**ΕΡΓΑΣΤΗΡΙΟ ΨΗΦΙΑΚΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ ΣΗΜΑΤΩΝ**

**Εργαστηριακή Άσκηση 4**

**Εργαστηριακή Άσκηση 4 A/D, D/A , Δειγματοληψία και FIR φίλτρα**

|  |  |  |  |
| --- | --- | --- | --- |
| **ΟΝΟΜΑ** | **ΕΠΩΝΥΜΟ** | **Α.Μ.** | **ΟΜΑΔΑ** |
|  |  |  |  |
|  |  |  |  |

**Άσκηση 4.1 - ρύθμιση McBSP1: Να δημιουργήσετε ένα CCS Project με τις κατάλληλες ρυθμίσεις και τα απαραίτητα αρχεία και να αναπτύξετε μια ρουτίνα σε assembly ώστε να ρυθμίσετε την McBSP1.**

Ο AIC23 Codec πραγματοποιεί A/D ή D/A μετατροπές μέσω της σειριακής θύρας McBSP1 του DSP.Για να γίνουν οι μετατροπές αυτές πρέπει πρώτα να ρυθμίσουμε την McBSP1,το οποιο γίνεται γράφοντας κατάλληλες τιμές στους καταχωρητές μνήμης που την ελέγχουν. Η συμπεριφορά της McBSP1 διαμορφώνεται από τους καταχωρητές SPCR, SRGR,PCR,XCR,RCR. O SRGR ελεγχει τις λειτούργιες του sample rate generator.Οι PCR και SPCR περιεχουν bit που περιγράφουν τις καταστάσεις του port,π.χ. διαβαζοντας τα καταληλλα bit του SPCR ελενχουμε αν εχουμε λαβει ή μεταδωσει στοιχεια.Τελος οι καταχωρητες RCR και XCR οριζουν τη μορφη που θα εχει η πληροφορια που λαμβανουμε και στελνούμε αντίστοιχα.Γράφοντας 0xA στους καταχωρητές βαζουμε την τιμή 5 στa bit 5-7 επιλεγουμε να λαμβάνουμε ενα word 32 bit και αφηνοντας 0 τα bit 8-14 λαμβάνουμε ενα word ανα φαση ωστε να παίρνουμε και να στελνουμέ τελικά 32 bit σε μια φάση.

**Άσκηση 4.2 - μετάδοση δεδομένων: Υποθέτοντας ότι τα 32-bit δεδομένα που θέλουμε να μεταδώσουμε μέσω της McBSP1 βρίσκονται στον A0, να γράψετε μια ρουτίνα σε γλώσσα assembly που να ελέγχει τον SPCR και να διαβιβάζει έπειτα τα δεδομένα στην McBSP1. Εάν ο DXR δεν είναι κενός, το πρόγραμμά σας πρέπει να περιμένει έως ότου τελειώσει η McBSP1 τη μεταβίβαση των υπόλοιπων δεδομένων.**

Αν θέλουμε να κάνουμε μετάδοση δεδομένων, θα χρησιμοποιήσουμε τον καταχωρητή DXR ο οποίος περιέχει τα δεδομένα εξόδου της σειριακής θύρας(αυτά μεταφέρονται σε σειριακή μορφή στο pin εξόδου του AIC23 Codec).Στον DXR γράφουμε μόνο εφόσον έχει τελειώσει η διαβίβαση των προηγούμενων δεδομένων,δηλαδή όταν ο είναι άδειος.Κάθε φορά λοιπόν που ο DXR είναι έτοιμος να πάρει νέα δεδομένα το 17o bit(XRDY) του SPCR γίνεται 1.

SPCR **.set** 0x1900008 ; διεύθυνση μνήμης SPCR

DXR **.set** 0x1900004 ; διεύθυνση μνήμης DXR

.def entry

**.text**

**entry:** MVKL SPCR, A2

MVKH SPCR, A2

MVKL 0x10000, A3

MVKH 0x10000, A3

MVKL DXR,A1

MVKH DXR,A1

**loopb:** LDW \*A2,B1 ; metaferw ta dedomena tous spcr ston B1

**NOP** 4

**AND** B1,A3,B1 ;apomononw to 17 bit twn dedomenwn gia na dw an einai 1 h 0

**XOR** B1,A3,B1 ;an einai 1 dhladi to apotelesma tis xor einai 0 ksanametadidw kainouria data

[B1] B loopb

**NOP** 5

STW A4,\*A1 ;apothikeuw tin eisodo ston dxr

**IDLE**

.end

Για το σκοπό αυτό απομονώνουμε το XRDY του SPCR με μια AND με τον αριθμό 0x10000 το οποίο έχει 1 ασσο μόνο στο 17ο bit.Μετά με μία XOR με τον ίδιο αριθμό του λέμε αν το XRDY είναι 0 τότε το αποτέλεσμα είναι 1, να συνεχίζει να μεταφέρει δεδομένα από τον SPCR ,ενώ όταν το 17ο bit γίνει 1 τότε το αποτελεσμα της XOR είναι 0 και μπορεί πλέον να αποθηκεύσει στον DXR τα δεδομένα εξόδου.

**Άσκηση 4.3 - λαμβάνοντας δεδομένα: Να γραφεί μια ρουτίνα σε γλώσσα assembly που να ελέγχει τον SPCR και να διαβάζει έπειτα τα δεδομένα από τον DRR. Εάν δεν υπάρχει κανένα δεδομένο στον DRR, το πρόγραμμά σας πρέπει να περιμένει έως ότου η McBSP1 λάβει τα νέα δεδομένα στον DRR.**

Για τη λήψη δεδομένων υπάρχει ένας καταχωρητής DRR που περιέχει τα δεδομένα εισόδου της McBSP1 που παραλαμβάνονται από το αντίστοιχο pin.Αφού γίνει η A/D μετροπή του σήματος εισόδου τότε το αποτέλεσμα διαβιβάζεται στη θύρα και καταχωρείται στον DRR.Αν τα δεδομένα είναι έτοιμα να διαβαστούν στον DRR τότε γίνεται 1 το 1ο bit του SPCR(RRDY).

SPCR **.set** 0x1900008 ; διεύθυνση μνήμης SPCR

DRR **.set** 0x1900000 ; διεύθυνση μνήμης DRR

.def entry

**.text**

**entry:** MVKL SPCR,A1 ;fortwnw tin dieuthinsi tou spcr

MVKH SPCR,A1

MVKL DRR,A2 ;fortwnw tin dieuthinsi tou drr

MVKH DRR,A2

**loop:** LDW \*A1,B0

**NOP** 4

**AND** B0,1,B0 ;apomonwnw to 1o bit

**XOR** B0,1,B0 ;to allazw wste otan mou dwsei 1 na vgw apo to loop

[B0] B loop

**NOP** 5

LDW \*A2,A3 ;diavazw tin eisodo

**IDLE**

.end

Με την ίδια λογική όπως και παραπάνω κάνουμε αρχικά AND με το 1 για να απομονώσουμε το 1o bit και στη συνέχεια με μια XOR ελέγχουμε αν ειναι 1 ή 0.Έτσι όταν το RRDY γίνει 1,το αποτέλεσμα της XOR είναι 0,βγαίνουμε απο το loop, και διαβάζουμε τα δεδομένα στον DRR.

**Άσκηση 4.4 - A/D, και D/A βρόχος (polling): Να γραφεί ένα πρόγραμμα σε γλώσσα assembly που να διαβάζει τα δεδομένα μετατροπής A/D και να γράφει άμεσα πίσω τα ίδια δεδομένα για τη μετατροπή D/A. Με τη σύνδεση μιας ηχητικής πηγής και ηχείων στο DSK board, πρέπει να είστε σε θέση να ακούσετε τον ήχο εισόδου. Ανάλογα με το επίπεδο έντασης των σημάτων εισόδου, μπορεί να είναι απαραίτητο να πολλαπλασιαστεί η είσοδος με μια σταθερά, συνήθως από 4 έως 8, πριν από τη μετατροπή D/A για να ενισχυθεί το σήμα.**

Σε αυτό το ερώτημα μας ζητάται να γράψουμε μια ρουτίνα που να επεξεργάζεται τα δεδομένα με τη μέθοδο polling,δηλαδή με τη χρήση ενός βρόγχου ο οποίος διαβάζει αρχικά τα δεδομένα που ερχονται απο την A/D μετατροπή και στη συνέχεια γράφει τα ίδια δεδομένα πίσω για τη μετατροπή D/A.Για το σκοπό αυτό θα συνδιάσουμε τα παραπάνω ερωτήματα,αφού αυτή ηδιαδικασία είναι μια λούπα λήψης και μετάδοσης δεδομένων.

Αρχικά θα διαβάζουμε τα δεδομένα απο τον DRR, στη συνέχεια γράφουμε τα ίδια δεδομένα στον DXR και όταν τελειώσει η λήψη και μετάδοση των συγκεκριμένων στοιχείων ξαναπάμε στην αρχή και επαναλαμβάνεται η διαδικασία για τα επόμενα

SPCR **.set** 0x1900008 ; διεύθυνση μνήμης SPCR

DRR **.set** 0x1900000 ; διεύθυνση μνήμης DRR

DXR **.set** 0x1900004 ; διεύθυνση μνήμης DXR

.def entry

**.text**

**entry:** MVKL DRR,A0

MVKH DRR,A0

MVKL DXR,A1

MVKH DXR,A1

MVKL SPCR,A2

MVKH SPCR,A2

MVKL 0x10000,A3

MVKH 0x10000,A3

**loopc:**

**loopa:**  LDW \*A2,B0

**NOP** 4

**AND** B0,1,B0 ;apomonwnoume to prwto bit

**XOR** B0,1,B0 ;to allazoume

[B0] B loopa

**NOP** 5

LDW \*A0,A4 ;diavazw tin eisodo

**NOP** 4

**loopb:** LDW \*A2,B1

**NOP** 4

**AND** B1,A3,B1 ;apomonwnoume to 17 bit

**XOR** B1,A3,B1 ;to allazοume

[B1] B loopb

**NOP** 5

STW A4,\*A1 ;apothikeuw tin eisodo ston dxr

**B** loopc

**NOP** 5

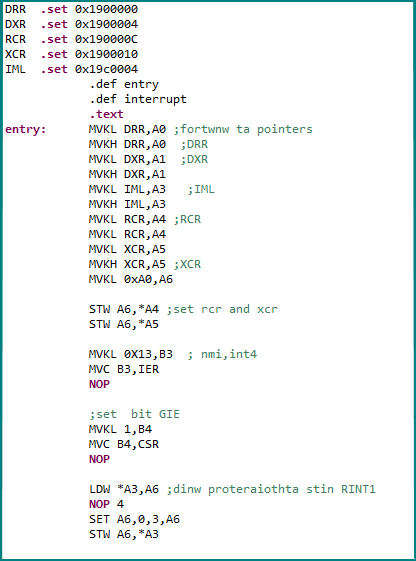
**IDLE**

.end

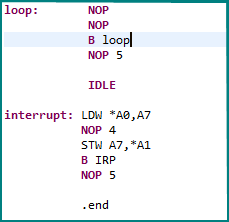
**Άσκηση 4.5 - A/D και D/A βρόχος (με διακοπές): Να γραφεί ένα πρόγραμμα σε γλώσσα assembly που να διαβάζει τα δεδομένα μετατροπής A/D και να γράφει άμεσα πίσω τα ίδια δεδομένα για τη μετατροπή D/A. Χρησιμοποιώντας την RINT1 διακοπή, το αποτέλεσμα A/D διαβάζεται μέσω της ρουτίνας εξυπηρέτησης διακοπής. Το πρόγραμμα απλά περιμένει την εμφάνιση αιτήσεων διακοπών. Επειδή και οι μετατροπές A/D και D/A εμφανίζονται σε υψηλό ρυθμό (48 kHz), αρκεί να εξυπηρετήσετε μόνο μια διακοπή (π.χ. RINT1) για να λάβετε και στη συνέχεια να αποστείλετε το ψηφιακό σήμα.**

Για να πραγματοποιήσουμε την ίδια διαδικασία λήψης και μετάδοσης δεδομένων,μόνο που τώρα με τη χρήση διακοπών θα πρέπει να τροποποιήσουμε το πρόγραμμα έτσι ώστε να περιμένει κάποια αίτηση διακοπής για να ξεκινήσει να “τρέχει”.Με τα interrupts εξασφαλίζουμε ότι το πρόγραμμα δε θα ελέγχει συνεχώς αν υπάρχουν δεδομένα για ανάγνωση αλλά θα ενεργοποιείται μόνο όταν έχουν φτάσει.

Αρχικά θα ορίσουμε τις διευθύνσεις των καταχωρητών που ελέγχουν και τη λήψη και τη μετάδοση δεδομένων.Για το σκοπό αυτό πρέπει να κάνουμε την αρχικοποιηση των RXR,XCR με 0x00Α0 για να λαμβανει και να μεταδιδει αντιστοιχα 32 bits σε μια φάση όπως και για το interrupt, δηλαδή να ορίσουμε το INT4,NMI καθώς και το enable bit του CSR.Επίσης ορίζουμε τη διακοπή RINT1 στο intsel4.



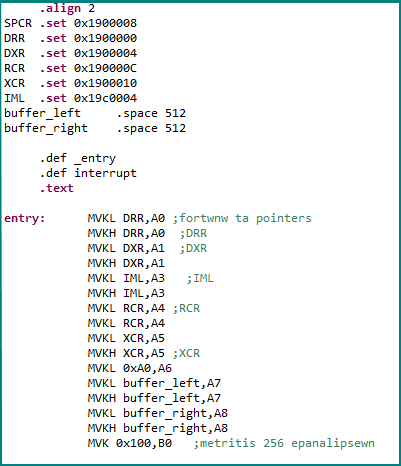
Προσθέτουμε έναν ατέρμονα βρόγχο για μην σταματήσει να εκτελείται η διακοπή.O ISR της RINT1 περιέχει τη διαδικασία που διαβάζει και γράφει επιτόπου τα δεδομένα και βρίσκεται κάτω από τον κώδικα γιατί μπαίνει από το αρχείο interrupt\_vectors με μία branch κάθε φορά που έρχεται η διακοπή.



**Άσκηση 4.6 - αποθηκεύοντας δεδομένα στον buffer**

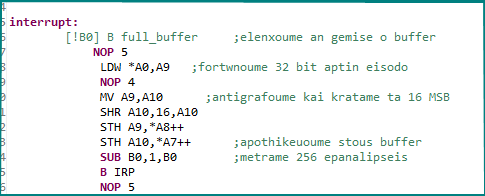
**1.Ορίστε δύο buffers μεγέθους 512 bytes ο καθένας, έναν για κάθε κανάλι. Ονομάστε τους buffer\_left και buffer\_right. 2. Κατόπιν, εισάγετε τις κατάλληλες γραμμές κώδικα έτσι ώστε κάθε φορά που o DSP διαβάζει από τον codec, η τιμή των 32-bit δεδομένων (χωρισμένη σε 2 halfwords αντίστοιχα για το αριστερό και το δεξιό κανάλι) να αποθηκεύεται κατάλληλα στους 2 buffers. 3. Αποθηκεύστε στους buffers όλα τα δεδομένα εισόδου αρχίζοντας από την κορυφή. Όταν ο buffer είναι πλήρης, αρχίζετε πάλι να γράφετε από την κορυφή, γράφοντας επάνω στα παλιά δεδομένα. 4. Γράψετε τον κώδικα σας και τρέξτε τον στο board. Εισάγετε ένα σήμα ημίτονου 12 kHz στον codec χρησιμοποιώντας κάποιο λογισμικό tone generator, είτε από τον υπολογιστή του εργαστηρίου είτε από κάποιο smartphone. 5. Ενώ η είσοδος παραμένει συνδεδεμένη, σταματήστε (pause) τον DSP. Οι τιμές δειγμάτων παραμένουν καταχωρημένες στις θέση μνήμης όπου δείχνουν τα buffer\_left και buffer\_right.**

Όπως είναι γνωστό ήδη για να δεσμεύσουμε κάποιο χώρο στη μνήμη πρέπει να ορίσουμε αντίστοιχους buffer με το επιθυμητό μέγεθος.Επίσης συμπεριλαμβάνουμε τις αρχικοποιήσεις των καταχωρητών για τη διαδικασία της λήψης και μετάδοσης δεδομένων.

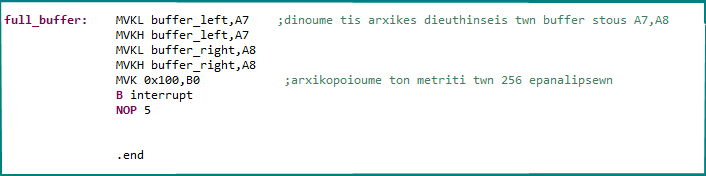


Κάνουμε τις ρυθμίσεις που έχουν αναφερθέι για RCR,XCR καθώς και για την ενεργοποίηση της διακοπής interrupt.

Τέλος έχουμε τον ατέρμονα βρόγχο.Έτσι κάθε φορά που έρχεται μια διακοπή η ροή του προγράμματος μεταφέρεται στην ISR της :



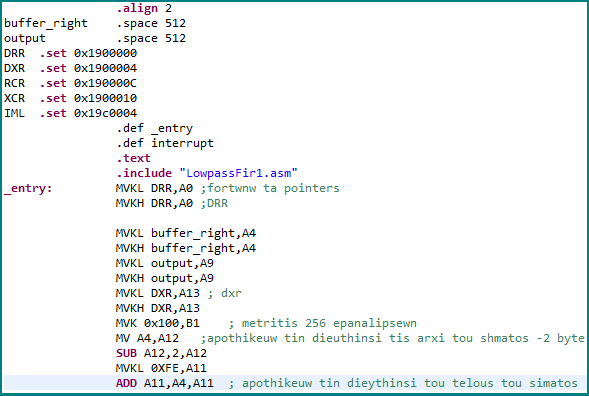
Εδώ φορτώνουμε το περιεχόμενο το οποίο βρίσκεται στον DRR προς ανάγνωση στον καταχωρητή Α9.Αντιγράφουμε τον Α9 στον Α10,και τα μετατοπίζουμε 1 Halfword δεξία με την εντολή SHR. Πλέον ο καταχωρητής Α9 περιέχει τα 16 LSB στοιχεία της εισόδου και ο Α10 τα 16 MSB στα 16 πρώτα τους bit.Τώρα μπορούμε να αποθηκεύσουμε τα δεδομένα στους 2 buffer,τα LSB στον \_right ενώ τα 16 MSB στον \_left.Όσο οι buffers γεμίζουν με δεδομένα μειώνουμε 1 counter τον οποίο έχουμε αρχικοποιήσει παραπάνω με την τιμή 256,δηλαδή μετράει 256 halfwords (512 bytes ) μέχρι να γεμίσουν οι buffers.Με τον έλεγχο στη αρχή της ISR , πετυχαίνουμε όταν αυτός ο counter φτάσει στο μηδέν,να μεταφέρεται σε ένα άλλο σημείο στο οποίο ο counter ξαναπαίρνει την τιμή 256,οι buffers θα παίρνουν τις αρχικές διευθύνσεις,θα μεταφέρεται στην αρχή του interrupt και τα επόμενα δεδομένα θα αρχίσουν να γράφονται πάνω στα παλιά.



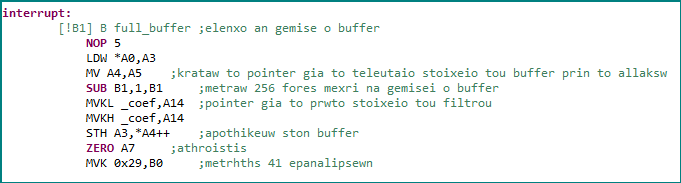
**Άσκηση 4.7 – Υλοποίηση FIR:**

**1. Γράψτε μια ρουτίνα assembly που υλοποιεί το FIR φίλτρο τροποποιώντας το πρόγραμμα εσωτερικού γινομένου που έχετε γράψει στο εργαστήριο 2. 2. Συνδυάστε τη λειτουργία φιλτραρίσματος FIR με την είσοδο δεδομένων από τον codec με την μέθοδο των διακοπών που γράψατε στο εργαστήρι 3. Είναι απαραίτητο να αρχικοποιήσετε τον AIC23 codec όπως κάνατε στο εργαστήριο 3. Ο κώδικάς σας θα πρέπει να φιλτράρει τα δεδομένα που έρχονται από τον codec και να στέλνει στην έξοδο κάθε φορά το αποτέλεσμα. Και τα δύο κανάλια υποβάλλονται σε επεξεργασία. Χρησιμοποιήστε τα αρχεία συντελεστών που σας παρέχονται. Αρχικά εφαρμόστε το lowpass φίλτρο τάξης 40 με αποκοπή στα 10kHz που έχει δημιουργηθεί με τη firpm. 3. Συνδέστε την έξοδο του υπολογιστή ή κάποιου smartphone που διαθέτουν λογισμικό tone generator, με την είσοδο (line in) του codec. Δημιουργήστε ένα ημιτονοειδές σήμα εισόδου στα 5kHz. Σχεδιάστε (γραφικά) μέσω του CCS και παρατηρήσετε την είσοδο και την έξοδο του codec.4. Εισάγετε λευκό θόρυβο (wn.wav στο eclass) στην είσοδο του codec και παρατηρήστε το φάσμα της εξόδου. Μπορείτε να δείτε την απόκριση συχνότητας του φίλτρου; 5. Επαναλάβετε τα παραπάνω βήματα χρησιμοποιώντας όλα τα άλλα φίλτρα που σας παρέχονται.**

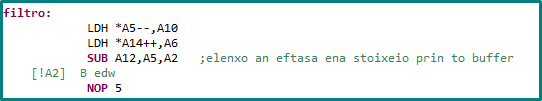
Για την υλοποίηση του FIR φίλτρου,αρχικά ορίζουμε 2 buffer για την είσοδο και για την έξοδο του.Συμπεριλαμβάνουμε με .include το LowpassFir1.asm που περιέχει τους συντελεστές του φίλτρου, κάνουμε τις απαραίτητες αρχικοποιήσεις και επιπλέον αποθηκεύουμε την αρχή και το τέλος του σήματος,για να μπορέσουμε να κάνουμε κυκλική διευθυνσιοδότηση.



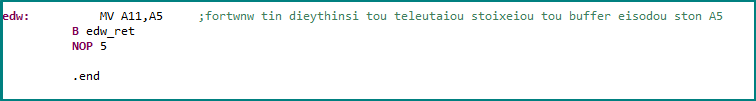
Παρακάτω παρουσιάζεται ο ISR του interrupt που «έρχεται».Σε αυτό διαβάζουμε την τιμή εισόδου από τον DRR,κρατάμε κάθε φορά τη διεύθυνση του τελευταίου στοιχείου σε έναν καταχωρητή,φορτώνουμε τα φίλτρα,μειώνουμε τον counter για το μέγεθος του buffer και αποθηκεύουμε το δεδομένο στον buffer\_right κάθε φορά στην επόμενη θέση.Τέλος, ορίζουμε άλλον έναν counter για να υπολογίζουμε 41 επαναλήψεις δηλαδή όσοι και συντελεστές του φίλτρου.



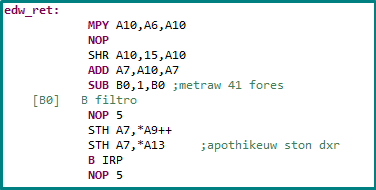
Παίρνουμε έτσι δείγματα από το τελευταίο πρoς τα πίσω και τα φίλτρα κανονικά.



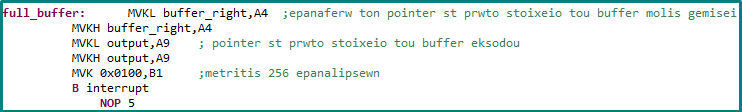
Σε περίπτωση που έχω φτάσει εξω απτον buffer τότε κάνω branch στο edw, όπου επαναφέρω στον Α5 τη διεύθυνση του τελευταίου στοιχείου,και συνεχίζω από το σημείο που ήμουν δηλαδη στο edw\_ret.



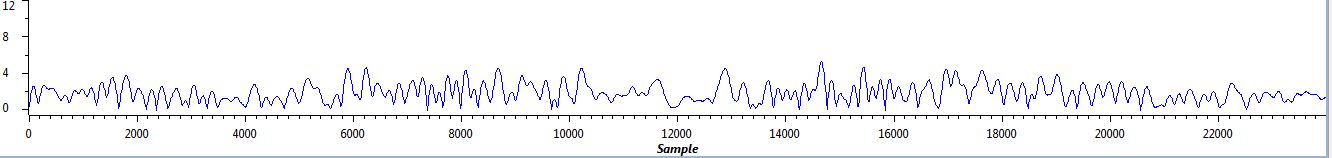
Αν δεν έχω φτάσει απλά συνεχίζω στο edw\_ret,όπου εκτελείται με τον γνωστό τρόπο η πράξη της συνέλιξης.Επίσης για κάθε στοιχείο που φιλτράρεται μειώνω τον counter των 41 επαναλήψεων, ώστε όσο έχω συντελεστές να επαναλαμβάνεται η διαδικασία του φιλτραρίσματος.



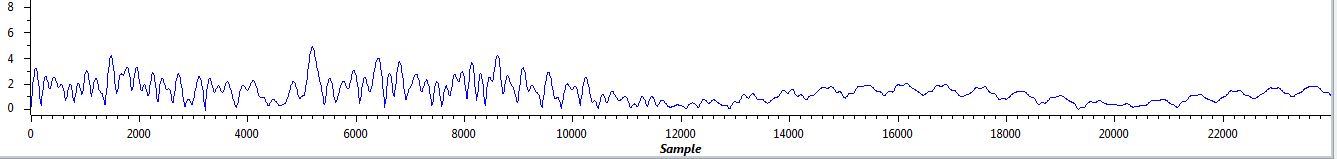
Τέλος, έχουμε τον full\_buffer από το προηγούμενο ερώτημα για να επαναφέρουμε τους buffers.



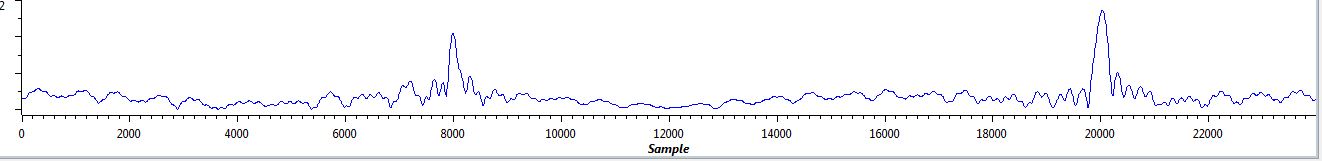
**FFT ΕΙΣΟΔΩΝ ΚΑΙ ΕΞΟΔΩΝ:**

Όταν έχουμε ως είσοδο λευκό θόρυβο : ****

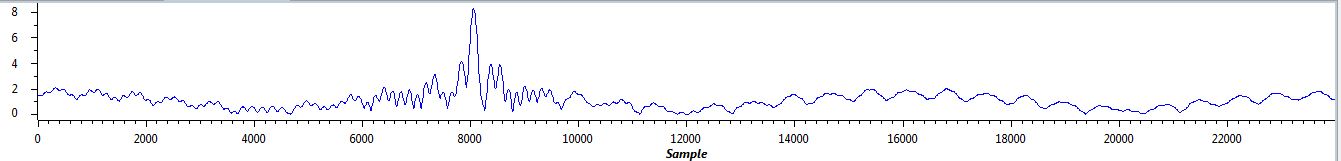
Η έξοδος: Το φίλτρο κόβει στα 10ΚHz

****

Όταν η είσοδος είναι 2 ημίτονα 8 και 20 KHz:

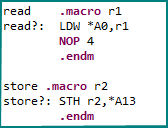
****

Στην έξοδο έχει κοπεί το ημίτονο στα 20 ΚΗz:

****

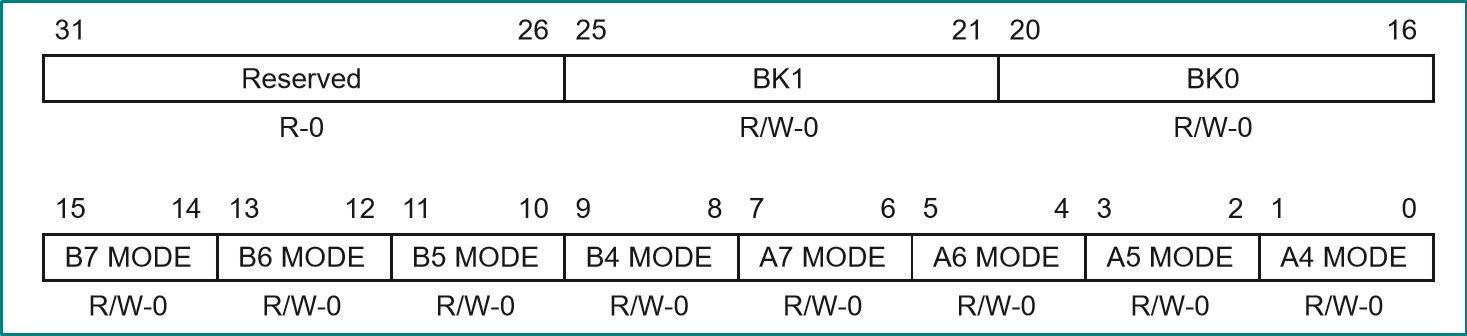
**Άκηση 4.8/4.9: Τα ερωτήματα παρουσιάζονται μαζί σε έναν καινούριο κώδικα που υλοποιείται με χρήση macro για τις διαδικασίες receive/transmit και κυκλικού buffer.**

Οι μακροεντολές που χρησιμοποιούνται ορίζονται στην αρχή εκτός της entry και καλούνται στα σημεία του προγράμματος που προηγουμένως κάναμε ανάγνωση των δεδομένων από τον DRR και μετάδοση των δεδομένων στον DXR αντίστοιχα.Το ερωτηματικό στα label που βρίσκονται μέσα στις macro χρειάζεται ώστε να μην συγχαίονται διάφορες κλήσεις (αν υπάρχουν). Για να καλέσουμε τις macro στο πρόγραμμα γράφουμε read Α3,write A7 που ειναι οι καταχωρητές που περιέχουν τα δεδομένα από την εισόδο και από το αποτέλεσμα της συνέλιξης(ορίζουμε δηλαδή τα r1,r2).

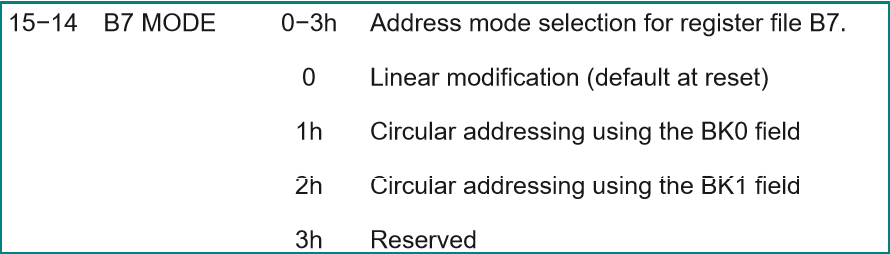


Το μεγαλύτερο ενδιαφέρον όμως παρουσιάζει η χρήση του κυκλικού buffer.Κάτι τέτοιο μας παρέχει τη διευκόλυνση να μην ελέγχουμε εμείς κάθε φορά το μέγεθος του buffer (με τις 256 επαναλήψεις)έτσι ώστε να γράφεται το δεδομένο στη σωστή θέση αν γεμίσει, αλλά γίνεται μόνο του.Αυτομάτως λοιπόν βγάζουμε από τον προηγούμενο κώδικα το label full\_buffer που επαναφέρει αρχικές συνθήκες στους buffers, όπως και το edw αφού πλέον δεν κρατάμε pointer με τη διεύθυνση του buffer\_right.

Για να ορίσουμε λοιπόν έναν κυκλικό buffer γράφουμε στον καταχωρητή AMR,ο οπoίος έχει προφανώς 32 bit.Όπως φαίνεται και παρακάτω τα 16 LSB χρησιμοποιούνται για να επιλέξουν τον τρόπο λειτουργίας για κάθε έναν από τους 8 επιτρεπόμενους καταχωρητές Α4-Α7,Β4-Β7 και τα 10 υψηλότερα ορίζουν το μέγεθος του buffer Ν.



Στην προκειμένη περίπτωση,χρησιμοποιούμε τους Β5,Β6,Β7, με Β5,Β6 να δείχνουν στον ΒΚ0 και Β7 νστον ΒΚ1.Γράφουμε δηλαδή από 1 στα αντίστοιχα Bit των Β5/6 και 2 στα bit του Β7 σύμφωνα με τον πίνακα παρακάτω.

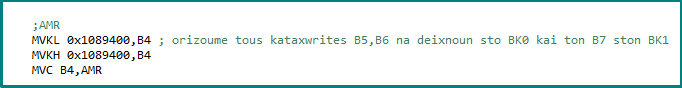


Το μέγεθος Ν που γράφεται στα bits 16-20,21 -25 υπολογίζεται από:

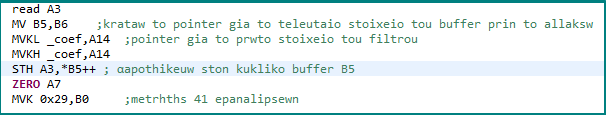
To μέγεθος του buffer είναι 2^(N+1) bytes. Επομένως 512= 2^(N+1) => N=8.

Τελικά ο αριθμός που γράφεται στον AMR είναι 0x1089400.

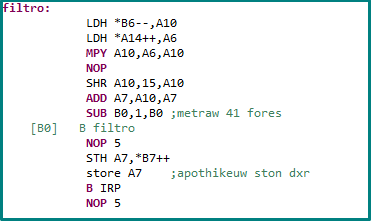
Στον κώδικα προσθέτουμε την κατάλληλη αρχικοποίηση για τον AMR μαζί αυτές των προηγούμενων ερωτημάτων.



Στη συνέχεια η δομή είναι η ίδια δηλαδή ένας ατέρμον βρόγχος και από κάτω η ISR της διακοπής.Στην ISR αρχικά τοποθετούνται τα στοιχεία της εισόδου και του φίλτρου σε καταχωρητές:



Και αμέσως μετά εκτελείται το φίλτρο,με την ίδια λογική,δηλαδή φορτώνω «προς τα πίσω» δείγματα και «προς τα εμπρός»,τα πολλαπλασίαζω,κάνω shift για να τα μεταφέρω μία θέση μέσα και αθροίζω το τελικό αποτέλεσμα στον Α7.Τέλος, αποθηκέυω στον κυκλικό buffer εξόδου αλλά και στον DXR με τη macro store A7.



ΣΗΜΕΙΩΣΗ: Οι κώδικες δίνονται ολόκληροι σε ξεχωριστά αρχεία.Εδώ χρησιμοποιήθηκαν μόνο τμήματα για την καλύτερη εξήγηση των υλοποιήσεων.