LAB4: Digital Modulation for Sound Signals

Δημήτριος Τσαλαπάτας, ΑΕΜ: 03246

22 Ιανουαρίου 2023

Περιεχόμενα

1 Περίληψη

Για την υλοποίηση του Sound-Driver χρειάζεται να δημιουργήσουμε δύο controllers. Ένα για την εγγραφή ήχου και αποθήκευση του στην BRAM και ένα για την αναπαραγωγή του μηνύματος που γράφτηκε στην BRAM μέσω της θύρας audio jack. Υλοποιήθηκαν 2 FSM's και κρίθηκε απαραίτητη η χρήση του MMCM για παραγωγή διαφορετικών ρολογιών πέρα από αυτό της πλακέτας.

2 Εισαγωγή

Η τέταρτη εργαστηριακή άσκηση έχει σαν στόχο την υλοποίηση ενός sound driver πάνω στην πλακέτα της FPGA. Πιο συγκεκριμένα απαιτείται με το πάτημα ενός κουμπιού να ηχογραφείται μέσω του μικροφώνου ένα μήνυμα διάρκειας 0.015 sec και να αποθηκεύεται σε μία μνήμη BRAM της FPGA. Στην συνέχεια θα πρέπει με το πάτημα ενός άλλου κουμπιού να διαβάζεται αυτό το μήνυμα από την BRAM και να μπορεί να αναπαραχθεί για περίπου 5 sec επαναλαμβανόμενο από την θύρα audio jack της πλακέτας. Χρειάζεται να χρησιμοποιηθεί το MMCM για παραλλαγή διαφορετικών ρολογιών λόγω του μικροφώνου κυρίως αλλά και το primitive της BRAM για να μπορέσουμε να την χρησιμοποιήσουμε. Επίσης η υλοποίηση που περιγράφεται παρακάτω αποτελείται από 5 βασικά modules και 3 testbenches ενώ έχουν υλοποιηθεί και δύο FSM's.

3 Μέρος Α - Υλοποίηση Οδηγού Μικορφώνου και Αποσειριοποίηση Δεδομένων

3.1 Υλοποίηση MMCM

Αρχικά για την υλοποίηση του οδηγού του μικροφώνου απαιτείται να παραχθεί ένα ρολόι με συχνότητα 2 MHz ώστε να στέλνετε στο μικρόφωνο και να δειγματολειπτεί. Για την παραγωγή του ρολογιού χρησιμοποιήθηκε η μονάδα MMCM όπου εφόσον δεν ήταν δυνατή η παραγωγή ρολογιού από τα 100 MHz της πλακέτας στα 2 που απαιτείται, χρειάζεται να γίνει χρήση της λειτουργίας `CASCADE`. Πιο συγκεκριμένα, με την μέθοδο cascading χρησιμοποιούμαι το port `CLKOUT_6` και το `CLKOUT_4` όπου προστίθεται ο όρος `CLKOUT_DIVIDE_4` για να φτάνει σε μεγαλύτερη τιμή από 128. Η συγκεκριμένη λειτουργία περιγράφεται αναλυτικά στο αντίστοιχο manual σελίδα 76 εδώ
Στην συνέχεια με την χρήση των εξισώσεων που φαίνονται παρακάτω, υπολογίζονται οι τιμές των υπόλοιπων μεταβλητών ώστε να παραχθεί το απαιτούμενο ρολόι. Πα-

$$F_{VCO} = F_{CLKIN} \times \frac{M}{D} \qquad \text{Equation 3-1}$$

$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O} \qquad \text{Equation 3-2}$$

Σχήμα 1: MMCM equations

ράλληλα θα πρέπει να παραχθεί από το ίδιο module του MMCM ένα ακόμα ρολόι από άλλο port (port 1) στα 200 MHz με το οποίο θα λειτουργεί το υπόλοιπο σύστημα.

Έτσι προκύπτουν οι παρακάτω τιμές στις μεταβλητές: $M = 10, CLKOUT1DIVIDE = 5, CLKOUT4DIVIDE = 5, CLKOUT6DIVIDE = 100$, ενώ $DIVCLKDIVIDE = 1$.
Με αυτόν τον τρόπο παράγονται τα δύο ρολόγια που χρειάζονται για την εργασία.

3.2 Υλοποίηση BRAM

Στην συνέχεια χρειάζεται να ορίσουμε ένα module μνήμης ώστε να αποθηκεύονται τα δεδομένα που λαμβάνουμε από το μικρόφωνο και να μπορούμε να τα αναπαράγουμε όποτε επιθυμούμε. Για την υλοποίηση χρησιμοποιήθηκε το module RAMB36E1 όπου αφού διαβάστηκε αναλυτικά το manual, επιλέχθηκε να δουλεύει στο TDP (True Dual Port) mode ώστε να χρησιμοποιηθεί μόνο το ένα port και για να γράφει και για να διαβάζει.
Ορίζουμε τα Read και Write width ίσα με 36 καθώς δηλώνει ότι έχουμε πρόσβαση σε 32 bit δεδομένων και 4 bit parity. Δεν χρησιμοποιούμε στην προκειμένη περίπτωση τα parity bits αλλά μόνο τα data. Επιπλέον, επιλέγουμε ”WRITE-First” ώστε να είναι εμφανές με την μία στο simulation η τιμή που γράφτηκε στο output. Από τα ports χρησιμοποιείται ένα για το output (`DOADO`), ένα για `clock`, ένα για `reset`, ένα για το `enable` της BRAM, το bus για το `input` των δεδομένων που πρόκειται να γραφτούν, το `WEA` (Write enable) όπου όταν είναι 1 γράφει στην BRAM και τέλος η διεύθυνση (`addr`). Ιδιαίτερη έμφαση απαιτείται στο WEA το οποίο για 32 bit width είναι 4bit και κάθε bit δηλώνει πόσα bytes από το input θα γραφτούν. Επίσης, προσοχή απαιτείται και στο addr καθώς είναι στα 16 bit όμως για την περίπτωση μας, valid είναι μόνο τα [14:5], όπως φαίνεται και στην παρακάτω εικόνα.

Table 1-13: **Port Aspect Ratio for RAMB36E1 (in TDP Mode)**

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
1	15	32,768	[14:0]	[0]	NA
2	14	16,384	[14:1]	[1:0]	NA
4	13	8,192	[14:2]	[3:0]	NA
9	12	4,096	[14:3]	[7:0]	[0]
18	11	2,048	[14:4]	[15:0]	[1:0]
36	10	1,024	[14:5]	[31:0]	[3:0]
1 (Cascade)	16	65536	[15:0]	[0]	NA

Σχήμα 2: BRAM table

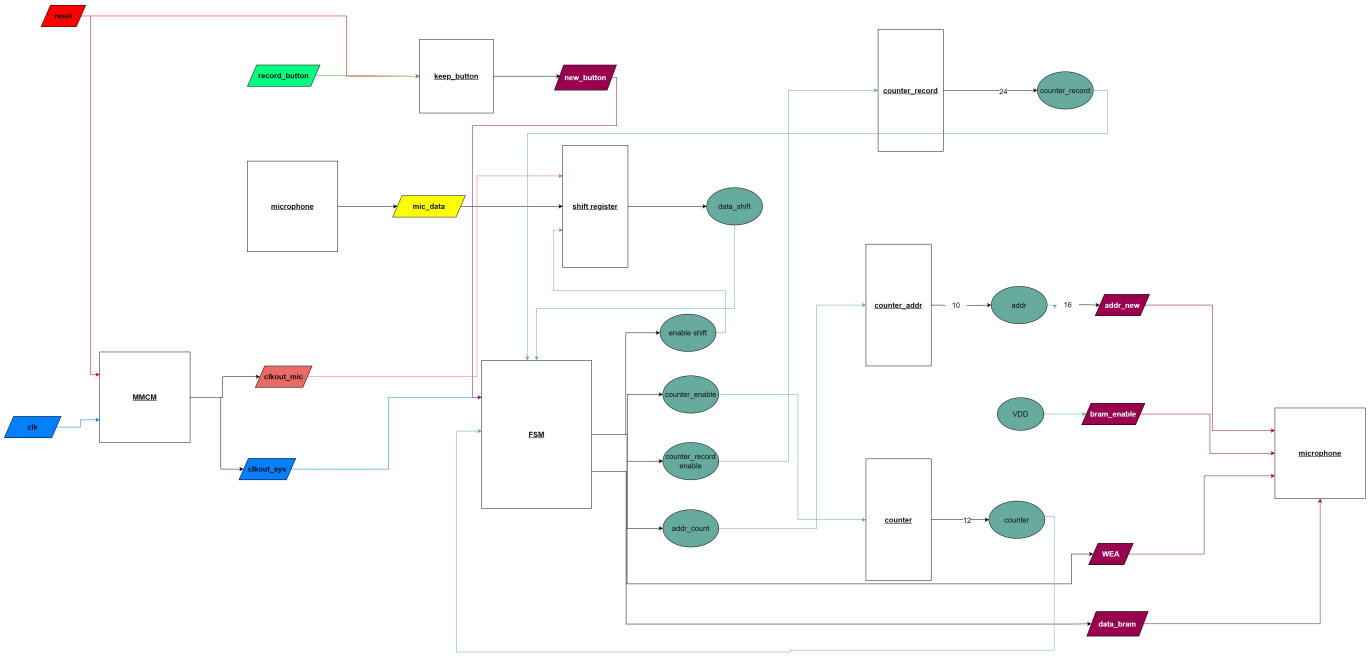
3.3 Υλοποίηση Deserializer

Για να μην γράφουμε συνεχώς δεδομένα στην BRAM με το που λαμβάνουμε κάθε bit από το μικρόφωνο, επιλέγουμε να χρησιμοποιήσουμε έναν deserializer δεδομένων ό-που λαμβάνει 1 προς 1 δεδομένα από το μικρόφωνο τα αποθηκεύει και τα μετατρέπει σε bus των 32 bit.
Για να υλοποιηθεί το συγκεκριμένο σύστημα χρησιμοποιείται ένας `Shift_Register` (Serial to Pararel) το οποίο λειτουργεί με το ρολόι του μικροφώνου ώστε να ενεργοποιείται κά-θε φορά που στέλνεται ένα δεδομένο. Παρόλα αυτά έχει και ένα σήμα enable το οποίο προέρχεται από την FSM και λόγω του ότι λειτουργούν σε διαφορετικά ρολόγια, μεταξύ

τους παρεμβάλετε ένας συγχρονιστής με 2 flip flop για να αποφευχθεί παραβίαση στο setup χρόνο του `shift_register`.

3.4 Υλοποίηση Microphone Controller

Στη συνέχεια, χρειάζεται να ενώσουμε όλα τα παραπάνω και να δημιουργήσουμε ένα σύστημα το οποίο με το πάτημα ενός κουμπιού θα αποθηκεύει στην μνήμη ένα μήνυμα που ηχογραφείται από το μικρόφωνο διάρκειας 0.015 sec.
Παρακάτω φαίνεται το αναλυτικό `Dataflow` για το μέρος Α, ενώ τα υποσυστήματα αναλύονται παρακάτω.



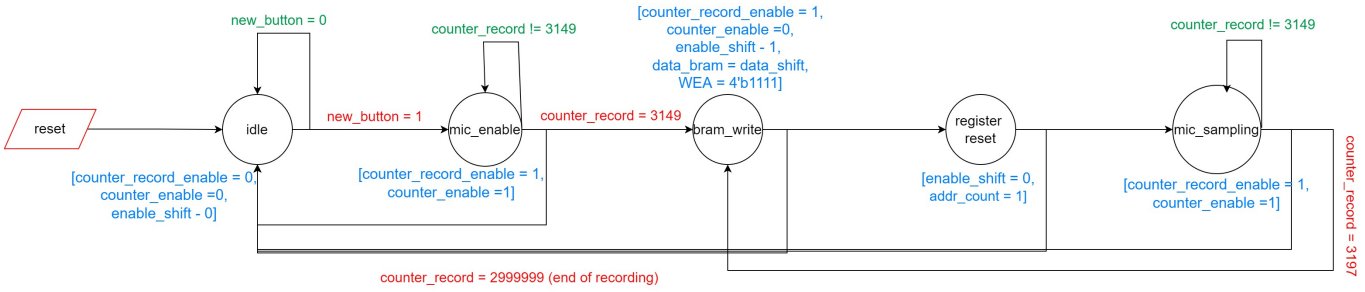
Σχήμα 3: Dataflow Part A

3.4.1 Always Blocks

Στην υλοποίηση μας χρειάστηκε εκτός από τον shift register που αναφέρθηκε προηγουμένως, να χρησιμοποιήσουμε και άλλα always blocks για τις λειτουργίες που χρειαζόμαστε. Χρησιμοποιούμε τρία ακολουθιακά always για να δημιουργήσουμε τρεις counters με το ρολόι του συστήματος. Ο πρώτος χρησιμοποιείται για να αλλάζει τις καταστάσεις στην FSM, ο δεύτερος για να μετρήσει την διάρκεια της εγγραφής ώστε να ξέρουμε πότε να σταματήσει το σύστημα, και ο τελευταίος για να αυξάνει την τιμή της διεύθυνσης που δείχνει στην BRAM. Όλοι έχουν σήμα enable το οποίο ελέγχεται από την FSM. Επιπλέον, έτσι όπως έχει δημιουργηθεί η υλοποίηση, απαιτείται να παραμένει μόνιμα στο πατημένο το κουμπί του record όταν γράφει, οπότε χρησιμοποιείται ένα ακολουθιακό always όπου όταν λαμβάνει το πάτημα του κουμπιού κρατάει τον παλμό στο 1, για όσο χρειάζεται.

3.4.2 FSM part A

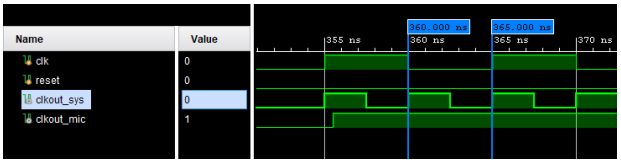
Για να ελέγξουμε όλες τις καταστάσεις του συστήματος δημιουργούμε μία FSM τύπου MOORE με 5 states. Αρχική κατάσταση αποτελεί η `idle` όπου βρίσκεται το σύστημα μετά από κάθε reset για αρχικοποίηση, ενώ με το πάτημα του αντίστοιχου κουμπιού περνάμε στην επόμενη κατάσταση `mic-enable` όπου αρχικοποιεί τους counters του συστήματος και σκοπός του είναι να μετράει τον χρόνο δειγματοληψίας των πρώτων 32 bit του μικροφώνου και να σταματήσει στην μέση του τελευταίου bit ώστε να προλάβει να κάνει τις αλλαγές που χρειάζεται προτού έρθει το επόμενο δεδομένο. Κάθε δεδομένο έχει διάρκεια 500 ns, οπότε έχουμε $500 * 32 = 16000ns$ όμως η περίοδος του ρολογιού του συστήματος είναι 5ns άρα ο counter θα πρέπει να έχει τιμή $16000/5 = 3200$, όμως εφόσον θέλουμε να φτάσουμε στην μέση του τελευταίου bit βάζουμε την τιμή 3150 και -1 για την αλλαγή του state προκύπτει 3149. Το επόμενο state είναι το `Bram-write` στο οποίο το WEA γίνεται ίσο με 1 και γράφεται η πρώτη 32 bit τιμή στην μνήμη. Ένα κύκλο μετά προχωράει στην κατάσταση `register-reset` όπου μηδενίζει τον shifter και ενεργοποιεί και τον counter της διεύθυνσης αυξάνοντας την κατά 1. Στον επόμενο κύκλο συνεχίζει στο `mic-sampling` όπου έχει την ίδια λειτουργία με το `mic_enable` αλλά πλέον εφόσον βρισκόταν ήδη στην μέση του τελευταίου δεδομένου ο counter διαρκεί 3200 κύκλους και αφαιρόντας τις μεταβάσεις για τα states γίνεται 3197. Στην συνέχει κάνει ένα loop πηγαίνοντας στο state `bram-write` και τρέχει έως ότου ο counter που μετράει τον χρόνο ηχογράφησης να γίνει ίσος με 3000000 δηλαδή 0.015 sec.
Παρακάτω φαίνεται το σχήμα της FSM για την συγκεκριμένη υλοποίηση.

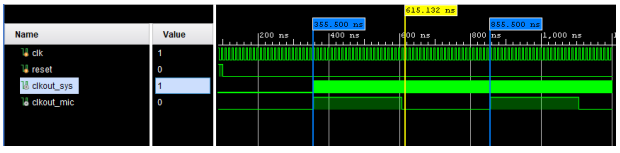


Σχήμα 4: Part A FSM

3.5 Επαλήθευση MMCM

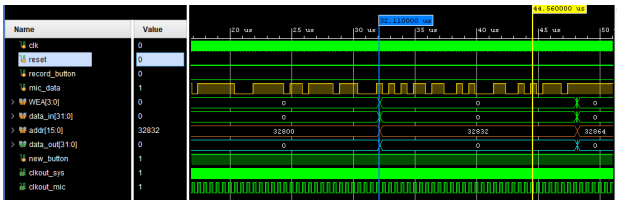
Η επαλήθευση του συστήματος ξεκίνησε από τον έλεγχο του MMCM για να σιγουρευτούμε ότι τα ρολόγια που παράγονται έχουν τις επιθυμητές τιμές. Δημιουργήθηκε ένα πολύ απλό testbench στο οποίο αρχικοποιήθηκε το ρολόι και το reset και προσομοιάστηκε το ρολόι της `FPGA` των 100 MHz και παρατηρήσαμε τις κυματομορφές που προκύπτουν, οι οποίες φαίνονται παρακάτω.

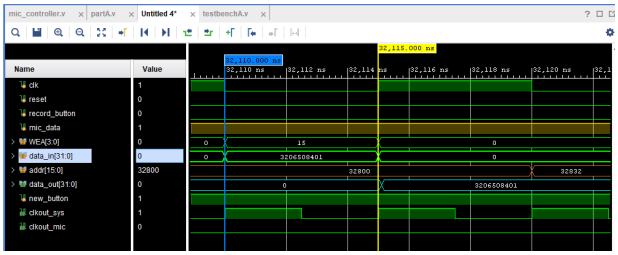




3.6 Επαλήθευση part A

Για να επαληθευτεί σωστά το `mic-controller` είναι απαραίτητο να προσομοιάσουμε το μικρόφωνο και τον τρόπο που μας δίνει δεδομένα. Στην συνέχεια αρκεί να παρατηρήσουμε τις κυματομορφές και να δούμε ότι γράφει το σωστό μήνυμα στην BRAM και στην σωστή διεύθυνση.
Αρχικά, αρχικοποιούμε τις τιμές του ρολογιού και του reset, ενώ επίσης και την τιμή του κουμπιού που πραγματοποιεί την εγγραφή. Στην συνέχεια στέλνουμε το πρώτο δεδομένο πάνω σε posedge clk του ρολογιού των 2 MHz και κάθε 500ns στέλνουμε το επόμενο bit. Στελνουμε 3 μηνύματα των 32 bit στην αρχή και στην συνέχεια άλλο ένα λίγο πιο μετά και ένα προς το τέλος για να βεβαιωθούμε ότι λειτουργεί σε όλη την διάρκεια. Παρατηρούμε λοιπόν τις κυματομορφές και πιο συγκεκριμένα τα σήματα addr και data out όπου δείχνουν σε ποια διεύθυνση γράφεται το bus και ποια είναι η τιμή του. Στις παρακάτω εικόνες φαίνεται η εγγραφή του δεύτερου δεδομένου που στέλνουμε στο testbench με τιμή δεκαδική '3206508401' οπότε παρατηρούμε ότι γράφεται σωστά.





Στην συνέχεια επαναλαμβάνουμε την διαδικασία για όλα τα δεδομένα που στείλαμε και επιβεβαιώνουμε την λειτουργικότητα του συστήματος.

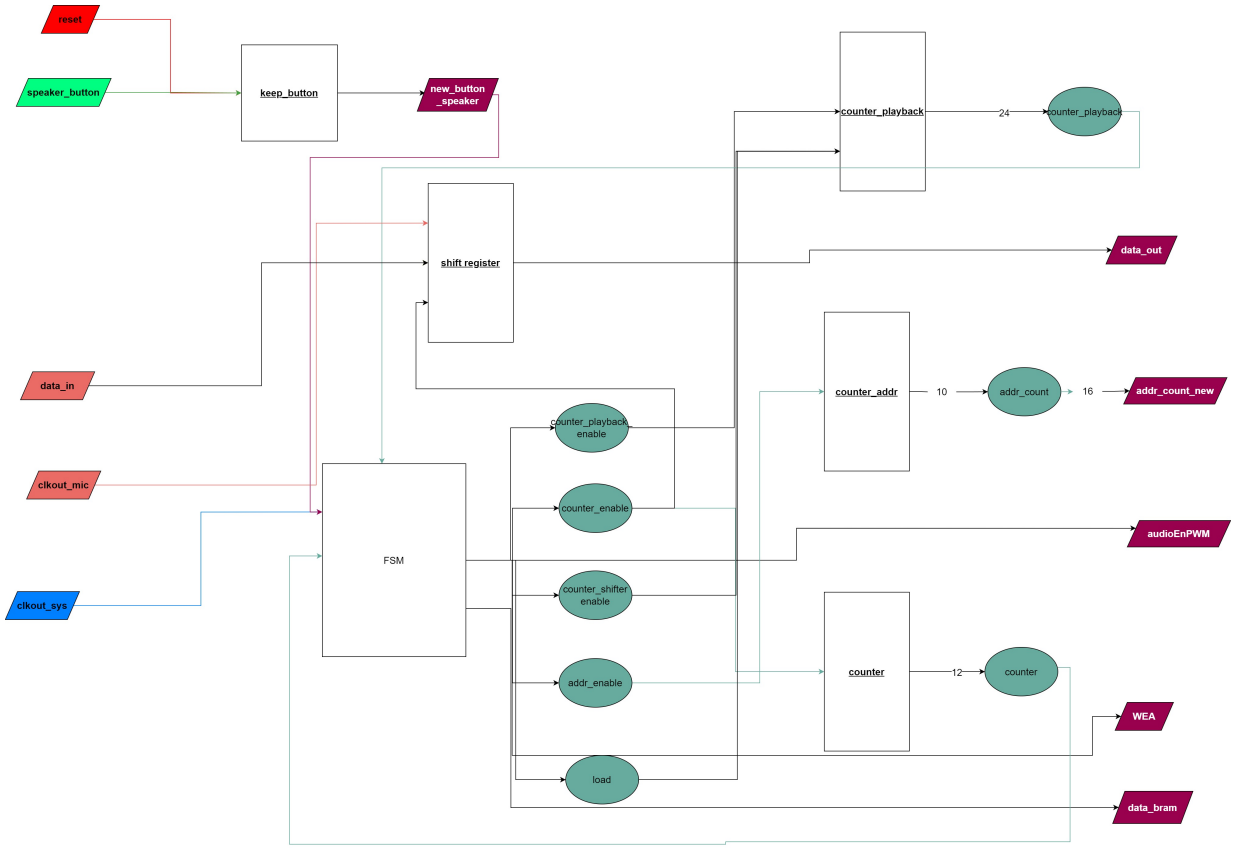
4 Μέρος Β - Υλοποίηση Οδηγού Αναπαραγωγής Ήχου και Σειριοποίηση Δεδομένων

4.1 Περιγραφή Υλοποίησης Σειριοποιητή Δεδομένων

Τα δεδομένα πρέπει να διαβάζονται από την BRAM σε busses των 32 bit, όμως στο φίλτρο του ηχείου πρέπει να στέλνονται σειριακά δηλαδή ένα προς ένα. Για να επιτευχθεί αυτό απαιτείται η υλοποίηση ενός `Shift␣Register␣Pararel␣to␣Serial`, όπου έχει αντίστροφη λειτουργία σε σχέση με εκείνον του μέρους Α. Αποτελείται από ένα ακολουθιακό always block και χρησιμοποιεί το ρολόι του συστήματος (200 MHz) ενώ επίσης έχει ένα σήμα enable το οποίο ελέγχεται από την FSM (αναλύεται παρακάτω). Επιπλέον, εφόσον λειτουργεί με το ρολόι του συστήματος είναι απαραίτητο να μην κάνει σε κάθε posedge clock shift αλλά μόνο όταν δέχεται δεδομένα από το μικρόφωνο, οπότε ελέγχεται και από ένα άλλο counter και μόνο όταν γίνει εκείνο ίσος με 100 (διότι $100 * 5 = 500ns$) κάνει shift τα δεδομένα και τα περνάει στο φίλτρο του ηχείου.

4.2 Περιγραφή Υλοποίησης Οδηγού Αναπαραγωγής Ήχου

Στην συνέχεια χρειάζεται να υλοποιηθεί ο οδηγός αναπαραγωγής ήχου, όπου έχει ως σκοπό να διαβάζει από την μνήμη που έχει γραφτεί το μήνυμα που έγινε ηχογράφηση και στην συνέχεια να το περνάει από τον `Deserializer` και να το στέλνει στο φίλτρο του ηχείου, ενώ επίσης θα πρέπει να επαναλαμβάνει το μήνυμα συνεχώς για διάρκεια περίπου 5 sec. Παρακάτω φαίνεται το αναλυτικό Dataflow για το μέρος B, ενώ τα υποσυστήματα και η FSM αναλύονται στην συνέχεια αναλυτικά



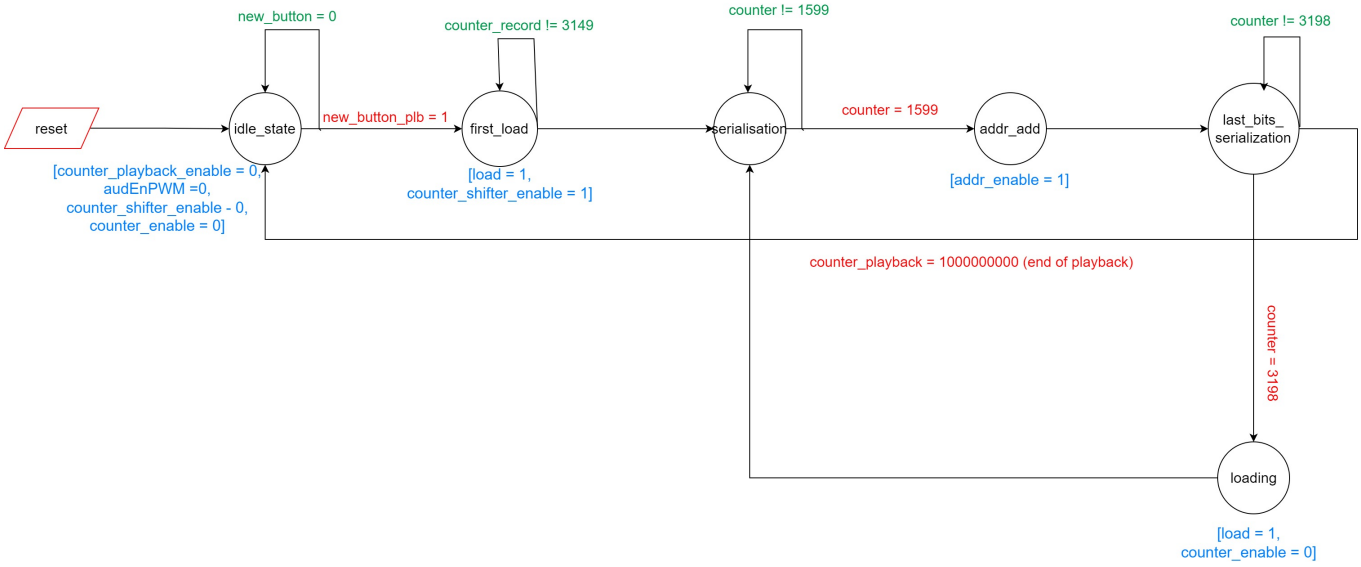
Σχήμα 5: Dataflow part B

4.2.1 Περιγραφή Always Blocks

Για την υλοποίηση του οδηγού απαιτείται η χρήση αρκετών υποκυκλωμάτων κυρίως `Counters` με την χρήση ακολουθιακών `Always␣Blocks`. Χρησιμοποιούνται τέσσερεις counters οι οποίοι λειτουργούν όλοι με το ρολόι του συστήματος (200 MHz). Ο πρώτος (`counter-shifter`) χρειάζεται για να μετράει πότε θα γίνει το shift των δεδομένων, ενώ ο δεύτερος counter (`counter`) μετράει χρόνο για το πότε θα αλλάζουν τα states της FSM. Επιπλέον, χρειαζόμαστε ακόμα έναν ώστε να αλλάζει την τιμή του address (`addr-count`) για την bram. Τέλος, ο τέταρτος (`counter-playback`) μετράει τον χρόνο της αναπαραγωγής όλου του μηνύματος, δηλαδή τα 5 sec.
Στην συνέχεια, χρησιμοποιείται ένα ακόμα ακολουθιακό always block για τον shift register που αναφέρθηκε προηγουμένως και άλλο ένα για να αλλάζει τις καταστάσεις της FSM. Τέλος, χρησιμοποιείται και ένα συνδιαστικό για να παράγει τα σήματα ελέγχου της FSM ανάλογα με το state.

4.2.2 FSM part B

Η FSM που χρησιμοποιούμε είναι τύπου `Moore` και αποτελείται από 6 βασικές καταστάσεις. Αρχικά, για την αρχικοποίηση της FSM είναι το state `idle-state` όπου μετά από κάθε reset μεταφέρεται το σύστημα εκεί και σταματάει όλους τους counters ενώ παράλληλα αρχικοποιεί τα σήματα. Στην συνέχεια όταν πατηθεί το κουμπί για το `Playback` μεταφερόμαστε στην κατάσταση `first-load` όπου κάνει το πρώτο load στον shifter και ενεργοποιεί τον `shifter-counter`. Στον αμέσως επόμενο κύκλο μεταφέρεται το σύστημα στο `Serialization` όπου παραμένει εκεί έως ότου περάσουν τα πρώτα 16 bit ενός μηνύματος. Ακολουθεί η κατάσταση `addr-add` όπου αυξάνει την διεύθυνση κατά 1, το οποίο γίνεται στην μέση του μηνύματος ώστε η διεύθυνση να έχει σταθερή τιμή αρκετά πριν χρειαστεί να διαβάσουμε ξανά από την μνήμη της. Ένα κύκλο μετά πηγαίνουμε στο `last-bit-serialization` όπου περιμένουμε μέχρι να γίνει σειριοποίηση και των υπόλοιπων bits από το bus και μετά όταν ο counter γίνει ίσος με 3200 όπου είναι ο χρόνος για να σειριοποιθούν και τα 32 bit, περνάμε στο επόμενο state. Έαν όμως, ο counter-playback γίνει ίσος με 1000000000 (δηλαδή φτάσει στα 5 sec) επιστρέφουμε στο state `idle-state`, αλλιώς προχωράμε στο `loading` όπου γίνεται load το επόμενο δεδομένο και μηδενίζεται ο `counter` για να συνεχίσει για το επόμενο bus στο state `serialization`. Παρακάτω παρατίθεται το αναλυτικό σχήμα για την FSM του μέρους B



Σχήμα 6: FSM Part B

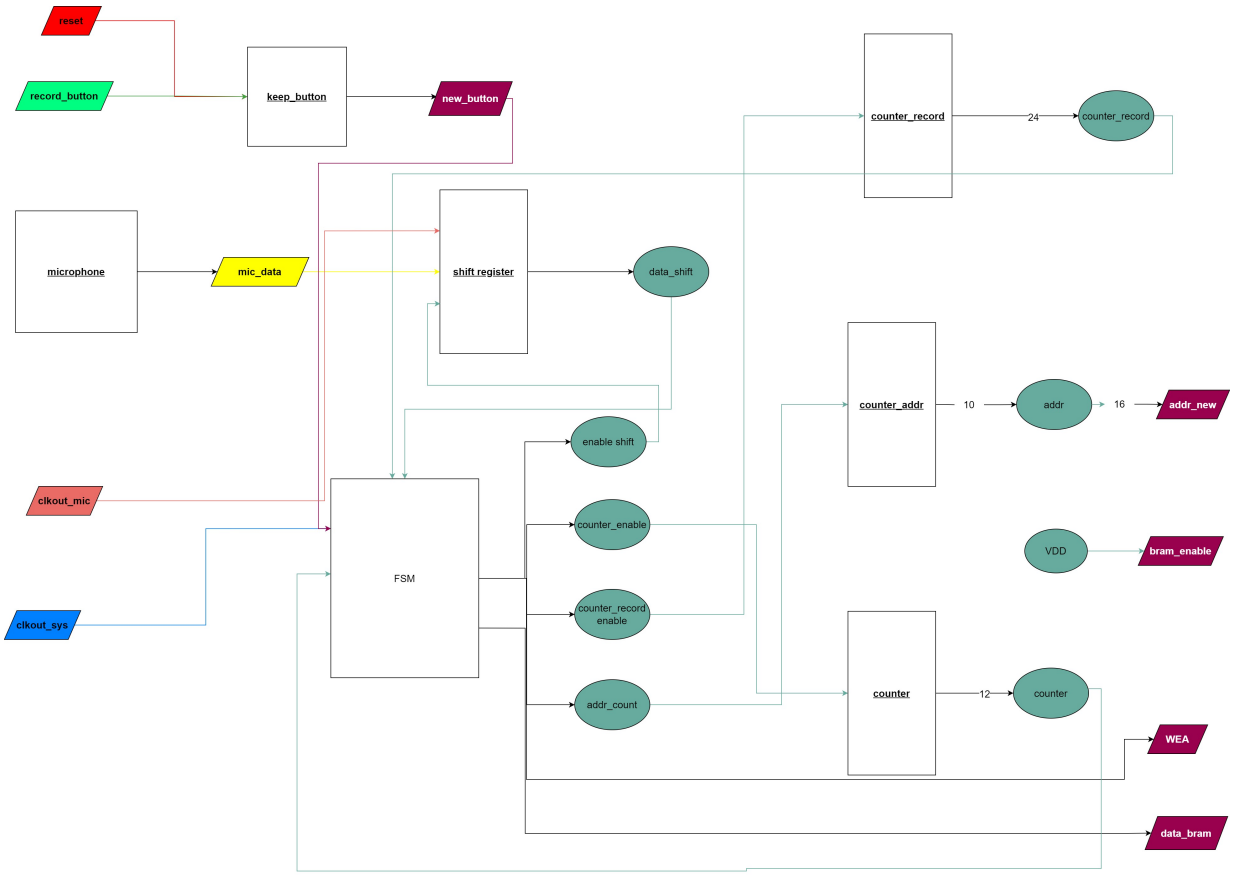
4.3 Έλεγχος Part B

Ο έλεγχος για το Part B έγινε όταν συνδέθηκε όλο το σύστημα μαζί ώστε να προσομοιωθεί πρώτα η διαδικασία της εγγραφής στην μνήμη και στην συνέχεια να παρατηρηθεί εάν διαβάζει σωστά από αυτήν. Οπότε παρατίθεται παρακάτω μετά από την περιγραφή της ένωσης όλου του κυκλώματος.

5 Ένωση Κυκλώματος - Υλοποίηση Sound-Driver

5.1 Περιγραφή υλοποίησης Sound-Driver

Για να ενωθεί όλο το κύκλωμα μαζί και να υλοποιηθεί ένα ορθό κύκλωμα `Sound-Driver` χρειάστηκε να γίνουν αλλαγές σε κάποια κομμάτια της υλοποίησης του μέρους Α. Αρχικά, εφόσον και τα δύο modules χρειάζονται πρόσβαση στην BRAM και στο MMCM δεν γίνεται instantiation τοπικά σε καθένα από αυτά, αλλά γίνεται στο top module και έχουν και τα δύο access στα inputs και outputs τους.
Οπότε το νέο dataflow του μέρους Α μετά τις αλλαγές παρατίθεται παρακάτω



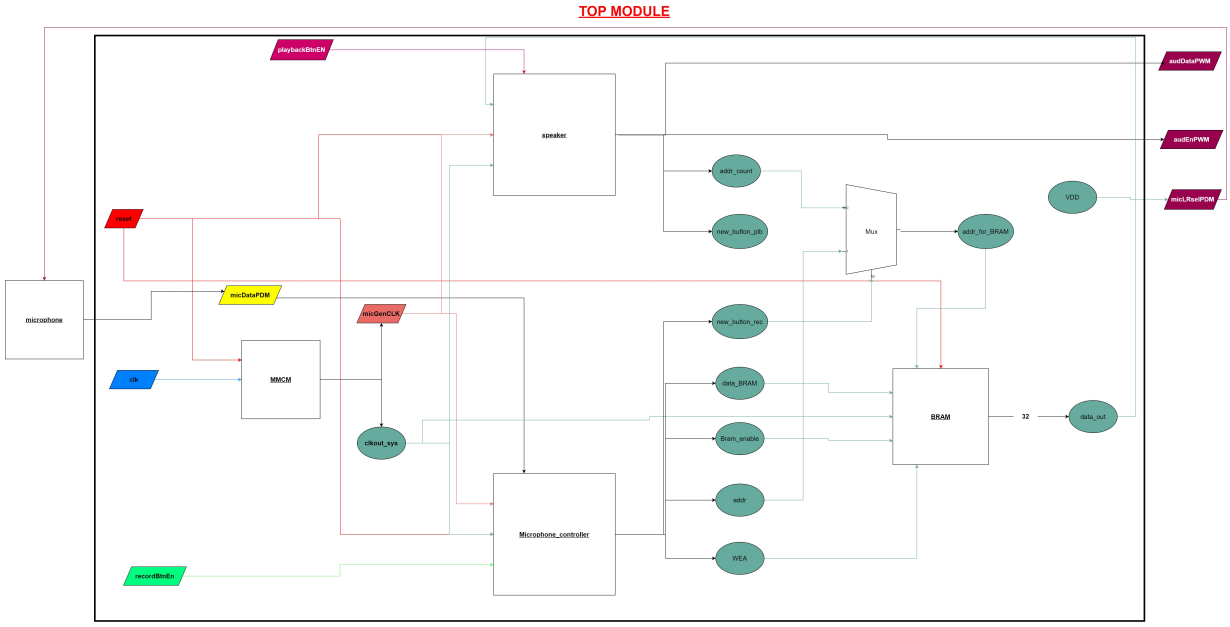
Σχήμα 7: Dataflow Part A second implementation

Ενδιαφέρον κομμάτι στο top module αποτελεί το γεγονός ότι εφόσον έχουμε ένα instantiation από την BRAM και πρέπει να αλλάζουμε το ποια διεύθυνση χρησιμοποιούμε με βάση το αν κάνουμε record ή playback, χρησιμοποιούμε ένα multiplexer όπου αποφασίζει με βάση το ποιο κουμπί είναι πατημένο την σωστή διεύθυνση.
Το κομμάτι του κώδικα φαίνεται παρακάτω

```
always @(posedge clkout_sys)
begin
    if(new_button_rec)
        addr_for_bram <= addr;
    else
        addr_for_bram <= addr_count;
end
```

Σχήμα 8: Multiplexer Code

Επιπλέον, παρακάτω φαίνεται το αναλυτικό dataflow για το ολικό σύστημα, δηλαδή για τον `Sound-Driver`

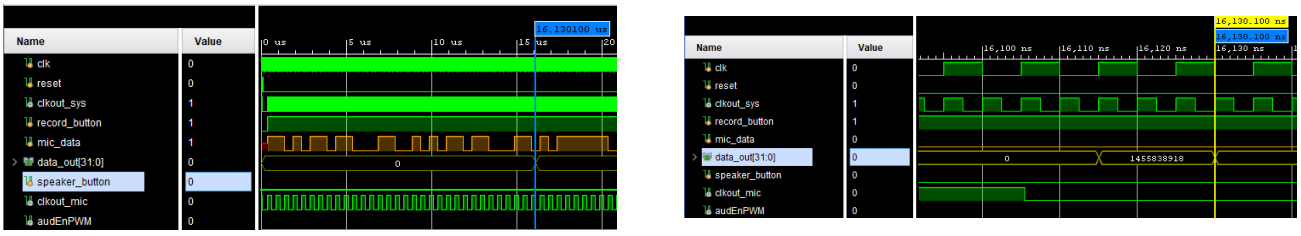


Σχήμα 9: Sound-Driver Dataflow

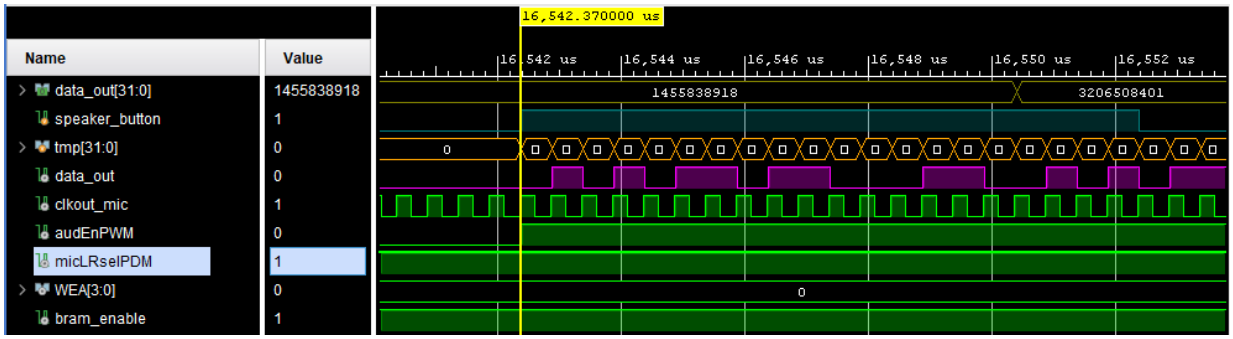
5.2 Έλεγχος Sound Driver

Για να πετύχουμε έναν ικανοποιητικό έλεγχο για το sound driver αλλά και για το μέρος Β, αρκεί να παρατηρήσουμε ότι τα δεδομένα αρχικά γράφονται σωστά στην μνήμη και στον χρόνο που πρέπει και ότι όταν πατηθεί το κουμπί για το playback, ότι διαβάζει σωστά από την μνήμη και ότι παράγει ένα προς ένα σωστά τα δεδομένα.
Για να το πετύχουμε αυτό δημιουργούμε ένα πλαίσιο δοκιμής στο οποίο προσομοιώνουμε το ρολόι της FPGA αρχικοποιούμε τα κατάλληλα σήματα και τοποθετούμε το sound driver module ώστε να το προσομοιώσουμε. Επιπλέον, προσομοιώνουμε το μικρόφωνο στέλνοντας όπως και στο μέρος Α δεδομένα με τον τρόπο που πρέπει να τα στέλνει δηλαδή πάνω στο posedge clock των 2 MHz, στέλνουμε τρία bus στην αρχή και μετά ένα στην μέση και ένα στο τέλος. Στην συνέχεια στέλνουμε σήμα για το playback

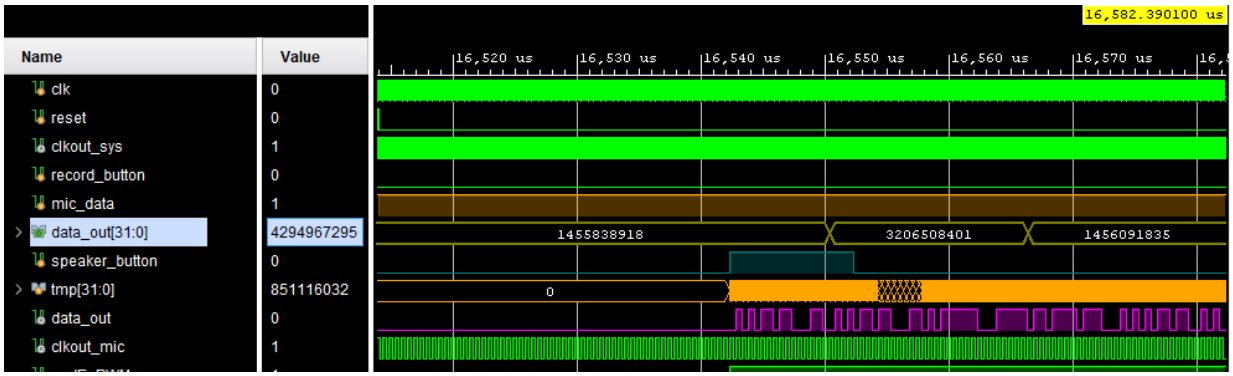
button και κοιτάμε αν το σειριακό output από τις κυματομορφές είναι τα δεδομένα που πρέπει.
Στις δύο εικόνες που ακολουθούν φαίνεται στο mic data τα δεδομένα που έρχονται από το μικρόφωνο και αρχικά στέλνουμε την τιμή ”1455838918” οπότε στην επόμενη εικόνα παρατηρούμε ότι την σωστή χρονική στιγμή γράφεται στην BRAM η σωστή τιμή. Επαναλαμβάνουμε την ίδια διαδικασία και για τα 5 busses δεδομένων που έχουμε ορίσει για να βεβαιωθούμε για την ορθή λειτουργία. Οι τιμές που στέλνουμε φαίνονται στο testbench.



Στην συνέχεια παρατηρούμε ότι όταν στείλουμε σήμα ότι πατήθηκε το κουμπί για αναπαραγωγή ήχου ξεκινάει να διαβάζει από την μνήμη και να στέλνει σειριακά δεδομένα. Ελέγχουμε λοιπόν ότι στο data out της BRAM υπάρχει η τιμή που επιθυμούμε και στην συνέχεια παρατηρούμε εάν ο σειριοποιητής κάνει σωστά την δουλειά του και βγάζει τα bits που πρέπει κοιτώντας τα πρώτα 32 bits. Επιπλέον, πρέπει να επαναλάβουμε την διαδικασία για όλες τις τιμές.
Στις παρακάτω εικόνες φαίνεται ότι ακολουθείται η παραπάνω λειτουργία.



Σχήμα 10: Waveforms Part B



Σχήμα 11: Waveforms Part B 2

Με τις παραπάνω προσομοιώσεις ελέγχουμε σε αρκετά μεγάλο βαθμό το σύστημά μας διότι ελέγχουμε την αρχή την μέση και το τέλος αφήνοντας έτσι μικρά περιθώρια για λάθος. Παρόλα αυτά αν και θεωρητικά και στα ενδιάμεσα στάδια λειτουργεί σωστά, το συγκεκριμένο testbench δεν μας καλύπτει. Επιπλέον, βάζοντας μικρότερους χρόνους από αυτούς των 0.015 και 5 sec ελέγχουμε ότι σταματάει την σωστή χρονική στιγμή.
Μετά από αυτήν την διαδικασία τρέχουμε στο Vivado synthesis και implementation και ξανά τρέχουμε τις προσομοιώσεις. Παρατηρούμε ότι δεν υπάρχουν latches ούτε και κάποιο ιδιαίτερο warning εξαιρώντας κάποια για σταθερές τιμές και κάποια warnings για τα ports του MMCM και της BRAM όπου μετά από έρευνα καταλήξαμε στο ότι δεν είναι απαραίτητα να συνδεθούν για τον σκοπό που τα χρειαζόμαστε. Μετά από αυτο παρατηρούμε ότι έχουμε τα ίδια αποτελέσματα και σε αυτές τις προσομοιώσεις και ήρθε η ώρα να δημιουργήσουμε το XDC και να παράξουμε το Bitstream.

5.3 Δημιουργία XDC και παραγωγή Bitstream

Για το `XDC` μελετάμε το manual της πλακέτας για να παρατηρήσουμε ποια pins αντιστοιχούν στα inputs και outputs που έχουμε. Αρχικά, τα δεδομένα από το μικρόφωνο λαμβάνονται από το pin H5, το micLRselPDM στο F5, το micGenCLK στο J5, ενώ το recordBtnEN επιλέγουμε να το βάλουμε στο κουμπί M17. Στην συνέχεια, το audDataPWM στο A11 και το playbackBtnEn επιλέγουμε να συνδεθεί στο κουμπί M18. Δυσεύρετο ήταν το pin για το σήμα audEnPWM όμως μετά από έρευνα στο internet σε ένα βίντεο αναφερόταν ότι είναι το D12. Το manual βρίσκεται εδώ προς το τέλος της σελίδας.
Στην συνέχεια παράγουμε το Bitstream το οποίο παράγεται χωρίς κάποιο πρόβλημα.

6 Προαιρετικό μέρος Εργασίας

6.1 Περιγραφή Υλοποίησης Προαιρετικού Μέρους

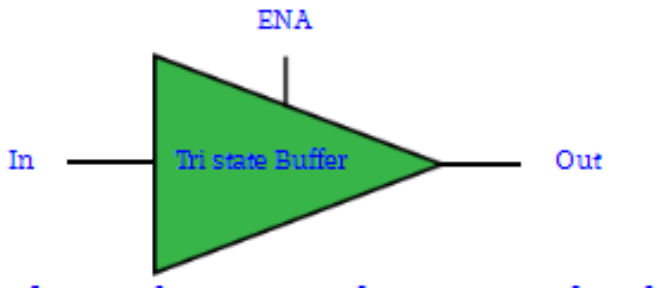
Για το προαιρετικό μέρος ζητείται από την εργασία να αποσυνδεθεί η BRAM και να συνδεθεί απευθείας το μικρόφωνο στο ηχείο όπου όμως χρειάζεται να γίνει σωστό conversion από PDM σε PWM ώστε να μπορεί να αναπαράγεται η φωνή σε πραγματικό χρόνο από το ηχείο.
Αρχικά, διαβάζοντας το manual πιο αναλυτικά παρατηρήθηκε ότι για το φίλτρο που υπάρχει πριν το ηχείο πρέπει όταν στέλνονται τα δεδομένα, να στέλνετε 0 όταν το bit είναι 0, αλλά να στέλνεται `High␣Impendance` δηλαδή Z όταν το bit είναι 1 ώστε να ρυθμίζει το φίλτρο την τάση που θα δώσει στο 1.
Παρατίθεται εικόνα από το manual

The on-board audio jack (J8) is driven by a Sallen-Key Butterworth Low-pass 4th Order Filter that provides mono audio output. The circuit of the low-pass filter is shown in Figure 15.1. The input of the filter (AUD_PWM) is connected to the FPGA pin A11. A digital input will typically be a pulse-width modulated (PWM) or pulse density modulated (PDM) open-drain signal produced by the FPGA. The signal needs to be driven low for logic '0' and left in high-impedance for logic '1'. An on-board pull-up resistor to a clean analog 3.3V rail will establish the proper voltage for logic '1'. The low-pass filter on the input will act as a reconstruction filter to convert the pulse-width modulated digital signal into an analog voltage on the audio jack output.

Σχήμα 12: manual filter

Μετά από έρευνα στο internet βρέθηκε ότι ένας τρόπος για να μπορέσεις να στείλεις σήμα Z σε RTL είναι μέσω ενός `tri-state-buffer`. Όπου ουσιαστικά είναι ένας buffer με ένα σήμα enable το οποίο όταν είναι 0 βγάζει Z.
Παρατίθεται το truth table του tri-state-buffer

ENA	IN	OUT
0	0	0
0	1	1
1	0	Z
1	1	Z

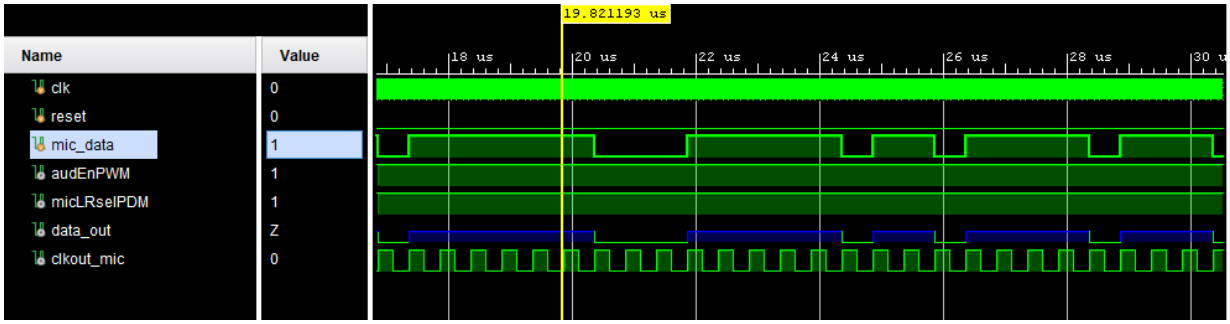


Σχήμα 13: tri state buffer

Για την υλοποίηση, λοιπόν, χρησιμοποιούμαι το module του MMCM, module που μας δόθηκε (convertPDMtoPWM) καθώς και ένα καινούριο module που αναλαμβάνει την αποσειριοποίηση των δεδομένων. Έτσι στο top module περνάμε τα δεδομένα από το μικρόφωνο στο convertPDMtoPWM και εκείνο μας τα επιστρέφει σε PWM και σε bus των 32 bits (το module αρχικά ήταν για 16 bit αλλά αλλάζοντας το width των busses το έκανα για 32 bits) οπότε το περνάμε στο module speaker το οποίο τα μετατρέπει σε σειριακά και τα περνάει από τον tri-state-buffer για να βγάλει σωστά τα data και πάλι στο top module. Έτσι στέλνονται τα data στο φίλτρο με το format που αναγράφεται.

6.2 Έλεγχος Προαιρετικού Μέρους

Για να ελέγξουμε το προαιρετικό μέρος αρκεί να αλλάξουμε ελάχιστα το testbench από το μέρος Β και να βάλουμε πάλι να στέλνονται δεδομένα όπως τα στέλνει το μικρόφωνο και να παρατηρήσουμε ότι το σήμα που οδηγείται στο φίλτρο του audio jack γίνεται 0 και Z όταν πρέπει.
Το παρατηρούμε εύκολα στην παρακάτω εικόνα



Σχήμα 14: Waveforms Προαιρετικό Μέρος

Στην συνέχεια πραγματοποιήθηκαν synthesis και implementation όπου και εκεί οι προσομοιώσεις φαινόντουσαν σωστές. Το bitstream παράχθηκε χωρίς κάποιο πρόβλημα ή σημαντικό warning.

7 Δοκιμές στην Πλακέτα

7.1 Δοκιμή Υποχρεωτικού Μέρους

Εφόσον είχε παραχθεί το bitsream και είχαν γίνει όλοι οι απαραίτητοι έλεγχοι, έμενε να γίνει έλεγχος πάνω στην πλακέτα. Στην πρώτη δοκιμή που έγινε δεν λειτούργησε σωστά καθώς φαινόταν να παράγει ήχο την ώρα που έγραφε και για πολύ λίγο χρόνο μετά. Δηλαδή πατούσα το κουμπί του Record και ενώ έγραφε, έπαιζε και ταυτόχρονα ήχο. Παρατηρήθηκε ότι υπήρχε πρόβλημα στην σύνδεση στο XDC το οποίο είχε γίνει από λάθος, ενώ επίσης και στις διευθύνσεις, δηλαδή στον τρόπο που επιλέγονταν ποιο module ορίζει την διεύθυνση της BRAM. Αλλάζοντας το XDC και προσθέτοντας τον multiplexer για την επιλογή της διεύθυνσης της BRAM, στην επόμενη δοκιμή έπαιξε πλήρως σωστά. Πατώντας το κουμπί για λίγο έγραφε και όταν πατούσα να παίξει έπαιζε έναν ήχο σαν θόρυβο για 5 sec. Για να απορρίψω την περίπτωση ότι είναι απλά ο θόρυβος που βγαίνει από τα ηχεία, παρατήρησα ότι διαφέρει ο ήχος που βγαίνει όταν δεν είναι πατημένο του κουμπί του playback με όταν έχει πατηθεί, οπότε όντως διάβαζε από την BRAM.

7.2 Δοκιμή Προαιρετικού Μέρους

Εφόσον βγήκε απροβλημάτιστα bitstream και για το προαιρετικό μέρος, έγινε έλεγχος και για αυτό στην πλακέτα. Δυστυχώς όμως έγινε τελευταία στιγμή και δεν υπήρχε αρκετός χρόνος για να σιγουρευτώ για τα αποτελέσματα. Όταν το συνέδεσα από τα ηχεία ακουγόταν μόνο ορισμένοι ήχοι τύπου θόρυβοι και ”φύσημα” στο μικρόφωνο, ή πάτημα πάνω στο μικρόφωνο. Οπότε δεν λειτούργησε σωστά η υλοποίηση. Παρόλα αυτά παρατηρήθηκε αργότερα ένα λάθος για το bus ενός από τα δεδομένα το οποίο είχε λάθος τιμή, οπότε ίσως εάν γίνει δοκιμή να παίξει σωστά.

8 Μέρος Γ - Υλοποίηση VSYNC και Κατακόρυφου Μετρητή Pixel

8.1 Περιγραφή υλοποίησης VSYNC

Στο τρίτο μέρος της εργασίας ζητείται η υλοποίηση του σήματος συγχρονισμού VSYNC και η ολοκλήρωση της σύνδεσης του `VGA_controller`. Το σήμα VSYNC αποτελεί το σήμα για τον κατακόρυφο χρονισμό της οθόνης τύπου `CRT` και προσδίδει τους χρόνους που πρέπει να διασχισθεί ο κατακόρυφος άξονας καθώς και τα Hz στα οποία θα τρέξει. Όπως και προηγουμένως για τις συγκεκριμένες τιμές ανάλυσης και Hz προκύπτουν οι αντίστοιχοι χρόνοι που φαίνονται παρακάτω.

9 Συμπεράσματα

Τα τελικά συμπεράσματα που προκύπτουν για την εργασία είναι ότι ενώ φαινόταν σχετικά εύκολη στην αρχή, υπήρχαν δυσκολίες που δεν ήταν αντιληπτές με την πρώτη ματιά. Παράδειγμα αποτελούν η BRAM όπου εφόσον απαιτήθηκε να χρησιμοποιηθεί το primitive είχε περισσότερες μεταβλητές και ήταν λίγο δυσνόητο στην αρχή, αλλά επίσης και τα ελλειπή manual της πλακέτας για ότι αφόρα το φίλτρο και το audio jack. Παρόλα αυτά, θεωρώ ότι επιτεύχθηκε να γίνει μία αρκετά καλή προσπάθεια και λειτουργική, αλλά σίγουρα φέρει βελτιώσεις κυρίως στο κομμάτι των testbenches (ώστε να γίνουν αυτόματα) αλλά και σε κάποια κομμάτια της υλοποίησης. Θετική νότα δίνει το ότι λειτούργησε στην πλακέτα, αλλά ότι και το προαιρετικό μέρος υπάρχει πιθανότητα να λειτουργήσει ή έστω να είναι κοντά. Μοναδικό αρνητικό της εργασίας ότι λόγω του ελάχιστου χρόνου ηχογράφησης δεν ήμασταν σε θέση να ακούσουμε κάποιον ήχο παρά μόνο κάποιου είδους θόρυβο.