

HY220: Εργαστήριο Ψηφιακών Κυκλωμάτων
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης
Χειμερινό Εξάμηνο 2020

3^η Σειρά Ασκήσεων

Ημερομηνία Παράδοσης: 31/01/2021 23:59

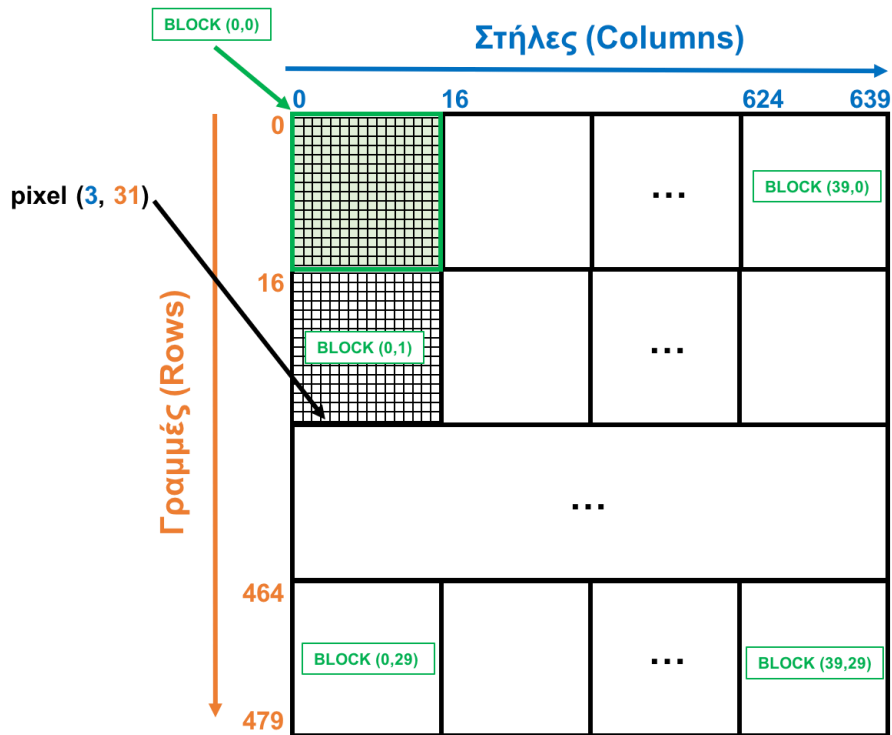
Άσκηση 3.1

Στην άσκηση αυτή θα υλοποιήσετε την δεύτερη φάση για το παιχνίδι λαβυρίνθος που περιλαμβάνει την εμφάνιση στην οθόνη VGA ενός λαβυρίνθου, ενός «παίκτη» στην αρχή του λαβυρίνθου και της εξόδου του λαβυρίνθου. Η άσκηση αυτή βασίζεται στο υλικό της Άσκησης 2.1 από την 2^η Σειρά Ασκήσεων και σας δίνεται επιπλέον κώδικας για ROMs που θα χρησιμοποιηθούν για τη σχεδίαση του λαβυρίνθου κτλ. Επίσης σας δίνεται ένα κατάλληλο testbench και reference outputs και θα μπορείτε να χρησιμοποιήσετε τον προσομοιωτή VGA (VGA Simulator) για να βλέπετε τι θα εμφανίζονταν σε μια πραγματική οθόνη.

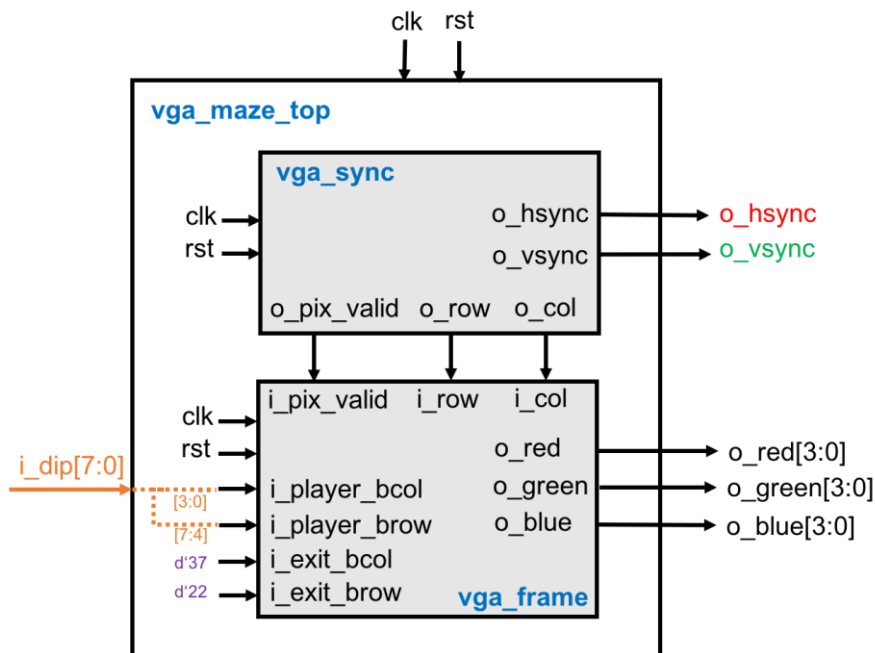
Δημιουργία Pixel Blocks 16x16 για VGA 640 x 480

Για την ευκολότερη διαχείριση των «πολλών» pixels ενός frame, συχνά ομαδοποιούμε τα pixels σε μπλοκ (block) και τα επεξεργαζόμαστε μαζί. Μια συχνή ομαδοποίηση είναι σε μπλοκ των 16x16 pixels (16 στήλες επί 16 γραμμές) την οποία θα χρησιμοποιήσουμε και εμείς για την άσκηση. Στην παρακάτω εικόνα φαίνεται η ομαδοποίηση ενός frame 640x480 σε blocks των 16x16. Με αυτή την ομαδοποίηση έχουμε δημιουργήσει ένα πλέγμα (grid) 40x30 (40 στήλες επί 30 γραμμές) και έτσι έχουμε να διαχειριστούμε 1200 blocks των 16x16 pixels αντί για 307200 μεμονωμένα pixels. Έχουμε λοιπόν 30 γραμμές από μπλοκ (block rows) όπου η κάθε γραμμή έχει 40 στήλες από μπλοκ (block columns).

Με αυτή την ομαδοποίηση σε block των 16x16 pixels έχοντας τη συντεταγμένη ενός pixel (στήλη, γραμμή) μπορούμε να βρούμε τον αριθμό του block στο οποίο ανήκει παίρνοντας το πηλίκο της ακέραιας διαίρεσης με το 16 και αγνοώντας το υπόλοιπο. Για παράδειγμα το pixel (3, 31) ανήκει στο block (3/16, 31/16) δηλαδή το block (0, 1) ενώ το pixel (536, 372) ανήκει στο block (33, 23). Σε hardware η διαίρεση με δυνάμεις του 2 όπως το 16 στην περίπτωση μας είναι μια απλή ολίσθηση κατά 4 θέσεις δεξιά ή απλό bit selection από MSBs, οπότε είναι πολύ «φθηνή» πράξη.



Για την Άσκηση 3.1 το σχέδιο είναι ελαφρώς τροποποιημένο σε σχέση με το σχέδιο της Άσκησης 2.1 και φαίνεται στην παρακάτω εικόνα:



Το σχέδιο έχει την εξής ιεραρχία:

- **vga_maze_top** (*vga_maze_top.sv*): σας δίνεται και σε σχέση με την Άσκηση 2.1 έχει απλά προστεθεί η πόρτα *i_dip* που είναι 8-bits και κάποιες συνδέσεις όπως φαίνονται στο σχήμα.
- **vga_sync** (*vga_sync.sv*): Το μπλοκ αυτό υλοποιεί το χρονισμό του πρωτοκόλλου VGA για ανάλυση 640 x 480 και πρέπει να είναι η υλοποίηση σας από την Άσκηση 2.1.

- **vga_frame** (*vga_frame.sv*): Το μπλοκ αυτό είναι υπεύθυνο για τη δημιουργία των χρωμάτων RGB (red/green/blue) για κάθε pixel που πρέπει να εμφανίζεται στην οθόνη για να εμφανιστεί ο λαβύρινθος και ο παίκτης στην οθόνη και είναι αυτό που θα πρέπει να υλοποιήσετε για την Άσκηση 3.1 με βάση τις προδιαγραφές που σας δίνονται παρακάτω.
- **rom** (*rom.sv*): Το μπλοκ σας δίνεται έτοιμο και δημιουργεί μια σύγχρονη ROM παραμετρικού μεγέθους και την αρχικοποιεί με τα περιεχόμενα ενός αρχείου που δίνεται επίσης ως παράμετρος. Οι πόρτες της ROM είναι: clk, en, addr, dout. Η πόρτα en (enable) σηματοδοτεί μια νέα ανάγνωση, η πόρτα addr (address) είναι για να δοθεί η είσοδος για τη διεύθυνση που θέλουμε να διαβάσουμε και η πόρτα εξόδου dout (data out) βγάζει τα περιεχόμενα της διεύθυνσης addr στον «επόμενο κύκλο». Η παράμετρος size ορίζει τον αριθμό των θέσεων της ROM, η παράμετρος width το πλάτος σε bits της κάθε θέσης της ROM (16-bits στην περίπτωσή μας) και η παράμετρος file δηλώνει το αρχείο που θα χρησιμοποιηθεί για αρχικοποίηση των περιεχομένων της ROM. Επίσης στο φάκελο roms σας δίνονται 3 αρχεία (maze1.rom, player.rom, exit.rom) που θα χρησιμοποιηθούν για αρχικοποίηση των δεδομένων των διαφόρων ROM που θα χρησιμοποιήσουμε για την άσκηση (περισσότερες λεπτομέρειες δίνονται παρακάτω).
- **vga_tb** (*vga_tb.sv*): ένα testbench για προσομοίωση που δημιουργεί το ρολόι (25 MHz – 40 ns), το reset, και τα σήματα από τα DIP switches. Το testbench αποθηκεύει την έξοδο του κυκλώματός σας σε ένα αρχείο (vga_log.txt) με το κατάλληλο format έτσι ώστε να μπορείτε να χρησιμοποιήσετε τον VGA Simulator για να βλέπετε τι θα εμφανίζεται στην οθόνη από τον κώδικά σας. Περισσότερες λεπτομέρειες για τον VGA Simulator παρακάτω.
- **Reference output**: Στο φάκελο reference υπάρχει ένα πρότυπο *vga_log.txt* output που είναι αυτό που θα πρέπει να δημιουργεί ένας σωστός κώδικας. Μπορείτε να κάνετε diff αυτό που παράγει ο δικός σας κώδικας με το reference output για να εντοπίσετε λάθη κατά την προσομοίωση. Το format είναι συμβατό με τον VGA Simulator και είναι πολύ απλό. Σε κάθε γραμμή περιέχει το χρόνο σε ns ακολουθούμενο από τις τιμές των σημάτων hsync (1-bit), vsync (1-bit), red (4-bits), green (4-bits), blue (4-bits).
- **VGA Simulator**: Μέσα στο φάκελο vga-simulator υπάρχει μια ιστοσελίδα που μπορείτε να ανοίξετε τοπικά στον web browser σας. Εκεί μπορείτε να επιλέξετε το log file που έχει δημιουργηθεί από την προσομοίωσή σας και όταν πατήσετε το κουμπί submit τότε θα εμφανιστεί σε μια εικονική VGA οθόνη η έξοδός σας. Μπορείτε να το δοκιμάσετε επίσης με το reference output. Μην αλλάξετε τις παραμέτρους που υπάρχουν ήδη στη σελίδα!
Credits: Ο VGA Simulator έχει δημιουργηθεί από τον Eric Eastwood στο παρακάτω website <http://ericeastwood.com/lab/vga-simulator/>

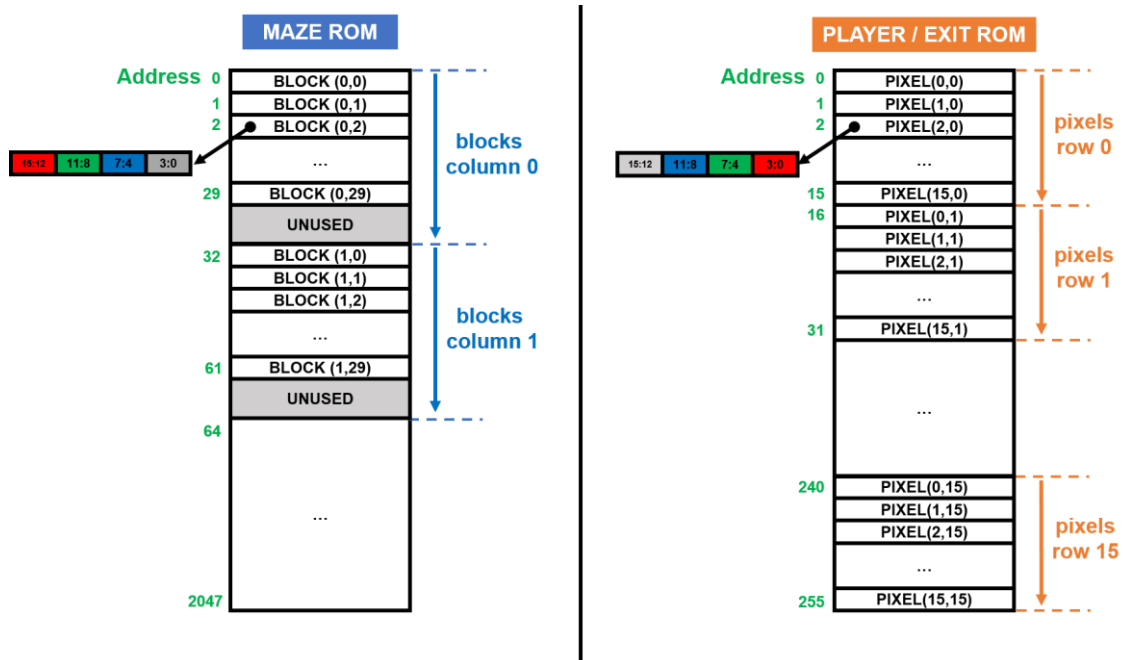
Τι πρέπει υλοποιήσετε για την Άσκηση 3.1:

Θα πρέπει να υλοποιήσετε το module **vga_frame** του οποίου ένας άδειος σκελετός σας δίνεται. Το module αυτό με βάση τις εισόδους του, *i_row*, *i_col*, και *i_pixel_valid* που προέρχονται από το module **vga_sync** (το υλοποιήσατε στην Άσκηση 2.1) και τις εισόδους *i_player_bcol*, *i_player_brow*, *i_exit_bcol*, και *i_exit_brow* (που προσθέτουμε τώρα), πρέπει να δημιουργήσει τις τιμές των χρωμάτων RGB (red/green/blue) για να εμφανιστεί/ζωγραφιστεί στην οθόνη ο λαβύρινθος, ο παίκτης και η έξοδος του λαβυρίνθου.

Η είσοδος *i_player_bcol* δίνει την οριζόντια θέση (column) του block που βρίσκεται ο παίκτης, με βάση την ομαδοποίηση σε πλέγμα (grid) και η είσοδος *i_player_brow* δίνει την κατακόρυφη θέση (row) του block του παίκτη. Η οριζόντια θέση (block column) του παίκτη στο grid 40x30 δίνεται από τα DIP switches [3:0] και η κατακόρυφη θέση (block row) από τα DIP switches [7:4] και μπορείτε να την αλλάξετε αν αλλάξετε τις τιμές τους. Αντίστοιχα οι εισοδοί

`i_exit_bcol` και `i_exit_brow` δίνουν την θέση της εξόδου του λαβυρίνθου στο πλέγμα. Η έξοδος στο λαβύρινθο είναι στην θέση (37,22) του grid είναι σταθερή στο `vga_maze_top` module.

Για τις τιμές RGB των pixels που χρειάζονται για τον λαβύρινθο, τον παίκτη και την έξοδο θα χρησιμοποιήσετε και θα κάνετε instantiate 3 ROM με βάση το module `rom.v` που σας δίνεται.



Η ROM του λαβυρίνθου (`maze_rom`) έχει 2048 θέσεις και πλάτος 16-bits και δίνει σε κάθε θέση μια τιμή RGB για κάθε block του λαβυρίνθου στο πλέγμα (grid 40x30). Όλα τα pixels (16x16) αυτού του block πρέπει να παίρνουν την ίδια τιμή και πρέπει να τη διαβάζετε από τη ROM δίνοντας την κατάλληλη διεύθυνση. Το περιεχόμενο της `maze_rom` σε κάθε διεύθυνση παρουσιάζεται στην παραπάνω εικόνα και το αρχείο αρχικοποίησης σας δίνεται (`roms/maze1.rom`). Επειδή το grid 40x30 (1200 blocks) έχει διαστάσεις που δεν είναι δυνάμεις του 2, για να διευκολύνουμε την διευθυνσιοδότηση χρησιμοποιούμε μια μεγαλύτερη ROM με 2048 θέσεις που μπορεί να υλοποιήσει ένα grid 64x32 και έτσι κάποιες θέσεις μένουν αχρησιμοποίητες όπως φαίνεται στην εικόνα. Αυτό που χρειάζεται να κάνετε για τη `maze_rom` είναι να βρείτε πως θα «γεννήσετε» τη σωστή διεύθυνση για να πάρετε τις σωστές τιμές RGB. Κάθε θέση της Maze ROM είναι 16-bits και το περιεχόμενο παρέχει τις τιμές RGB ως εξής: (i) τα bits 7 έως 4 δίνουν την τιμή για το 4-bit μπλε χρώμα, (ii) τα bits 11 έως 8 δίνουν την τιμή για το 4-bit πράσινο χρώμα, (iii) τα bits 15 έως 12 δίνουν την τιμή για το 4-bit κόκκινο χρώμα, (iv) τα υπόλοιπα bits δεν χρησιμοποιούνται.

Ο παίκτης πρέπει να εμφανίζεται μόνο στο σημείο (block) του grid που δίνεται από τις εισόδους `i_player_bcol` και `i_player_brow`. Ο παίκτης θα εμφανίζεται σε ένα μόνο block από 16x16 pixels και για κάθε pixel αυτού του block οι τιμές RGB για εμφάνιση πρέπει να διαβάζονται από μια άλλη ROM (την `player_rom`). Η ROM του παίκτη (`player_rom`) έχει 256 θέσεις, πλάτος 16-bits και το περιεχόμενό της σε κάθε διεύθυνση παρουσιάζεται στην εικόνα, το αρχείο αρχικοποίησης σας δίνεται (`roms/player.rom`). Η `player_rom` χρησιμοποιείται πλήρως, δηλαδή σε κάθε θέση έχει διαφορετικές RGB τιμές για καθένα από τα 16x16 pixels έτσι ώστε να ζωγραφιστεί ο παίκτης (ο Μάριος ☺) με καλύτερη λεπτομέρεια. Για την `player_rom` πρέπει να βρείτε πως θα «γεννήσετε» τη σωστή διεύθυνση για να πάρετε τις κατάλληλες τιμές RGB.

Αντίστοιχα με τον παίκτη, η έξοδος πρέπει να εμφανίζεται μόνο στο σημείο (block) του grid που δίνεται από τις εισόδους `i_exit_bcol` και `i_exit_brow`. Η έξοδος θα εμφανίζεται σε ένα μόνο block από 16x16 pixels και για κάθε pixel αυτού του block οι τιμές RGB για εμφάνιση πρέπει να διαβάζονται από μια άλλη ROM (την `exit_rom`). Η ROM της εξόδου (`exit_rom`) έχει 256 θέσεις, πλάτος 16-bits και το περιεχόμενό της σε κάθε διεύθυνση παρουσιάζεται στην εικόνα, το αρχείο αρχικοποίησης σας δίνεται (`roms/exit.rom`). Η `exit_rom` χρησιμοποιείται πλήρως, δηλαδή σε κάθε θέση έχει διαφορετικές RGB τιμές για καθένα από τα 16x16 pixels έτσι ώστε να ζωγραφιστεί η έξοδος (το Αστéρι ☺) με καλύτερη λεπτομέρεια. Για την `exit_rom` πρέπει να βρείτε πως θα «γεννήσετε» τη σωστή διεύθυνση για να πάρετε τις κατάλληλες τιμές RGB.

Στην υλοποίηση που περιγράφεται, ο λαβύρινθος εμφανίζεται στα 40x30 blocks του grid σαν background και «πάνω» από αυτόν σε συγκεκριμένα blocks, που δίνονται από τις εισόδους, εμφανίζονται ο παίκτης και η έξοδος του λαβυρίνθου (τέτοια αντικείμενα είναι γνωστά και ως sprites). Όταν ο παίκτης και η έξοδος είναι στο ίδιο block πρέπει να εμφανίζεται μόνο ο παίκτης. Άρα τελικά η προτεραιότητα εμφάνισης είναι (1)παίκτης, (2)έξοδος, (3)λαβύρινθος. Το module `vga_frame` λοιπόν με βάση τις εισόδους πρέπει να διαβάζει τις κατάλληλες διευθύνσεις από τις 3 ROM και να επιλέγει για εμφάνιση την κατάλληλη RGB τιμή ανάλογα με την προτεραιότητα που έχει το κάθε αντικείμενο και έτσι να δημιουργεί τις εξόδους του `o_red`, `o_green`, `o_blue`. Θυμηθείτε ότι το module αυτό πρέπει να παράγει έξοδο στον «επόμενο κύκλο». Επίσης, όταν οι εισοδοί `i_col` και `i_row` δεν είναι έγκυρες (δηλαδή η είσοδος `i_pix_valid` είναι 0) τότε η έξοδος πρέπει είναι το μαύρο χρώμα RGB(0,0,0).

Θα πρέπει να υλοποιήσετε σε SystemVerilog RTL το module `vga_frame`. Θα πρέπει να προσομοιώσετε και να επαληθεύσετε με το έτοιμο testbench και τα reference outputs τον κώδικά σας. Θα πρέπει να μπορείτε να δείτε την έξοδο που θα έβγαζε το κύκλωμα με τον προσομοιωτή VGA Simulator. Μη ξεχάσετε να βάλετε reset στους καταχωρητές!

Τι πρέπει να παραδώσετε:

Μέχρι την ημερομηνία της προθεσμίας πρέπει να παραδώστε σε ένα zip/tar.gz αρχείο (e.g. hw3.zip ή hw3.tar.gz) τα παρακάτω:

1. Τον SystemVerilog RTL κώδικα του μπλοκ `vga_frame`

Στείλτε το **hw3.zip** σας με e-mail στο hy220@csd.uoc.gr με τίτλο: “HW3 – Ονοματεπώνυμο – AM”.

Οι κώδικες θα ελέγχονται για αντιγραφές με ειδικό λογισμικό!