_re_in : in STD_LOGIC;
      c1_im_out : out STD_LOGIC;
      c1_re_out : out STD_LOGIC;
      c2_im_out : out STD_LOGIC;
      c2_re_out : out STD_LOGIC;
      s_im_out : out STD_LOGIC;
      s_re_out : out STD_LOGIC;
      x_im_out : out STD_LOGIC;
      x_re_out : out STD_LOGIC
  );
end component;

component CFA1_cell is
  port(
      a_im : in STD_LOGIC;
      a_re : in STD_LOGIC;
      b_im : in STD_LOGIC;
      b_re : in STD_LOGIC;
      c_im_in : in STD_LOGIC;
      c_re_in : in STD_LOGIC;
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      c_im_out : out STD_LOGIC;
      c_re_out : out STD_LOGIC;
      s_im : out STD_LOGIC;
      s_re : out STD_LOGIC
  );
end component;

component CFA2_cell is
  port(
      a_im : in STD_LOGIC;
      a_re : in STD_LOGIC;
      b_im : in STD_LOGIC;
      b_re : in STD_LOGIC;
      c_im_in : in STD_LOGIC;
      c_re_in : in STD_LOGIC;
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      c_im_out : out STD_LOGIC;
      c_re_out : out STD_LOGIC;
      s_im : out STD_LOGIC;
      s_re : out STD_LOGIC
  );
end component;

begin
  -- <<enter your statements here>>
```

```
  -- cell generation
          -- CMC cell generation
row1:for i in 0 to NY-2 generate
                -- CMC1 cell generation
col1:      for j in NX-2 downto 0 generate
CMC1:           CMC1_cell port map (
                       c1_im_in => c1_im_in(i)(j),
                       c1_re_in => c1_re_in(i)(j),
                       c2_im_in => c2_im_in(i)(j),
                       c2_re_in => c2_re_in(i)(j),
                       clear => clear,
                       clk => clk,
                       s_im_in => s_im_in(i)(j),
                       s_re_in => s_re_in(i)(j),
                       x_im_in => x_im_in(i)(j),
                       x_re_in => x_re_in(i)(j),
                       y_im_in => y_im_in(i)(j),
                       y_re_in => y_re_in(i)(j),
                       c1_im_out => c1_im_out(i)(j),
                       c1_re_out => c1_re_out(i)(j),
                       c2_im_out => c2_im_out(i)(j),
                       c2_re_out => c2_re_out(i)(j),
                       s_im_out => s_im_out(i)(j),
                       s_re_out => s_re_out(i)(j),
                       x_im_out => x_im_out(i)(j),
                       x_re_out => x_re_out(i)(j)
                     );
                end generate;
                -- CMC2_cell generation ( for X MSB )
CMC2:      CMC2_cell port map (
                  c1_im_in => c1_im_in(i)(NX-1),
                  c1_re_in => c1_re_in(i)(NX-1),
                  c2_im_in => c2_im_in(i)(NX-1),
                  c2_re_in => c2_re_in(i)(NX-1),
                  clear => clear,
                  clk => clk,
                  s_im_in => s_im_in(i)(NX-1),
                  s_re_in => s_re_in(i)(NX-1),
                  x_im_in => x_im_in(i)(NX-1),
                  x_re_in => x_re_in(i)(NX-1),
                  y_im_in => y_im_in(i)(NX-1),
                  y_re_in => y_re_in(i)(NX-1),
                  c1_im_out => c1_im_out(i)(NX-1),
                  c1_re_out => c1_re_out(i)(NX-1),
                  c2_im_out => c2_im_out(i)(NX-1),
                  c2_re_out => c2_re_out(i)(NX-1),
                  s_im_out => s_im_out(i)(NX-1),
                  s_re_out => s_re_out(i)(NX-1),
                  x_im_out => x_im_out(i)(NX-1),
                  x_re_out => x_re_out(i)(NX-1)
                );
          end generate;

          -- CMC3_cell generation (for Y MSB)
col2:for j in NX-2 downto 0 generate
CMC3:      CMC3_cell port map (
                  c1_im_in => c1_im_in(NY-1)(j),
                  c1_re_in => c1_re_in(NY-1)(j),
                  c2_im_in => c2_im_in(NY-1)(j),
                  c2_re_in => c2_re_in(NY-1)(j),
                  clear => clear,
                  clk => clk,
                  s_im_in => s_im_in(NY-1)(j),
                  s_re_in => s_re_in(NY-1)(j),
                  x_im_in => x_im_in(NY-1)(j),
                  x_re_in => x_re_in(NY-1)(j),
                  y_im_in => y_im_in(NY-1)(j),
                  y_re_in => y_re_in(NY-1)(j),
                  c1_im_out => c1_im_out(NY-1)(j),
                  c1_re_out => c1_re_out(NY-1)(j),
```

```
                c2_im_out => c2_im_out(NY-1)(j),
                c2_re_out => c2_re_out(NY-1)(j),
                s_im_out => s_im_out(NY-1)(j),
                s_re_out => s_re_out(NY-1)(j),
                x_im_out => x_im_out(NY-1)(j),
                x_re_out => x_re_out(NY-1)(j)
             );
        end generate;

        -- CMC4_cell generation ( for X and Y MSB )
CMC4:CMC4_cell port map (
                c1_im_in => c1_im_in(NY-1)(NX-1),
                c1_re_in => c1_re_in(NY-1)(NX-1),
                c2_im_in => c2_im_in(NY-1)(NX-1),
                c2_re_in => c2_re_in(NY-1)(NX-1),
                clear => clear,
                clk => clk,
                s_im_in => s_im_in(NY-1)(NX-1),
                s_re_in => s_re_in(NY-1)(NX-1),
                x_im_in => x_im_in(NY-1)(NX-1),
                x_re_in => x_re_in(NY-1)(NX-1),
                y_im_in => y_im_in(NY-1)(NX-1),
                y_re_in => y_re_in(NY-1)(NX-1),
                c1_im_out => c1_im_out(NY-1)(NX-1),
                c1_re_out => c1_re_out(NY-1)(NX-1),
                c2_im_out => c2_im_out(NY-1)(NX-1),
                c2_re_out => c2_re_out(NY-1)(NX-1),
                s_im_out => s_im_out(NY-1)(NX-1),
                s_re_out => s_re_out(NY-1)(NX-1),
                x_im_out => x_im_out(NY-1)(NX-1),
                x_re_out => x_re_out(NY-1)(NX-1)
        );

        -- CFA_cell generation

        -- CFA1 type cell generation
col3:for j in NX-1 downto 0 generate
CFA1:      CFA1_cell port map (
                    a_im => a_im(NY)(j),
                    a_re => a_re(NY)(j),
                    b_im => b_im(NY)(j),
                    b_re => b_re(NY)(j),
                    c_im_in => c_im_in(NY)(j),
                    c_re_in => c_re_in(NY)(j),
                    clear => clear,
                    clk => clk,
                    c_im_out => c_im_out(NY)(j),
                    c_re_out => c_re_out(NY)(j),
                    s_im => s_im(NY)(j),
                    s_re => s_re(NY)(j)
             );
        end generate;

        -- CFA2 type cell generation
row4:for i in 1 to NX-1 generate
CFA2:      CFA2_cell port map (
                    a_im => a_im(NY+i)(i),
                    a_re => a_re(NY+i)(i),
                    b_im => b_im(NY+i)(i),
                    b_re => b_re(NY+i)(i),
                    c_im_in => c_im_in(NY+i)(i),
                    c_re_in => c_re_in(NY+i)(i),
                    clear => clear,
                    clk => clk,
                    c_im_out => c_im_out(NY+i)(i),
                    c_re_out => c_re_out(NY+i)(i),
                    s_im => s_im(NY+i)(i),
                    s_re => s_re(NY+i)(i)
             );
        end generate;
```

```
          -- CFA intermediate dff generation and connection
row5:for i in 1 to NX-2 generate
col5:     for j in NX-1 downto i+1 generate
                  -- dff generation for CFA a inputs
gdCFAare:       dff generic map (
                        CLEARQ => '0'
                  ) port map (
                        clear => clear,
                        clk => clk,
                        d => a_re(NY+i)(j),
                        q => a_re(NY+i+1)(j)
                  );
gdCFAaim:       dff generic map (
                        CLEARQ => '0'
                  ) port map (
                        clear => clear,
                        clk => clk,
                        d => a_im(NY+i)(j),
                        q => a_im(NY+i+1)(j)
                  );
                  -- dff generation for CFA b inputs
cond1:                  if j<NX generate
gdCFAbre:         dff generic map (
                              CLEARQ => '1'
                        ) port map (
                              clear => clear,
                              clk => clk,
                              d => b_re(NY+i)(j),
                              q => b_re(NY+i+1)(j)
                        );
gdCFAbim:         dff generic map (
                              CLEARQ => '1'
                        ) port map (
                              clear => clear,
                              clk => clk,
                              d => b_im(NY+i)(j),
                              q => b_im(NY+i+1)(j)
                        );
                  end generate;
               end generate;
          end generate;

          -- output m dff generation
row7:for i in 1 to NY+NX-1 generate
col7:     for j in i-1 downto 0 generate
cond2:          if j /= NY generate
gdmre0:               dff generic map (
                              CLEARQ => '0'
                        ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_re(i-1)(j),
                              q => dm_re(i)(j)
                        );
gdmim0:               dff generic map (
                              CLEARQ => '0'
                        ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_im(i-1)(j),
                              q => dm_im(i)(j)
                        );
                  end generate;
cond3:            if j = NY generate
gdmre1:               dff generic map (
                              CLEARQ => '1'
                        ) port map (
                              clear => clear,
                              clk => clk,
```

```
                              d => dm_re(i-1)(j),
                              q => dm_re(i)(j)
                         );
gdmim1:                  dff generic map (
                              CLEARQ => '1'
                         ) port map (
                              clear => clear,
                              clk => clk,
                              d => dm_im(i-1)(j),
                              q => dm_im(i)(j)
                         );
                    end generate;
               end generate;
          end generate;

  -- intermediate wires generation
          -- CMC intermediate wires generation
row8:for i in 1 to NY-1 generate
               c1_im_in(i) <= c1_im_out(i-1);
               c1_re_in(i) <= c1_re_out(i-1);
               c2_im_in(i) <= c2_im_out(i-1);
               c2_re_in(i) <= c2_re_out(i-1);
               x_im_in(i) <= x_im_out(i-1);
               x_re_in(i) <= x_re_out(i-1);
               s_im_in(i)(NX-1 downto 0) <= '0'&s_im_out(i-1)(NX-1 downto 1);
               s_re_in(i)(NX-1 downto 0) <= '0'&s_re_out(i-1)(NX-1 downto 1);

          end generate;
row9:for i in 0 to NY-1 generate
               y_re_in(i)(NX-2 downto 0) <= (others => y_re_in(i)(NX-1));
               y_im_in(i)(NX-2 downto 0) <= (others => y_im_in(i)(NX-1));
          end generate;

          -- input y CMC from dy dff
row10:    for i in 0 to NY-1 generate
               y_im_in(i)(NX-1) <= dy_im(0)(i);
               y_re_in(i)(NX-1) <= dy_re(0)(i);
          end generate;

          -- CFA intermediate wires generation
          --inputs CFA from CMC
          b_re(NY)(NX-1 downto 0) <= '0'&s_re_out(NY-1)(NX-1 downto 1);
          b_im(NY)(NX-1 downto 0) <= '0'&s_im_out(NY-1)(NX-1 downto 1);
          a_re(NY)(NX-1 downto 0) <= c1_re_out(NY-1)(NX-1 downto 0);
          a_im(NY)(NX-1 downto 0) <= c1_im_out(NY-1)(NX-1 downto 0);
          c_re_in(NY)(NX-1 downto 0) <= c2_re_out(NY-1)(NX-1 downto 0);
          c_im_in(NY)(NX-1 downto 0) <= c2_im_out(NY-1)(NX-1 downto 0);
          -- from CFA row to CFA_dff intermediate wires generation
          b_re(NY+1)(NX-1 downto 1) <= s_re(NY)(NX-1 downto 1);
          b_im(NY+1)(NX-1 downto 1) <= s_im(NY)(NX-1 downto 1);
          a_re(NY+1)(NX-1 downto 1) <= c_re_out(NY)(NX-2 downto 0);
          a_im(NY+1)(NX-1 downto 1) <= c_im_out(NY)(NX-2 downto 0);
          -- carry CFA
          c_im_in(NY+1)(1)<=s_im(NY)(0);
          c_re_in(NY+1)(1)<=s_re(NY)(0);
row11:    for i in 2 to NX-1 generate
               c_im_in(NY+i)(i)<=c_im_out(NY+i-1)(i-1);
               c_re_in(NY+i)(i)<=c_re_out(NY+i-1)(i-1);
          end generate;


          -- output m dff intermediate wires generation
col8:for j in NY-1 downto 0 generate
               dm_re(j)(j) <= s_re_out(j)(0);
               dm_im(j)(j) <= s_im_out(j)(0);
          end generate;
col9:for j in NY+NX-1 downto NY generate
               dm_re(j)(j) <= s_re(j)(j-NY);
               dm_im(j)(j) <= s_im(j)(j-NY);
           end generate;
```

```
  -- input connections
          c1_im_in(0)(NX-1) <= '1';
          c1_im_in(0)(NX-2 downto 0) <= (others => '0');
          c1_re_in(0) <= (others => '1');
          c2_im_in(0)(NX-1) <= '1';
          c2_im_in(0)(NX-2 downto 0) <= (others => '0');
          c2_re_in(0) <= (others => '0');
          s_im_in(0) <= (others => '0');
          s_re_in(0) <= (others => '0');
row12:    for i in 1 to NY-1 generate
               s_im_in(i)(NX-1) <= '0';
               s_re_in(i)(NX-1) <= '0';
          end generate;
          x_im_in(0) <= x_im;
          x_re_in(0) <= x_re;
          dy_re(0) <= y_re;
          dy_im(0) <= y_im;

  -- output connections
col10:    for j in NY+NX-2 downto NX generate
cond4:         if j=NY generate
                    m_re(j-NX)<= (not dm_re(j)(j));
                    m_im(j-NX)<= (not dm_im(j)(j));
               end generate;
cond5:         if j/=NY generate
                    m_re(j-NX)<=dm_re(j)(j);
                    m_im(j-NX)<=dm_im(j)(j);
               end generate;
          end generate;

end Complex_Multiplier_Structural;
```

E.4 Μιγαδικός καταχωρητής ολίσθησης

Σε αυτήν την μονάδα δεν έχουμε κάποια αλλαγή και παραμένει η ίδια.

E.5 Μονάδα Butterfly

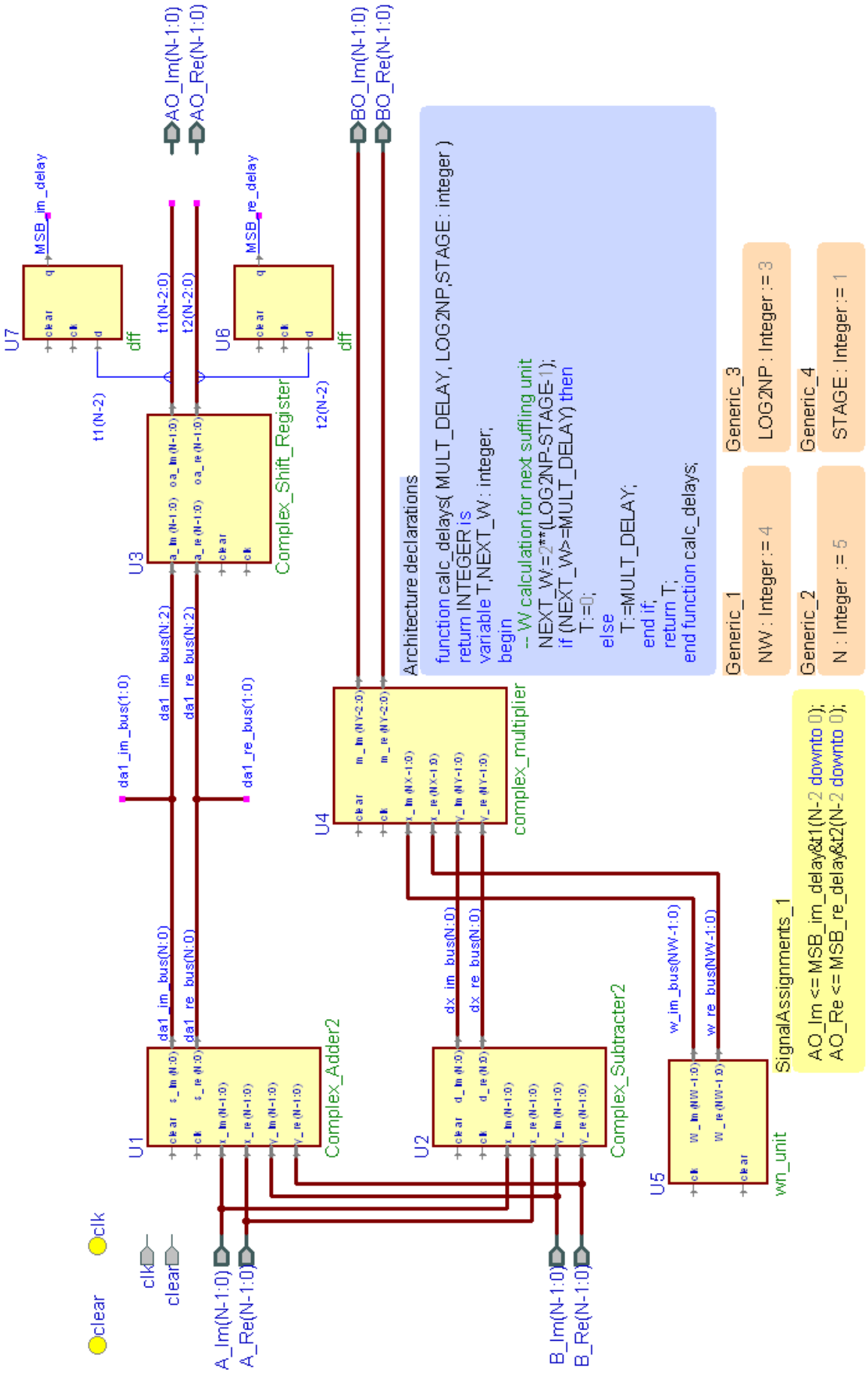
Ακολουθούν τα αρχεία που τροποποιούνται σε αυτή την περιγραφή σε VHDL.

WN_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity WN_Unit is
  generic(
        N : Integer := 8;    -- bits for inputs on FFT
       NW : Integer := 8;    -- bits for twiddle factors
       LOG2NP : Integer := 3;  -- LOG2NP=log2(NP) NP=N-Point DFT;
       STAGE : Integer := 1       -- stage of butterfly where WN_Unit are
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       W_im : out STD_LOGIC_VECTOR(NW-1 downto 0);
       W_re : out STD_LOGIC_VECTOR(NW-1 downto 0)
  );
end WN_Unit;
```

```
architecture WN_Unit of WN_Unit is

---- Component declarations -----

component BU_Tff is
      generic (
          CLEAR_BIT : STD_LOGIC := '0'
      );
      port(
           T : in STD_LOGIC;
           clear : in STD_LOGIC;
           clk : in STD_LOGIC;
           Q : out STD_LOGIC
          );
end component;
component BU_addr_counter
  generic (
           N : Integer := 5;
           START : Integer := 7
  );
  port (
       clk : in STD_LOGIC;
       reset : in STD_LOGIC;
       addr : out INTEGER range 0 to 2**N-1
  );
end component;
component BU_Rom
  generic(
       NW : INTEGER := 8;
       LOG2NP : INTEGER := 3;
       STAGE : INTEGER := 1
  );
  port (
       Address : in INTEGER range 0 to 2**(LOG2NP-STAGE)-1;
       clk : in STD_LOGIC;
       Data_Im : out STD_LOGIC_VECTOR(0 to NW-1);
       Data_Re : out STD_LOGIC_VECTOR(0 to NW-1)
  );
end component;

---- Signal declarations used on the diagram ----

signal a : INTEGER;

function calc_counter_start( N,NW, LOG2NP, STAGE : integer )
return INTEGER is
variable T,W,S,SUM,
           MULT_DELAY,
           BUTTERFLY_DELAY : integer;
begin
     MULT_DELAY:=NW-1;
     BUTTERFLY_DELAY:=NW+2;
     W:=2**(LOG2NP-2);
     T:=0;
     SUM:=0;
     S:=2;
     while S<=STAGE loop
          if W>=MULT_DELAY then
               T:=T+1;
          end if;
          SUM:=SUM+W;
          W:=W/2;
          S:=S+1;
     end loop;
     T:=-(STAGE-1)*BUTTERFLY_DELAY-SUM+T*MULT_DELAY;
     return T;
end function calc_counter_start;

begin
```

```
----  Component instantiations  ----

U1 : BU_Rom
  generic map (
       NW => NW,
       LOG2NP => LOG2NP,
       STAGE => STAGE
  )
  port map(
       Address => a,
       Data_Im => W_im( NW-1 downto 0 ),
       Data_Re => W_re( NW-1 downto 0 ),
       clk => clk
  );

if1:  if LOG2NP-STAGE = 0 generate
           a<=0;
       end generate if1;
if2:  if LOG2NP-STAGE/=0 generate
U2 :   BU_addr_counter
          generic map (
               N => LOG2NP-STAGE,
               START => calc_counter_start(N,NW,LOG2NP,STAGE)
          )
          port map(
                 addr => a,
                 clk => clk,
                 reset => clear
          );
      end generate if2;

end WN_Unit;
```



Σχήμα Ε.1 Σχεδιάγραμμα της πρώτης μονάδας butterfly, της τροποποιημένης αρχιτεκτονικής, που βρίσκετε στο αρχείο First_Butterfly_Unit.bde

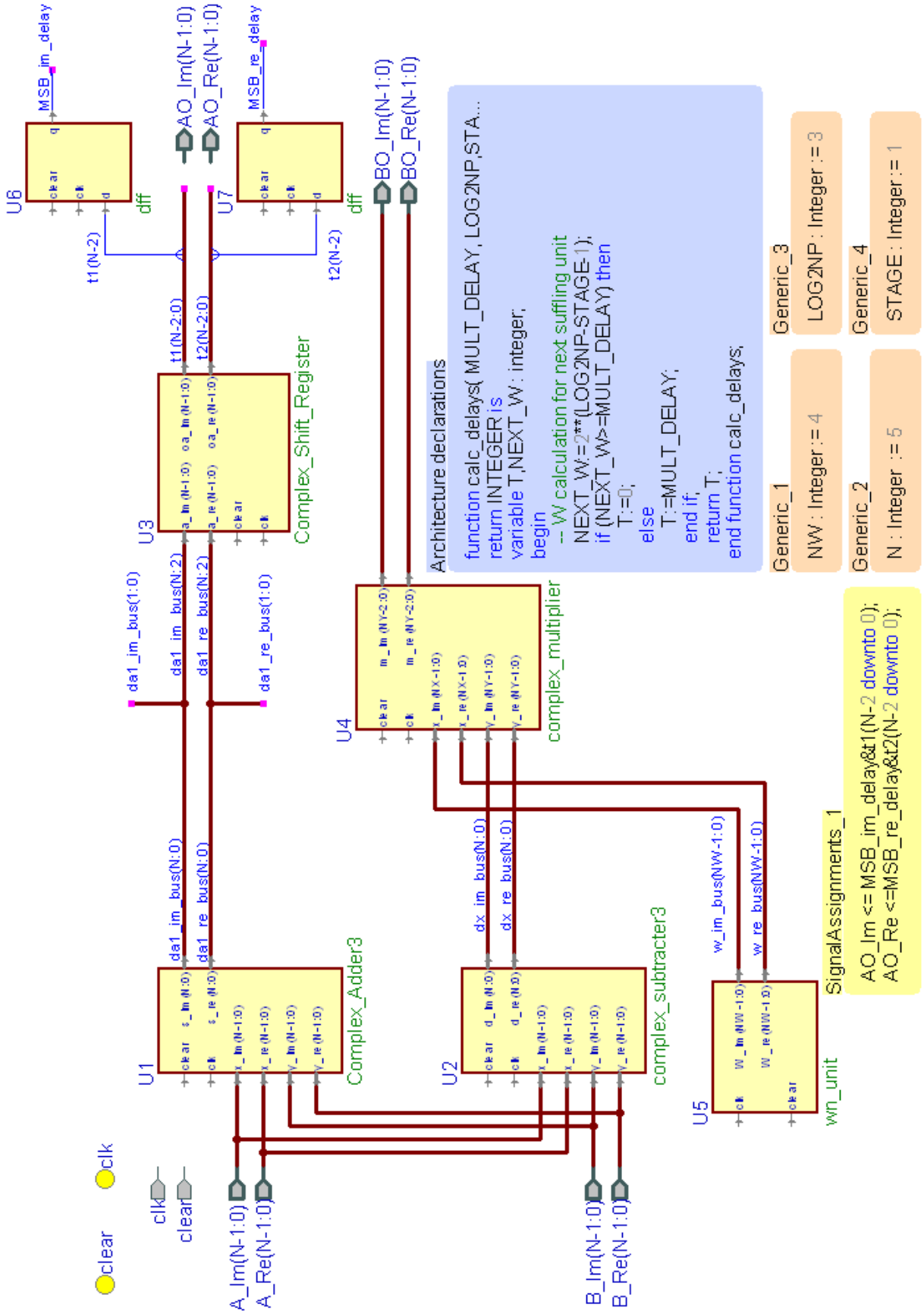
First_Butterfly_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


-- other libraries declarations
library COMPLEX_ADDER;
library COMPLEX_MULTIPLIER;
library COMPLEX_SHIFT_REGISTER;
library COMPLEX_SUBTRACTER;

entity First_Butterfly_Unit is
  generic(
       NW : Integer := 4;
       N : Integer  := 5;
       LOG2NP : Integer := 3;
       STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end First_Butterfly_Unit;

architecture First_Butterfly_Unit of First_Butterfly_Unit is

---- Component declarations -----

component dff
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component WN_Unit
  generic(
       LOG2NP : INTEGER := 3;
       N : INTEGER := 8;
       NW : INTEGER := 8;
       STAGE : INTEGER := 1
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       W_im : out STD_LOGIC_VECTOR(NW-1 downto 0);
       W_re : out STD_LOGIC_VECTOR(NW-1 downto 0)
  );
end component;
component Complex_Adder2
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
```

```
      x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      s_im : out STD_LOGIC_VECTOR(N downto 0);
      s_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;
component Complex_Multiplier
  generic(
      NX : POSITIVE := 4;
      NY : POSITIVE := 5
  );
  port (
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      x_im : in STD_LOGIC_VECTOR(NX-1 downto 0);
      x_re : in STD_LOGIC_VECTOR(NX-1 downto 0);
      y_im : in STD_LOGIC_VECTOR(NY-1 downto 0);
      y_re : in STD_LOGIC_VECTOR(NY-1 downto 0);
      m_im : out STD_LOGIC_VECTOR(NY-2 downto 0);
      m_re : out STD_LOGIC_VECTOR(NY-2 downto 0)
  );
end component;
component Complex_Shift_Register
  generic(
      N : INTEGER := 8;
      W : INTEGER := 2
  );
  port (
      a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      oa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
      oa_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component Complex_Subtracter2
  generic(
      N : POSITIVE := 4
  );
  port (
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      d_im : out STD_LOGIC_VECTOR(N downto 0);
      d_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;

---- Architecture declarations -----
function calc_delays( MULT_DELAY, LOG2NP,STAGE : integer )
return INTEGER is
variable T,NEXT_W : integer;
begin
   -- W calculation for next suffling unit
     NEXT_W:=2**(LOG2NP-STAGE-1);
     if (NEXT_W>=MULT_DELAY) then
          T:=0;
     else
          T:=MULT_DELAY;
     end if;
     return T;
end function calc_delays;
```

```
---- Signal declarations used on the diagram ----

signal MSB_im_delay : STD_LOGIC;
signal MSB_re_delay : STD_LOGIC;
signal da1_im_bus : STD_LOGIC_VECTOR (N downto 0);
signal da1_re_bus : STD_LOGIC_VECTOR (N downto 0);
signal dx_im_bus : STD_LOGIC_VECTOR (N downto 0);
signal dx_re_bus : STD_LOGIC_VECTOR (N downto 0);
signal t1 : STD_LOGIC_VECTOR (N-2 downto 0);
signal t2 : STD_LOGIC_VECTOR (N-2 downto 0);
signal w_im_bus : STD_LOGIC_VECTOR (NW-1 downto 0);
signal w_re_bus : STD_LOGIC_VECTOR (NW-1 downto 0);

---- Configuration specifications for declared components

for U1 : Complex_Adder2 use entity complex_adder.Complex_Adder2;
for U2 : Complex_Subtracter2 use entity complex_subtracter.Complex_Subtracter2;
for U3 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;
for U4 : complex_multiplier use entity complex_multiplier.complex_multiplier;

begin

---- User Signal Assignments ----
AO_Im <= MSB_im_delay&t1(N-2 downto 0);
AO_Re <= MSB_re_delay&t2(N-2 downto 0);

----  Component instantiations  ----

U1 : Complex_Adder2
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       s_im => da1_im_bus( N downto 0 ),
       s_re => da1_re_bus( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U2 : Complex_Subtracter2
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       d_im => dx_im_bus( N downto 0 ),
       d_re => dx_re_bus( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U3 : Complex_Shift_Register
  generic map (
       N => N-1,
       W => calc_delays(NW-1,LOG2NP,STAGE)
  )
  port map(
       a_im => da1_im_bus( N downto 2 ),
       a_re => da1_re_bus( N downto 2 ),
       clear => clear,
       clk => clk,
       oa_im => t1( N-2 downto 0 ),
```

```
        oa_re => t2( N-2 downto 0 )
  );

U4 : Complex_Multiplier
  generic map (
        NX => NW,
        NY => N+1
  )
  port map(
        clear => clear,
        clk => clk,
        m_im => BO_Im( N-1 downto 0 ),
        m_re => BO_Re( N-1 downto 0 ),
        x_im => w_im_bus( NW-1 downto 0 ),
        x_re => w_re_bus( NW-1 downto 0 ),
        y_im => dx_im_bus( N downto 0 ),
        y_re => dx_re_bus( N downto 0 )
  );

U5 : WN_Unit
  generic map (
        LOG2NP => LOG2NP,
        N => N,
        NW => NW,
        STAGE => STAGE
  )
  port map(
        W_im => w_im_bus( NW-1 downto 0 ),
        W_re => w_re_bus( NW-1 downto 0 ),
        clear => clear,
        clk => clk
  );

U6 : dff
  port map(
        clear => clear,
        clk => clk,
        d => t2(N-2),
        q => MSB_re_delay
  );

U7 : dff
  port map(
        clear => clear,
        clk => clk,
        d => t1(N-2),
        q => MSB_im_delay
  );


end First_Butterfly_Unit;
```



Σχήμα Ε.2 Σχεδιάγραμμα της τροποποιημένης μονάδας butterfly, που χρησιμοποιείται στις ενδιάμεσες μονάδες butterfly, και βρίσκετε στο αρχείο butterfly_unit.bde

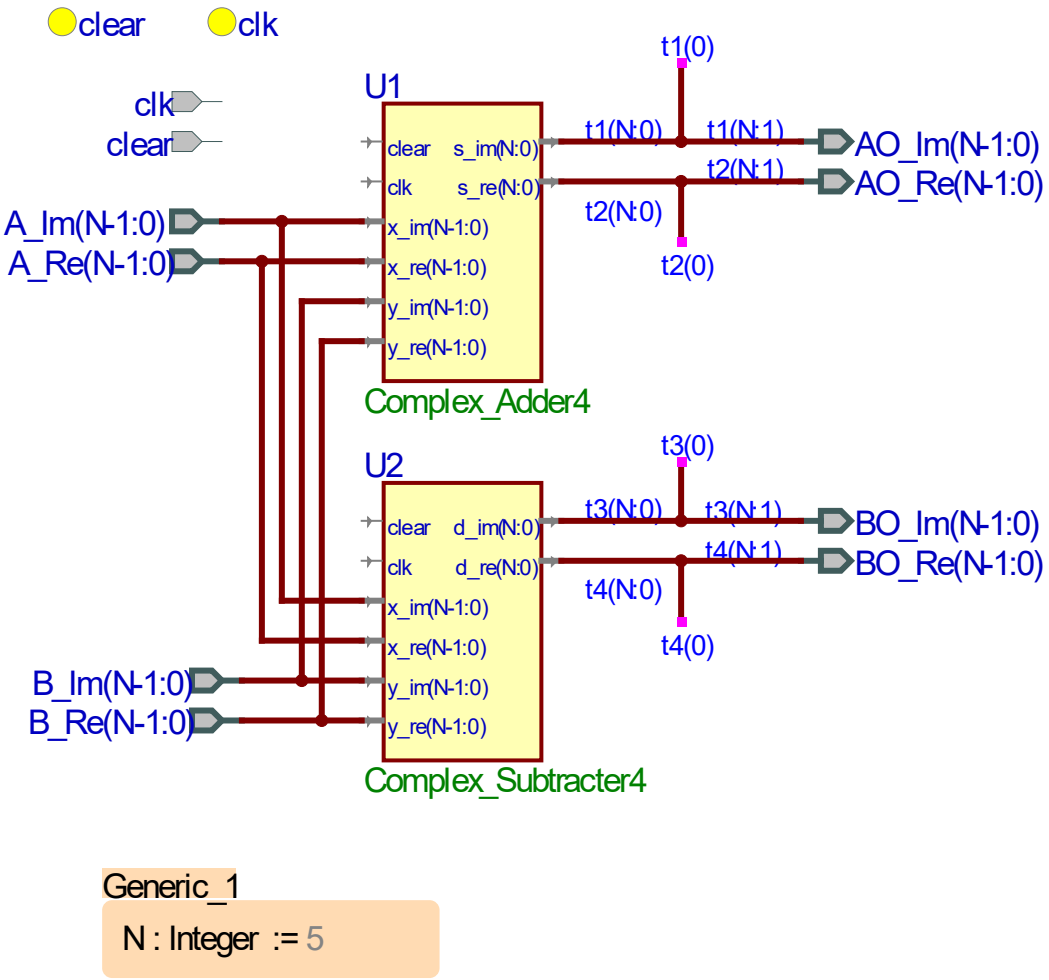
butterfly_unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


-- other libraries declarations
library COMPLEX_ADDER;
library COMPLEX_MULTIPLIER;
library COMPLEX_SHIFT_REGISTER;
library COMPLEX_SUBTRACTER;

entity Butterfly_Unit is
  generic(
       NW : Integer := 4;
       N : Integer  := 5;
       LOG2NP : Integer := 3;
       STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Butterfly_Unit;

architecture Butterfly_Unit of Butterfly_Unit is

---- Component declarations -----

component dff
  generic(
       CLEARQ : std_logic := '0'
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       d : in STD_LOGIC;
       q : out STD_LOGIC
  );
end component;
component WN_Unit
  generic(
       LOG2NP : INTEGER := 3;
       N : INTEGER := 8;
       NW : INTEGER := 8;
       STAGE : INTEGER := 1
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       W_im : out STD_LOGIC_VECTOR(NW-1 downto 0);
       W_re : out STD_LOGIC_VECTOR(NW-1 downto 0)
  );
end component;
component Complex_Adder3
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
```

```
      clk : in STD_LOGIC;
      x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      s_im : out STD_LOGIC_VECTOR(N downto 0);
      s_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;
component Complex_Multiplier
  generic(
      NX : POSITIVE := 4;
      NY : POSITIVE := 5
  );
  port (
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      x_im : in STD_LOGIC_VECTOR(NX-1 downto 0);
      x_re : in STD_LOGIC_VECTOR(NX-1 downto 0);
      y_im : in STD_LOGIC_VECTOR(NY-1 downto 0);
      y_re : in STD_LOGIC_VECTOR(NY-1 downto 0);
      m_im : out STD_LOGIC_VECTOR(NY-2 downto 0);
      m_re : out STD_LOGIC_VECTOR(NY-2 downto 0)
  );
end component;
component Complex_Shift_Register
  generic(
      N : INTEGER := 8;
      W : INTEGER := 2
  );
  port (
      a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      oa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
      oa_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component complex_subtracter3
  generic(
      N : POSITIVE := 4
  );
  port (
      clear : in STD_LOGIC;
      clk : in STD_LOGIC;
      x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
      y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
      d_im : out STD_LOGIC_VECTOR(N downto 0);
      d_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;

---- Architecture declarations -----
function calc_delays( MULT_DELAY, LOG2NP,STAGE : integer )
return INTEGER is
variable T,NEXT_W : integer;
begin
   -- W calculation for next suffling unit
     NEXT_W:=2**(LOG2NP-STAGE-1);
     if (NEXT_W>=MULT_DELAY) then
          T:=0;
     else
          T:=MULT_DELAY;
     end if;
     return T;
end function calc_delays;
```

```
---- Signal declarations used on the diagram ----

signal MSB_im_delay : STD_LOGIC;
signal MSB_re_delay : STD_LOGIC;
signal da1_im_bus : STD_LOGIC_VECTOR (N downto 0);
signal da1_re_bus : STD_LOGIC_VECTOR (N downto 0);
signal dx_im_bus : STD_LOGIC_VECTOR (N downto 0);
signal dx_re_bus : STD_LOGIC_VECTOR (N downto 0);
signal t1 : STD_LOGIC_VECTOR (N-2 downto 0);
signal t2 : STD_LOGIC_VECTOR (N-2 downto 0);
signal w_im_bus : STD_LOGIC_VECTOR (NW-1 downto 0);
signal w_re_bus : STD_LOGIC_VECTOR (NW-1 downto 0);

---- Configuration specifications for declared components

for U1 : Complex_Adder3 use entity complex_adder.Complex_Adder3;
for U2 : complex_subtracter3 use entity Complex_Subtracter.complex_subtracter3;
for U3 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;
for U4 : complex_multiplier use entity complex_multiplier.complex_multiplier;

begin

---- User Signal Assignments ----
AO_Im <= MSB_im_delay&t1(N-2 downto 0);
AO_Re <=MSB_re_delay&t2(N-2 downto 0);

----  Component instantiations  ----

U1 : Complex_Adder3
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       s_im => da1_im_bus( N downto 0 ),
       s_re => da1_re_bus( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U2 : complex_subtracter3
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       d_im => dx_im_bus( N downto 0 ),
       d_re => dx_re_bus( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U3 : Complex_Shift_Register
  generic map (
       N => N-1,
       W => calc_delays(NW-1,LOG2NP,STAGE)
  )
  port map(
       a_im => da1_im_bus( N downto 2 ),
       a_re => da1_re_bus( N downto 2 ),
       clear => clear,
       clk => clk,
```

```
        oa_im => t1( N-2 downto 0 ),
        oa_re => t2( N-2 downto 0 )
  );

U4 : Complex_Multiplier
  generic map (
        NX => NW,
        NY => N+1
  )
  port map(
        clear => clear,
        clk => clk,
        m_im => BO_Im( N-1 downto 0 ),
        m_re => BO_Re( N-1 downto 0 ),
        x_im => w_im_bus( NW-1 downto 0 ),
        x_re => w_re_bus( NW-1 downto 0 ),
        y_im => dx_im_bus( N downto 0 ),
        y_re => dx_re_bus( N downto 0 )
  );

U5 : WN_Unit
  generic map (
        LOG2NP => LOG2NP,
        N => N,
        NW => NW,
        STAGE => STAGE
  )
  port map(
        W_im => w_im_bus( NW-1 downto 0 ),
        W_re => w_re_bus( NW-1 downto 0 ),
        clear => clear,
        clk => clk
  );

U6 : dff
  port map(
        clear => clear,
        clk => clk,
        d => t1(N-2),
        q => MSB_im_delay
  );

U7 : dff
  port map(
        clear => clear,
        clk => clk,
        d => t2(N-2),
        q => MSB_re_delay
  );


end Butterfly_Unit;
```



Σχήμα Ε.3 Σχεδιάγραμμα της τροποποιημένης μονάδας butterfly, που χρησιμοποιείται στην τελευταία μονάδας butterfly, και βρίσκεται στο αρχείο Last_Butterfly_Unit.vhd

Last_Butterfly_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


-- other libraries declarations
library COMPLEX_ADDER;
library COMPLEX_SUBTRACTER;

entity Last_Butterfly_Unit is
  generic(
       N : Integer  := 5
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
```

```
end Last_Butterfly_Unit;

architecture Last_Butterfly_Unit of Last_Butterfly_Unit is

---- Component declarations -----

component Complex_Adder4
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       s_im : out STD_LOGIC_VECTOR(N downto 0);
       s_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;
component Complex_Subtracter4
  generic(
       N : POSITIVE := 4
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       x_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       y_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       d_im : out STD_LOGIC_VECTOR(N downto 0);
       d_re : out STD_LOGIC_VECTOR(N downto 0)
  );
end component;

---- Signal declarations used on the diagram ----

signal t1 : STD_LOGIC_VECTOR (N downto 0);
signal t2 : STD_LOGIC_VECTOR (N downto 0);
signal t3 : STD_LOGIC_VECTOR (N downto 0);
signal t4 : std_logic_vector (N downto 0);

---- Configuration specifications for declared components

for U1 : Complex_Adder4 use entity complex_adder.Complex_Adder4;
for U2 : Complex_Subtracter4 use entity complex_subtracter.Complex_Subtracter4;

begin

----  Component instantiations  ----

U1 : Complex_Adder4
  generic map (
       N => N
  )
  port map(
       clear => clear,
       clk => clk,
       s_im => t1( N downto 0 ),
       s_re => t2( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );

U2 : Complex_Subtracter4
  generic map (
```

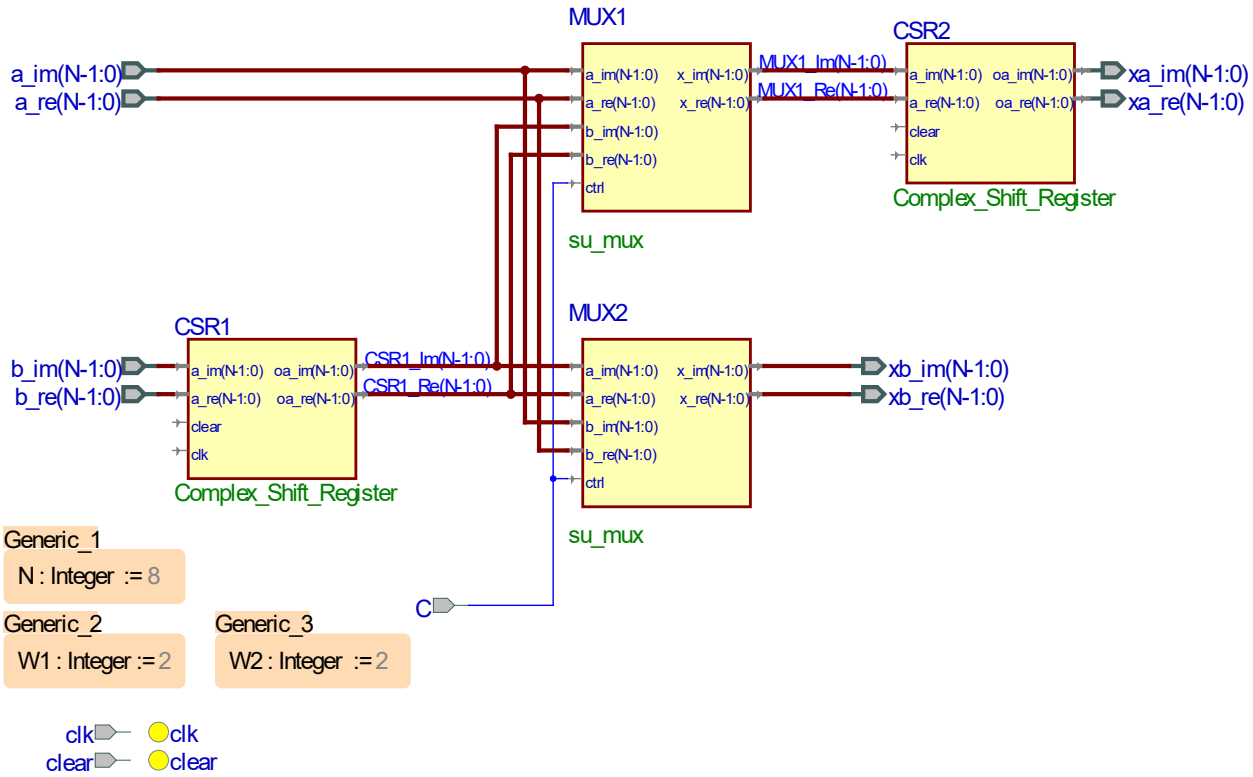
```
      N => N
  )
  port map(
       clear => clear,
       clk => clk,
       d_im => t3( N downto 0 ),
       d_re => t4( N downto 0 ),
       x_im => A_Im( N-1 downto 0 ),
       x_re => A_Re( N-1 downto 0 ),
       y_im => B_Im( N-1 downto 0 ),
       y_re => B_Re( N-1 downto 0 )
  );


---- Terminal assignment ----

    -- Output\buffer terminals
     AO_Im <= t1( N downto 1 );
     AO_Re <= t2( N downto 1 );
     BO_Im <= t3( N downto 1 );
     BO_Re <= t4( N downto 1 );


end Last_Butterfly_Unit;
```

E.6 Μονάδα αναδιάταξης

Ακολουθούν τα αρχεία που τροποποιούνται σε αυτή την περιγραφή σε VHDL.

Mux.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;


entity Mux is
      generic   (
          N : Integer := 8
      );
      port(
           a_re : in STD_LOGIC;
           a_im : in STD_LOGIC;
           b_re : in STD_LOGIC;
           b_im : in STD_LOGIC;
           ctrl : in STD_LOGIC;
           x_re : out STD_LOGIC;
           x_im : out STD_LOGIC
          );
end Mux;


architecture Mux_behav of Mux is
begin
     process (a_re,a_im,b_re,b_im,ctrl)
     begin
I1:             if ctrl='0' then
                     x_re<=a_re;
                     x_im<=a_im;
                elsif ctrl='1' then
                     x_re<=b_re;
                     x_im<=b_im;
                end if I1;
     end process;
end Mux_behav;
```



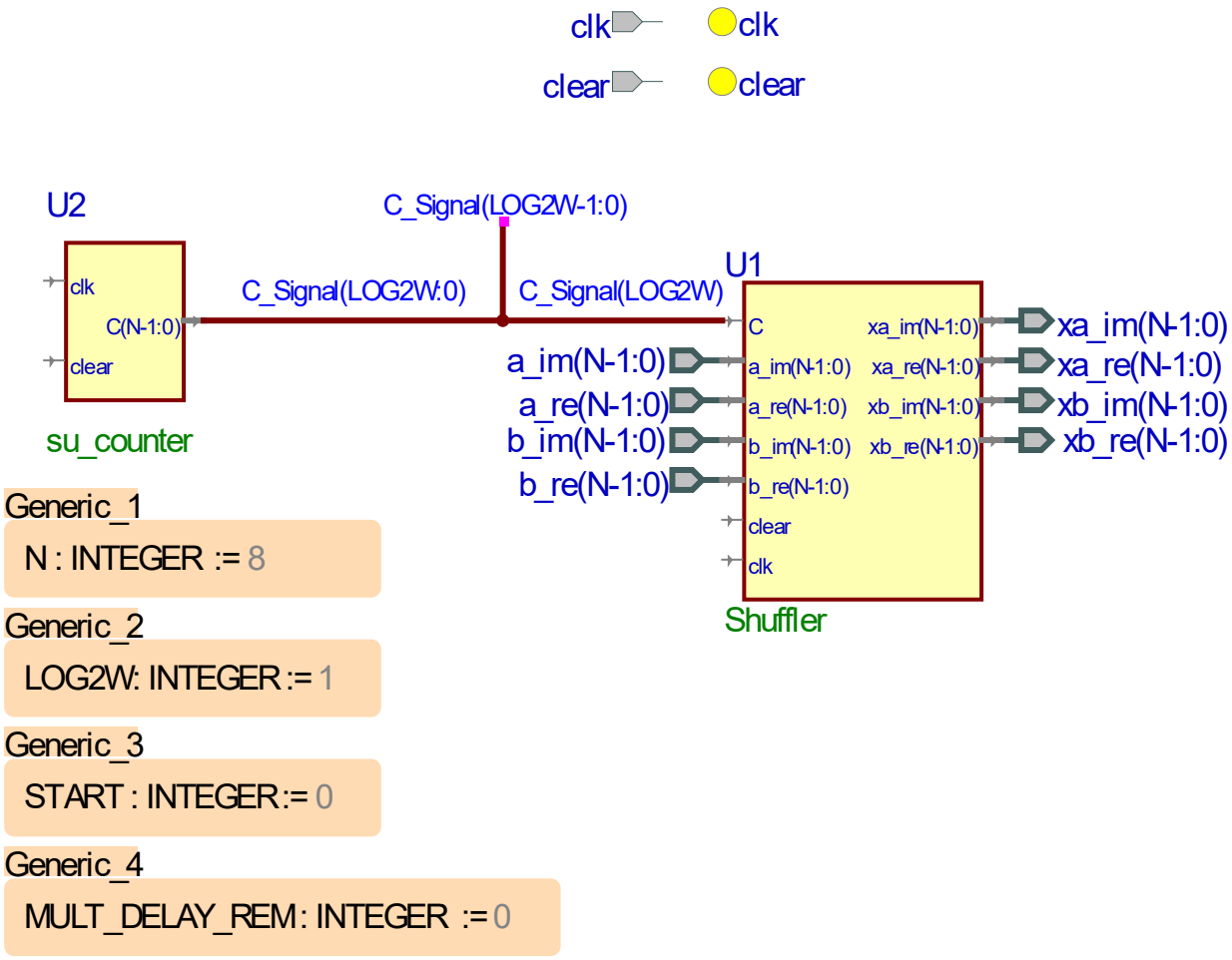
Σχήμα Ε.4 Τροποποιημένη μονάδα αναδιάταξης χωρίς τον μετρητή που οδηγεί τους πολυπλέκτες για σωστό συγχρονισμό και βρίσκεται στο αρχείο Shuffler.bde

Shuffler.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


-- other libraries declarations
library COMPLEX_SHIFT_REGISTER;

entity Shuffler is
  generic(
        N : Integer  := 8;
        W1 : Integer := 2;
        W2 : Integer  := 2
  );
  port(
        C : in STD_LOGIC;
        clear : in STD_LOGIC;
        clk : in STD_LOGIC;
        a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
        a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
        b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
        b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
        xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
        xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
        xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
        xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Shuffler;

architecture Shuffler of Shuffler is

---- Component declarations -----
```

```
component SU_Mux
  generic(
       N : INTEGER := 8
  );
  port (
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       ctrl : in STD_LOGIC;
       x_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       x_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component Complex_Shift_Register
  generic(
       N : INTEGER := 8;
       W : INTEGER := 2
  );
  port (
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       oa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       oa_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

---- Signal declarations used on the diagram ----

signal CSR1_Im : STD_LOGIC_VECTOR (N-1 downto 0);
signal CSR1_Re : STD_LOGIC_VECTOR (N-1 downto 0);
signal MUX1_Im : STD_LOGIC_VECTOR (N-1 downto 0);
signal MUX1_Re : STD_LOGIC_VECTOR (N-1 downto 0);

---- Configuration specifications for declared components

for CSR1 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;
for CSR2 : Complex_Shift_Register use entity
complex_shift_register.Complex_Shift_Register;

begin

----  Component instantiations  ----

CSR1 : Complex_Shift_Register
  generic map (
       N => N,
       W => W1
  )
  port map(
       a_im => b_im( N-1 downto 0 ),
       a_re => b_re( N-1 downto 0 ),
       clear => clear,
       clk => clk,
       oa_im => CSR1_Im( N-1 downto 0 ),
       oa_re => CSR1_Re( N-1 downto 0 )
  );

CSR2 : Complex_Shift_Register
  generic map (
       N => N,
       W => W2
  )
  port map(
       a_im => MUX1_Im( N-1 downto 0 ),
```

```
      a_re => MUX1_Re( N-1 downto 0 ),
      clear => clear,
      clk => clk,
      oa_im => xa_im( N-1 downto 0 ),
      oa_re => xa_re( N-1 downto 0 )
  );

MUX1 : SU_Mux
  generic map (
      N => N
  )
  port map(
      a_im => a_im( N-1 downto 0 ),
      a_re => a_re( N-1 downto 0 ),
      b_im => CSR1_Im( N-1 downto 0 ),
      b_re => CSR1_Re( N-1 downto 0 ),
      clear => clear,
      clk => clk,
      ctrl => C,
      x_im => MUX1_Im( N-1 downto 0 ),
      x_re => MUX1_Re( N-1 downto 0 )
  );

MUX2 : SU_Mux
  generic map (
      N => N
  )
  port map(
      a_im => CSR1_Im( N-1 downto 0 ),
      a_re => CSR1_Re( N-1 downto 0 ),
      b_im => a_im( N-1 downto 0 ),
      b_re => a_re( N-1 downto 0 ),
      clear => clear,
      clk => clk,
      ctrl => C,
      x_im => xb_im( N-1 downto 0 ),
      x_re => xb_re( N-1 downto 0 )
  );


end Shuffler;
```



Σχήμα Ε.5 Τροποποιημένη μονάδα αναδιάταξης όπως σχεδιάστηκε στο αρχείο Shuffling_Unit.bde

Shuffling_Unit.vhd

```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;


entity Shuffling_Unit is
  generic(
       N : INTEGER  := 8;
       LOG2W: INTEGER := 1;
       START : INTEGER := 0;
       MULT_DELAY_REM : INTEGER  := 0
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end Shuffling_Unit;

architecture Shuffling_Unit of Shuffling_Unit is
```

```
---- Component declarations -----

component Shuffler
  generic(
       N : INTEGER := 8;
       W1 : INTEGER := 2;
       W2 : INTEGER := 2
  );
  port (
       C : in STD_LOGIC;
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
component su_counter
  generic(
       N : INTEGER := 5;
       START : INTEGER := 0
  );
  port (
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       C : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

---- Signal declarations used on the diagram ----

signal C_Signal : STD_LOGIC_VECTOR (LOG2W downto 0);

begin

----  Component instantiations  ----

U1 : Shuffler
  generic map (
       N => N,
       W1 => (2**LOG2W)-MULT_DELAY_REM,
       W2 => 2**LOG2W
  )
  port map(
       C => C_Signal(LOG2W),
       a_im => a_im( N-1 downto 0 ),
       a_re => a_re( N-1 downto 0 ),
       b_im => b_im( N-1 downto 0 ),
       b_re => b_re( N-1 downto 0 ),
       clear => clear,
       clk => clk,
       xa_im => xa_im( N-1 downto 0 ),
       xa_re => xa_re( N-1 downto 0 ),
       xb_im => xb_im( N-1 downto 0 ),
       xb_re => xb_re( N-1 downto 0 )
  );

U2 : su_counter
  generic map (
       N => LOG2W+1,
       START => START
  )
  port map(
       C => C_Signal( LOG2W downto 0 ),
```

```
       clear => clear,
       clk => clk
  );


end Shuffling_Unit;
```

E.7 Μονάδα βάσης-2 γρήγορου μετασχηματισμού Fourier με συνδυασμό του μιγαδικού αφαιρέτη-πολλαπλασιαστή

Ακολουθεί η περιγραφή της τροποποιημένης μονάδας υπολογισμού του βάσης-2 γρήγορου μετασχηματισμού Fourier.

FFT_Unit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

library BUTTERFLY_UNIT;
library SHUFFLING_UNIT;

entity FFT_Unit is
      generic(
          NW : Integer := 8;      -- bits for twiddle factors
          N : Integer  := 16;    -- bits for input of FFT
          LOG2NP : Integer := 3  -- NP-Points FFT
      );
      port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
      );
end FFT_Unit;

architecture FFT_Unit of FFT_Unit is

component First_Butterfly_Unit is
  generic(
          NW : Integer := 8;
          N : Integer  := 8;
          LOG2NP : Integer := 3;
          STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;
```

```
component Butterfly_Unit is
  generic(
           NW : Integer := 8;
           N : Integer  := 8;
           LOG2NP : Integer := 3;
           STAGE : Integer := 1
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

component Last_Butterfly_Unit is
  generic(
       N : Integer  := 5
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

component Shuffling_Unit is
  generic(
       N : INTEGER  := 8;
        LOG2W: INTEGER := 1;
        START : INTEGER := 0;
       MULT_DELAY_REM : INTEGER  := 0
  );
  port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       a_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       a_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_im : in STD_LOGIC_VECTOR(N-1 downto 0);
       b_re : in STD_LOGIC_VECTOR(N-1 downto 0);
       xa_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xa_re : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_im : out STD_LOGIC_VECTOR(N-1 downto 0);
       xb_re : out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end component;

subtype num_word is STD_LOGIC_VECTOR (N-1 downto 0);
type num_word_array is array(natural range <>) of num_word;
signal Up_Re_in,Up_Im_in,Down_Re_in,Down_Im_in,
        Up_Re_out,Up_Im_out,Down_Re_out,Down_Im_out : num_word_array(1 to LOG2NP);


function calc_counter_start( N,NW, LOG2NP, STAGE : integer )
return INTEGER is
variable T,W,S,SUM,
           MULT_DELAY,
```

```
            BUTTERFLY_DELAY : integer;
begin
     MULT_DELAY:=NW-1;
     BUTTERFLY_DELAY:=NW+2;
     W:=2**(LOG2NP-2);
     T:=0;
     SUM:=0;
     S:=2;
     while S<=STAGE loop
          if W>=MULT_DELAY then
               T:=T+1;
          end if;
          W:=W/2;
          SUM:=SUM+2**(LOG2NP-STAGE);
          S:=S+1;
     end loop;
     T:=-(STAGE-1)*BUTTERFLY_DELAY+T*MULT_DELAY;
     return T;
end function calc_counter_start;

function calc_delays_rem( MULT_DELAY, LOG2NP, STAGE : integer )
return INTEGER is
variable T,W_NEXT : integer;
begin
   -- W calculation for suffling unit
     W_NEXT:=2**(LOG2NP-STAGE);
     if (W_NEXT>=MULT_DELAY) then
          T:=MULT_DELAY;
     else
          T:=0;
     end if;
     return T;
end function calc_delays_rem;

begin

     --Batterfly Units generation
FirstBU:   First_Butterfly_Unit
               generic map (
                    NW => NW,
                    N => N,
                    LOG2NP => LOG2NP,
                    STAGE => 1
               )
               port map(
                    clear => clear,
                    clk => clk,
                    A_Im => Up_Im_in(1),
                    A_Re => Up_Re_in(1),
                    B_Im => Down_Im_in(1),
                    B_Re => Down_Re_in(1),
                    AO_Im => Up_Im_out(1),
                    AO_Re => Up_Re_out(1),
                    BO_Im => Down_Im_out(1),
                    BO_Re => Down_Re_out(1)
               );
BUgi:for i in 2 to LOG2NP-1 generate
BU:            Butterfly_Unit
               generic map (
                    NW => NW,
                    N => N,
                    LOG2NP => LOG2NP,
                    STAGE => i
               )
               port map(
                    clear => clear,
                    clk => clk,
                    A_Im => Up_Im_in(i),
                    A_Re => Up_Re_in(i),
                    B_Im => Down_Im_in(i),
```

```
                    B_Re => Down_Re_in(i),
                    AO_Im => Up_Im_out(i),
                    AO_Re => Up_Re_out(i),
                    BO_Im => Down_Im_out(i),
                    BO_Re => Down_Re_out(i)
               );
          end generate BUgi;
LastBU:   Last_Butterfly_Unit
               generic map (
                    N => N
               )
               port map(
                    clear => clear,
                    clk => clk,
                    A_Im => Up_Im_in(LOG2NP),
                    A_Re => Up_Re_in(LOG2NP),
                    B_Im => Down_Im_in(LOG2NP),
                    B_Re => Down_Re_in(LOG2NP),
                    AO_Im => Up_Im_out(LOG2NP),
                    AO_Re => Up_Re_out(LOG2NP),
                    BO_Im => Down_Im_out(LOG2NP),
                    BO_Re => Down_Re_out(LOG2NP)
               );

          -- Shuffling untis generation
SUgi:for i in 2 to LOG2NP generate
SU:            Shuffling_Unit
               generic map (
                    N => N,
                    LOG2W => LOG2NP-i,
                    START => calc_counter_start(N,NW,LOG2NP,i),
                    MULT_DELAY_REM => calc_delays_rem(NW-1, LOG2NP, i)
               )

               port map (
                    clear => clear,
                    clk => clk,
                    a_im => Up_Im_out(i-1),
                    a_re => Up_Re_out(i-1),
                    b_im => Down_Im_out(i-1),
                    b_re => Down_Re_out(i-1),
                    xa_im => Up_Im_in(i),
                    xa_re => Up_Re_in(i),
                    xb_im => Down_Im_in(i),
                    xb_re => Down_Re_in(i)
               );
          end generate SUgi;

          -- input signals
          Up_Im_in(1) <= A_Im;
          Up_Re_in(1) <= A_Re ;
          Down_Im_in(1) <= B_Im;
          Down_Re_in(1) <= B_Re;

          -- output signals
          AO_Im <= Up_Im_out(LOG2NP);
          AO_Re <= Up_Re_out(LOG2NP);
          BO_Im <= Down_Im_out(LOG2NP);
          BO_Re <= Down_Re_out(LOG2NP);

end FFT_Unit;
```

FFT_Unit_TB.vhd

```
library ieee,butterfly_unit,shuffling_unit;
use ieee.std_logic_1164.all;
```

```
     -- Add your library and packages declaration here ...
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use std.textio.all;
use IEEE.math_real.all;

entity fft_unit_tb is
     -- Generic declarations of the tested unit
          generic(
          NW : INTEGER := 12;
          N : INTEGER := 13;
          LOG2NP : INTEGER := 5 );
end fft_unit_tb;

architecture TB_ARCHITECTURE of fft_unit_tb is
     file OUTPUT_FILE : TEXT open WRITE_MODE is "outputs.txt";

     -- Component declaration of the tested unit
     component fft_unit
          generic(
          NW : INTEGER := 8;
          N : INTEGER := 16;
          LOG2NP : INTEGER := 3 );
     port(
       clear : in STD_LOGIC;
       clk : in STD_LOGIC;
       A_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       A_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Im : in STD_LOGIC_VECTOR(N-1 downto 0);
       B_Re : in STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       AO_Re : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Im : out STD_LOGIC_VECTOR(N-1 downto 0);
       BO_Re : out STD_LOGIC_VECTOR(N-1 downto 0) );
     end component;

     constant DELAY1 : time := 1 ns;

     -- Stimulus signals - signals mapped to the input and inout ports of tested
entity
     signal clk : std_logic;
     signal clear : std_logic;
     signal A_Im : std_logic_vector((N-1) downto 0);
     signal A_Re : std_logic_vector((N-1) downto 0);
     signal B_Im : std_logic_vector((N-1) downto 0);
     signal B_Re : std_logic_vector((N-1) downto 0);
     -- Observed signals - signals mapped to the output ports of tested entity
     signal AO_Im : std_logic_vector((N-1) downto 0);
     signal AO_Re : std_logic_vector((N-1) downto 0);
     signal BO_Im : std_logic_vector((N-1) downto 0);
     signal BO_Re : std_logic_vector((N-1) downto 0);

     -- Add your code here ...

function calc_f( t : positive ) return integer is
variable f : real;
begin
     f := real(2**(N-3)-1)*COS( -real(t)*2.0*MATH_PI/3.0 )+real(2**(N-3)-1)*COS( -
real(t)*2.0*MATH_PI/11.0 );
     return integer(f);
end function calc_f;

function calc_fft_delay( N,NW, LOG2NP : integer )
return INTEGER is
variable T,W,S,
           MULT_DELAY,
           BUTTERFLY_DELAY : integer;
begin
     MULT_DELAY:=NW-1;
     BUTTERFLY_DELAY:=NW+2;
```

```
     W:=2**(LOG2NP-2);
     T:=0;
     S:=2;
     while S<=LOG2NP-1 loop
          if W>=MULT_DELAY then
               T:=T+1;
          end if;
          W:=W/2;
          S:=S+1;
     end loop;
     return (LOG2NP-1)*BUTTERFLY_DELAY-T*MULT_DELAY+(N)+(2**(LOG2NP-1)-1);
end function calc_fft_delay;

begin

     -- Unit Under Test port map
     UUT : fft_unit
          generic map (
               NW => NW,
               N => N,
               LOG2NP => LOG2NP
          )

          port map (
               clk => clk,
               clear => clear,
               A_Im => A_Im,
               A_Re => A_Re,
               B_Im => B_Im,
               B_Re => B_Re,
               AO_Im => AO_Im,
               AO_Re => AO_Re,
               BO_Im => BO_Im,
               BO_Re => BO_Re
          );

     -- Add your stimulus here ...
     CLK_Generation: process
     begin
          clk<='0';wait for DELAY1;
          clk<='1';wait for DELAY1;
     end process;

     INPUT_Generation: process
     variable seed1 : integer := 11111111;
     variable seed2 : integer := 22222222;
     variable rnd_num : real;
     variable rnd_num_v1 : std_logic_vector(N-1 downto 0);
     variable rnd_num_v2 : std_logic_vector(N-1 downto 0);
     variable rnd_num_v3 : std_logic_vector(N-1 downto 0);
     variable rnd_num_v4 : std_logic_vector(N-1 downto 0);
     variable t,ft1,ft2 : integer :=0;

     begin
          A_Re <= CONV_STD_LOGIC_VECTOR(0,N);
          A_Im <= CONV_STD_LOGIC_VECTOR(0,N);
          B_Re <= CONV_STD_LOGIC_VECTOR(0,N);
          B_Im <= CONV_STD_LOGIC_VECTOR(0,N);
          clear <= '0';
          wait until (clk'event) and (clk='1');
          clear <= '1';

          while 1>0 loop
               t:=t+1;
               ft1 := integer(calc_f(t));
               ft2 := integer(calc_f(t+2**(LOG2NP-1)));
               UNIFORM(seed1,seed2,rnd_num);
               rnd_num_v1 :=
conv_std_logic_vector(integer(real(ft1)*(1.0+0.00*rnd_num*2.0**(0))),N);
               A_Re <= rnd_num_v1;
```

```
                UNIFORM(seed1,seed2,rnd_num);
                rnd_num_v2 :=
conv_std_logic_vector(integer(real(0)*(1.0+0.00*rnd_num*2.0**(0))),N);
                A_Im <= rnd_num_v2;
                UNIFORM(seed1,seed2,rnd_num);
                rnd_num_v3 :=
conv_std_logic_vector(integer(real(ft2)*(1.0+0.00*rnd_num*2.0**(0))),N);
                B_Re <= rnd_num_v3;
                UNIFORM(seed1,seed2,rnd_num);
                rnd_num_v4 :=
conv_std_logic_vector(integer(real(0)*(1.0+0.00*rnd_num*2.0**(0))),N);
                B_Im <= rnd_num_v4;
                wait until (clk'event) and (clk='1');
          end loop;
     end process;

     OUTPUT_Checking:process
          type N_Matrix is array (NATURAL range <>) of std_logic_vector(N-1 downto 0);

          constant FFT_DELAY : integer := calc_fft_delay(N,NW,LOG2NP);

          variable A_Re_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));
          variable A_Im_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));
          variable B_Re_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));
          variable B_Im_Matrix : N_Matrix (0 to FFT_DELAY):=(others=>(others=>'0'));

          variable report_line : line;
          variable report_str : string(1 to 160);
          variable i,j,re_part, im_part,temp_part : integer := 0;


     begin

          wait until (clk'event) and (clk='1') and (clear='1');

          for i in 0 to FFT_DELAY-1 loop
               A_Re_Matrix(i) := A_Re_Matrix(i+1);
               A_Im_Matrix(i) := A_Im_Matrix(i+1);
               B_Re_Matrix(i) := B_Re_Matrix(i+1);
               B_Im_Matrix(i) := B_Im_Matrix(i+1);
          end loop;
          A_Re_Matrix(FFT_DELAY) := A_Re;
          A_Im_Matrix(FFT_DELAY) := A_Im;
          B_Re_Matrix(FFT_DELAY) := B_Re;
          B_Im_Matrix(FFT_DELAY) := B_Im;

          report_line:=null;

          write(report_line,conv_integer(A_Re_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(A_Im_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(B_Re_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(B_Im_Matrix(0)));
          write(report_line,string'(","));
          write(report_line,conv_integer(AO_Re));
          write(report_line,string'(","));
          write(report_line,conv_integer(AO_Im));
          write(report_line,string'(","));
          write(report_line,conv_integer(BO_Re));
          write(report_line,string'(","));
          write(report_line,conv_integer(BO_Im));
          write(report_line,string'("       "));

          for i in report_line'range loop
               report_str(i) := report_line(i);
          end loop;
          report report_str;
```

```
            writeline(OUTPUT_FILE,report_line);
      end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_fft_unit of fft_unit_tb is
      for TB_ARCHITECTURE
            for UUT : fft_unit
                  use entity work.fft_unit(fft_unit);
            end for;
      end for;
end TESTBENCH_FOR_fft_unit;
```