Σταθουδάκης Σταύρος, AEM:03512 17/1/2024 Lab 3 VGA

Περίληψη

Υλοποίηση του πρωτόκολλου επικοινωνίας VGA σε ανάλυση 640χ480 με την χρήση μνήμης VRAM χωρητικότητας 128χ96. Χρησιμοποιήθηκαν 6 αρχεία συνθέσιμης Verilog τα οποία ελέγχθηκαν για την ορθή λειτουργία τους πάνω σε μία Nexys A7-100T FPGA.

Εισαγωγή

Ο στόχος της εργασίας ήταν να καταφέρουμε να οδηγήσουμε μία οθόνη VGA να δείχνει την εικόνα που είχαμε αποθηκεύσει στην μνήμη της VRAM. Αυτό έπρεπε να γίνει σε ανάλυση 640χ480 και σε ρυθμό ανανέωσης 60 Hz. Λόγω μικρής χωρητικότητας της εσωτερικής μνήμης της FPGA, αναγκαστήκαμε να χρησιμοποιήσουμε μία μικρότερη μνήμη 128χ96 για κάθε χρώμα διαχειρίζοντας κατάλληλα την διεύθυνση πρόσβασης της μνήμης για σωστή απεικόνιση της. Αυτός ο στόχος επιτεύχθηκε με την χρήση 6 συνθέσιμων αρχείων Verilog, 2 FSM’s και 3 μνήμες VRAM.

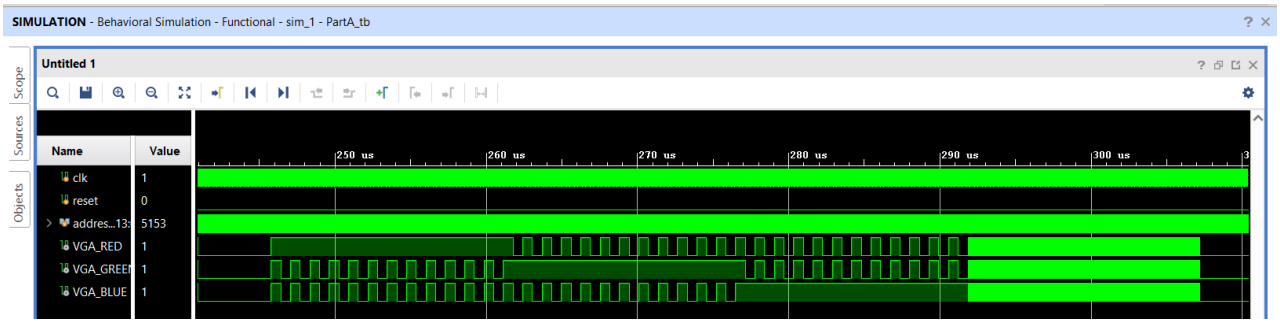
Μέρος Α - Υλοποίηση VRAM

Υλοποίηση

- Χρησιμοποιήθηκε το έτοιμο αρχείο που περιέχει την (κενή αρχικά) VRAM το οποίο προσφέρει το Vivado. Αρχικοποίηθηκαν 3 μνήμες στα 16 KB, μία για κάθε χρώμα.
- Μετατράπηκε ο κώδικας για να λειτουργεί ως διαφορετικό module έχοντας ως input το clk και το reset της FPGA καθώς και την διεύθυνση πρόσβασης της μνήμης. Ως output έχει μόνο την τιμή της μνήμης στην εισερχόμενη διεύθυνση. Έχω θέσει την τιμή του write enable μόνιμα στο 0 καθώς δεν θα χρειαστεί να αλλάξουμε δυναμικά την μνήμη, αντίστοιχα έχω θέσει την μεταβλητή DI στην τιμή 0 καθώς είναι αυτή που ορίζει το μέγεθος της μνήμης το οποίο πρόκειται να γίνει write και εφόσον εμείς δεν κάνουμε write δεν έχει σημασία την τιμή της. Επίσης έχω θέσει την τιμή του enable στο 1 μόνιμα διότι αυτή είναι που επιτρέπει την πρόσβαση στην μνήμη ή όχι και δεν ειπώθηκε κάποιος περιοσμός στο πότε θα μπορούμε ή οχι να μπορούμε να έχουμε πρόσβαση στην μνήμη απο την εκφώνηση της εργασίας. Ακόμα η τιμή του REGCE register είναι και αυτή μόνιμα αρχικοποιμένη στο 0.
- Όσο για την αρχικοποίηση της μνήμης για κάθε χρώμα, εφόσον λειτουργούμε σε ανάλυση όπου κάθε γραμμή αποτελείται από τα ίδια 128 bit και μία γραμμή στην μνήμη είναι 256 bit, η κάθε γραμμή της μνήμης αντιστοιχεί σε 2 σειρές στην οθόνη. Έτσι όταν σε κάποια γραμμή θέλω να φαίνεται μόνο ένα χρώμα, έχω μόνο εκείνο το χρώμα σε τιμή F και τα υπόλοιπα τα έχω στην τιμή 0. Αντίστοιχα όταν θέλω να έχω άσπρο έχω όλα τα χρώματα «ενεργά», δηλαδή στην τιμή F και όταν θέλω μαύρο χρώμα έχω όλα τα χρώματα «ανενεργά» δηλαδή στην τιμή 0. Συνολικά για κάθε χρώμα καταλαμβάνονται 48 γραμμές στην μνήμη (που αντιστοιχούν σε 96 γραμμές στην οθόνη) όπου κάθε μία αποτελείται από 256 bite. Άρα συνολικά χρησιμοποιούμε 48*256=12,288 BYTES για κάθε ένα από τα 3 χρώματα.

Επαλήθευση

- Η στρατηγική επαλήθευσης είναι αρκετά απλή. Απλά καλώ και τις 3 VRAM αλλάζοντας την διεύθυνση πρόσβασης συνεχόμενα και έπειτα βλέπω στις κυματομορφές εάν τα output της κάθε VRAM είναι σωστά συγχρονισμένα.
- Τα διανύσματα ελέγχου είναι τα output από τις VRAM. Το ποσοστό κάλυψης του κώδικα είναι στο 100% καθώς αλλάζοντας την διεύθυνση συνεχώς διαπερνάμε ολόκληρη την VRAM, δηλαδή όλες τις τιμές που μας ενδιαφέρουν και μας επηρεάζουν.
- Δεν αντιμετώπισα κάποιο σοβαρό πρόβλημα σε αυτό το μέρος.
- Παρακάτω ακολουθεί ένα στιγμιότυπο από το simulation αυτού του μέρους στο οποίο μπορεί να παρατηρήσει κανείς ότι τα σήματα για κάθε χρώμα πάνε έναλλάξ, δηλαδή όταν έχω κόκκινες και άσπρες γραμμές, το κόκκινο παραμένει μόνιμα ενεργό και τα υπόλοιπα 2 χρώματα είναι ενεργά μόνο όταν θέλω να έχω άσπρη γραμμή. Έτσι αντίστοιχα αυτό είναι αλήθεια και για το μπλε και το πράσινο. Στην συνέχεια όπου θέλουμε να έχουμε και κάθετες λωρίδες χρωμάτων δεν φαίνονται με λεπτομέρεια οι αλλαγές που γίνονται σε κάθε χρώμα διότι είναι πολύ γρήγορες και θα έπρεπε το scale να είναι πολύ μικρότερο. Πάντως και εκεί επικρατεί η ίδια πρακτική.



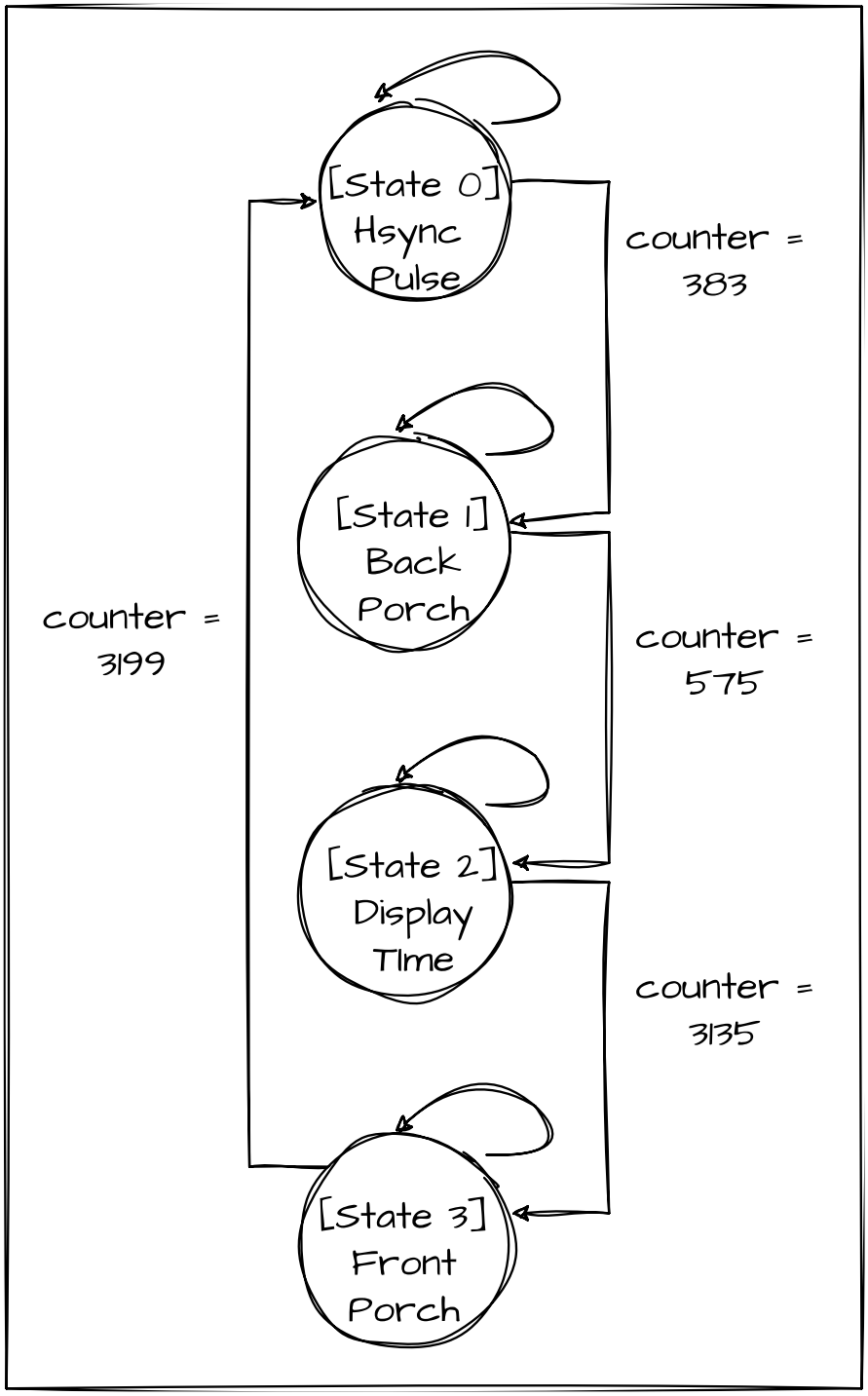
Μέρος Β - Υλοποίηση Hsync και Hpixel

Υλοποίηση

- Σε αυτό το μέρος καλούμαστε να υλοποιήσουμε ένα σήμα το οποίο θα είναι υπεύθυνο για τον συγχρονισμό του display time ανά γραμμή και την υλοποίηση ενός οριζόντιου μετρητή που θα είναι δείκτης για το εκάστοτε ενεργό bit της VRAM για κάθε χρώμα. Ο κώδικας αποτελείται ουσιαστικά από μία FSM και για αυτό δεν υπάρχει κάποιο dataflow για ολόκληρο το μέρος Β, αλλά μόνο για την FSM που παρουσιάζετε παρακάτω.
- Βασική δομή:
    - Η FSM έχει 4 state, όπου το καθένα αντιπροσωπεύει μία κατάσταση του HSYNC και του σήματος όπου επιτρέπει την προβολή της VRAM στην οθόνη. Χρησιμοποιώ έναν counter για να μετατοπίσω την FSM στο επόμενο state, ο οποίος προκύπτει αθροιστικά από τους απαραίτητους χρόνους για να προβάλλεται το μήνυμα στα 60 Hz. Αυτοί οι χρόνοι δίνονται στην εκφώνηση και για να τους μετατρέψω σε κύκλους ρολογιού τους μετέτρεψα σε ns και τους διαίρεσα με το 10 καθώς το ρολόι της FPGA είναι 10 ns. Έτσι καταλήγουμε με τις ακόλουθες τιμές για τον counter:
        - A (Total time) :3200 cycles
        - B (Hsync Pulse):384 cycles
        - C (Back Porch):192 cycles
        - D (Dislay Time):2560 cycles
        - E (Front Porch):64 cycles

Σε περίπτωση όπου ο counter δεν έχει την επιθυμητή τιμή για να προχωρήσει το επόμενο state, τότε θα παραμείνει στο ίδιο μέχρι να αυξηθεί ο counter επαρκώς.

Ακολουθεί το κατάλληλο dataflow:

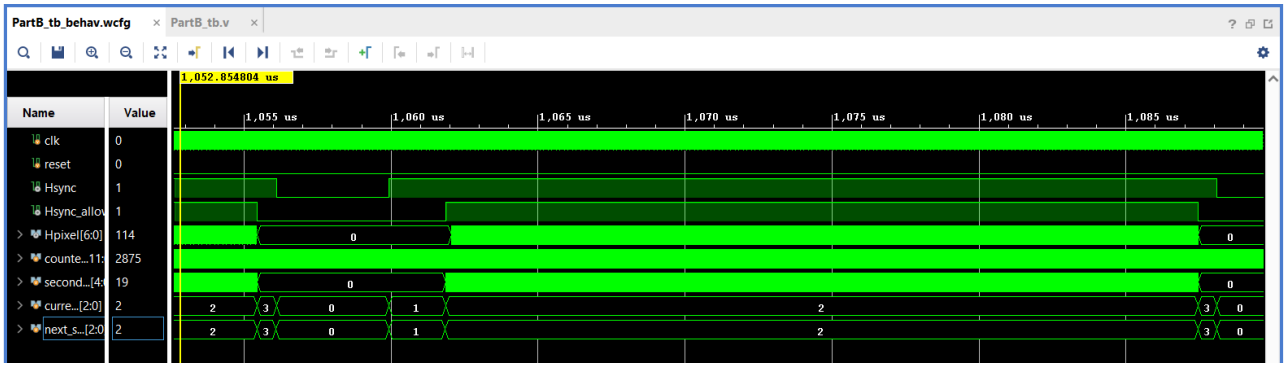


- Εκτός από την συνδυαστική always όπου περιέχει τα states της FSM, υπάρχει και μία ακολουθιακή always που ανήκει στην FSM όπου το μόνο που κάνει είναι τα εξισώνει το τρέχων state με το επόμενο όταν το reset δεν είναι 0. Ακόμα έχω ακόμα μία εξωτερική ακολουθιακή always όπου παίζει τον βοηθητικό ρόλο προς την FSM και είναι υπεύθυνη για την αλλαγή τιμής του counter και της μεταβλητής Hpixel.

- Η τιμή του counter αυξάνεται κατά 1 σε κάθε θετική ακμή του ρολογιού όταν το reset δεν είναι 0. Η τιμή του Hpixel αλλάζει μόνο όταν η FSM βρίσκεται στο state 2, δηλαδή όταν θέλουμε να διατρέξουμε την μνήμη για να την προβάλουμε στην οθόνη. Η τιμή Hpixel αυξάνεται κατά 1 κάθε 2560(display cycles) / 128(μήκος γραμμής σε bits) = 20 θετικές ακμές του ρολογιού. Με αυτόν τον τρόπο καταφέρνουμε να έχουμε μεγαλύτερη ανάλυση οθόνης σε σχέση με αυτό που αντιστοιχεί στην μνήμη που έχουμε διαθέσιμη.

Επαλήθευση

- Για να επαληθεύσω την ορθή λειτουργία αυτού τους μέρους του κώδικα απλά καλώ το PartB στο testbench και παρατηρώ τις κυματομορφές των ακόλουθων σημάτων.
- Τα διανύσματα τα οποία κοιτάω για να σιγουρευτώ την ορθή λειτουργία του κώδικα είναι καταρχάς τα current_state και next_state της FSM και έπειτα εάν τα σήματα που έχω ως έξοδο ακολουθούν αυτά. Δηλαδή το Hsync θα πρέπει να είναι μόνο στο state 0 (Hsync Pulse) ίσο με 0 και το Hsync_allow θα πρέπει να είναι ίσο με 1 μόνο στο state 2 (Display time). Επίσης συμπληρωματικά έχω και τον εσωτερικό counter της FSM για να δω σε ποια τιμή αλλάζει η FSM state. Ακόμα εμφανίζω και το Hpixel για να δω ότι μόνο όσο το Hsync_allow είναι 1, το Hpixel αλλάζει και τερματίζει στο 127 γιατί αλλιώς θα βγαίναμε εκτός ορίων στην μνήμη. Το ποσοστό κάλυψης του κώδικα για τα ζητούμενα της εργασίας είναι στο 100% καθώς έτσι μπορούμε να μετρήσουμε όλους του χρόνους που δίνονται από την εκφώνηση με ακρίβεια επαληθεύοντας έτσι την ορθή λειτουργία του όλου συστήματος. Είναι πολύ σημαντικό να έχω πετύχει τους χρόνους τους οποίους είναι ενεργό το κάθε state, σε ακρίβεια κύκλου μηχανής, διότι άμα δεν είναι ακριβώς η οθόνη δεν αναγνωρίζει την ανάλυση όπου θέλουμε να προβάλουμε το μύνημα και μπαίνει σε power save mode.
- Στην δοκιμή αυτή στο behavioral simulation κατάλαβα μετρώντας τους χρόνους ότι έχανα έναν κύκλο μηχανής και έτσι η οθόνη δεν έδειχνε το επιθυμητό μήνυμα όπως ανέφερα παραπάνω. Αυτό γινόταν επειδή στον counter της FSM δεν υπολόγιζα ότι αρχίζει από το 0 και τον τερμάτιζα έναν κύκλο πιο μετά απ' ότι πρέπει. Κατά τ' αλλα δεν αντιμετώπισα κάποιο ιδιαίτερο πρόβλημα.
- Παρακάτω φαίνεται ένα στιγμιότυπο από την προσομοίωση του συγκεκριμένου μέρους στο οποίο μπορούμε να παρατηρήσουμε όσα ανέφερα παραπάνω:



Μέρος Γ - Υλοποίηση Vync και Vpixel

Υλοποίηση

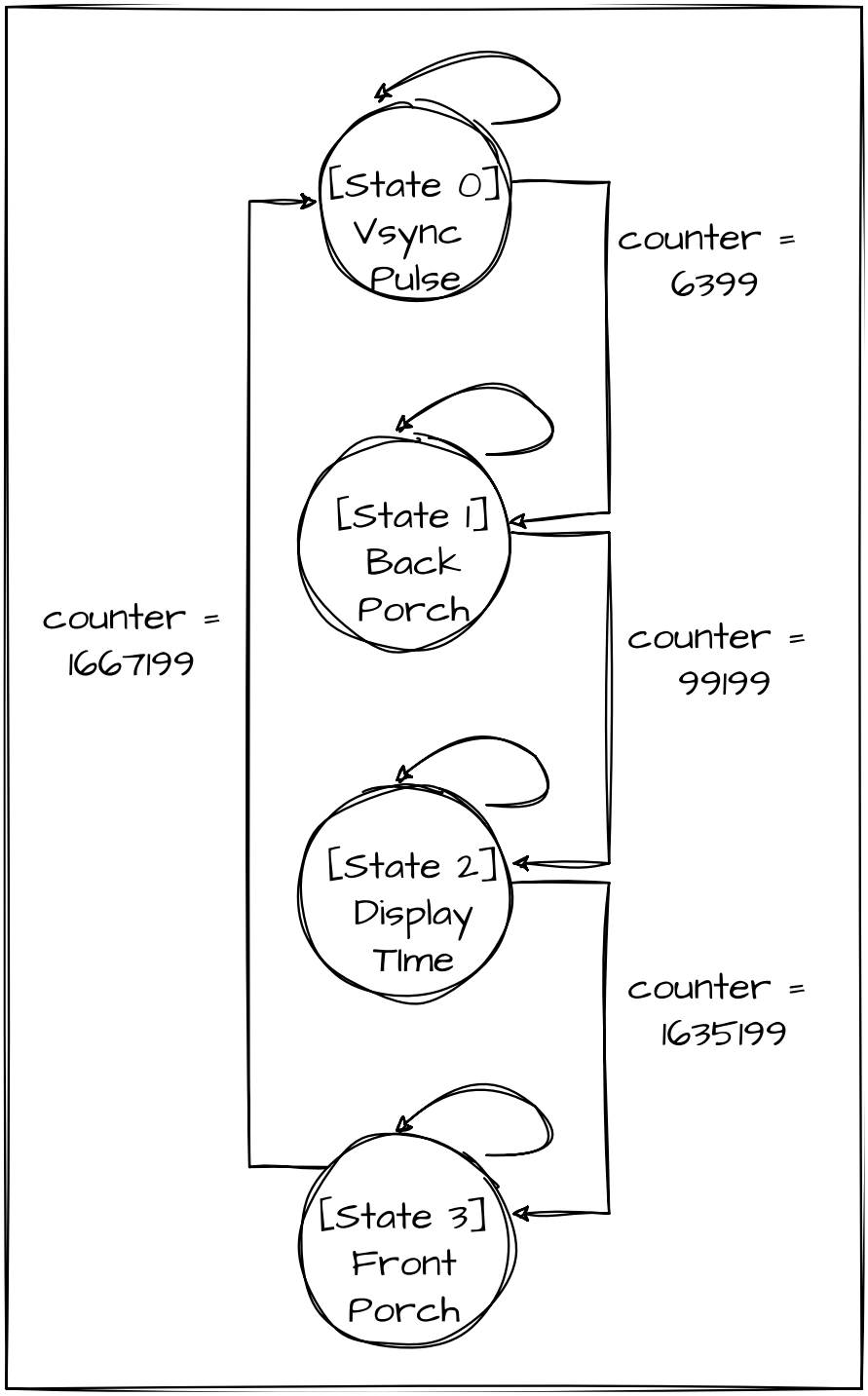
- Σε αυτό το μέρος καλούμαστε να υλοποιήσουμε ένα σήμα το οποίο θα είναι υπεύθυνο για τον συγχρονισμό του display time ανά οθόνη και την υλοποίηση ενός κάθετου μετρητή που θα είναι δείκτης για το εκάστοτε ενεργό bit της VRAM για κάθε χρώμα. Ο κώδικας

αποτελείται ουσιαστικά από μία FSM και για αυτό δεν υπάρχει κάποιο dataflow για ολόκληρο το μέρος Γ, αλλά μόνο για την FSM που παρουσιάζετε παρακάτω.

- Βασική δομή:
    - Η FSM έχει 4 state, όπου το καθένα αντιπροσωπεύει μία κατάσταση του VSYNC και του σήματος όπου επιτρέπει την προβολή της VRAM στην οθόνη. Χρησιμοποιώ έναν counter για να μετατοπίσω την FSM στο επόμενο state, ο οποίος προκύπτει αθροιστικά από τους απαραίτητους χρόνους για να προβάλλεται το μήνυμα στα 60 Hz. Αυτοί οι χρόνοι δίνονται στην εκφώνηση και για να τους μετατρέψω σε κύκλους ρολογιού τους μετέτρεψα σε ns και τους διαίρεσα με το 10 καθώς το ρολόι της FPGA είναι 10 ns. Έτσι καταλήγουμε με τις ακόλουθες τιμές για τον counter:
        - O (Tota frame) :1667000 cycles
        - P (Vsync Pulse) :6400 cycles
        - Q (Back Porch) :92800 cycles
        - R (Active Video):1536000 cycles
        - S (Front Porch):32000 cycles

      Σε περίπτωση όπου ο counter δεν έχει την επιθυμητή τιμή για να προχωρήσει το επόμενο state, τότε θα παραμείνει στο ίδιο μέχρι να αυξηθεί ο counter επαρκώς.

Ακολουθεί το κατάλληλο dataflow:



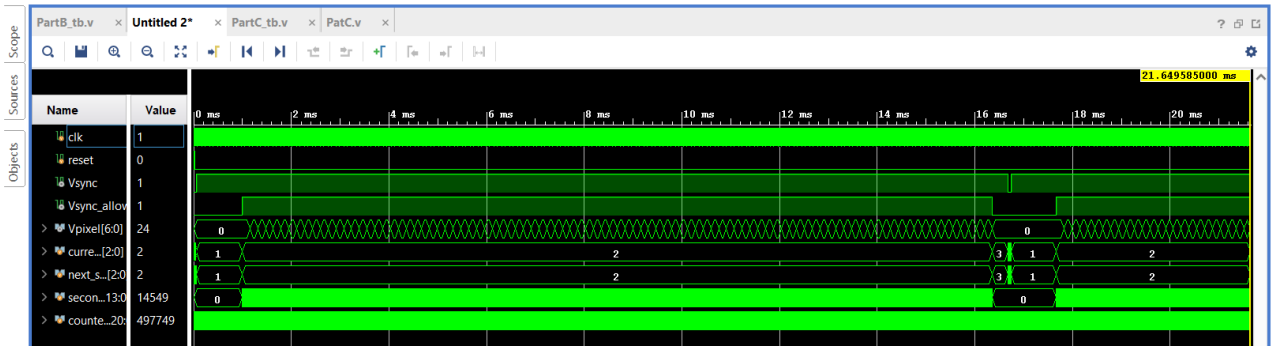
- Εκτός από την συνδυαστική always όπου περιέχει τα states της FSM, υπάρχει και μία ακολουθιακή always που ανήκει στην FSM όπου το μόνο που κάνει είναι τα εξισώνει το τρέχων state με το επόμενο όταν το reset δεν είναι 0. Ακόμα έχω ακόμα μία εξωτερική ακολουθιακή always όπου παίζει τον βοηθητικό ρόλο προς την FSM και είναι υπεύθυνη για την αλλαγή τιμής του counter και της μεταβλητής Vpixel.

- Η τιμή του counter αυξάνεται κατά 1 σε κάθε θετική ακμή του ρολογιού όταν το reset δεν είναι 0. Η τιμή του Vpixel αλλάζει μόνο όταν η FSM βρίσκεται στο state 2, δηλαδή όταν

θέλουμε να διατρέξουμε την μνήμη για να την προβάλουμε στην οθόνη. Η τιμή Vpixel αυξάνεται κατά 1 κάθε 1536000 (active video cycles) / 96(σύνολο διαθέσιμων γραμμών στην μνήμη) = 16000 θετικές ακμές του ρολογιού. Με αυτόν τον τρόπο καταφέρνουμε να έχουμε μεγαλύτερη ανάλυση οθόνης σε σχέση με αυτό που αντιστοιχεί στην μνήμη που έχουμε διαθέσιμη.

Επαλήθευση

- Για να επαληθεύσω την ορθή λειτουργία αυτού τους μέρους του κώδικα απλά καλώ το PartC στο testbench και παρατηρώ τις κυματομορφές των ακόλουθων σημάτων.
- Τα διανύσματα τα οποία κοιτάω για να σιγουρευτώ την ορθή λειτουργία του κώδικα είναι καταρχάς τα current_state και next_state της FSM και έπειτα εάν τα σήματα που έχω ως έξοδο ακολουθούν αυτά. Δηλαδή το Vsync θα πρέπει να είναι μόνο στο state 0 (Vsync Pulse) ίσο με 0 και το Vsync_allow θα πρέπει να είναι ίσο με 1 μόνο στο state 2 (Active video time). Επίσης συμπληρωματικά έχω και τον εσωτερικό counter της FSM για να δω σε ποια τιμή αλλάζει η FSM state. Ακόμα εμφανίζω και το Vpixel για να δω ότι μόνο όσο το Vsync_allow είναι 1, το Vpixel αλλάζει και τερματίζει στο 95 γιατί αλλιώς θα βγαίναμε εκτός ορίων στην μνήμη. Το ποσοστό κάλυψης του κώδικα για τα ζητούμενα της εργασίας είναι στο 100% καθώς έτσι μπορούμε να μετρήσουμε όλους του χρόνους που δίνονται από την εκφώνηση με ακρίβεια επαληθεύοντας έτσι την ορθή λειτουργία του όλου συστήματος. Είναι πολύ σημαντικό να έχω πετύχει τους χρόνους τους οποίους είναι ενεργό το κάθε state, σε ακρίβεια κύκλου μηχανής, διότι άμα δεν είναι ακριβώς η οθόνη δεν αναγνωρίζει την ανάλυση όπου θέλουμε να προβάλουμε το μύνημα και μπαίνει σε power save mode.
- Στην δοκιμή αυτή στο behavioral simulation κατάλαβα μετρώντας τους χρόνους ότι έχανα έναν κύκλο μηχανής και έτσι η οθόνη δεν έδειχνε το επιθυμητό μήνυμα όπως ανέφερα παραπάνω. Αυτό γινόταν επειδή στον counter της FSM δεν υπολόγιζα ότι αρχίζει από το 0 και τον τερμάτιζα έναν κύκλο πιο μετά απ' ότι πρέπει. Κατά τ' αλλα δεν αντιμετώπισα κάποιο ιδιαίτερο πρόβλημα.
- Παρακάτω φαίνεται ένα στιγμιότυπο από την προσομοίωση του συγκεκριμένου μέρους στο οποίο μπορούμε να παρατηρήσουμε όσα ανέφερα παραπάνω:



VGA - Top module

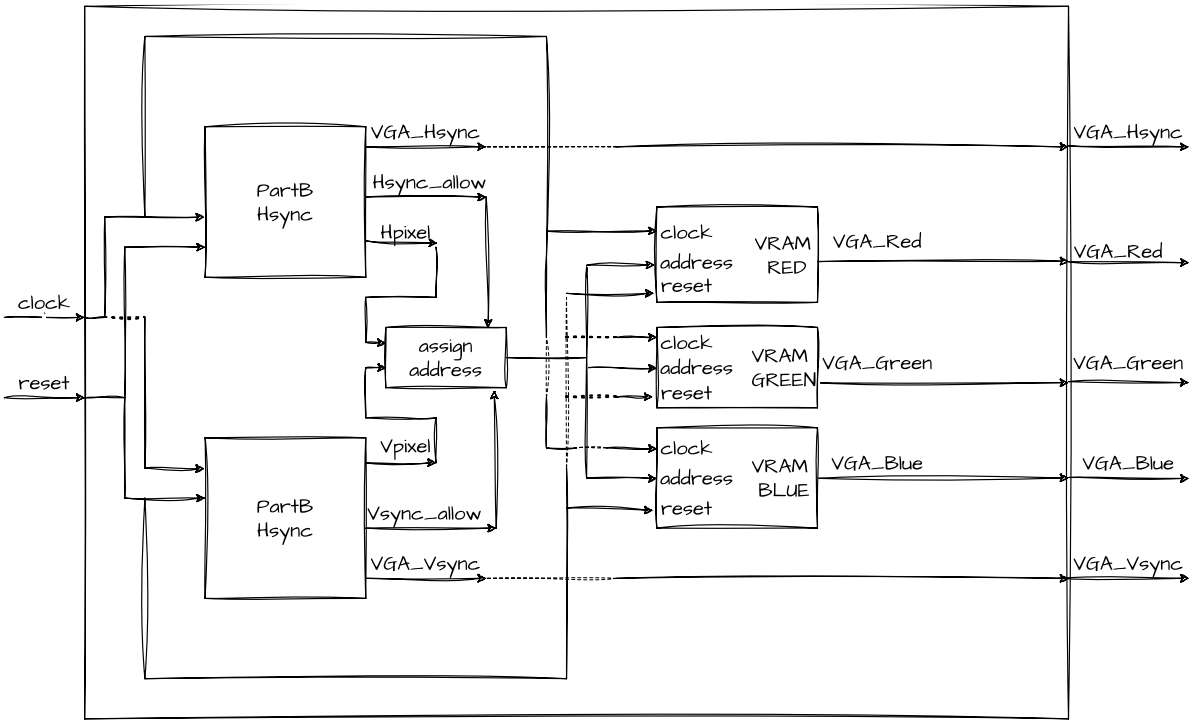
Υλοποίηση

- Σε αυτό το μέρος (το οποίο δεν είναι μέρος την εκφώνησης) ουσιαστικά ενώνω τα 3 προηγούμενα μέρη, διότι για να μπορέσω να προβάλλω οποιοδήποτε μήνυμα στην οθόνη

μέσω VGA χρειάζεται να έχω και τα 3 προηγούμενα μέρη ήδη υλοποιημένα. Έτσι σε αυτό το module περιλαμβάνονται:

- 3 μνήμες για κάθε χρώμα (PartA)
- Module για την διαχείριση του Hsync (PartB)
- Module για την διαχείριση του Vsync (PartC)

Ακολουθεί το κατάλληλο dataflow:

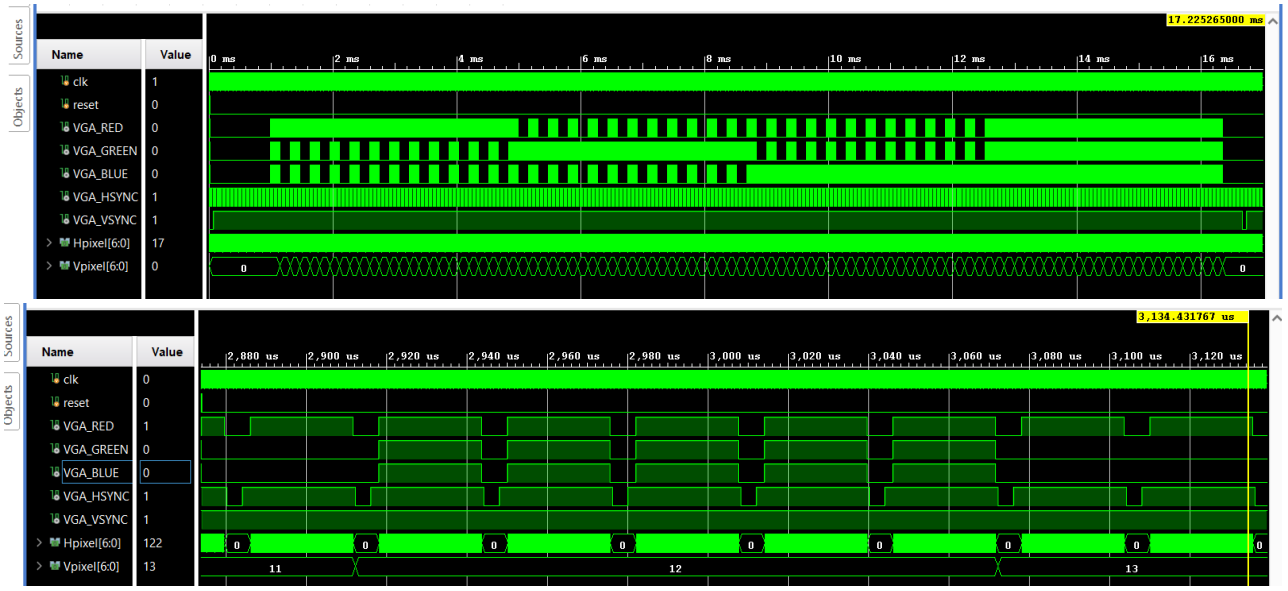


- Στο αρχείο αυτό υπάρχει και μια assign όπου διαχειρίζεται την διεύθυνση όπου εισάγεται μέσα στην μνήμη για να γίνει εξαγωγή του κατάλληλου bit. Αυτή η assign θα έχει την τιμή {Vpixel, Hpixel} μόνο όταν οι δύο FSM βρίσκονται ταυτόχρονα στα state 2 δηλαδή στο display time και active video time αντίστοιχα. Στην περίπτωση που δεν ισχύει κάποιος από τους 2 περιορισμούς τότε η διεύθυνση παίρνει την τιμή 13000 (>12.288) δηλαδή μία τιμή όπου όλες οι μνήμες έχουν τιμή 0.

Επαλήθευση

- Για το testbench πάλι απλά καλώ την μονάδα τροφοδοτώντας την με ένα ρολόι που να προσομοιώνει αυτό της FPGA και παρατηρώ τις κυματομορφές των σημάτων που αναφέρω παρακάτω
- Εμφανίζω όλα τα inputs και outputs του module για να δω πως λειτουργούν σαν σύνολο και όχι μεμονωμένα όπως έκανα στα προηγούμενα μέρη. Πιο συγκεκριμένα θέλω να δω μέσα σε μία τιμή του Vpixel (πχ 12) πόσα Hpixel έχω. Επίσης Εμφανίζω και τις εξόδους από τα χρώματα για να δω πως συμπεριφέρονται με την assign. Για να είναι σωστό θα πρέπει η τιμή του κάθε χρώματος να είναι το εκάστοτε bit που δείχνει μέσα στην μνήμη όσο το Hpixel αυξάνεται, αλλιώς να είναι 0.
- Από την στιγμή που έφτιαξα το πρόβλημα που είχα με τους counters στα μέρη Β και Γ, δεν υπήρχε κάποιο πρόβλημα στην υλοποίηση αυτού του top module.
- Παρακάτω είναι 2 στιγμιότυπα από την προσομοίωση αυτού του module. Το 1 είναι για να αποκτήσουμε μία πιο γενική άποψη και να καταλάβουμε αν κάτι πιο γενικό πάει στραβά, και το δέυτερο είναι μια μεγέθυνση στην γραμμή 12 όπου πρέπει να είναι άσπρη. Και από

τα δύο παρατηρούμε ότι όλα φαίνονται να λειτουργούν και είναι βάση όσα ανέφερα στο προηγούμενο bullet.



Πείραμα / Τελική υλοποίηση

- Εφόσον Για να μπορέσω να προβάλω κάποιο μήνυμα στην οθόνη έπρεπε να είχα όλα τα parts έτοιμα, αυτό το module είναι το μόνο που πέρασα μέσα στην πλακέτα.
- Η πορεία προγραμματισμού πήγε αρκετά ομαλά καθώς δεν χρειάστηκαν πάνω από 5 επαναλήψεις. Γενικά οι δυσκολίες ήταν να πετύχεις ακριβώς τους χρόνους στην FSM όπου είχα λάθος όπως και ανέφερα προηγουμένως.
- Για τον λόγο που ανέφερα παραπάνω, την πρώτη φορά που προσπάθησα να προβάλω το μήνυμα στην οθόνη, η οθόνη μπήκε σε power save mode, κάτι που με παρότρυνε να ελέγξω εξονοχυστικά τους χρόνους και στις 2 FSM. Έπειτα είχα ξεχάσει να περάσω στο constraints file τα υπόλοιπα πινάκια για να φαίνεται πιο έντονα το κάθε χρώμα και για αυτό τον λόγο μου φαινόντουσαν τα χρώματα πολύ αμυδρά. Έπειτα παρατήρησα ότι είχα κάνει μία ελαφρώς λάθος αρχικοποίηση της VRAM για το χρώμα μπλε και μου φαινόταν σε μία κάθετη στήλη κάποιο βιολετί χρώμα αλλά λύθηκε πολύ γρήγορα.
- Γενικά καθώς έτρεχα τον κώδικα στην πλακέτα το μόνο που άλλαξα ουσιαστικά είναι αυτό το μικρό μέρος στην μνήμη του μπλε χρώματος και οι counters στις FSM.

Συμπεράσματα

Γενικά μετά από 2 εργασίες Verilog και Vivado, αυτή η εργασία βγήκε αρκετά πιο ανώδυνα από τις υπόλοιπες. Οι προκλήσεις που αντιμετώπισα ήταν αρκετά περιορισμένες και για όσες υπήρχαν τις ξεπέρασα αρκετά εύκολα και γρήγορα. Ο κώδικας γράφτηκε και επαληθεύτηκε μέσα σε λίγες μέρες και η δοκιμή σε πλακέτα χρειάστηκε να γίνει μόνο σε ένα εργαστήριο από τα 3 που είχαμε διαθέσιμα, και μία ώρα πλακέτας.