

HY220: Εργαστήριο Ψηφιακών Κυκλωμάτων

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης
Εαρινό Εξάμηνο 2026

Εργαστήριο 2 (2 εβδομάδες)

- **Εβδομάδα 1** (αναλόγως το εργαστηριακό τμήμα που έχετε δηλώσει)
(Γκρουπ Α) Τρίτη 05/05 13:00 – 15:00 στην αίθουσα B.110
(Γκρουπ Β) Παρασκευή 08/05 11:00 – 13:00 στην αίθουσα B.110
- **Εβδομάδα 2** (αναλόγως το εργαστηριακό τμήμα που έχετε δηλώσει)
(Γκρουπ Α) Τρίτη 12/05 13:00 – 15:00 στην αίθουσα B.110
(Γκρουπ Β) Παρασκευή 15/05 11:00 – 13:00 στην αίθουσα B.110

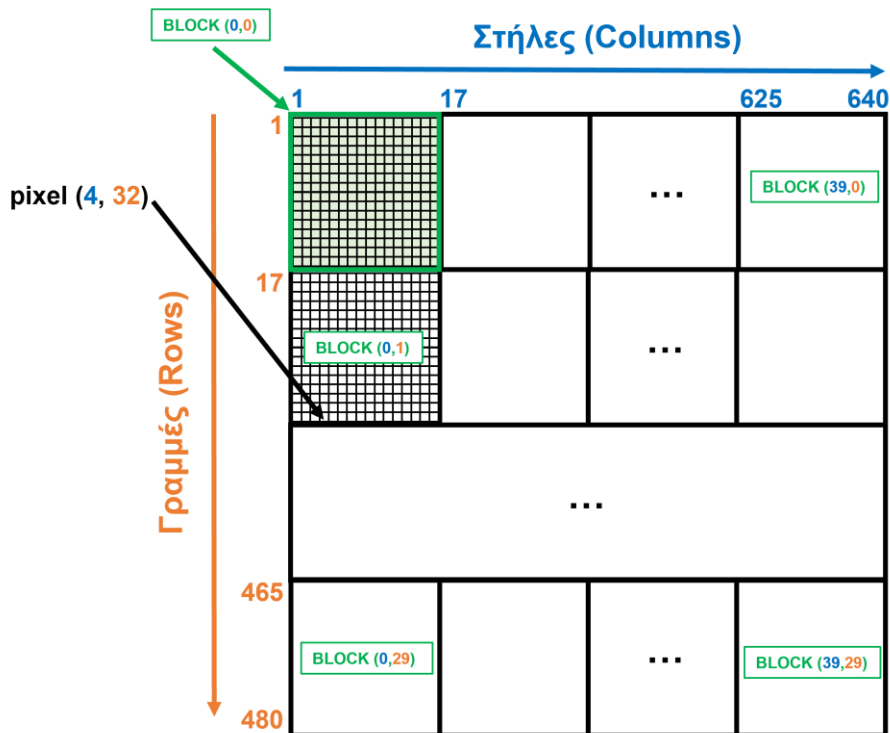
Κατα τη διάρκεια των εργαστηρίων θα υλοποιήσετε σε τρεις φάσεις το παιχνίδι «Λαβύρινθος» (Maze). Περιληπτικά, το τελικό παιχνίδι θα εμφανίζει σε VGA οθόνη ένα λαβύρινθο και τη φιγούρα ενός παίκτη. Ο χρήστης θα μπορεί να μετακινήσει τον παίκτη με τα κουμπιά που υπάρχουν πάνω στην πλακέτα και ο σκοπός είναι να τον οδηγήσει στην έξοδο το ταχύτερο δυνατόν.

Στο *Εργαστήριο 2* θα υλοποιήσετε την δεύτερη φάση (σε **2 εβδομάδες εργαστηρίων**) που περιλαμβάνει την εμφάνιση στην οθόνη VGA ενός λαβυρίνθου, ενός «παίκτη» στην αρχή του λαβυρίνθου και την έξοδο του λαβυρίνθου. Το εργαστήριο αυτό βασίζεται στο υλικό του Εργαστηρίου 1 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 από τον κώδικά σας.

Δημιουργία Pixel Blocks 16x16 (παράδειγμα 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 το εργαστήριο. 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 χειριστούμε 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 του «αριθμού μείον 1» με το 16 και αγνοώντας το υπόλοιπο. Για παράδειγμα το pixel (4, 32) ανήκει στο block ((4-1)/16, (32-1)/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 περίπτωσή μας είναι μια απλή ολίσθηση κατά 4 θέσεις δεξιά ή απλό bit selection από MSBs, οπότε είναι πολύ «φθηνή» πράξη.



Πρόσβαση στον κώδικα του εργαστηρίου:

Για τον κώδικα και τις παραδόσεις έχουν δημιουργηθεί git repositories στο gitlab του Τμήματος. Στους εγγεγραμμένους φοιτητές στο εργαστήριο αντιστοιχεί ένα private git repository της μορφής:

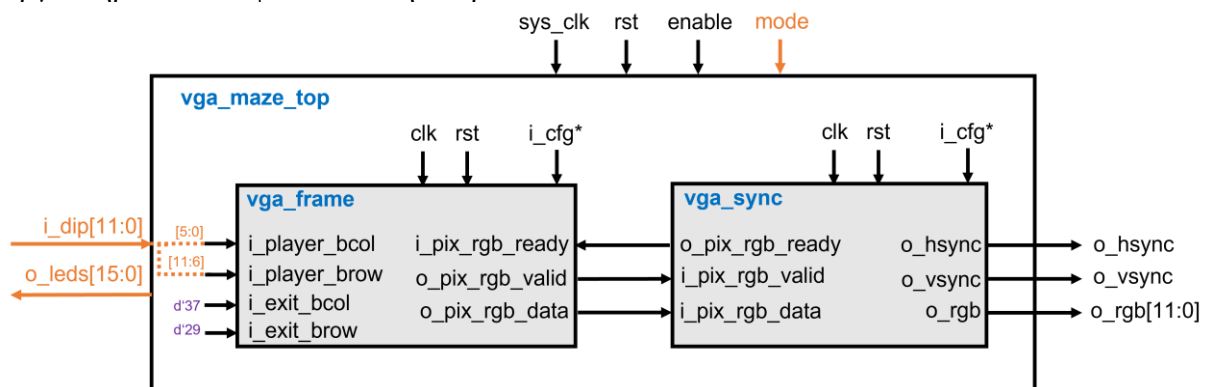
https://gitlab-csd.datacenter.uoc.gr/hy220_2026s/lab2_submits/lab2-csdXYZW

Κάντε `git clone` το προσωπικό σας repository με το username σας (csdXYZW@csd.uoc.gr και το password σας). Η πρόσβαση είναι εφικτή από το δίκτυο του Πανεπιστημίου και το VPN.

MHN KANETE FORK – Εργαστείτε απευθείας στο repository που σας δίνεται

Το σχέδιο του Εργαστηρίου 2:

Για το *Εργαστήριο 2* το σχέδιο είναι ελαφρώς τροποποιημένο σε σχέση με το σχέδιο του Εργαστηρίου 1 και φαίνεται στην παρακάτω εικόνα:



Το σχέδιο έχει την εξής ιεραρχία:

- **vga_maze_top** (*src/vga_maze_top.sv*): σας δίνεται και σε σχέση με το *Εργαστήριο 1* έχουν προστεθεί πόρτες `i_dip` που είναι 12-bits (οι 12 τελευταίοι διακόπτες από δεξιά), ο `leds` που αντικατοπτρίζουν σε `led` τις τιμές των `dip`, το σήμα `i_enable` (δεύτερος διακόπτης από αριστερά), ένα σήμα `i_mode` (τρίτος διακόπτης από αριστερά) για αλλαγή ανάλυσης στην οθόνη, και κάποιες συνδέσεις όπως φαίνονται ενδεικτικά στο σχήμα. Στο νέο αυτό top γίνεται χρήση ενός SystemVerilog package (*src/pkg/vga_pkg.sv*) που περιέχει ένα τύπο `struct packed vga_cfg_t` και παραμέτρους για 2 αναλύσεις VGA (640x480 και 1024x768) που επιλέγονται στο σχέδιο με βάση το διακόπτη `i_mode` και τροφοδοτούνται στα modules. Επιπλέον δημιουργούνται 2 ρολόγια για τις διαφορετικές αναλύσεις και ένας ειδικός πολυπλέκτης επιλογής. Το σήμα `rst` προέρχεται πλέον από διακόπτη (πρώτος από αριστερά).
- **vga_sync** (*src/vga_sync.sv*): Το μπλοκ αυτό υλοποιεί το χρονισμό του πρωτοκόλλου VGA και πρέπει να είναι η υλοποίηση σας από το *Εργαστήριο 1* με προσαρμογή για τη χρήση του `vga_cfg_t` struct και άλλες προσαρμογές όπου χρειαστούν.
- **vga_frame** (*src/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 το *Εργαστήριο 2* με βάση τις προδιαγραφές που σας δίνονται παρακάτω.
- **rom** (*src/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 περίπτωση μας) και η παράμετρος `file` δηλώνει το αρχείο που θα χρησιμοποιηθεί για αρχικοποίηση των περιεχομένων της ROM. Επίσης στο φάκελο *src/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 το εργαστήριο (περισσότερες λεπτομέρειες δίνονται παρακάτω).
- **vga_tb** (*src/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 `ref` 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 `ref/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/>

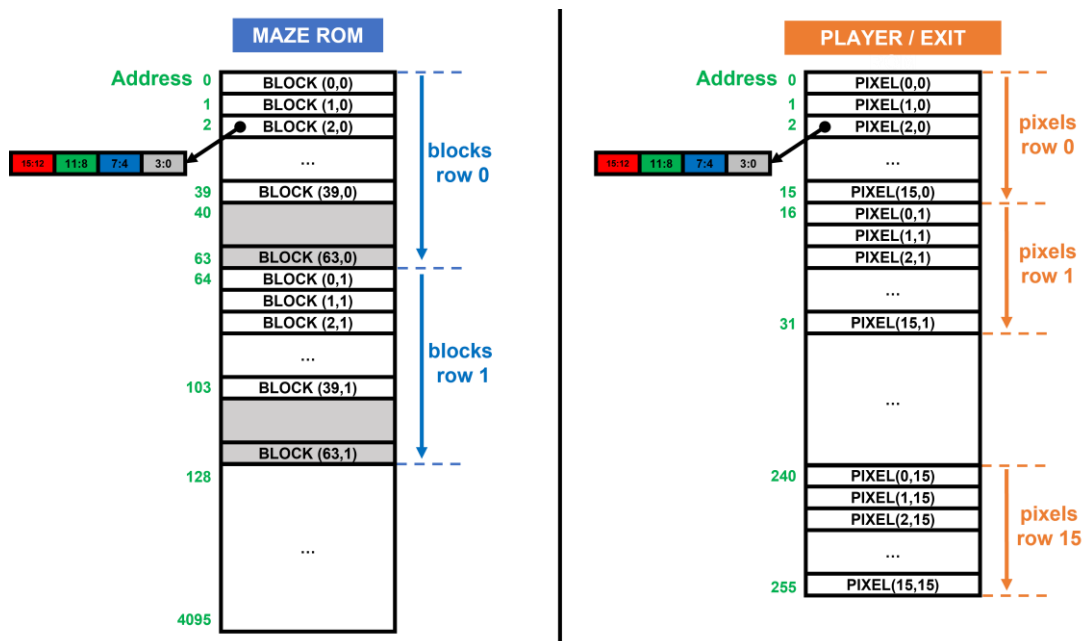
- Δίνεται επίσης ο φάκελος **run** με έτοιμο Makefile για να τρέξετε την προσομοίωση και να δείτε αν βγάζετε σωστά αποτελέσματα (make και μετά make check) με τους υποστηριζόμενους προσομοιωτές όπως αναφέρονται στη σελίδα του μαθήματος.

Τι πρέπει υλοποιήσετε και να προσομοιώσετε πριν πάτε στα εργαστήρια:

Για το εργαστήριο αυτό θα πρέπει να υλοποιήσετε το module **vga_frame** του οποίου ένας άδειος σκελετός σας δίνεται. Το module αυτό με βάση 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. Το module αυτό επικοινωνεί με τον **vga_sync** (το υλοποιήσατε στο Εργαστήριο 1) με βάση το πρωτόκολλο «χειραψίας/handshake» `pix_rgb_valid & pix_rgb_ready` που εξηγήθηκε στο Εργαστήριο 1.

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 δίνεται από τα DIP switches [5:0] και η κατακόρυφη θέση (block row) από τα DIP switches [11:6] και μπορείτε να την αλλάζετε αν μετακινείτε τους διακόπτες στην πλακέτα. 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 (37,29) του grid για την ανάλυση 640x480 ενώ είναι σε διαφορετική θέση για την ανάλυση 1024x768 που επιλέγεται μέσω του σήματος `i_mode` 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 4096 θέσεις και πλάτος 16-bits και δίνει σε κάθε θέση μια τιμή RGB για κάθε block του λαβυρίνθου στο πλέγμα. 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 (*src/roms/maze1.rom*). Επειδή το grid έχει διαστάσεις που δεν είναι δυνάμεις του 2, για να διευκολύνουμε την διευθυνσιοδότηση χρησιμοποιούμε μια μεγαλύτερη ROM με 4096 θέσεις που μπορεί να υλοποιήσει ένα grid 64x64 και έτσι κάποιες θέσεις μπορεί να μένουν αχρησιμοποίητες σε κάποιες αναλύσεις. 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 (*src/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 (*src/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 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 την έξοδο *o_pix_rgb_data*. Θυμηθείτε ότι το module αυτό, επειδή διαβάζει δεδομένα από τη ROM, παράγει τα δεδομένα στον «επόμενο κύκλο» και θα πρέπει να είναι κάπως «συγχρονισμένο» με το module *vga_sync*.

Θα πρέπει να υλοποιήσετε σε SystemVerilog RTL το module **vga_frame** και να κάνετε προσαρμογές στο module **vga_sync**. Πριν πάτε στο εργαστήριο 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 κάνετε στο εργαστήριο:

Θα πρέπει να πάτε στο εργαστήριο με τον **κώδικά σας έτοιμο και προσομοιωμένο από πριν** και να ακολουθήσετε την ροή του εργαλείου Xilinx Vivado και τα βήματα που χρειάζεται για να «κατεβάσετε» το σχέδιο στην FPGA και να το δείτε να δουλεύει. Αλλάξτε τα DIP switches και δείτε ότι ο παίκτης αλλάζει θέση στην οθόνη. Δείξτε το κύκλωμα που δουλεύει στο βοηθό.

Σημείωση: Το ρολόι του κυκλώματος πρέπει να είναι 25MHz (περίοδος ρολογιού 40ns) για τη σωστή υλοποίηση του χρονισμού VGA με ανάλυση 640x480 (resolution) και ρυθμό ανανέωσης 60Hz (refresh rate), δηλαδή 60 VGA frames per second. Ενώ για την ανάλυση 1024x768 στα 60Hz η συχνότητα πρέπει να είναι 65MHz. Η πλακέτα του εργαστηρίου όμως έχει εξωτερικό ρολόι 100 MHz οπότε υπάρχει ένα επιπλέον block από τον «Clocking Wizard» του Vivado (*src/clk_wiz_x2.sv*) το οποίο είναι ήδη συνδεδεμένο στο top, ενεργοποιείται μόνο κατά την σύνθεση, και με έναν ειδικό πολυπλέκτη επιλέγει το κατάλληλο ρολόι με βάση το *i_mode*. Για την ανάθεση pins (pin assignment) χρησιμοποιείτε το constraint file που σας δίνεται (*constr/lab2.xdc*).

Αφού δουλέψει το κύκλωμά σας για την ανάλυση 640x480, τότε ενεργοποιήστε το reset, απενεργοποιήστε το enable, αλλάξτε το mode σε 1 για ανάλυση 1024x768, και μετά απενεργοποιήστε το reset και ενεργοποιήστε το enable. Το κύκλωμά σας τώρα πρέπει να εμφανίζει στην οθόνη ένα μεγαλύτερο λαβύρινθο στη νέα ανάλυση.

Παράδοση του κώδικα του εργαστηρίου:

Πρέπει να παραδώσετε στο τέλος του εργαστηρίου τα αρχεία στον **φάκελο src** όπως έχουν προκύψει/αλλάξει. Η παράδοση θα πρέπει να γίνει **μέχρι τη 2^η εβδομάδα των εργαστηρίων (αναλόγως το Γκρουπ που ανήκετε)** στο τέλος της ώρας που έχετε εργαστήριο **και πρέπει να έχετε εξεταστεί από τον βοηθό** – δεν αρκεί μόνο να παραδώσετε στο gitlab!

**Οι παραδόσεις θα γίνονται μόνο μέσω του gitlab.
ΜΗΝ προσθέσετε το Vivado project ή το φάκελο “run” στο git !!!**

Ενδεικτικές εντολές

- git status (για να δείτε τι έχει αλλάξει)
- git config user.email csdXYZW@csd.uoc.gr (ΜΗΝ χρησιμοποιήσετε το --global flag)
- git config user.name "Name Surname" (ΜΗΝ χρησιμοποιήσετε το --global flag)
- git add src (για να προσθέσετε τα αρχεία που αλλάξατε στο φάκελο src)
- git commit -m “lab2 delivery” (commit και μήνυμα)
- git push (θα πρέπει πάλι να χρησιμοποιήσετε το username και το password σας)
- **Επιβεβαιώστε ότι το push έγινε και ότι τα αρχεία έχουν ανέβει στο gitlab στο repository που σας δόθηκε και OXI ΣΕ FORK**

**Μετά την πάροδο της ημερομηνίας του Εργαστηρίου σας
ΔΕΝ θα μπορείτε να παραδώσετε και να κάνετε push στο gitlab !**