FPGsnAke

A SNAKE GAME FOR THE ATLYS FPGA

Λίγα λόγια για το Project

- Είναι το κλασικό ‘φιδάκι’ των παλαιών κινητών.
- Το φιδάκι πρέπει να φάει τα μήλα.
- Δεν πρέπει να δαγκώσει την ουρά του.
- Δεν πρέπει να χτυπήσει σε κάποιον τοίχο.

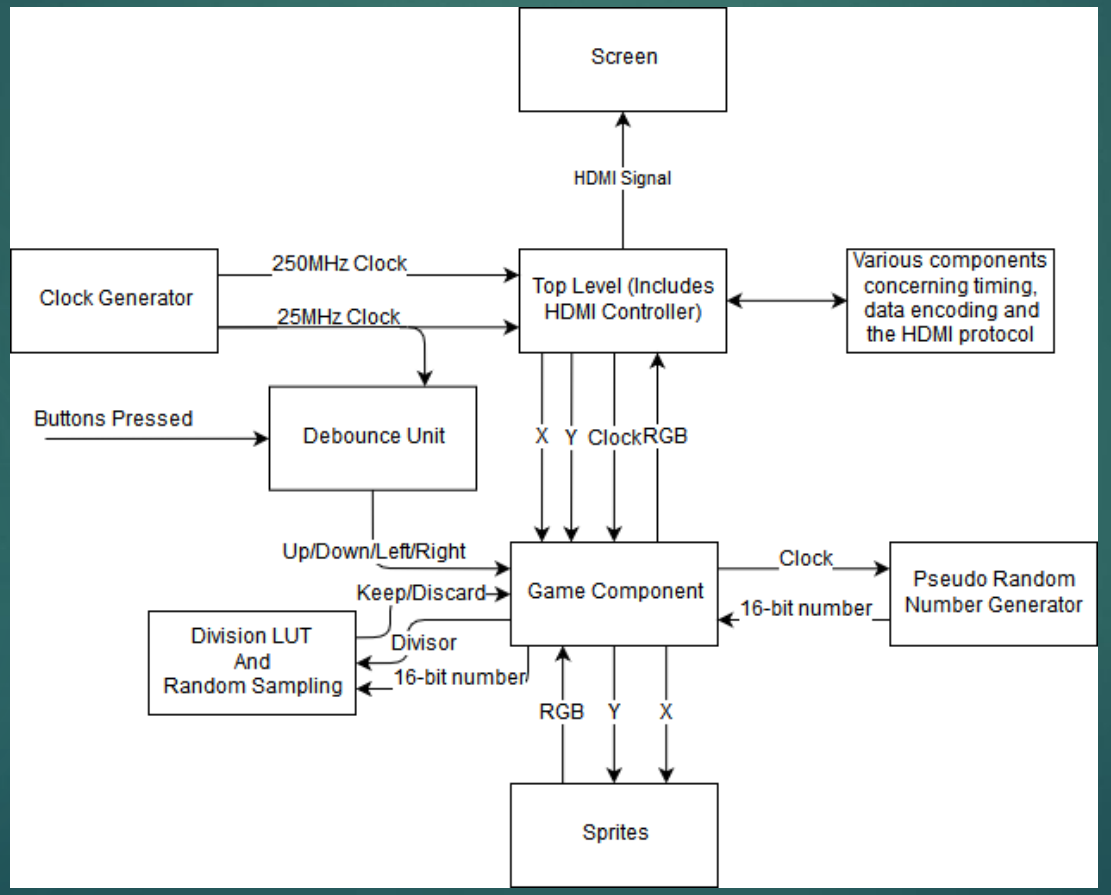
Εργαλεία που χρησιμοποιήθηκαν

- Η γλώσσα περιγραφής υλικού που χρησιμοποιήθηκε είναι η ‘Verilog’.
- Το περιβάλλον ανάπτυξης είναι το ‘ISE 13.4’ και το ‘Vim’.
- Χρησιμοποιήθηκε ‘Version Control’ (git).
- Ο κώδικας είναι διαθέσιμος στο ‘GitHub’ (και οι ρυθμίσεις του Vim).
- Επίσης χρησιμοποιήθηκε python για την διευκόλυνση διεργασιών.

Τι τεχνολογία χρησιμοποιήθηκε έτοιμη

- Ο κώδικας για την χρήση της θύρας HDMI καθώς και το πρωτόκολλο επικοινωνίας πάρθηκαν έτοιμα από τις βιβλιοθήκες που παρέχει η XILINX για την συγκεκριμένη FPGA (ATLYS).
- Ο αλγόριθμος για την μετατροπή binary σε binary coded decimals (για το score).
- Η μέθοδος παραγωγής ψεύδο-τυχαίων 16-bit αριθμών μέσω 'Linear Feedback Shift Register'.

Block Diagram



Ανάλυση των λειτουργιών

- Η γεννήτρια ρολογιού παράγει 2 ρολόγια, ένα στα 25MHz και ένα στα 250MHz
- Το ρολόι των 250MHz είναι για το πρωτόκολλο του HDMI, ενώ το άλλο είναι Pixel Clock.

Debounce Unit

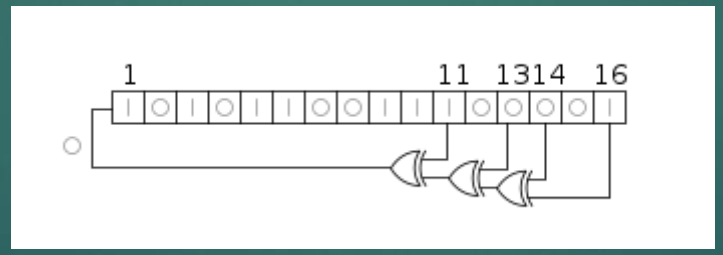
- Αυτό το module έχει την ευθύνη του να κάνει debounce την είσοδο που παίρνει το board μας από τα 5 push buttons.
- Ουσιαστικά θυμάται για κάθε κουμπί την προηγούμενη ευσταθή του κατάσταση και όποτε δημιουργείται αναταραχή σε σχέση με την προηγούμενη κατάσταση, αρχίζει και μετράει κύκλους.
- Αν η διαταραχή στην είσοδο παραμείνει για παραπάνω από 2^16 κύκλους τότε θεωρούμε ότι το σήμα είναι ευσταθές και το προωθούμε στην έξοδο.
- Διαφορετικά, σημαίνει ότι υπάρχουν διακυμάνσεις από το σήμα εισόδου του push button και έτσι δεν αλλάζει η τιμή εξόδου στο debounce unit.

Sprites

- Εδώ έχουμε φορτωμένες τις εικόνες που μπορούμε να προβάλουμε στην οθόνη (π.χ. το μήλο, το φίδι, τα τούβλα, οι αριθμοί).
- Κάθε Sprite αποτελείτε από ένα μεγάλο case statement όπου για κάθε πιθανή τιμή εισόδου (X, Y), ανατίθεται η κατάλληλη RGB έξοδος
- Προφανώς αυτό δεν μπορεί να γίνει με το χέρι, αφού για μία εικόνα 32x32 έχουμε 1024, 24-bit τιμές.
- Συνεπώς υλοποιήθηκε ένα πρόγραμμα σε python που παρήγαγε αυτόματα αρχεία σε Verilog με είσοδο μια εικόνα.

Pseudo Random Number Generator

- Για την παραγωγή τυχαίων αριθμών χρησιμοποιήθηκε η μέθοδος γνωστή ως ‘Linear Feedback Shift Register’.
- Το κύκλωμα λειτουργεί με το να γίνεται shift στις τιμές ενός shift register και η καινούργια τιμή που μπαίνει μέσα να είναι μια συνάρτηση της προηγούμενης κατάστασης.



Pseudo Random Number Generator (Συνέχεια)

- Το κύκλωμα αυτό όπως είναι προφανές δεν παράγει τυχαίους αριθμούς, αφού αν γνωρίζουμε την τιμή για μία χρονική στιγμή τότε μπορούμε να υπολογίσουμε την επόμενη.
- Αυτό που είναι τυχαίο όμως είναι το πότε κοιτάμε την έξοδο της γεννήτριας.

Επιλογή νέας θέσης για το μήλο

- Τους πρώτους 169^2=28.561 κύκλους κάθε 25.000.000 κύκλους ξεκινάει ένας έλεγχος.
- Υπάρχουν συνολικά 169 διαθέσιμες θέσεις, και συνεπώς το φιδάκι μπορεί να έχει μέγεθος μέχρι 169.
- Άρα σε κάθε μία από τις 169 θέσεις ελέγχουμε αν κάποιο από τα 169 μέρη του φιδιού βρίσκονται πάνω στην θέση που εξετάζουμε.
- Αν δεν είναι διαθέσιμη πάμε στην επόμενη.
- Αν είναι ελεύθερη τότε τρέχουμε έναν σειριακό αλγόριθμο για random sampling, ο οποίος κρατάει ισοπίθανα μία από όλες τις τιμές που περνάνε ως είσοδος.

Division LUT και Random Sampling

- Το random sampling λειτουργεί με το να αναθέτει στο i-οστό αντικείμενο πιθανότητα να το κρατήσει ίση με 1/i.
- Άρα εύκολα μπορεί να αποδειχτεί ότι στο τέλος μετά από N στοιχεία, το στοιχείο που θα έχει κρατηθεί θα είναι ένα από αυτά τα N με ίση πιθανότητα 1/N.
- Συνεπώς θα πρέπει να λυθεί και το πρόβλημα της διαίρεσης, το οποίο κυκλωματικά είναι δύσκολο (παίρνει πολλούς κύκλους).
- Όμως το να φτιάξουμε ένα Look Up Table που να μας δίνει το πηλίκο μιας διαίρεσης για κάθε διαιρέτη (1 ως 169) είναι εύκολο.

Κίνηση φιδιού

- Κρατάμε σε registers την θέση του κάθε μέρος του φιδιού.
- Μια απλή FSM με βάση τα push buttons ελέγχει την επόμενη θέση του κεφαλιού του φιδιού.
- Η κίνηση μεταδίδεται στα υπόλοιπα μέρη του φιδιού με το να μεταδίδουμε την παλιά τιμή του κεφαλιού ένα βήμα κάτω και ούτω καθεξής.

Score

- Κάθε φορά που το φιδάκι τρώει ένα μήλο αυξάνεται η τιμή ενός μετρητή.
- Αυτό επίσης ‘ξεκλειδώνει’ επιπλέον register με αποτέλεσμα να ζωγραφίζεται το φιδάκι μεγαλύτερο.
- Το δυαδικό score προκειμένου να αναπαρασταθεί πρέπει να μετατραπεί σε BCD μορφή, όπου εκεί χρησιμοποιήθηκε έτοιμος κώδικας.

Ζωγραφική

- Σε κάθε κύκλο δίνεται ως είσοδος στο Game Component μια X και μία Y συντεταγμένη που αντιστοιχούν σε ποια θέση στην οθόνη βρίσκεται ο ‘pointer’ που ‘ζωγραφίζει’.
- Εξετάζεται λοιπόν, αν βρίσκεται κάτι σε αυτήν την (X, Y) θέση και επιστρέφεται κατάλληλο χρώμα με βάση το Sprite, αλλιώς μαύρο.

Συνθήκες τερματισμού

- Το να χτυπήσει το κεφάλι του φιδιού στα σύνορα, το οποίο ελέγχεται πολύ εύκολα κάθε φορά που πάει να κινηθεί το φίδι.
- Το να χτυπήσει το κεφάλι του φιδιού σε κάποιο μέρος της ουράς του.
- Αυτό είναι πιο δύσκολο να ελεγχτεί αλλά υλοποιείται παρόμοια με την αναζήτηση νέας θέσης για το μήλο.
- Ουσιαστικά ελέγχουμε μέσα σε 168 κύκλους αν το κεφάλι βρίσκεται σε θέση όπου και κάποιο άλλο μέρος του φιδιού βρίσκεται.

Τι απομένει / Μελλοντικά σχέδια

- Main Menu οθόνη με δυνατότητα επιλογής δυσκολίας
- Game over οθόνη.
- Καλύτερα γραφικά.

Σχόλια, απορίες, ιδέες;

ΕΥΧΑΡΙΣΤΩ ΓΙΑ ΤΟΝ ΧΡΌΝΟ ΣΑΣ

Πηγές

- Ο κώδικας μου: https://github.com/mouroutzoglou/FPGsnAke
- Xilinx documentation και drivers για το HDMI: https://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf
- Pseudo RNG: https://en.wikipedia.org/wiki/Linear-feedback_shift_register
- Binary σε BCD: http://www.johnloomis.org/ece314/notes/devices/binary_to_BCD/bin_to_bcd.html