

# **Final Project: ADXL-362 Controller**

17 Φεβρουαρίου 2023

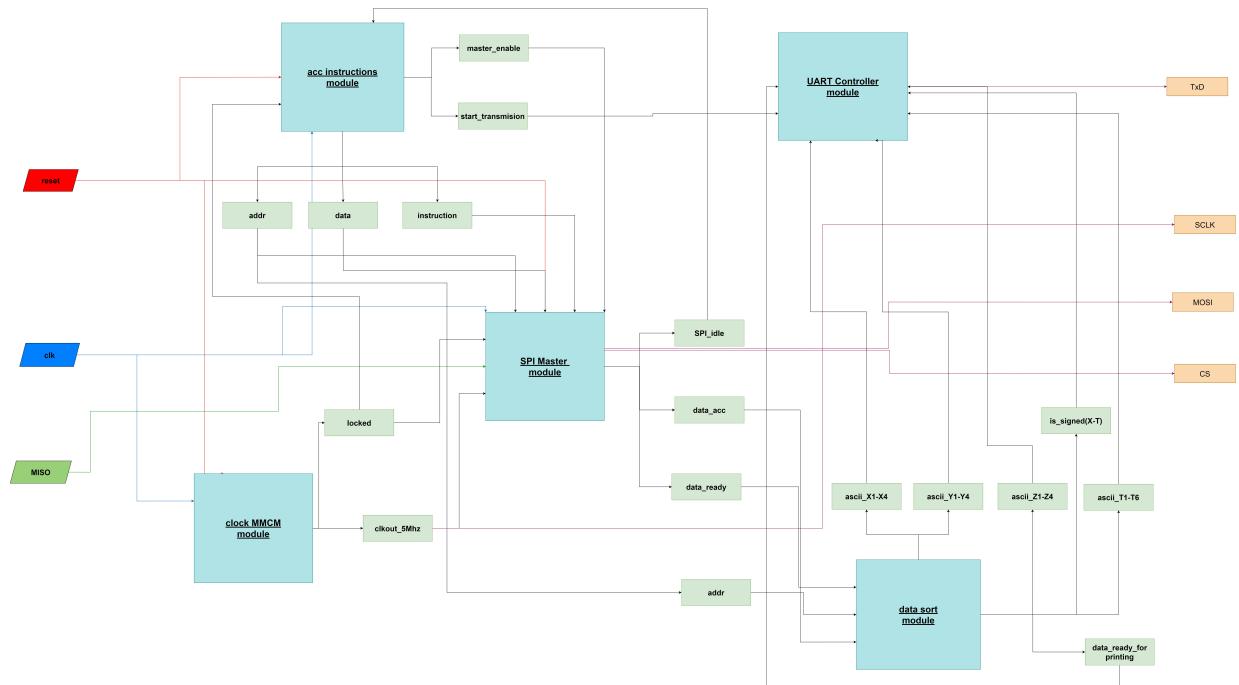
## **Περιεχόμενα**

<b>1 Εισαγωγή</b>	<b>1</b>
<b>2 Μέρος Α: Υλοποίηση UART Transmitter και baud rate Controller</b>	<b>2</b>
2.1 Υλοποίηση BAUD controller . . . . .	2
2.2 Υλοποίηση Transmitter . . . . .	2
2.3 Έλεγχος Transmitter . . . . .	4
<b>3 Μέρος Β - Υλοποίηση SPI Master</b>	<b>5</b>
3.1 Υλοποίηση . . . . .	5
3.1.1 Υλοποίηση MMCM . . . . .	5
3.2 Έλεγχος SPI Master . . . . .	9
<b>4 Μέρος Γ - Υλοποίηση Ελεγκτή ADXL362</b>	<b>9</b>
4.1 Υλοποίηση Accelerometer Instructions . . . . .	9
4.2 Επαλήθευση Accelerometer Instructions - SPI Master . . . . .	13
4.3 Υλοποίηση Data Sort Module . . . . .	14
4.3.1 Υλοποίηση Double Dabble module . . . . .	16
4.3.2 Επαλήθευση Double Dabble Module . . . . .	19
4.4 Επαλήθευση Data Sort - SPI Master . . . . .	21
<b>5 Υλοποίηση Transmitter Controller</b>	<b>22</b>
5.1 Έλεγχος Transmitter Controller . . . . .	24
5.2 Generate Bitstream και δοκιμή στήν πλακέτα . . . . .	24
5.2.1 Constraints . . . . .	24
5.2.2 Δοκιμή στην πλακέτα . . . . .	24

## **1 Εισαγωγή**

Το τελικό project αφορούσε την υλοποίηση ενός κυκλώματος το οποίο διαβάζει δεδομένα από ένα επιταχυνσιόμετρο της FPGA πλακέτας (ADXL 362) μέσω του πρωτοκόλλου SPI και τα εκτυπώνει τα δεδομένα σε ένα εικονικό τερματικό μέσω UART.

Για την επίτευξη των στόχων δημιουργήθηκαν 7 modules και 6 FSM's τα οποία θα αναλυθούν παρακάτω. Αρχικά παρατίθεται το διάγραμα ροής για την συνολική υλοποίηση, ώστε να υπάρχει μία πρώτη εικόνα για το κύκλωμα και την ανάλυση που ακολουθεί.



Σχήμα 1: dataflow ADXL-362 Controller

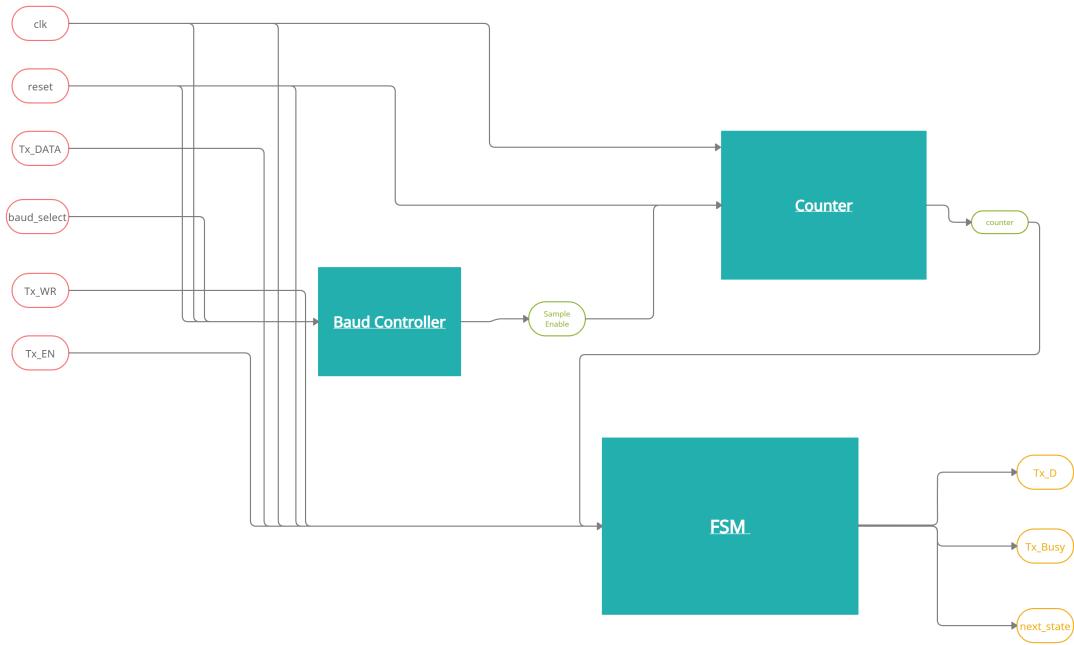
## 2 Μέρος Α: Υλοποίηση UART Transmitter και baud rate Controller

### 2.1 Υλοποίηση BAUD controller

Αρχικά χρειάζεται να υλοποιήσουμε ένα Baud controller για την σωστή λειτουργία του transmitter όπου ανάλογα με το input που θα παίρνει από μία 3-bit μεταβλητή (baud select) θα βγάζει την ανάλογη ταχύτητα επικοινωνίας για το UART. Από έναν πίνακα δίνεται το baud rate (bits/sec) αλλά για να το προσαρμόσουμε στο ρολόι της FPGA και να βρούμε το όριο (δηλαδή το πόσες περίοδοι περνάνε για να σταθεί δε- δομένο) αρκεί να κάνουμε την ακόλουθη πράξη  $\text{limit} = 10\text{-}8/(16 * \text{BaudRate})$ . Ουσιαστικά στον αριθμητή βάζουμε το 1 sec σε nsec αλλά επειδή στην συνέχεια πολλαπλασιάζουμε με την πε- ρίοδο, 10ns, προκύπτει 10-8. Επειδή όμως προκύπτει δεκαδικός αριθμός στρογγυλοποιούμε στον πιο κοντινό ακέραιο και έ- χουμε κάποιο σφάλμα. Για να το υπολογίσουμε αρκεί να χρησιμοποιήσουμε τον τύπο:  $\text{Error} = (\text{limit} - \text{int(limit)}) * 1$ ) Για το πρώτο baud rate παραδείγματος χάριν προκύπτει  $\text{limit} = 20833.3333$  και το σφάλμα βγαίνει ίσο με 3.333ης. Για να υλοποιηθεί αυτό χρησιμοποιήθηκε ένας counter των 15bit όπου μετράει θετικούς παλ- μούς του ρολογιού και όταν γίνει ίσος με το `limit` παράγει έναν παλμό για ένα κύκλο sample- enable. Το όριο καθορίζεται από μια case με βάση το input από το top module. Τέλος χρησι- μοποιήθηκε και ένα σήμα σαν enable (check) το οποίο μηδενίζει τον counter όταν φτάσει το `limit`.

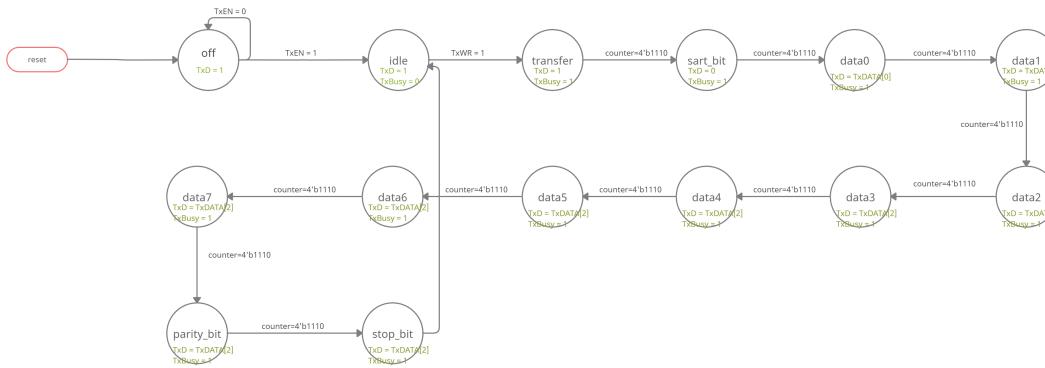
### 2.2 Υλοποίηση Transmitter

Στο μέρος Α της εργασίας ζητείται η δημιουργία ενός Transmitter για το UART. Για την επίτευξη αυτού δημιουργήθηκε ένα module το οποίο περιέχει το `Baud_controller` για να παρέχει το σήμα του baud rate, μία FSM όπου διαχειρίζεται όλες τις καταστάσεις του transmitter και τέλος ένας counter όπου παρέχει το πότε πρέπει να στείλει το επόμενο bit ο αποστολέας. Παρατίθεται το αντίστοιχο dataflow για τον Transmitter



Σχήμα 2: dataflow Transmitter

Για τον σκοπό της εργασίας δημιουργήθηκε μία FSM τύπου Moore ώστε να διαχειρίζεται όλες τις καταστάσεις του αποστολέα. Αποτελείται από 14 states τα οποία διαχωρίζονται στις καταστάσεις όπου ο transmitter είναι off, idle και όλα τα μηνύματα που πρέπει να στείλει (Start\_bit - Stop\_bit). Στην off κατάσταση σημαίνει ότι το σήμα  $TxEN = 0$  ενώ στην idle ότι είναι 1, αλλά δεν έχει δωθεί ακόμα σήμα ώστε να ξεκινήσει να στέλνει δεδομένα. Στην συνέχεια ακολουθεί το Transfer state το οποίο βρίσκεται εκεί ώστε από την στιγμή που θα έρθει σήμα για αποστολή δεδομένων να περιμένει 16 κύκλους του sample-enable ώστε να στείλει το start bit, μετά από συγκεκριμένο χρόνο. Στην συνέχεια βρίσκονται όλα τα states από τα data  $data0 - data7$  όπου σε καθένα από αυτά στέλνετε το αντίστοιχο bit από το TxDATA που περιέχει το μήνυμα προς αποστολή. Τέλος, υπάρχει το parity bit και το stop bit όπου το πρώτο στέλνει 1 αν ο αριθμός των άσσων είναι μονός και 0 εάν είναι ζυγός. Για να το πετύχει αυτό περνάμε όλο το μήνυμα από μια πύλη xor. Το stop bit σηματοδοτεί το τέλος της μετάδοσης και στέλνει τον αριθμό 1, ενώ μετά από 16 κύκλους επιστρέφει στην κατάσταση idle περιμένοντας το επόμενο μήνυμα. Οι εναλλαγές στις καταστάσεις γίνονται μέσα από κάποια σήματα ελέγχου, τα οποία είναι συγκεκριμένα το  $TxEN$ ,  $TxWR$  (σήμα που ελέγχει πότε δίνεται εντολή από το σύστημα να ξεκινήσει να στέλνει ο αποστολέας), και το send-out όπου γίνεται 1 όταν έχουν περάσει 16 κύκλοι από το sample-enable, δηλαδή όταν πρέπει να σταλθεί το επόμενο bit. Παρακάτω φαίνεται το διάγραμμα της FSM, όπου στην έξοδο φαίνονται μόνο τα μηνύματα όπου αλλάζουν σε κάθε κατάσταση και όχι όλα ώστε να είναι πιο ευανάγνωστο.



Σχήμα 3: FSM Transmitter

Η υλοποίηση του transmitter έγινε σε ένα module το οποίο αποτελείτε από ένα ακολουθιακό always block και είναι ένας counter ο οποίος μετράει τις φορές που λαμβάνει το σήμα Sample\_Enable από το Baud Controller και όταν γίνει ίσος με 16 στέλνει ένα σήμα send = 1 που σημαίνει ότι ο transmitter πρέπει να στείλει το επόμενο bit. Επιπλέον λόγω της FSM που χρησιμοποιήσαμε δημιουργούμε 2 always blocks, το πρώτο ακολουθιακό όπου αλλάζει τις καταστάσεις με βάση την μεταβλητή Next\_state και το δεύτερο συνδιαστικής λογικής καθώς καθορίζει τι output θα έχουμε σε κάθε κατάσταση και αλλάζει και το Next\_State με βάση τα σήματα ελέγχου.

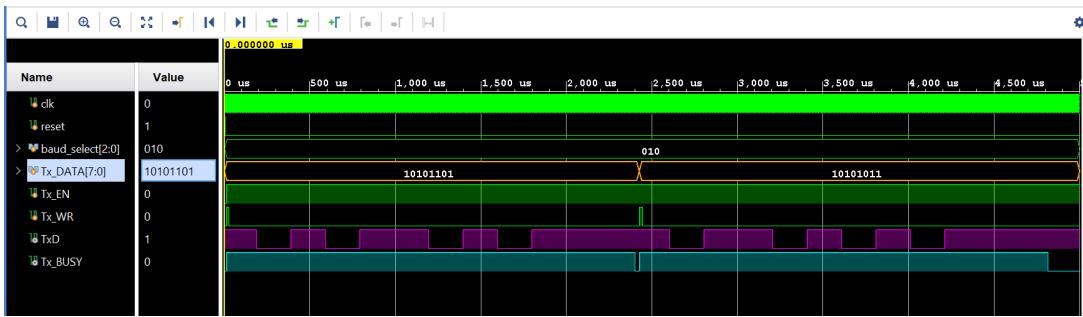
Επιπλέον, θεωρήθηκε αναγκαίο ο παλμός send που θα άλλαζε τα states να έχει διάρκεια μίας περιόδου του ρολογιού ώστε να μην δημιουργεί θέμα στην FSM, οπότε χρησιμοποιήθηκαν 2 flip flop στην σειρά και μία πύλη and ώστε να παίρνουν τον παλμό send και να βγάζουν έναν send\_out με διάρκεια μία περίοδο. Η υλοποίηση στον κώδικα έγινε όπως φαίνεται δεξιά.

## 2.3 Έλεγχος Transmitter

Αρχικά, έπρεπε να παρατηρήσω τι ακριβώς πρέπει να ελεγχθεί ώστε να σιγουρευτώ ότι λειτουργεί σωστά. Πρέπει λοιπόν να παρατηρήσω ότι μόλις αλλάζει κάποιο από τα σήματα ελέγχου όπως το TxEN, TxWR αλλάζει αμέσως η κατάσταση του Transmitter και ότι όταν στέλνει δεδομένα, στέλνονται όντως με την σωστή σειρά και μεσολαβεί ο απαραίτητος χρόνος μεταξύ δύο δεδομένων, και τέλος ότι το σήμα TxBUSY είναι όταν πρέπει στο 0 και αντίστοιχα στο 1.

Αρχικά λοιπόν επιλέγω ένα πακέτο για το TxDATA 8'b11101011 με αρκετές εναλλαγές μεταξύ 0 και 1 και βάζω συγκεκριμένο baud\_select = 3'b010. Αρχικοποιώ το clock, το TxEN στο 1 και το TxWR επίσης στο 1, και στην συνέχεια κάνω reset. Από την προσομοίωση σε συμπεριφορικό επίπεδο παρατηρώντας τις κυματομορφές και πιο συγκεκριμένα τα σήματα TxD και TxBUSY παρατηρούμε ότι εναλλάσσονται σωστά, και στέλνεται το μήνυμα που έπρεπε με βάση το TxDATA. Παρατίθεται η εικόνα.

Παρατηρούμε ότι το TxD στέλνει σωστά το μήνυμα δηλαδή το TxDATA από LSB->MSB και αντίστοιχα το TxBUSY μένει στο 0 μέχρι να έρθει το επόμενο TxWR = 1 για να σταλθεί το επόμενο μήνυμα. Επίσης, το Parity βρίσκεται σωστά στο 1 εφόσον το μήνυμα έχει μονό αριθμό από 1. Τέλος για να υπολογίσουμε τον χρόνο, το testbench εκτυπώνει τον χρόνο όπου μένει το TxD στο 1 μέχρι να πέσει πάλι στο 0 οπότε αφού γνωρίζουμε το μήνυμα που θα σταλθεί μπορούμε να υπολογίσουμε ανάλογα με το πόσα 1 έχουμε στην σειρά, αν διαρκεί το κάθε bit τον χρόνο που επιθυμούμε. Η διάρκεια προκύπτει από τον υπολογισμό του  $16 * limit * 10ns$  οπότε για baud rate = 010 προκύπτει 201120ns



Σχήμα 4: Waveforms Transmitter Simulation

Όπως παρατηρούμε και από την εικόνα πράγματι τα μηνύματα διαρκούν όσο χρειάζεται και αρκεί ο έλεγχος ενός μόνο bit καθώς όλα αλλάζουν με το ίδιο σήμα ελέγχου. Μόνο το πρώτο bit (start bit) δεν διαρκεί όσο πρέπει καθώς το TxWR είναι ασύγχρονο σήμα και δεν εγγυάται κανείς πότε ακριβώς θα έρθει το σήμα sample enable.

Ένα πρόβλημα που παρατηρήθηκε από το testbench είναι ο χρόνος για το start bit, ότι δεν έρχεται μετά από 16 κύκλους του sample enable για το οποίο δεν βρέθηκε λύση για τον λόγο που αναφέρθηκε προηγουμένως. Επιπλέον, ένα ακόμη πρόβλημα που παρατηρήθηκε ήταν ότι αρχικά τις καταστάσεις της FSM τις άλλαζε το σήμα send το οποίο όμως διαρκούσε πολύ παραπάνω από μία περίοδο του clock οπότε επειδή άλλαζε το current state και το send ήταν ακόμα 1, άλλαζε πολύ γρήγορα όλες τις καταστάσεις μέχρι να γίνει το send = 0. Το συγκεκριμένο πρόβλημα λύθηκε με την χρήση των 2 flip flop που έγινε αναφορά προηγουμένως και έτσι παράγεται ένας παλμός αποτέλεσμα του send το send\_out όπου διαρκεί για μία περίοδο του clock. Ο έλεγχος του κυκλώματος ολοκληρώθηκε με την πραγματοποίηση synthesis and implementation όπου παρατηρήθηκε ότι το κύκλωμα λειτουργεί επίσης κανονικά και δεν υπάρχουν latches!.

## 3 Μέρος Β - Υλοποίηση SPI Master

### 3.1 Υλοποίηση

Το Β μέρος ζητούσε την υλοποίηση ενός SPI Master ώστε να μπορεί να επικοινωνεί με το επιταχυνσιόμετρο το οποίο αποτελεί τον SPI slave.

Οι βασικές συνδέσεις του SPI είναι:

- MOSI: Η θύρα στην οποία στέλνει ο Master στον Slave δεδομένα
- MISO: Η θύρα στην οποία στέλνει ο Slave στον Master δεδομένα
- SCLK: Το ρολόι που συγχρονίζει την επικοινωνία (1-8 Mhz για την συγκεκριμένη περίπτωση)
- CS (Chip Select): ξεκινάει την επικοινωνία, και σε περίπτωση όπου υπάρχουν πολλοί slaves, επιλέγει με ποιον θα γίνει η επικοινωνία

#### 3.1.1 Υλοποίηση MMCM

Όπως βλέπουμε το SCLK, χρειάζεται να είναι από 1-8 MHZ οπότε πρέπει να γίνει χρήση μίας MMCM μονάδας ώστε να ρίξουμε το ρολόι της FPGA από τα 100 Mhz στα 5 Mhz όπου επιλέξαμε σαν ρολόι διότι έχει περίοδο 200ns πολλαπλάσια δηλαδή του 10 ns που είναι το ρολόι της πλακέτας.

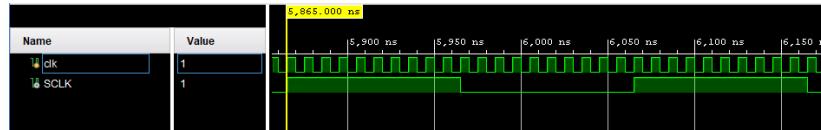
Με την χρήση των τύπων από το datasheet που φαίνονται παρακάτω προκύπτουν οι τιμές  $M(\text{CLKFBOUT MULT F}) = 32$ ,  $D(\text{DIVCLK DIVIDE}) = 5$  και  $O(\text{CLKOUT DIVIDE}) = 128$ . Με αυτές τις τιμές το ρολόι μας φτάνει τα 5 Mhz δηλαδή περίοδος 200 ns.

$$F_{vco} = F_{clkin} \times \frac{M}{D}$$

$$F_{clkout} = F_{clkin} \times \frac{M}{D \times O}$$

Σχήμα 5: MMCM formulas

Επίσης παρατίθεται και η προσομοίωση που αποδικνύει ότι το ρολόι έχει πέσει στα 5 Mhz

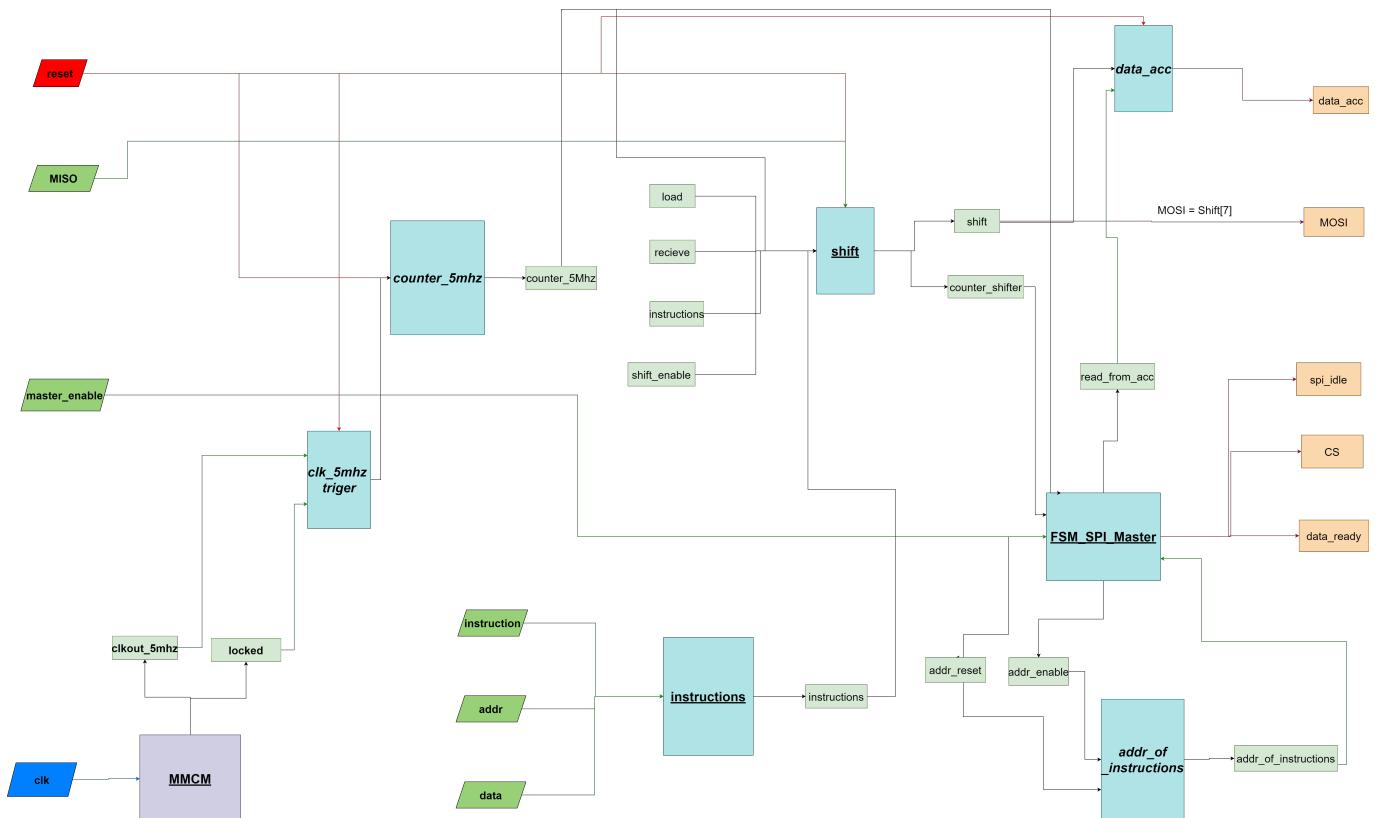


Σχήμα 6: MMCM Waveform

Παρακάτω φαίνεται το dataflow για τον SPI Master ώστε να αναλυθούν τα κομμάτια του στην συνέχεια

Βασικό κομμάτι του SPI Master αποτελεί ένας Shift register των 8-bits όπου κρατάει την πληροφορία που πρόκειται να μεταδοθεί, ενώ επίσης αποθηκεύει και τα δεδομένα που έρχονται από τον Slave. Σε περίπτωση λοιπόν ταυτόχρονης επικοινωνίας, κάθε φορά που στέλνεται ένα δεδομένο, στην θέση που αδειάζει, γεμίζει την τιμή που έρχεται από τον Slave. Στην προκειμένη περίπτωση όμως δεν χρειάστηκε κάπου να υπάρξει ταυτόχρονη επικοινωνία.

Επιπλέον, για να γίνει σωστά η επικοινωνία με το επιταχυνσιόμετρο, απαιτείται να πληρούνται κάποιοι περιορισμοί που δίνονται στο datasheet, οπότε είναι απαραίτητο να γνωρίζουμε την θέση του SCLK σε σχέση με το clock της πλακέτας για να γνωρίζουμε πότε μπορούμε να στείλουμε τα αντίστοιχα σήματα π.χ. πότε να γίνει 0 το CS. Για να το επιτύχουμε αυτό χρησιμοποιούμε έναν counter όπου όταν έρχεται posedge SCLK γίνεται 0 και μετράει με βάση το κανονικό ρολόι φορές (στην προκειμένη περίπτωση 20) μέχρι το επόμενο posedge SCLK.



Σχήμα 7: dataflow SPI Master

Table 10. SPI Timing ( $T_A = 25^\circ\text{C}$ ,  $V_S = 2.0 \text{ V}$ ,  $V_{DD\text{ I/O}} = 2.0 \text{ V}$ )

Parameter	Limit <sup>1,2</sup>		Unit	Description
	Min	Max		
$f_{CLK}^3$	2.4	8000	kHz	Clock Frequency
$t_{CS}$	100		ns	$\overline{CS}$ Setup Time
$t_{CSH}$	20		ns	$\overline{CS}$ Hold Time
$t_{CSD}$	20		ns	$\overline{CS}$ Disable Time
$t_{SU}$	20		ns	Data Setup Time
$t_{HD}$	20		ns	Data Hold Time
$t_{HIGH}$	50		ns	Clock High Time
$t_{LOW}$	50		ns	Clock Low Time
$t_{CLE}$	25		ns	Clock Enable Time
$t_V$	0	35	ns	Output Valid from Clock Low
$t_{DIS}$	0	25	ns	Output Disable Time

## Σχήμα 8: timing constraints

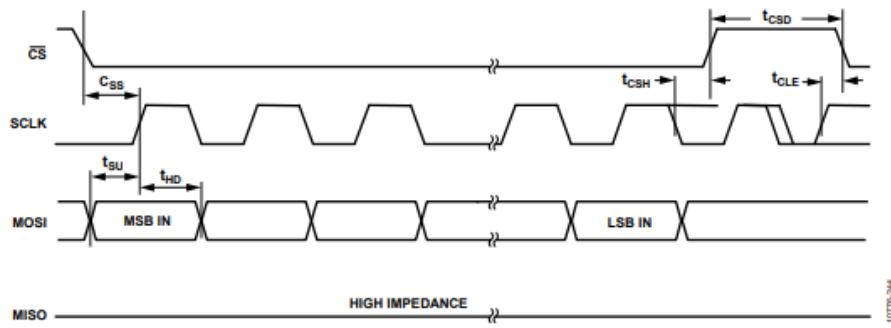


Figure 41. Timing Diagram for SPI Receive Instructions

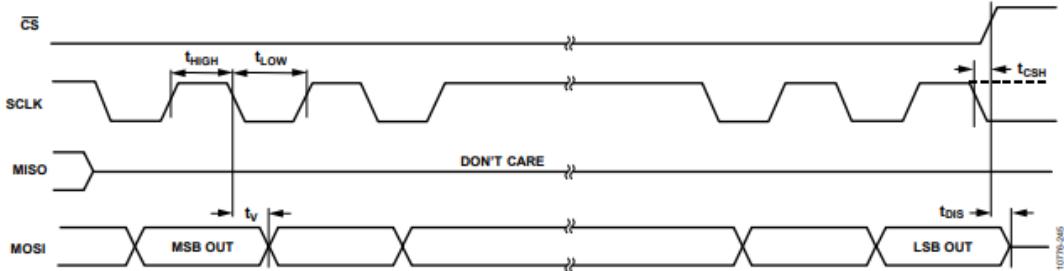
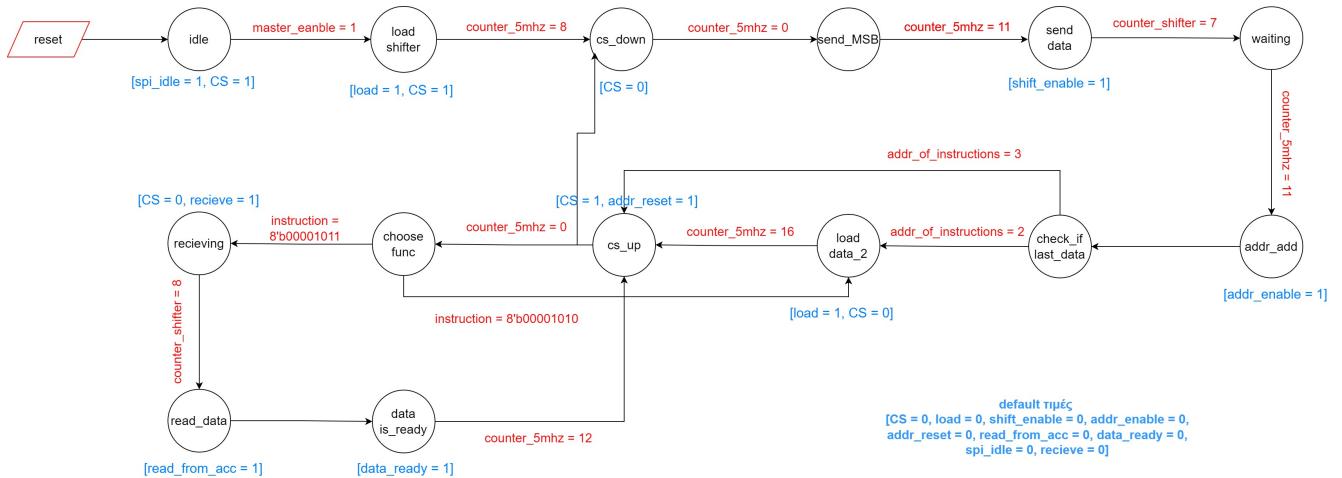


Figure 42. Timing Diagram for SPI Send Instructions (Shaded Portions of Figure 36, Figure 38, and Figure 40)

## Σχήμα 9: timing constraints spi

Για την δημιουργία του SPI Master χρησιμοποιήθηκε μία FSM τύπου Moore με 14 states, το σχηματικό της οποίας φαίνεται παρακάτω



## Σχήμα 10: FSM SPI Master

Ουσιαστικά η συγκεκριμένη FSM ξεκινάει από το state αδράνειας και όταν δοθεί σήμα από το επόμενο module που την ελέγχει φορτώνει τα δεδομένα στον Shifter και όταν είναι ο counter είναι ίσος με 8, δηλαδή βρισκόμαστε λίγο πριν το negedge του SCLK κατεβαίνει το CS ώστε να υπάρχει το απαραίτητο περιθώριο των 100 ns για setup που απαιτείται. Στην συνέχεια στέλνεται το MSB από το πρώτο μήνυμα και λίγο μετά το negedge του SCLK πρέπει να σταλθεί το επόμενο bit. Όταν σταλθούν και τα 8 bits από το μήνυμα, το οποίο το καταλαβαίνουμε από τον counter-shifter όπου μετράει πόσες φορές έχει γίνει shift. Στην συνέχεια αυξάνεται η διεύθυνση που δείχνει στον πίνακα που κρατούνται όλα τα δεδομένα προς αποστολή, ώστε να σταλθεί το επόμενο. Γίνεται ένας έλεγχος εάν το addr\_of\_instructions είναι 3, δηλαδή βρισκόμαστε στο τελευταίο δεδομένο προς αποστολή ώστε να γίνει το CS 1 και να σταματήσει η επικοινωνία, αλλιώς φορτώνει το επόμενο δεδομένο προς αποστολή και συνεχίζει να στέλνει. Επίσης γίνεται έλεγχος με βάση το Instruction για το εάν στέλνονται ή διαβάζονται δεδομένα, ώστε είτε να συνεχίσει να στέλνει είτε να κρατήσει στον shift register το δεδομένο και να το αποθηκεύσει σε τιμή για να το στείλει.

### **3.2 Έλεγχος SPI Master**

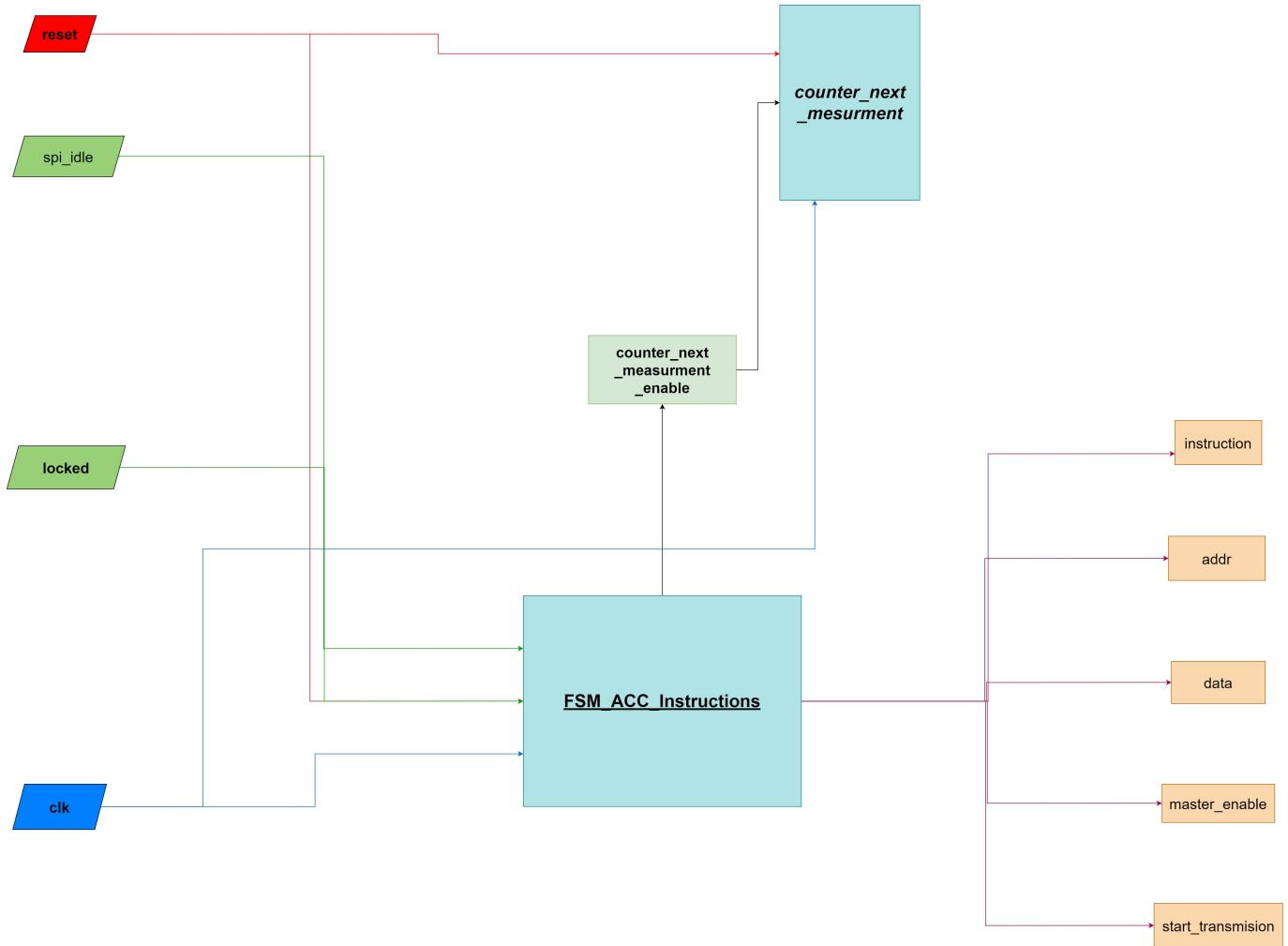
Για τον έλεγχο του SPI Master χρειάζεται να δημιουργήσουμε ένα testbench το οποίο έχει συμπεριφορά παρόμοια με του ADXL 362 και να παρατηρήσουμε ότι όταν στέλνουμε, στέλνει τα σωστά δεδομένα και όταν στέλνει το επιταχυνσιόμετρο, αποθηκεύουμε την σωστή τιμή. Στην προκειμένη περίπτωση ο έλεγχος του SPI περιλαμβάνει την περίπτωση που εισάγουμε δεδομένα προς αποστολή και παρατηρούμε από τις κυματομορφές ότι στέλνονται σωστά, ενώ επίσης τις χρονικές στιγμές που περιμένει δεδομένα από το επιταχυνσιόμετρο, στέλνουμε ένα ένα τα bits για να παρατηρήσουμε ότι αποθηκεύει την σωστή τιμή. Πιο ακριβής έλεγχος γίνεται μαζί με το επόμενο κομμάτι της εργασίας, που είναι πιο εύκολο να ελεγχθεί και πρακτικά.

## 4 Μέρος Γ - Υλοποίηση Ελεγκτή ADXL362

#### **4.1 Υλοποίηση Accelerometer Instructions**

Στην συνέχεια, χρειάζεται να στείλουμε στο ADXL-362 ορισμένες εντολές ώστε να αρχικοποιηθεί και να ορίσει τα όρια στις τιμές που θα διαβάζει, καθώς επίσης και την ταχύτητα αποστολής δεδομένων.

Υλοποιούμε λοιπόν ένα module το οποίο στέλνει στο SPI πότε και ποιες τιμές πρέπει να στείλει στο επιταχυνσιόμετρο κάθε φορά. Παρατίθεται το dataflow παρακάτω, ώστε να αναλυθεί πιο αναλυτικά το κύκλωμα



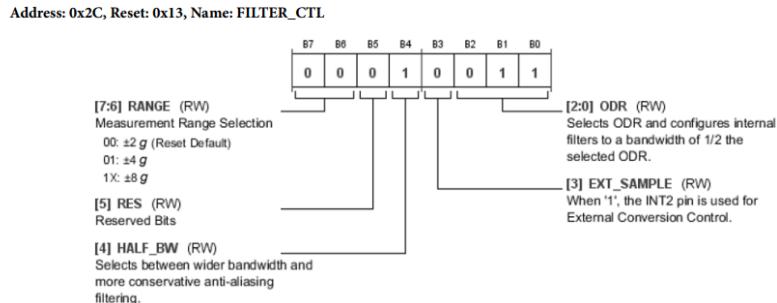
Σχήμα 11: Dataflow part C

Η διαδικασία για επικοινωνία με το accelerometer είναι η εξής, αρχικά το CS γίνεται 0 και στην συνέχεια στέλνουμε το `instruction` το οποίο θα έχει την τιμή 0xA εάν θέλουμε να στείλουμε δεδομένα, και 0xB εάν θέλουμε να διαβάσουμε. Στην συνέχεια στέλνουμε την διεύθυνση του register που επιθυμούμε να γράψουμε ή να διαβάσουμε και εάν εκτελούμε αποστολή, την τιμή των `data`, τέλος το CS γίνεται 1 και τερματίζεται η επικοινωνία.

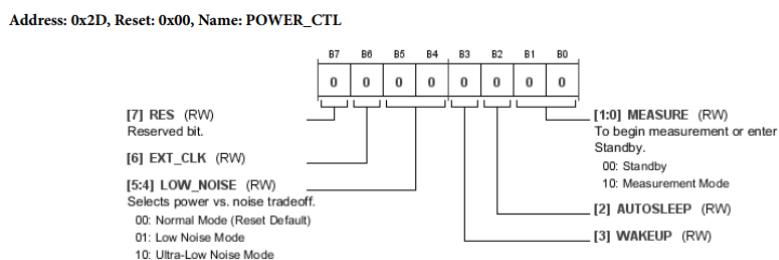
Οι εντολές διαμόρφωσης προς το accelerometer είναι απαραίτητο να είναι η εξής:

- Soft Reset Register: Στέλνοντας την τιμή 0x52 στην διεύθυνση 0x1F γίνονται `reset` και clear όλοι οι registers του επιταχυνσιόμετρου. Πρέπει να δοθεί προσοχή στο γεγονός ότι όταν γίνεται `reset` πρέπει να μην υπάρχει επικοινωνία για περίπου 0.5 ms όπως αναφέρεται στο datasheet.
- Power Control Register: Στέλνουμε την τιμή 8'b00000010 στην διεύθυνση 0x2D ώστε να θέσουμε το επιταχυνσιόμετρο σε κατάσταση μέτρησης.
- Filter Control Register: Στέλνουμε την τιμή 8'b00010100 στην διεύθυνση 0x2C ώστε να θέσουμε όρια μέτρησης τα +-2g, bandwidth 1/4 της ταχύτητας και ταχύτητα αποστολής

(ODR) ίσο με 200 Hz.



Σχήμα 12: Filter Register

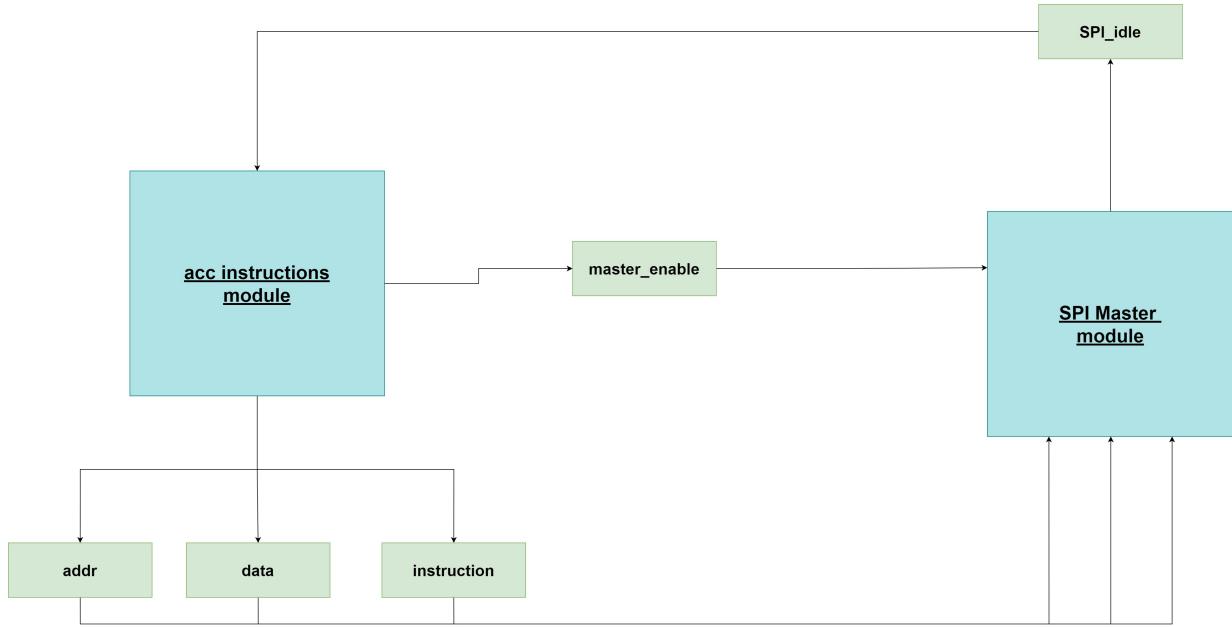


Σχήμα 13: Power control register

Υπάρχουν δύο επιλογές για να διαβάσουμε τις τιμές επιτάχυνσης στους άξονες X Y Z και την εσωτερική θερμοκρασία. Πρώτος τρόπος είναι μία FIFO που υπάρχει στην οποία μπορούν να αποθηκευτούν όλα αυτά τα απαραίτητα δεδομένα και να διαβάζονται με μόνο εντολή όλα μαζί, αλλά απαιτούνται επιπλέον εντολές για να διαμορφώσουμε την FIFO και να την ρυθμίσουμε, και επιπλέον υπάρχουν πιο αυστηρά timing constraints όσον αφορά την επικοινωνία με το SPI. Δεύτερη μέθοδος είναι να διβάζεται ένας ή ένας οι registers που μας αφορούν (συνολικά τέσσερεις) το οποίο απαιτεί 4 εντολές SPI για να διαβαστούν όλα τα δεδομένα, αλλά δεν χρειάζεται επιπλέον διαμόρφωση. Στην συγκεκριμένη υλοποίηση επιλέχτηκε ο δεύτερος τρόπος.

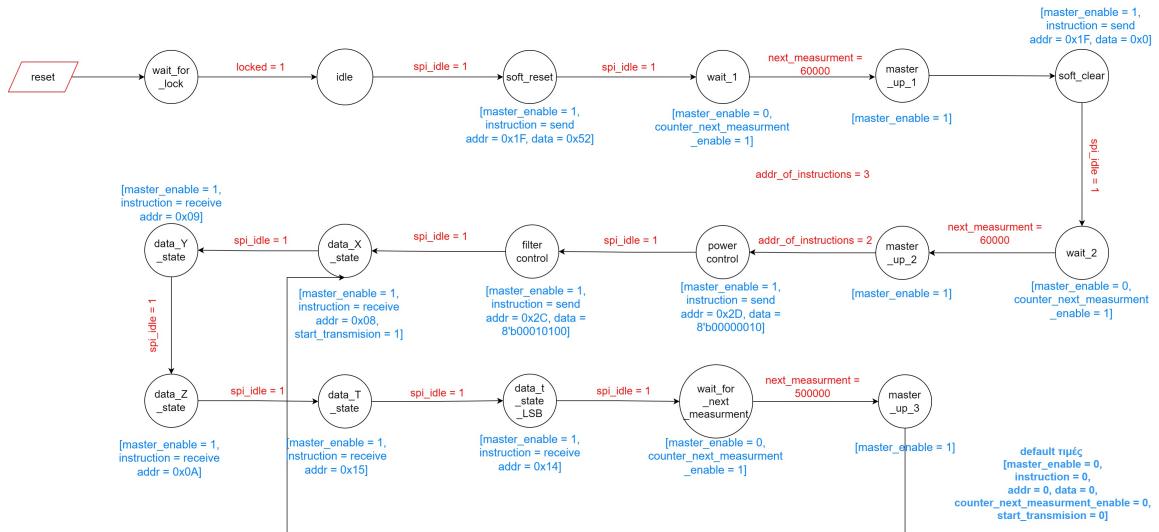
Οι τιμές των μετρήσεων κανονικά αναπαριστούνται από έναν signed αριθμό 12-bits, αλλά για τις τιμές των X, Y, Z άλλος register κρατάει τα 8 MSB της μέτρησης προσφέροντας μικρότερη ακρίβεια, αλλά λιγότερη κατανάλωση εφόσον διαβάζεται ένας register και όχι δύο. Για τους σκοπούς της εργασίας εφόσον δεν είναι κύριο μέλημα η ακρίβεια, επιλέχτηκε να διαβάζονται τα 8 MSB. Αυτή ή δυνατότητα δεν υπάρχει για την θερμοκρασία, οπότε διαβάζονται τα 12-bits.

Αρχικά παρατίθεται ένα dataflow που υποδηλώνει τις συνδέσεις μεταξύ των SPI Master και Accelerometer Instructions ώστε να γίνουν πιο εύκολα κατανοητά κάποια σήματα της FSM.



Σχήμα 14: SPI Master connections with Accelerometer Instructions

Η FSM για το module Accelerometer<sub>l</sub>Instructions φαίνεται και αναλύεται παρακάτω



Σχήμα 15: FSM Accelerometer Instructions

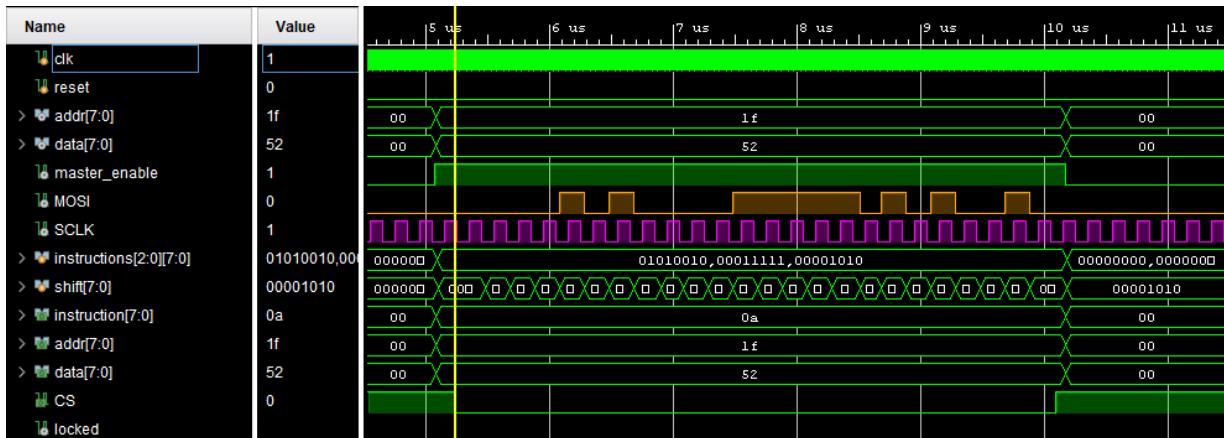
Το αρχικό state μετά από κάθε reset είναι to wait\_for\_lock στο οποίο το κύκλωμα αναμένει να σταθεροποιηθεί το ρολόι που παράγετε από το MMCΜ για να ξεκινήσει σωστά η λειτουργία. Στην συνέχεια στο idle το σύστημα είναι σε αδράνεια και όταν το spi στείλει σήμα (spi\_idle) ότι δεν κάνει κάποια αποστολή, προχωράει στο state soft\_reset όπου στέλνονται οι εντολές που αναφέρθηκαν προηγουμένως για reset στο SPI, το οποίο στην συνέχεια θα τις στείλει στο accelerometer. Στην συνέχεια ακολουθεί ένα state αναμονής wait\_1 όπου ουσιαστικά αναμένει τον χρόνο 0.5ms όπου αναφέραμε ότι πρέπει να είναι σε αδράνεια το σύστημα μετά από το reset. Ακολούθως, το state master\_up\_1 ξανά ενεργοποιεί τον SPI Master, σε ξεχωριστό state, ώστε να αρχικοποιηθεί το SPI και να στείλει το σήμα spi\_idle ώστε να προχωρήσει στην επόμενη μετάδοση. Επαναλαμβάνεται η διαδικασία για τις υπόλοιπες εντολές αρχικοποίησης και στην συνέχεια ξεκινάνε να στέλνονται εντολές για να λάβουμε δεδομένα από το επιταχυνσιόμετρο, α-

πό τα states `data_X_state`, `data_Y_state`, `data_Z_state`, `data_T_state`, `data_T_state_LSB`. Στέλνουμε το instruction που δηλώνει ανάγνωση τιμών και την ανάλογη διεύθυνση σε κάθε περίπτωση. Εφόσον, το accelerometer στέλνει δεδομένα κάθε 200 Hz (5 ms) δεν έχει νόημα να δειγματοληπτούμε συνεχώς, αλλά μόνο κάθε 5 ms και αυτό ακριβώς επιτελούν τα states (`wait_for_next_measurement`), ενώ μετά από αυτό συνεχίζουμε στο state που δειγματοληπτεί το X ώστε να ξαναλάβουμε για όλα μετρήσεις.

## 4.2 Επαλήθευση Accelerometer Instructions - SPI Master

Μαζί με την επαλήθευση του Accelerometer Instructions επαληθεύουμε και το SPI από το προηγούμενο μέρος. Για την επαλήθευση αρκεί να συνδέσουμε τα δύο modules και να παρατηρήσουμε τις κυματομορφές εάν στέλνονται σωστά τα bits ένα ένα. Το ίδιο πρέπει να κάνουμε και για την περίπτωση λήψης δεδομένων οπότε θα προσομοιώσουμε το πως στέλνει δεδομένα το επιταχυνσιόμετρο και τις χρονικές στιγμές που απαιτείται.

Υλοποιούμαι λοιπόν ένα testbench στο οποίο συνδέουμε τα δύο modules και παρατηρούμε ότι στέλνονται οι τιμές αρχικοποίησης όπως ακριβώς πρέπει μία προς μία μέσω του MOSI. Οι τιμές προς αποστολή βρίσκονται μέσα στον πίνακα instructions και με πορτοκαλί χρώμα αναπαρίτατε το MOSI όπου με βάση το SCLK στέλνει σωστά πρώτα το instruction 8'b00001010 και στην συνέχεια το address και τα data.



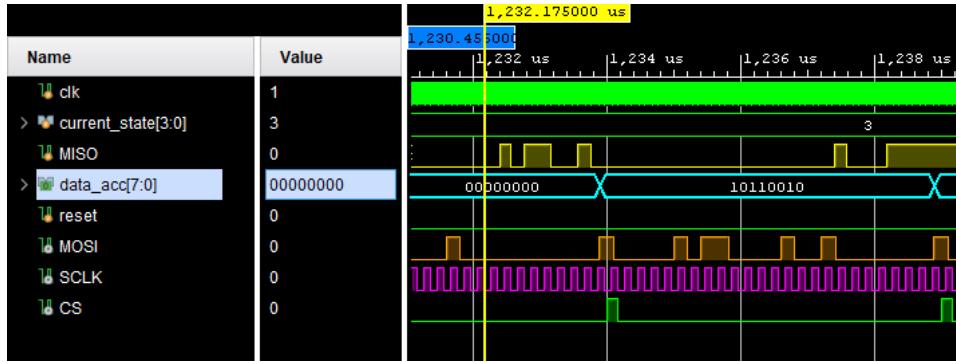
Σχήμα 16: SPI and Accelerometer Instructions Waveforms

Επιπλέον, παρατηρούμε ότι το CS κατεβαίνει τουλάχιστον 100 ns πριν το πρώτο psoedge SCLK όπως απαιτείται από το datasheet.



Σχήμα 17: CS test waveforms

Τέλος, στέλνοντας δεδομένα από το MOSI με τους χρόνους που θα έστελνε το επιταχυνσιόμετρο, όταν διαβάζει από το X, παρατηρούμε ότι γράφεται σωστά η τιμή στο data-acc.



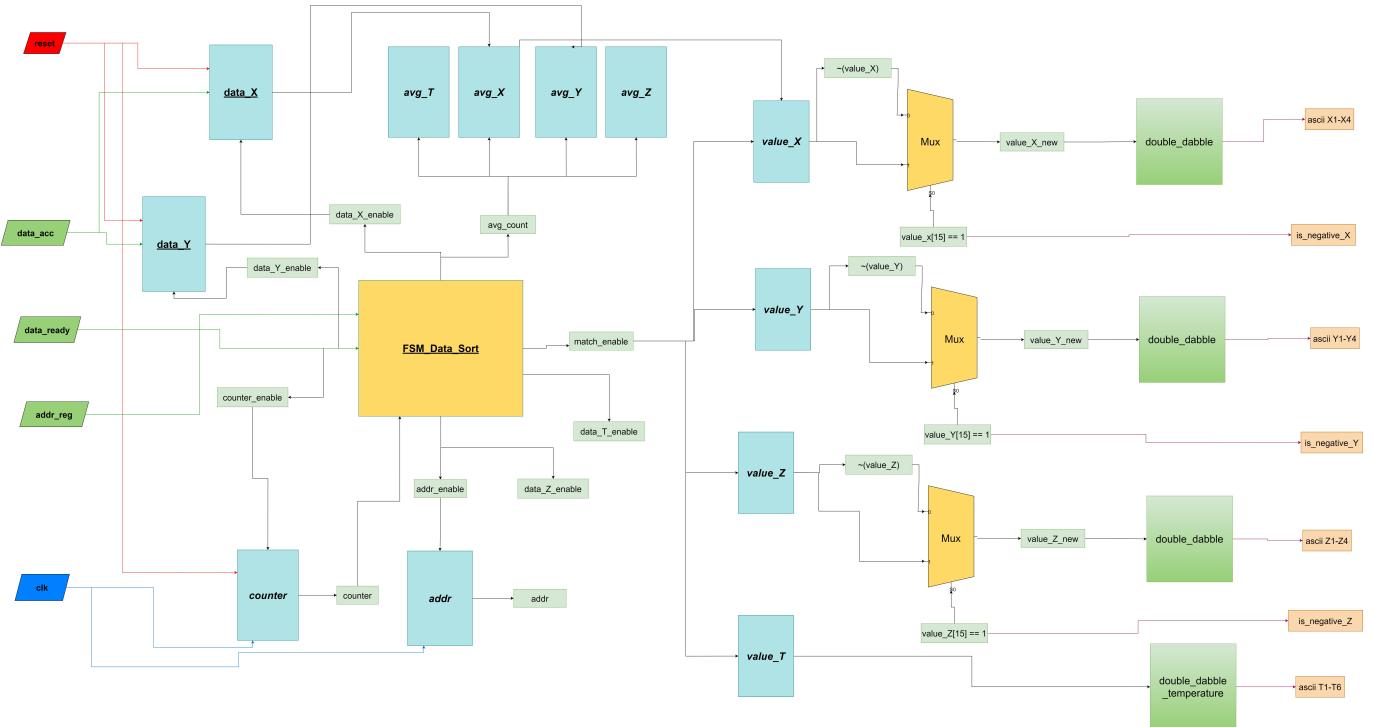
Σχήμα 18: SPI read test waveforms

Επαναλαμβάνοντας αρκετές φορές και για άλλους registers επιβεβαιώνουμε την λειτουργία του κυκλώματος.

Εφόσον μπορούμε να λάβουμε δεδομένα από το accelerometer σειρά έχει το πως θα επεξεργαστούμε τα δεδομένα ώστε να λάβουμε το αποτέλεσμα που επιθυμούμε. Το module που επιτελεί αυτή την διεργασία είναι το Data\_Sort\_Module.

### 4.3 Υλοποίηση Data Sort Module

Η υλοποίηση που πραγματοποιήθηκε για το Data Sort συνοψίζεται στο παρακάτω dataflow



Σχήμα 19: dataflow data sort

Αρχικά, εφόσον οι τιμές από τις μετρήσεις που θα λάβουμε θέλουμε να εκτυπώνονται, θα πρέπει να τις φιλτράρουμε ώστε να μην υπάρχει θόρυβος στο απεικονισμένο αποτέλεσμα από

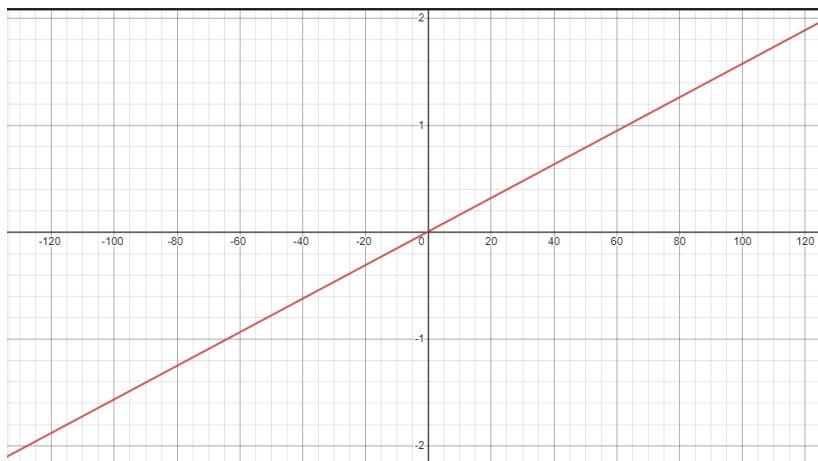
μικροαλλαγές στις τιμές από αστάθμητους παράγοντες. Μία λύση για αυτό είναι να δειγματοληπτούμε έναν μεγάλο αριθμό από μετρήσεις και να παίρνουμε τον μέσο όρο από αυτές. Στην συγκεκριμένη περίπτωση επιλέχθηκε να δειγματοληπτούμε 128 τιμές για κάθε X Y Z και T. Για τον λόγο αυτό χρησιμοποιούνται 4 πίνακες (ένας για κάθε άξονα και θερμοκρασία) με 128 θέσεις των 16 bits. Κάθε φορά που έρχεται μία τιμή αποθήκευται στον αντίστοιχο πίνακα στην θέση που πρέπει και προστίθεται με την τιμή στην προηγούμενη θέση, έτσι ώστε τελικά στην θέση 128 να υπάρχει το άθροισμα όλων των τιμών για να μπορέσει να προκύψει ο μέσος όρος. Επιλέχτηκε μέγεθος τιμών ίσο με 16 bits καθώς οι τιμές που έρχονται είναι 8 bit signed οπότε μέγιστη τιμή  $128 * 128 = 16384$  τιμή που χωράει σε 15 bits αριθμό, αλλά εφόσον οι τιμές είναι signed επιλέγουμε μέγεθος 16 bits. Για την θερμοκρασία που είναι 12 bit κάθε τιμή με τον ίδιο τρόπο προκύπτει πίνακας με μέγεθος 20 bits. Τέλος, για να προκύψει ο μέσος όρος επιλέγουμε νά κάνουμε shift αριστερά κατά 7 εφόσον  $2^7 = 128$  (χρησιμοποιούμε τον τελεστή «< και όχι « διότι θέλουμε signed τιμή). Ο λόγος που επιτελούμε διαίρεση με shift είναι προφανώς για να γλιτώσουμε χώρο και χρόνο που χρειάζεται ένα κύκλωμα που κάνει κανονικές διαιρέσεις με τον τελεστή /, αυτός είναι και ο λόγος που επιλέχτηκε δείγμα τιμών που είναι δύναμη του 2 (128).

Εφόσον λάβουμε λοιπόν τον μέσο όρο πρέπει να μετατρέψουμε την τιμή από raw data που έρχονται από το επιταχυνσιόμετρο σε τιμή επιτάχυνσης (g) και θερμοκρασίας (c). Για την επιτάχυνση, χρησιμοποιούμε την μέθοδο Linear Interpolation, ώστε να προκύψει μία συνάρτηση που θα μας δίνει την τιμή που επιθυμούμε. Τα όρια της signed τιμής 8-bit είναι (-128, 127) και τα όρια της μέτρησης της επιτάχυνσης που έχουμε ορίσει είναι (-2, 2) οπότε έχουμε δύο σημεία A(127, 2) και B(-128, -2) και εφαρμόζοντας τον τύπο που φαίνεται στην εικόνα προκύπτει η συνάρτηση  $A(127, 2) + B(-128, -2) * \text{avg}$  από την μορφή  $y = a + bx$ .

$$a = \frac{y_0 \cdot x_1 - y_1 \cdot x_0}{x_1 - x_0}$$

$$b = \frac{y_1 - y_0}{x_1 - x_0}$$

Σχήμα 20: Linear interpolation formula



Σχήμα 21: Linear interpolation graph

Για την τιμή της θερμοκρασίας εφόσον διαβάζουμε και τα 12 bits της τιμής από τα datashhet προκύπτει ότι για να μετατραπεί σε βαθμούς celsius απαιτείται να πολλαπλασιάσουμε με 0.065,

δηλαδή  $value_{temp} = 0.065 * avgT$ .

TEMPERATURE SENSOR			
Bias Average	@ 25°C	350	LSB
Standard Deviation		290	LSB
Sensitivity Average		0.065	°C/LSB
Standard Deviation		0.0025	°C/LSB
Sensitivity Repeatability		±0.5	°C
Resolution		12	Bits

Σχήμα 22: Temperature Sensor

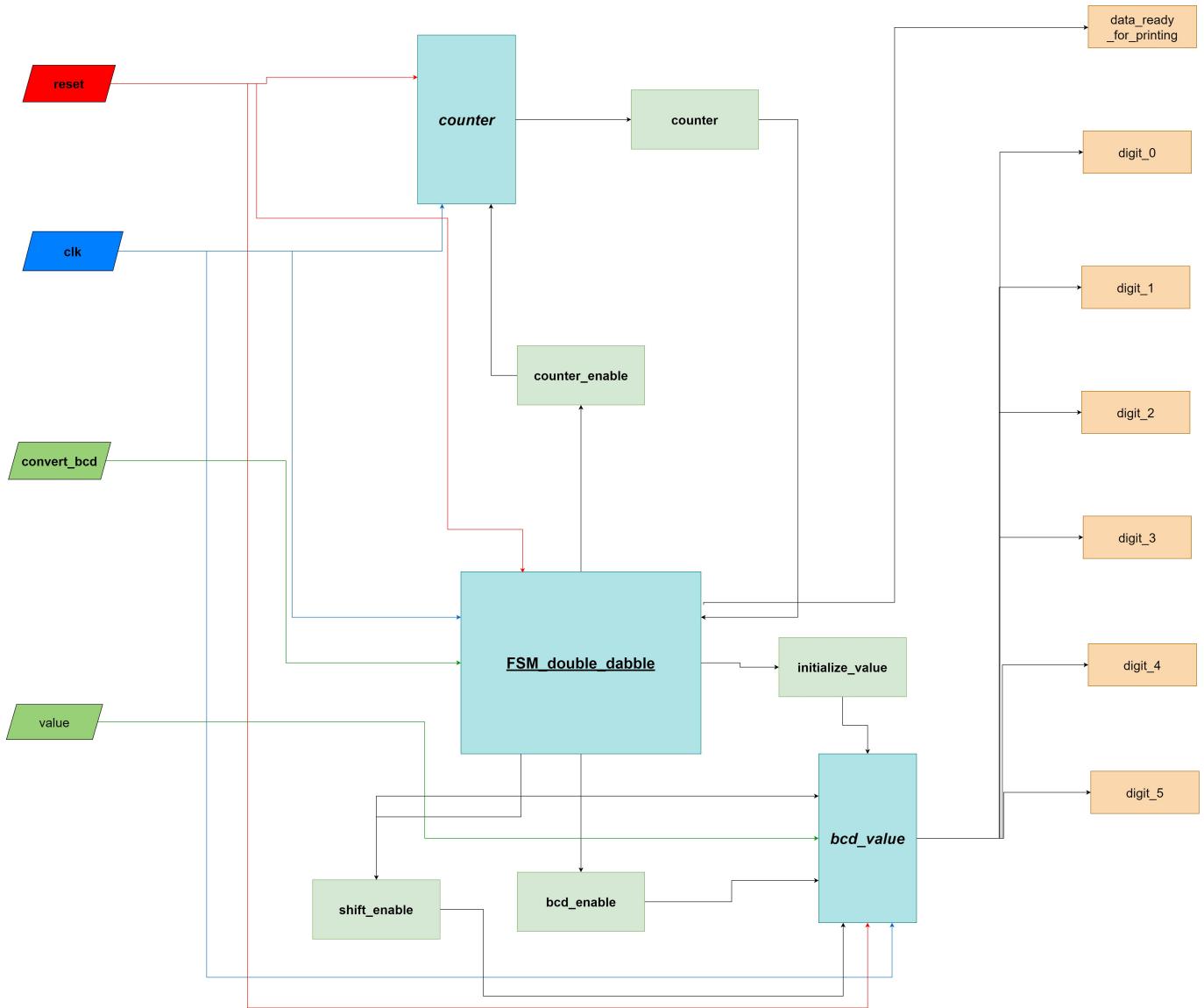
Για να αποφύγουμε τις πράξεις που προκύπτουν με δεκαδικούς και να δυσκολέψει την υλοποίηση μας θεωρήθηκε προτιμότερο να μετατρέψουμε τις συναρτήσεις σε ακέραιες τιμές, δηλαδή  $value_{acc} = 78 + 157 * avg_{acc}$  και  $value_{temp} = 65 * avgT$  γνωρίζοντας ότι οι τιμές που προκύπτουν είναι \*10000 g και \*1000 C αντίστοιχα.

Στην παρούσα φάση της υλοποίησης έχουμε τις τιμές έτοιμες στην μονάδα που επιθυμούμε και το μόνο που απομένει είναι να τους μετατρέψουμε σε μορφή ώστε να μπορούν να εκτυπωθούν στο τερματικό.

#### 4.3.1 Υλοποίηση Double Dabble module

Έχουμε ακέραιες τιμές οι οποίες επιθυμούμε να εκτυπωθούν στο τερματικό και για να επιτευχθεί αυτό, πρέπει αρχικά να διαχωρίσουμε τα τμήματα του αριθμού σε μονάδες, δεκάδες, εκατοντάδες κλπ ώστε εάν έχουμε π.χ. τον αριθμό 250, να έχουμε ξεχωριστά τα 2 5 0 ώστε καθένα από αυτά να μετατραπεί σε ascii και να εκτυπωθεί. Την συγκεκριμένη διεργασία υλοποιεί ο αλγόριθμος `double_dabble` γνωστός και ως `shift-and-add-3`. Ουσιαστικά άυτό που κάνει ο αλγόριθμος είναι ότι του εισάγουμε μία binary τιμή και την κάνει συνεχώς shift κατά 1 αριστερά όπου έχει τετράδες χωρισμένες για να χωράνε ένα ένα τα ψηφία του αριθμού. Στην συνέχεια ελέγχει εάν οποιαδήποτε από τις τετράδες έχει τιμή μεγαλύτερη ή ίση από 5 και προσθέτει 3 και συνεχίζει για η φορές, όπου η το μέγεθος του αριθμού που εισάγουμε. Έτσι προκύπτουν ξεχωριστά τα ψηφία σε 4-bit τιμές.

Παρακάτω φαίνεται η υλοποίηση του module double dabble



Σχήμα 23: dataflow data sort

Αρχικά χρειαζόμαστε έναν register που να χωράει την τιμή που θα εισάγουμε προς μετατροπή καθώς επίσης και τα παράγωγά τους. Τα παράγωγα είναι μονοψήφιοι δεκαδικοί αριθμοί οπότε κάθε ψηφίο χρειάζεται το πολύ 4 bits αριθμό. Για να υπολογίσουμε το μέγεθος του register χρησιμοποιούμε τον τύπο  $n + 4 * \text{ceil}(n/3)$  όπου  $n$  είναι το μέγεθος του αριθμού που εισάγουμε. Στην περίπτωση των επιταχύνσεων προκύπτει μέγεθος 40 (16 bits input), ενώ για την θερμοκρασία 48 (20 bits input). Στην συνέχεια ο αλγόριθμος ξεκινάει να κάνει το shift και να ελέγχει εάν οποιαδήποτε από τις τετράδες είναι μεγαλύτερη ή ίση με 5 για να προσθέσει 3 και να συνεχίσει για η φορές όπως αναφέρθηκε.

Παρακάτω φαίνεται το κομμάτι του κώδικα που υλοποιεί την συγκεκριμένη λειτουργία

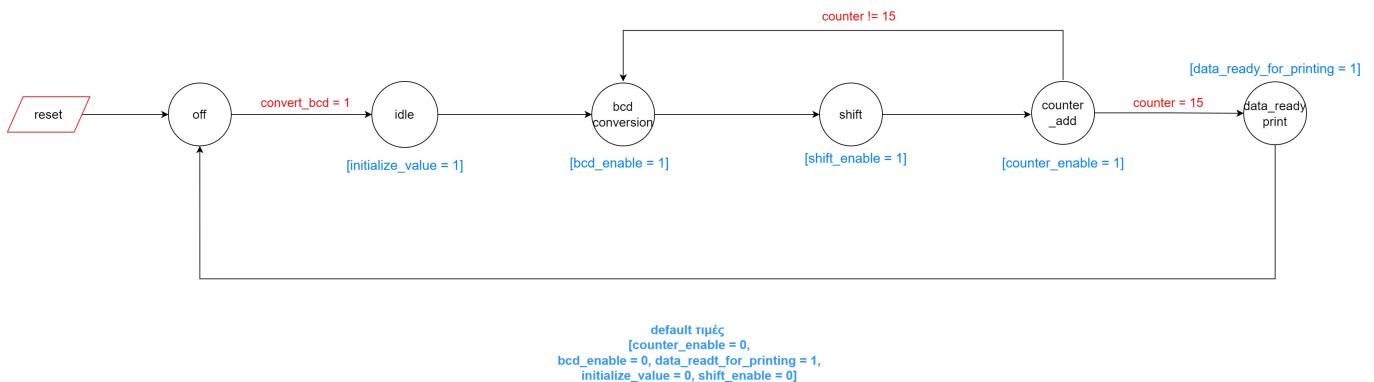
```

always @(posedge clk or posedge reset)
begin
    if(reset)
        bcd_value <= 40'b0;
    else
        begin
            if(initialize_value)
                bcd_value <= {24'b0, value};
            else if(bcd_enable)
                begin
                    if(bcd_value[39:36] >= 4'd5)
                        bcd_value[39:36] <= bcd_value[39:36] + 4'd3;
                    if(bcd_value[35:32] >= 4'd5)
                        bcd_value[35:32] <= bcd_value[35:32] + 4'd3;
                    if(bcd_value[31:28] >= 4'd5)
                        bcd_value[31:28] <= bcd_value[31:28] + 4'd3;
                    if(bcd_value[27:24] >= 4'd5)
                        bcd_value[27:24] <= bcd_value[27:24] + 4'd3;
                    if(bcd_value[23:20] >= 4'd5)
                        bcd_value[23:20] <= bcd_value[23:20] + 4'd3;
                    if(bcd_value[19:16] >= 4'd5)
                        bcd_value[19:16] <= bcd_value[19:16] + 4'd3;
                end
            else if (shift_enable)
                bcd_value <= bcd_value << 1;
        end
end

```

Σχήμα 24: double dabble code

Το συγκεκριμένο κομμάτι ελέγχεται από μία FSM του κυκλώματος που παράγει όλα τα απαραίτητα σήματα ελέγχου για την λειτουργία του.

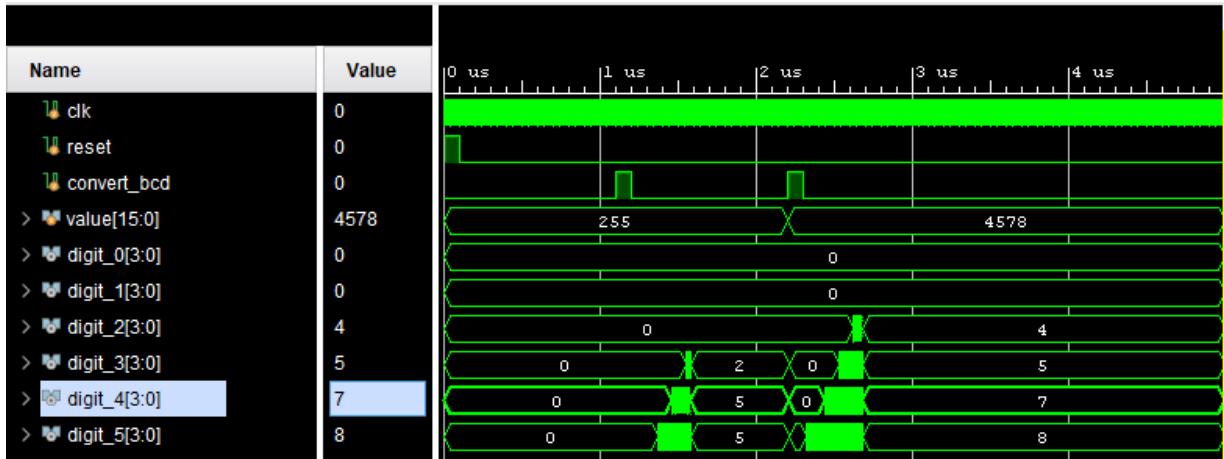


Σχήμα 25: dataflow data sort

Η συγκεκριμένη FSM είναι τύπου Moore και αποτελείται από 6 states. Αρχικά, στο state `off` το κύκλωμα είναι σε αδράνεια και όταν δοθεί εντολή από το module `data sort` ότι τα δεδομένα είναι έτοιμα για μετατροπή σε BCD μορφή αρχικοποιεί τον register με την τιμή που λαμβάνει και ξεκινάει το `conversion` στο state `bcd_conversion`. Στην συνέχεια προχωράει στο state `shift` όπου κάνει το `shift` αριστερά που χρειάζεται και προχωράει στο `counter_add` όπου αυξάνει έναν counter για να μετράει το πόσες φορές έτρεξε ο αλγόριθμος. Η διαδικασία επαναλαμβάνεται διαρκώς έως ότου οι επαναλήψεις γίνονται ίσες με n (15) όπου προχωράει στο state `data_read_print` όπου στέλνει ένα σήμα ότι η μετατροπή έχει ολοκληρωθεί. Με αυτόν τον τρόπο λαμβάνουμε τα δεδομένα μας σε μορφή `bcd` που είναι το επιθυμητό.  
Ένα τέτοιο module για κάθε άξονα γίνεται instantiate στο module `data sort` για να γίνεται ταυτόχρονα η μετατροπή, ενώ δημιουργείται ένα ίδιο module για την θερμοκρασία με διαφορετικά όρια τιμών.

#### 4.3.2 Επαλήθευση Double Dabble Module

Για την επαλήθευση του συγκεκριμένου module αρκεί να εισάγουμε μερικές τιμές και να παρατηρήσουμε από τις κυματομορφές εάν τα παράγωγά του είναι σωστά. Υλοποιούμε ένα testbench που προσομοιώνει το ρολό της FPGA των 100 Mhz και στέλνει στο module double dabble τις τιμές 255 και μετά από λίγο 4578. Από τις κυματομορφές παρατηρούμε ότι το κύκλωμά μας λειτουργεί, εφόσον δίνει σωστό αποτέλεσμα.



Σχήμα 26: Waveforms Double Dabble

Έχοντας υλοποιήσει την παραπάνω μετατροπή, επιστρέφουμε στο module Data\_Sort όπου πλέον έχουμε τις τιμές σε μορφή BCD. Απαιτείτε προσοχή στον τρόπο που περνάνε τα δεδομένα στο module που κάνει την μετατροπή, καθώς εάν η τιμή είναι αρνητική ο αλγόριθμος δεν θα λειτουργήσει σωστά. Οπότε πριν περάσουμε τις τιμές ελέγχοντας το MSB από κάθε τιμή καταλαβαίνουμε εάν είναι αρνητικός (εάν το MSB είναι 1) και μετατρέπουμε την τιμή σε 2's compliment και προσθέτουμε 1 για να λαβουμε την ακέραια τιμή του αριθμού.

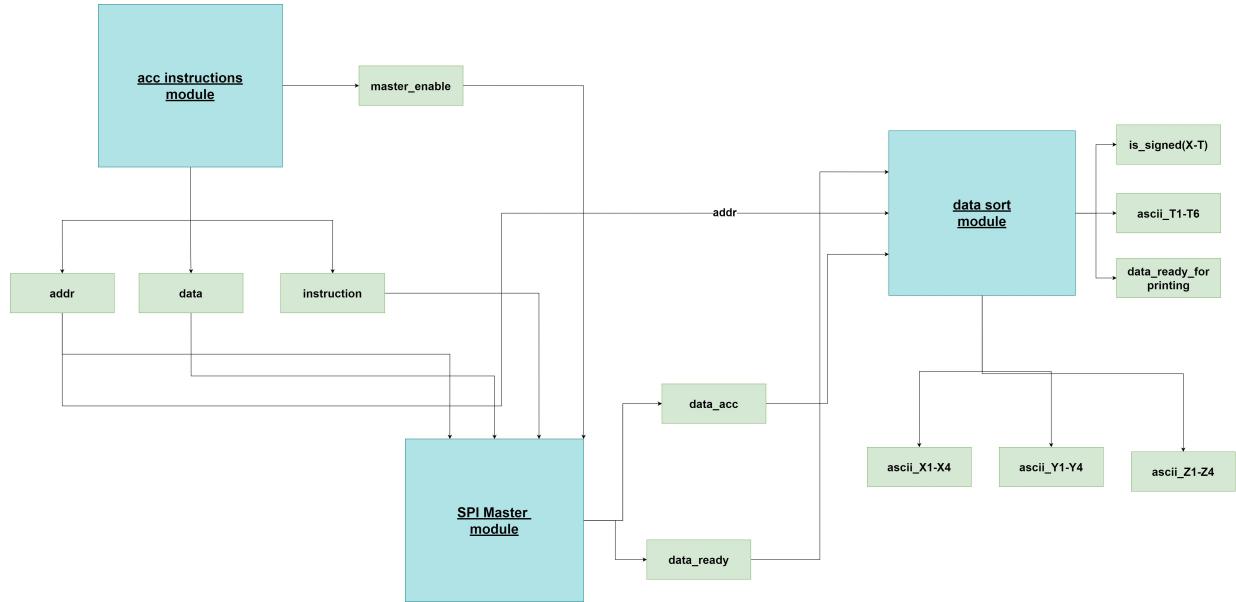
Στην παρούσα φάση αυτό που μένει για να είναι οι τιμές έτοιμες για εκτύπωση είναι να μετατρέψουμε το κάθε ψηφίο σε ASCII δηλαδή εκτυπώσιμο χαρακτήρα. Οι τιμές ascii των αριθμών 0-9 είναι στην σειρά γνωρίζοντας την τιμή του 0 (8'b000110000) αρκεί να προσθέτουμε σε αυτόν τον αριθμό την τιμή του κάθε ψηφίου και λαμβάνουμε τον αντίστοιχο εκτυπώσιμο χαρακτήρα.

48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9

Σχήμα 27: Ascii Values

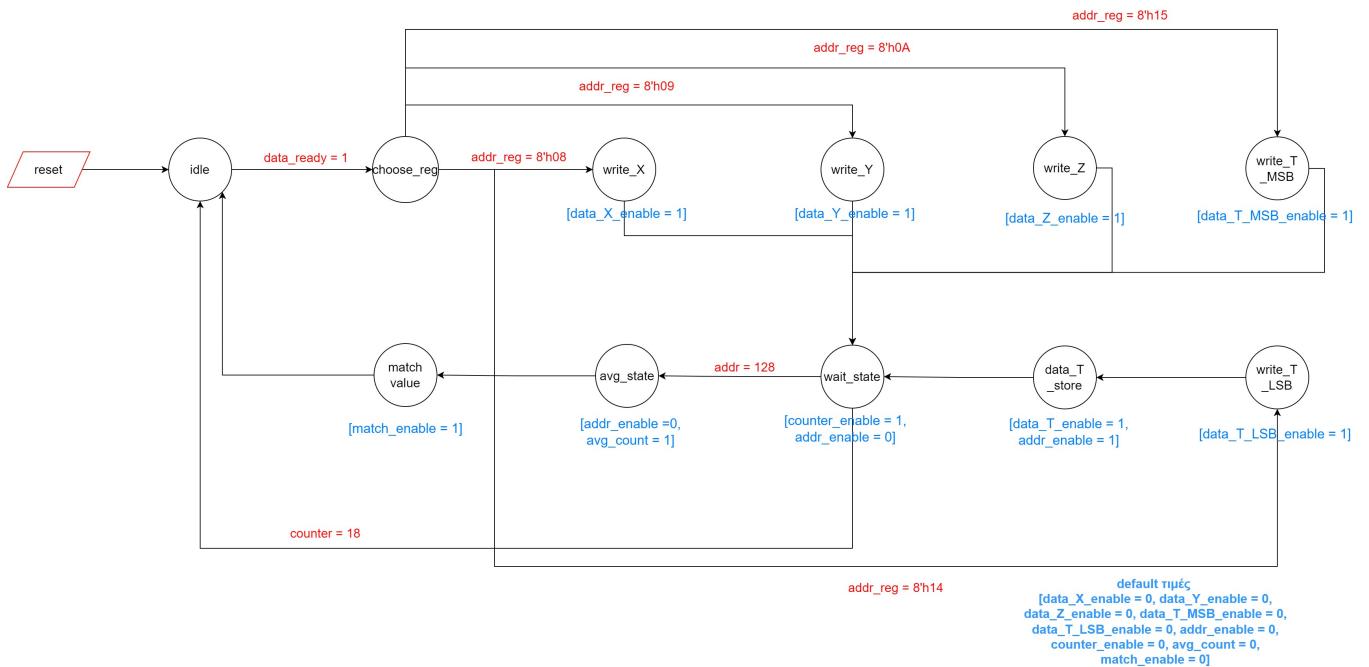
Όλες αυτές οι λειτουργίες ελέγχονται από σήματα ελέγχου που στέλνει η FSM που υπάρχει στο module Data\_Sort

Προτού αναλυθεί η FSM παρατίθεται ένα dataflow με τις συνδέσεις των modules SPI\_Master, Accelerometer\_instructions, Data\_Sort και τα σήματα που στέλνει το ένα στο άλλο.



Σχήμα 28: Specific connections for Data Sort

Στην συνέχεια ακολουθεί το σχήμα για την FSM



Σχήμα 29: FSM Data sort

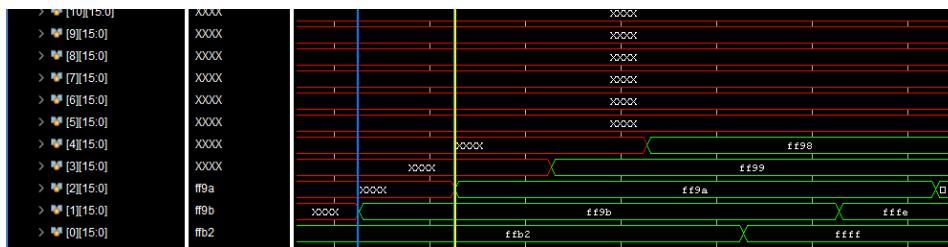
Η συγκεκριμένη FSM είναι τύπου Moore και αποτελείται από 11 states.

Αρχικά, το κύκλωμα βρίσκεται στο state `idle` και είναι σε αδράνεια, έως ότου το σήμα `data ready` από το SPI Master γίνει 1, όπου προχωράει στο state `choose_reg` στο οποίο γίνεται έλεγχος με βάση την διεύθυνση των registers του accelerometer σε ποιον ανήκουν τα δεδομένα που έρχονται και γράφονται στον αντίστοιχο πίνακα προχωρόντας στο ανάλογο state π.χ. `write_X`. Ιδιαίτερότητα έχει η θερμοκρασία όπου πρώτα γράφουμε τα MSB δεδομένα που έρχονται σε έναν 16 bits register, και στην συνέχεια τα LSB στις αντίστοιχες θέσεις του register. Εφόσον

έχουν γραφτεί όλες οι τιμές και η διεύθυνση των πινάκων δείχνουν 128 προχωράμε στο state avg\_state όπου στέλνετε σήμα για παραγωγή του AVG και τέλος στο state match\_value όπου στέλνετε σήμα για αντιστοίχηση της τιμής σε g και celsius. Τέλος η FSM επιστρέφει στο idle όπου περιμένει ξανά σήμα από το SPI για να συνεχίσει.

#### 4.4 Επαλήθευση Data Sort - SPI Master

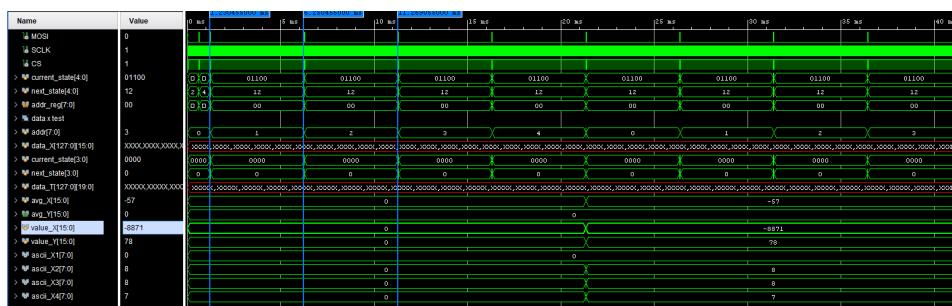
Για την επαλήθευση του Data Sort χρειάζεται να συνδέσουμε όλο το κύκλωμα μαζί μέχρι αυτό το σημείο για να παρατηρήσουμε ότι στέλνοντας κάποιες τιμές στους registers του X ας πούμε όντως βγάζει τον μέσο όρο και το μετατρέπει αρχικά στην σωστή τιμή και στην συνέχεια σε ascii χαρακτήρες. Οπότε υλοποιούμε ένα testbench παρόμοιο με εκείνο του SPI και στέλνουμε τιμές 4 φορές όταν διαβάζουμε X και το Y δειγματοληπτώντας 4 φορές (αντί για 128) ώστε να παρατηρήσουμε τις κυματομορφές. Αρχικά πάρατηρούμε ότι γράφονται τιμές στον πίνακα X όπως φαίνεται στην παρακάτω εικόνα



### Σχήμα 30: Data on array X

Στην συνέχεια στέλνουμε τις δεκαδικές τιμές για το X -76 , -46, -36, -66 ώστε να παρατηρήσουμε εάν προκύπτει σωστός μέσος όρος. Ο μέσος όρος που πρέπει να προκύψει είναι περίπου -57, οπότε αυτή είναι και η τιμή που παρατηρούμε στις κυματομορφές. Στην συνέχεια το value μετά από τον τύπο πρέπει να έχει τιμή -8871, οπότε τρέχει σωστά και αυτό το κομμάτι. Τέλος παρατηρούμε τις τιμές ascii όπου έχουμε βάλει στην προσομοίωση να φαίνονται σαν ascii ότι φαίνεται ο αριθμός 887 το οποίο είναι σωστό καθώς έχουμε βγάλει το τελευταίο ψηφίο διότι δεν χρειάζεται τόσο μεγάλη ακρίβεια.

Όλα τα παραπάνω φαίνονται στην παρακάτω εικόνα της προσομοίωσης



Σχήμα 31: Waveforms on top module simulation

Εφαρμόζοντας την συγκεκριμένη μέθοδο και για τους υπόλοιπους άξονες και για περισσότερες τιμές, καταλήγουμε στο αποτέλεσμα ότι το κύκλωμα σε αρκετά μεγάλο βαθμό λειτουργεί σωστά, καθώς έχουμε τηρήσει τα constraints που αναγράφονται στο datasheet όσον αφορά το πως ακριβώς στέλνει δεδομένα το επιταχυνσιόμετρο, όπως φαίνεται και στην εικόνα παρακάτω τα σήματα tv και tdis.

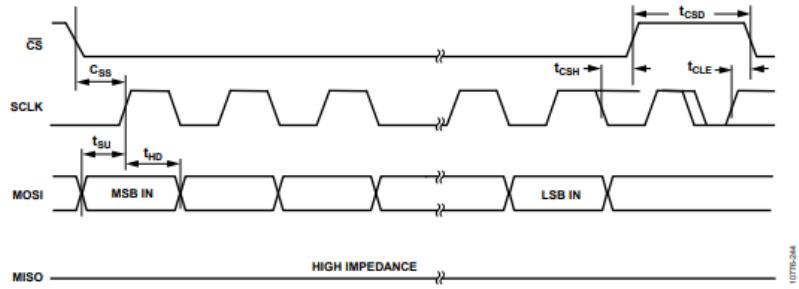


Figure 41. Timing Diagram for SPI Receive Instructions

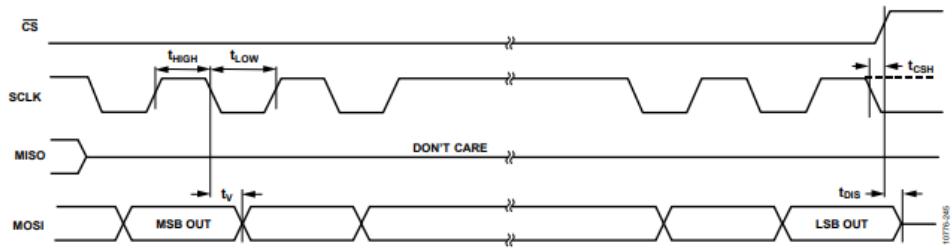
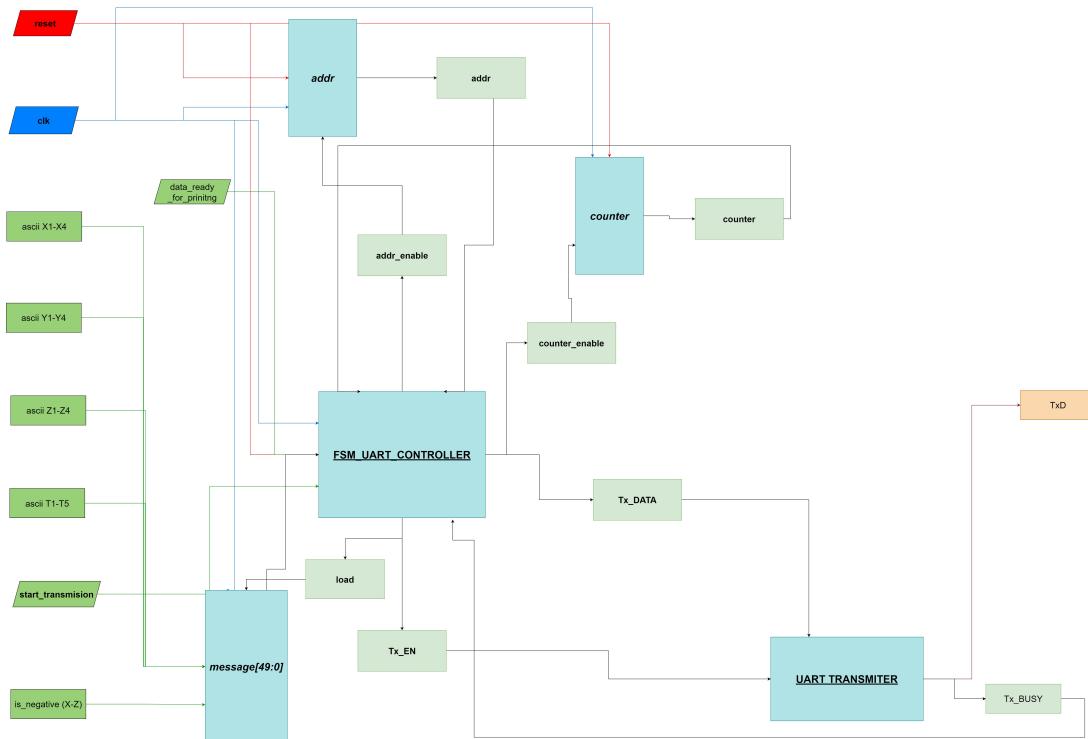


Figure 42. Timing Diagram for SPI Send Instructions (Shaded Portions of Figure 36, Figure 38, and Figure 40)

### Σχήμα 32: Accelerometer timing Constraints

## 5 Υλοποίηση Transmitter Controller

Χρειαζόμαστε μία μονάδα η οποία θα ελέγχει τι τιμές θα στέλνει το UART Transmitter και πότε. Αυτή ή μονάδα είναι το module Transmitter Controller η υλοποίηση του οποίου φαίνεται παρακάτω.

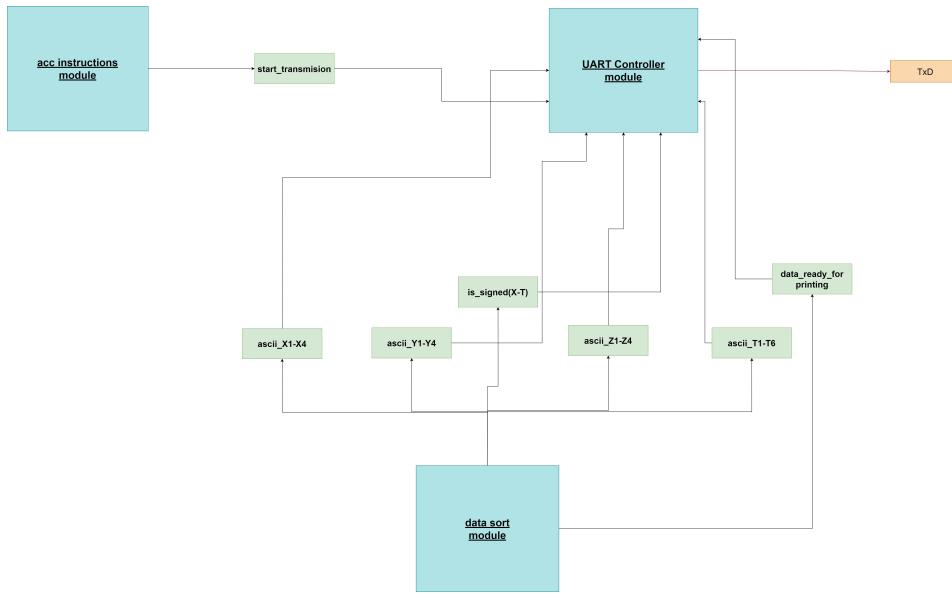


Σχήμα 33: Dataflow Transmitter Controller

Η συγκεκριμένη υλοποίηση έχει έναν πίνακα 50 θέσεων με 8 bits η κάθε μία, όπου έχει

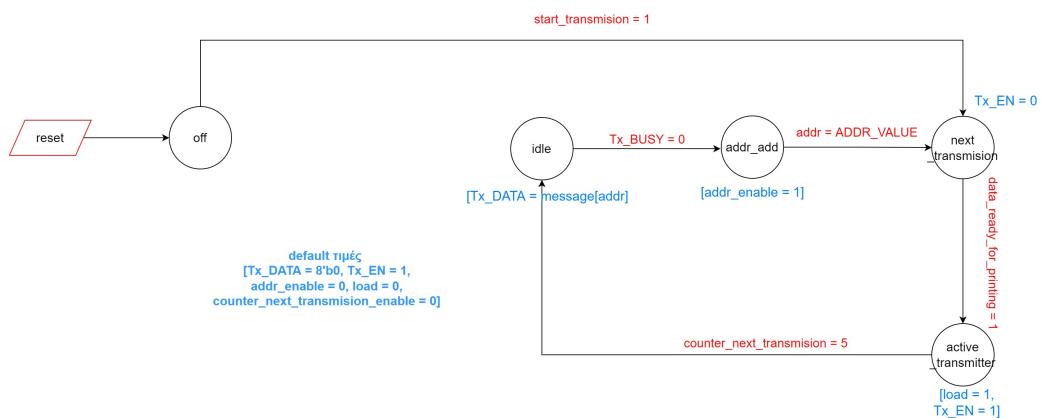
αποθηκευμένο το μήνυμα προς αποστολή. Κάθε χαρακτήρας αποτελεί μία ascii τιμή των 8 bits. Αρχικά αποθηκεύουμε σε κάθε θέση τον χαρακτήρα που θέλουμε να εκτυπώσουμε και όπου είναι να εισάγουμε ψηφίο από τιμή για κάποια μέτρηση την ανανεώνουμε όταν έρθει το αντίστοιχο σήμα από το module data sort. Επιπλέον από το data sort έρχεται και σήμα για το εάν η τιμή είναι θετική ή αρνήτική ώστε να εισαχθεί (-) ή κενό.

Οι συνδέσεις του Transmitter Controller με το υπόλοιπο κύκλωμα φαίνονται στο παρακάτω dataflow.



Σχήμα 34: Dataflow Transmitter Controller connections

Όλα αυτά ελέγχονται από μία FSM που υπαρχει στο κύκλωμα και το σχηματικό της φαίνεται παρακάτω



Σχήμα 35: FSM Transmitter Controller

Η FSM είναι τύπου Moore και αποτελείται από 5 states τα οποία ουσιαστικά, απλά άυξάνουν την διεύθυνση που δείχνει στον πίνακα με αποθηκευμένη την τιμή και φορτώνουν το Tx-DATA με την αντιστοιχη τιμή ώστε να στείλει τα δεδομένα o Transmitter.

Επιθυμούμε να πετύχουμε μία σταθερή εικόνα οπότε επιλέγουμε υψηλό baud rate και σβήνουμε τα data και τα γραφουμε στην ίδια θέση ώστε να αλλάζει μόνο η τιμή από τις μετρήσεις.

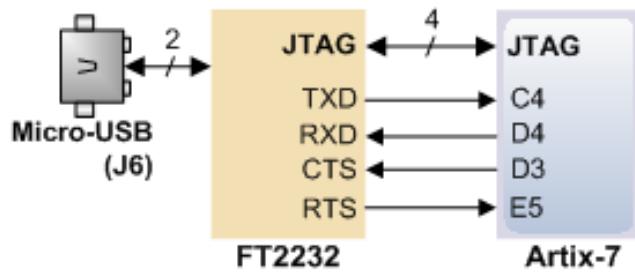
## 5.1 Έλεγος Transmitter Controller

Το κύκλωμα εφόσον ήταν υλοποιημένο από προηγούμενο εργαστήριο ελέγθηκε απευθείας στην πλακέτα στέλνοντας το μήνυμα που επιθυμούμε χωρίς όμως τις τιμές από τις μετρήσεις, και λειτούργησε κανονικά.

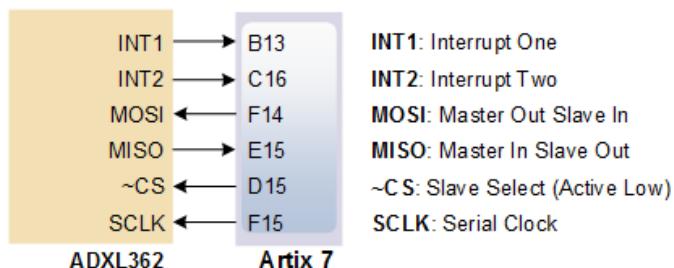
## 5.2 Generate Bitstream και δοκιμή στήν πλακέτα

### 5.2.1 Constraints

Για να δημιουργηθεί το αρχείο των constraints, διαβάστηκε αρχικά το datasheet της πλακέτας και βρίσκοντας τα αντίστοιχα σημεία, έγινε σωστή αντιστοιχία των βασικών σημάτων που είναι τα εξής: Clk, reset, SCLK, MISO, MOSI, CS, TxD. Παρακάτω φαίνονται οι εικόνες που αποδεικνύουν τις συνδέσεις.



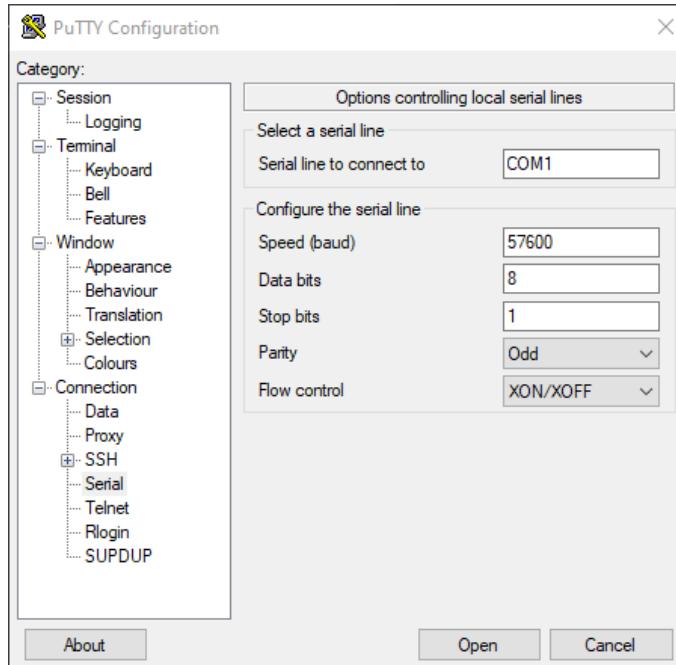
Σχήμα 36: Transmitter connections on FPGA



Σχήμα 37: ADXL 362 SPI connections on FPGA

### 5.2.2 Δοκιμή στήν πλακέτα

Αφού καταφέραμε να παράξουμε bitstream κάνουμε την δοκιμή μας στην πλακέτα. Χρησιμοποιήσαμε το εικονικό τερματικό putty το οποίο αρχικοποιήσαμε με το αντίστοιχο baud rate, 8 bits data και ένα bit parity.



Σχήμα 38: Dataflow Transmitter Controller connections

Αρχικά, το κύκλωμά δεν λειτουργούσε σωστά γιατί εκτύπωνε λάθος τιμές ακόμα και όταν ήταν στατική και ακίνητη η πλακέτα. Οπότε θεωρήθηκε ότι υπάρχει πρόβλημα στο SPI Master. Για να ελεγχθεί διαβάστηκαν κάποιοι registers που έχουν σταθερή τιμή μέσα, οπότε εάν διάβαζε αυτό ήταν σωστό. Δεν διάβασε αυτό και βρέθηκε ότι υπήρχε σφάλμα με τους χρόνους του SPI, οπότε άλλαξε και ξαναδοκιμάστηκε. Επιτέλους, λειτούργησε σωστά και διαβάζει τις τιμές όπως ακριβώς αναφέρεται στο manual ανάλογα με τις κλίσεις. Μοναδικό σφάλμα είναι στην θερμοκρασία όπου ενω φαίνεται να διαβάζει σωστά η θέση όπου υπολογίζεται ότι πρέπει να μπει το κόμα είναι μία πριν, οπότε ίσως κάποιο σφάλμα στις πράξεις που δεν βρέθηκε.

Ονοματεπώνυμο: Δημήτριος Τσαλαπάτας

AEM: 03246