Σχολή Θετικών Επιστημών και Τεχνολογίας

Τμήμα Πληροφορικής

Πτυχιακή εργασία

Σχεδιασμός και υλοποίηση ενός ρετρό παιχνιδιού σε FPGA με εξωτερική είσοδο και απεικόνιση σε οθόνη

Χατζηαβανίδου Κυριακή

Επιβλέπων καθηγητής: Δασυγένης Μηνάς

Πάτρα, Ιούλιος 2026

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχεδιασμός και υλοποίηση ενός ρετρό παιχνιδιού σε FPGA με εξωτερική είσοδο και απεικόνιση σε οθόνη

Χατζηαβανίδου Κυριακή

Επιτροπή Επίβλεψης Πτυχιακής / Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Δασυγένης Μηνάς

Αναπληρωτής καθηγητής στο Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας

Συν-Επιβλέπων Καθηγητής:

Κακαρούντας Αθανάσιος

Καθηγητής στο Πανεπιστήμιο Θεσσαλίας, στο Τμήμα Πληροφορικής και Βιοϊατρικής Πληροφορικής.

Συν-Επιβλέπων Καθηγητής:

Κονοφάος Νικόλαος

Καθηγητής στη Σχολή Θετικών Επιστημών του Αριστοτέλειου Πανεπιστημίου Θεσσαλονίκης

Πάτρα, Ιούλιος 2026

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Δασυγένη Μηνά, για την πολύτιμη καθοδήγηση, την υποστήριξη και την εμπιστοσύνη που μου παρείχε καθ' όλη τη διάρκεια εκπόνησης της παρούσας πτυχιακής εργασίας.

Η παρούσα εργασία αφιερώνεται στο σύντροφό μου, που πίστεψε σε μένα και τον ευχαριστώ για τη συνεχή του στήριξη, την κατανόηση και την υπομονή που έδειξε καθ' όλη τη διάρκεια των σπουδών μου.

Περίληψη

Η παρούσα πτυχιακή εργασία έχει ως αντικείμενο τον σχεδιασμό και την υλοποίηση του κλασικού ρετρό παιχνιδιού Tetris σε FPGA, με χρήση της πλακέτας DE23 Lite, εξωτερικής εισόδου από χειριστήριο τύπου arcade και απεικόνισης σε εξωτερική οθόνη μέσω θύρας HDMI, με στόχο την αναβίωση ενός δημοφιλούς ρετρό παιχνιδιού μέσα από μια σύγχρονη υλική υλοποίηση. Το Tetris αποτελεί ένα από τα πιο αναγνωρίσιμα και διαχρονικά παιχνίδια, με ιδιαίτερο ενδιαφέρον λόγω της χαρακτηριστικής λογικής του, που περιλαμβάνει τη δημιουργία και πτώση σχημάτων, την περιστροφή τους, τον έλεγχο, τη διαγραφή γραμμών και τη βαθμολόγηση. Η υλοποίηση απαιτεί τον συνδυασμό διαφορετικών τομέων ψηφιακής σχεδίασης, όπως την αναγνώριση εισόδων από το χειριστήριο, την επεξεργασία και αποθήκευση της κατάστασης του παιχνιδιού καθώς και την απεικόνιση γραφικών σε πραγματικό χρόνο μέσω HDMI. Βασικό ρόλο κατέχει η χρήση γλώσσας HDL (VHDL/Verilog) για την περιγραφή του συστήματος, καθώς και η αξιοποίηση εργαλείων ανάπτυξης και προσομοίωσης για τον έλεγχο σωστής λειτουργίας. Η πρόκληση εστιάζει στον συγχρονισμό όλων των επιμέρους μονάδων, από την αναγνώριση των κινήσεων του παίκτη έως την οπτικοποίηση του αποτελέσματος στην οθόνη, με τρόπο που να διασφαλίζει την ομαλή εμπειρία του παιχνιδιού. Τα αποτελέσματα επιβεβαιώνουν ότι το σύστημα μπορεί να αναπαράγει το Tetris με πλήρη λειτουργικότητα, προσφέροντας σταθερό σήμα HDMI, ακριβή απεικόνιση του πεδίου του παιχνιδιού, ομαλή απόκριση στις κινήσεις του χειριστηρίου και αξιόπιστη εφαρμογή των κανόνων του παιχνιδιού. Η τελική υλοποίηση αποδεικνύει ότι οι πλατφόρμες FPGA μπορούν να χρησιμοποιηθούν αποτελεσματικά για την ανάπτυξη διαδραστικών εφαρμογών πραγματικού χρόνου, ενώ ταυτόχρονα αναδεικνύει τις δυνατότητες τους σε εκπαιδευτικά και ερευνητικά έργα που συνδυάζουν ηλεκτρονικά, προγραμματισμό και ψυχαγωγία. Τέλος, προτείνονται πιθανές μελλοντικές επεκτάσεις της εργασίας, όπως η υποστήριξη παιχνιδιού για δύο παίκτες, η ενσωμάτωση ήχου, καθώς και η προσαρμογή της αρχιτεκτονικής για την υλοποίηση επιπλέον ρετρό παιχνιδιών στην ίδια πλατφόρμα.

Λέξεις – Κλειδιά

FPGA, VHDL, Ρετρό παιχνίδια, Tetris, HDMI, DE23 Lite, χειριστήριο τύπου arcade

Abstract

This present undergraduated thesis focuses on the design and implementation of the classic retro game Tetris on an FPGA, utilizing the DE23 Lite development board, external input from an arcade-style controller, and video output to an external display through an HDMI interface. The aim is to recreate a popular retro game through a modern hardware-based implementation. Tetris is one of the most recognizable and timeless games, notable for its characteristic logic, which involves the generation and falling of shapes, their rotation and control, the clearing of lines, and the scoring mechanism. The implementation requires the integration of multiple domains of digital design, such as input recognition from the controller, processing and storage of the game state, and real-time graphical rendering via HDMI. A central role is played by the use of HDL (VHDL/Verilog) for system description, as well as development and simulation tools for verifying correct functionality. The main challenge lies in synchronizing all functional units—from player input detection to the visualization of results on the display—in a way that ensures a smooth and responsive gameplay experience. The results confirm that the system is capable of reproducing Tetris with full functionality, offering stable HDMI output, accurate rendering of the game field, smooth response to controller actions, and reliable implementation of the game's rules. The final implementation demonstrates that FPGA platforms can be effectively used for developing real-time interactive applications, while also highlighting their potential in educational and research projects that combine electronics, programming, and entertainment. Finally, potential future extensions of the work are proposed, such as enabling two-player support, adding sound integration, and adapting the architecture for the implementation of additional retro games on the same platform.

Keywords

FPGA, VHDL, Retro games, Tetris, HDMI, DE23 Lite, Arcade-style controller

Περιεχόμενα

Κατάλογος Εικόνων / Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες & Ακρωνύμια

ARR	Auto Repeat Rate
ALMs	Adaptive Logic Modules
BRAM	Block RAM
CEC	Consumer Electronics Control
CLB	Configurable Logic Block
CPU	Central Processing Unit
DAS	Delay Auto Shift
DDC	Display Data Channel
FF	Flip Flop
FPGA	Field-Programmable Gate Array
FSM	Finite State Machines
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HPS	Hard Processor System
HSSI	High-Speed Serial Interface
IC	Intergrated Circuit
I2C	Inter-Integrated Circuit
I/O	Input/Output
ISP	In-system programmable
LB	Logic Block
LE	Logic Element
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
MMCM	Mixed-Mode Clock Manager
OTP	One-time programmable
PLL	Phase-Locked Loop
RC	Resistor-Capacitor
TMDS	Transition Minimized Differential Signaling
VHDL	Very High-Speed Intergrated Circuits (VHSIC) Hardware Description Language

1. Εισαγωγή

Η ανάπτυξη ψηφιακών συστημάτων σε προγραμματιζόμενη λογική αποτελεί έναν από τους σημαντικότερους τομείς της σύγχρονης ηλεκτρονικής σχεδίασης, καθώς επιτρέπει την υλοποίηση πολύπλοκων κυκλωμάτων με υψηλή απόδοση, παραλληλία και ευελιξία. Στο πλαίσιο της παρούσας εργασίας, η τεχνολογία αυτή αξιοποιείται για την υλοποίηση ενός κλασικού ηλεκτρονικού παιχνιδιού, του Tetris, όπου μέσα από τη διαδικασία ανάπτυξής του θα προσφέρει καλύτερη κατανόηση των βασικών αρχών της ψηφιακής σχεδίασης. Στο κεφάλαιο αυτό παρουσιάζονται το αντικείμενο και ο σκοπός της εργασίας, τα κίνητρα επιλογής, σχετική έρευνα και συναφείς εργασίες, καθώς και οι στόχοι και τα αναμενόμενα αποτελέσματα, ώστε να δοθεί ένα ολοκληρωμένο πλαίσιο για τα επόμενα κεφάλαια.

1.1 Αντικείμενο και σκοπός της εργασίας

Η ταχεία εξέλιξη της τεχνολογίας των ψηφιακών συστημάτων και των ολοκληρωμένων κυκλωμάτων έχει καταστήσει δυνατή την υλοποίηση πολύπλοκων λειτουργιών απευθείας σε επίπεδο υλικού. Τα Field Programmable Gate Arrays (FPGAs) [1], αποτελούν χαρακτηριστικό παράδειγμα τέτοιας τεχνολογίας, καθώς επιτρέπουν την επαναδιαμόρφωση του κυκλώματος μετά την κατασκευή, προσφέροντας ευελιξία και υψηλή απόδοση σε εφαρμογές όπως η επεξεργασία σήματος, η τεχνητή νοημοσύνη και τα ηλεκτρονικά παιχνίδια.

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός ρετρό ηλεκτρονικού παιχνιδιού Tetris σε πλακέτα FPGA, με χρήση γλώσσας περιγραφής υλικού HDL (VHDL/ Verilog) [2], αξιοποιώντας εξωτερικό χειριστήριο τύπου arcade και έξοδο προβολής σε οθόνη μέσω θύρας HDMI. Στόχος είναι η δημιουργία ενός πλήρως λειτουργικού παιχνιδιού που θα υλοποιείται αποκλειστικά με ψηφιακή λογική, χωρίς τη χρήση κλασικού μικροεπεξεργαστή ή έτοιμου λειτουργικού συστήματος.

Η εργασία επικεντρώνεται τόσο στην εκπαιδευτική και ερευνητική αξία της χρήσης FPGA για εφαρμογές πολυμέσων, όσο και στην κατανόηση της αρχιτεκτονικής και των δυνατοτήτων των σύγχρονων προγραμματιζόμενων κυκλωμάτων. Μέσω αυτής της εφαρμογής, επιδιώκεται η εμπέδωση εννοιών όπως η παράλληλη επεξεργασία, η χρονική ακρίβεια συγχρονισμού, η δημιουργία γραφικών σε πραγματικό χρόνο και η διαχείριση εξωτερικών συσκευών εισόδου/εξόδου.

Ο τελικός σκοπός είναι η ανάπτυξη ενός αυτόνομου συστήματος παιχνιδιού, το οποίο θα αναπαριστά το Tetris με τρόπο πιστό στην κλασική εκδοχή του, αποδεικνύοντας παράλληλα τη δυνατότητα των FPGA να χρησιμοποιούνται και σε εφαρμογές ψυχαγωγίας.

1.2 Κίνητρα επιλογής του Tetris ως ρετρό παιχνίδι για υλοποίηση

Το Tetris [3] αποτελεί ένα από τα πλέον αναγνωρίσιμα παιχνίδια όλων των εποχών, με απλή λογική και μηχανισμό που εξακολουθεί να ελκύει το ενδιαφέρον παικτών και προγραμματιστών εδώ και τέσσερις δεκαετίες. Δημιουργήθηκε το 1984 από τον Alexey Pajitnov και από τότε έχουν κυκλοφορήσει πολλές εκδόσεις και προσαρμογές σε διάφορες πλατφόρμες. Η επιλογή του Tetris σε σύγκριση με άλλα κλασικά ρετρό παιχνίδια, βασίζεται σε μια σειρά από δομικά, τεχνικά, εκπαιδευτικά και ερευνητικά πλεονεκτήματα που το καθιστούν ιδανικό υποψήφιο για μελέτη και υλοποίηση σε ακαδημαϊκά πλαίσια.

Τα βασικά κίνητρα επιλογής του Tetris ως αντικείμενο υλοποίησης είναι τα εξής:

- ***Απλότητα δομής και λογικής***

Το Tetris διαθέτει εξαιρετικά απλή αλλά πλήρη δομή παιχνιδιού, η οποία επιτρέπει την εστίαση στον σχεδιασμό υλικού. Αποτελείται από ένα σταθερό πλέγμα 10×20 κυψελών, επτά βασικά σχήματα (tetrominoes) και απλούς κανόνες όπως περιστροφή και πτώση σχημάτων, διαγραφή πλήρους γραμμών και βαθμολόγηση [4].

- ***Τεχνικά κίνητρα***

Η ανάπτυξη ενός παιχνιδιού σε FPGA, σε αντίθεση με υλοποίηση σε μικροεπεξεργαστή, απαιτεί σχεδιασμό σε επίπεδο υλικού. Η επιλογή του Tetris επιτρέπει την εξερεύνηση τεχνικών θεμάτων όπως την αναπαράσταση πλέγματος δύο διαστάσεων, το σχεδιασμό κίνησης και τη διαχείριση συγκρούσεων, την ανάγνωση εισόδων από χειριστήριο τύπου arcade και τον ακριβή χρονισμό για τη δημιουργία σήματος HDMI, ώστε να απεικονίζεται στην οθόνη. Το Tetris, με τη σχετικά απλή ροή του παιχνιδιού, δίνει τη δυνατότητα να σχεδιαστεί ένα πλήρες λειτουργικό σύστημα χωρίς να απαιτείται εξωτερική μνήμη προγράμματος ή μεγάλης πολυπλοκότητας λογισμικό. Απαιτεί περιορισμένους πόρους υλικού, καθιστώντας το εφικτό για υλοποίηση σε FPGA χαμηλού ή μεσαίου κόστους.

- ***Εκπαιδευτική Αξία***

Η υλοποίηση του Tetris συνδυάζει έννοιες ψηφιακής σχεδίασης, χρονοπρογραμματισμού, γραφικών και συστημάτων εισόδου/εξόδου σε ένα πρακτικό περιβάλλον. Συγκεκριμένα, απαιτεί την κατανόηση και υλοποίηση των βασικών στοιχείων ψηφιακής σχεδίασης όπως finite state machines (FSM), διαχείριση χρονισμού, μνήμης, σχεδίαση μονάδων εισόδου/εξόδου και αλγορίθμους ελέγχου. Επιπλέον, αναδεικνύει τη σημασία της παράλληλης επεξεργασίας, καθώς το FPGA εκτελεί ταυτόχρονα πολλές λειτουργίες, για παράδειγμα ενημέρωση πλέγματος, απόδοση εικόνας HDMI και ανάγνωση χειριστηρίου. Κάθε ενέργεια του χρήστη, όπως η μετακίνηση ή περιστροφή ενός σχήματος, μεταφράζεται άμεσα σε αλλαγές στα ψηφιακά κυκλώματα που αντιστοιχούν στη μνήμη του πλέγματος, και αυτές οι αλλαγές εμφανίζονται σε πραγματικό χρόνο στην οθόνη. Αυτό προσφέρει έναν ορατό τρόπο κατανόησης της αλληλεπίδρασης μεταξύ υλικού και λογισμικού λόγω του άμεσου αποτελέσματος στην οθόνη.

- ***Ερευνητικό ενδιαφέρον***

Το Tetris, ως ρετρό παιχνίδι, προκαλεί νοσταλγία και επιτρέπει την εξερεύνηση ιστορικών πτυχών της τεχνολογίας του παιχνιδιού, ενώ αναδεικνύει πώς οι πλατφόρμες FPGA δεν περιορίζονται σε αυστηρά βιομηχανικές ή επιστημονικές εφαρμογές, αλλά μπορούν να αποτελέσουν πλατφόρμα δημιουργικότητας και ψυχαγωγίας. Η υλοποίηση του Tetris σε FPGA προσφέρει πολλές ευκαιρίες έρευνας, ιδιαίτερα σε τεχνικά και θεωρητικά θέματα που σχετίζονται με τη σχεδίαση υλικού, την ανάλυση απόδοσης και χρονισμού, την ψηφιακή απεικόνιση, τη χρήση περιφερειακών καθώς και τις επεκτάσεις λειτουργικότητας (π.χ. χρήση δυο παικτών, ήχος, επίπεδα δυσκολίας, βελτιωμένα γραφικά). Επιπλέον, προσφέρεται η δυνατότητα σύγκρισης με άλλες πλατφόρμες για να εξεταστούν η ενεργειακή απόδοση, ο χρόνος απόκρισης, η ποιότητα γραφικών, η ευκολία υλοποίησης καθώς και η εμπειρία χρήσης μεταξύ FPGA, μικροελεγκτών και μικροεπεξεργαστών.

- ***Προσωπικά κίνητρα***

Σε προσωπικό επίπεδο, η επιλογή του Tetris, αντανακλά μια σειρά προσωπικών κινήτρων που συνδυάζουν νοσταλγική σύνδεση, εκπαιδευτική αξία και τεχνική πρόκληση. Ως κλασικό και αναγνωρίσιμο από πολλές γενιές παιχνίδι, προσφέρει ιδανικό πεδίο εφαρμογής βασικών εννοιών ψηφιακού σχεδιασμού, όπως η διαχείριση πεπερασμένων μηχανών καταστάσεων (FSM), χρονισμού και συγχρονισμού, η χρήση μνήμης για την αποθήκευση του πλέγματος, καθώς και η επεξεργασία εισόδου με χειριστήριο και εξόδου

μέσω διεπαφής HDMI. Η μέτρια πολυπλοκότητά του επιτρέπει την ολοκληρωμένη ανάπτυξη ενός λειτουργικού συστήματος χωρίς να υπερβαίνει τους πόρους τυπικών πλατφόρμων FPGA, ενώ παράλληλα ενθαρρύνει την προσωπική δημιουργικότητα μέσω επεκτάσεων όπως η προσθήκη βαθμολογίας, επιπέδων δυσκολίας ή ηχητικών εφέ. Η οπτική άμεση ανατροφοδότηση του παιχνιδιού ενισχύει την αίσθηση επίτευξης και αποτελεί ισχυρό κίνητρο ικανοποίησης και ενίσχυσης τεχνικών δεξιοτήτων.

1.3 Σχετική έρευνα και συναφείς εργασίες

Η έρευνα σε υπάρχουσες υλοποιήσεις του παιχνιδιού Tetris, καθώς και άλλων κλασικών ρετρό παιχνιδιών έχει ως στόχο την αποτύπωση των διαφορετικών τεχνολογικών προσεγγίσεων και τον εντοπισμό βέλτιστων πρακτικών. Η βιβλιογραφική και διαδικτυακή αναζήτηση περιλαμβάνει έργα που έχουν υλοποιηθεί σε διαφορετικές πλατφόρμες, όπως Arduino, μικροεπεξεργαστές, FPGA με εξόδους VGA ή HDMI.

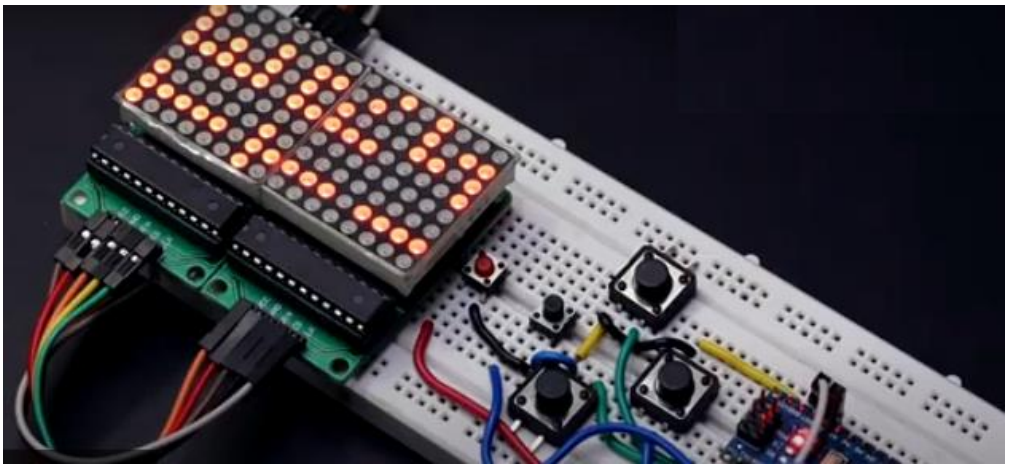
Η συγκριτική μελέτη αυτών των έργων επιτρέπει την κατανόηση των τεχνικών περιορισμών και δυνατοτήτων κάθε πλατφόρμας, αναδεικνύει συνήθη προβλήματα όπως ο συγχρονισμός, η διαχείριση μνήμης, η διαχείριση εισόδων και η σχεδίαση γραφικών, και συμβάλλει στον καθορισμό μιας βέλτιστης αρχιτεκτονικής για το σύστημα που αναπτύσσεται. Επιπλέον, αναδεικνύει ευκαιρίες για πρωτοτυπία, όπως η αξιοποίηση ψηφιακής εξόδου HDMI, η χρήση arcade-style χειριστηρίου και η υλοποίηση πλήρως παραμετροποιήσιμης αρχιτεκτονικής στο FPGA, στοιχεία που διαφοροποιούν την παρούσα εργασία από συνηθισμένες ή πιο απλουστευμένες προσεγγίσεις.

Παρακάτω ακολουθούν έρευνες σχετικές με υλοποιήσεις Tetris σε μικροελεγκτές, σε μικροεπεξεργαστές, σε λογισμικό υπολογιστή, σε FPGA καθώς και συγκριτική ανάλυση αυτών. Τέλος, ακολουθεί μια σύντομη αναφορά υλοποιήσεων άλλων κλασικών παιχνιδιών σε FPGA.

1.3.1 Υλοποίηση Tetris με μικροελεγκτές (Arduino)

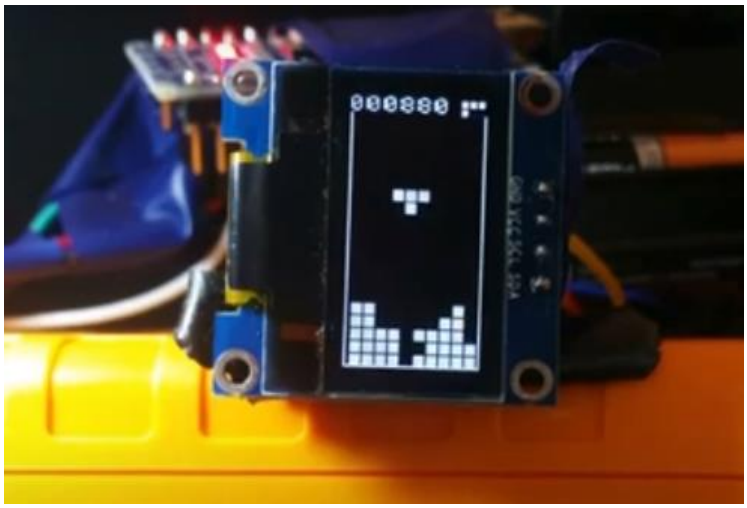
Οι μικροελεγκτές αποτελούν τη συνηθέστερη λύση για απλές ή εκπαιδευτικές υλοποιήσεις παιχνιδιών. Το Tetris είναι από τα πιο δημοφιλή παραδείγματα λόγω της σχετικά μικρής υπολογιστικής πολυπλοκότητάς του. Οι υλοποιήσεις του Tetris σε πλατφόρμες Arduino

[5] αποτελούν χαρακτηριστικό παράδειγμα του πώς ένας μικροελεγκτής χαμηλού κόστους μπορεί να υποστηρίξει πλήρως ένα κλασικό παιχνίδι. Στο έργο "8×8 Tetris" [6], το παιχνίδι αποδίδεται σε μια LED matrix 8×8, όπως φαίνεται στην Εικόνα 1, με χρήση 3 push buttons για μετακίνηση και περιστροφή, αναδεικνύοντας πως ακόμη και μια τόσο μικρή ανάλυση μπορεί να υποστηρίξει την πλήρη λειτουργικότητα του παιχνιδιού.



Εικόνα 1 Tetris σε 8x8 LED matrix MAX7219[1]

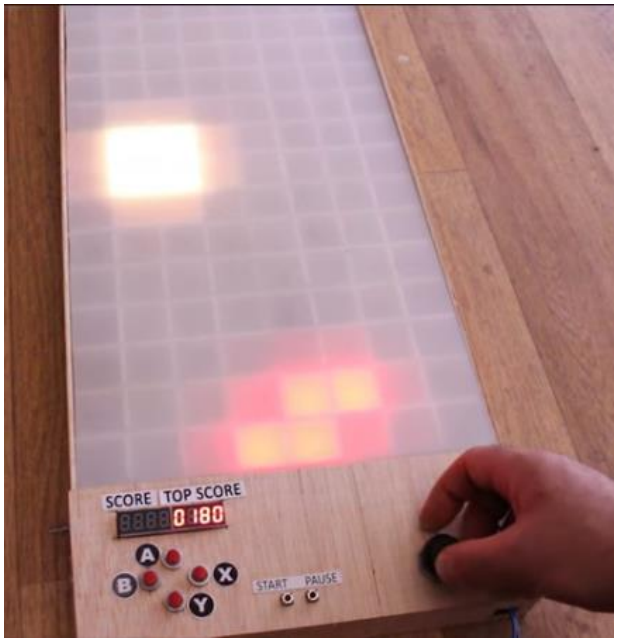
Αντίστοιχα, το "Tetris Clone With OLED SSD1306 For Arduino Nano/Uno" [7] παρουσιάζει μια πιο εξελιγμένη προσέγγιση, χρησιμοποιώντας OLED οθόνη 128×64, αναλογικό πληκτρολόγιο και μνήμη EEPROM για αποθήκευση υψηλού σκορ (Εικόνα 2).



Εικόνα 2 Tetris σε OLED SSD1306[2]

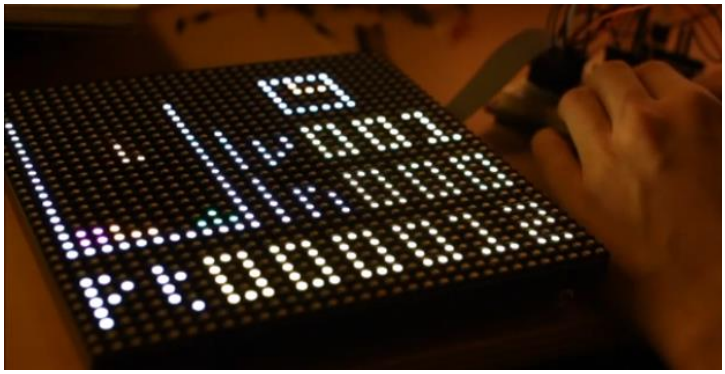
[1]Η εικόνα είναι από στιγμιότυπο του video από τη διεύθυνση
https://www.viralsciencecreativity.com/post/arduino-tetris-game-max7219-8x8-matrix-display

Πιο εντυπωσιακή οπτικά είναι η υλοποίηση “Big and Glowy Tetris via Arduino” [8], όπου μια LED matrix 8×16 (Εικόνα 3) και ένα Arduino Mega δίνουν έγχρωμη απεικόνιση των σχημάτων, ο χειρισμός γίνεται με joystick και κουμπιά στο πάνελ για κίνηση και περιστροφή κομματιών, ενώ η ενσωμάτωση ήχου προσφέρει ολοκληρωμένη εμπειρία.



Εικόνα 3 Tetris σε LED matrix 8×16[3]

Τέλος, το “LX' Arduino Tetris” [9] χρησιμοποιεί μια μεγάλη 32×32 RGB LED matrix (Εικόνα 3), επιτρέποντας υψηλότερη ανάλυση και καθαρότερο οπτικό αποτέλεσμα. Είναι υλοποιημένο σε Arduino/GenuinoUno και ο χειρισμός των κινήσεων γίνεται με 4 push-buttons.



Εικόνα 4 Tetris σε 32×32 RGB LED matrix panel[4]

[2] Η εικόνα είναι από στιγμιότυπο του video από τη διεύθυνση https://www.instructables.com/Tetris-Clone-With-OLED-SSD1306I2C-for-Arduino-Nano

[3] Η εικόνα είναι από στιγμιότυπο του video από τη διεύθυνση https://hackaday.com/2019/09/06/big-and-glowy-tetris-via-arduino

Αυτές οι υλοποιήσεις αποδεικνύουν τη μεγάλη ευελιξία της πλατφόρμας Arduino και την προσαρμογή σε διαφορετικά επίπεδα πολυπλοκότητας. Επομένως, αναδεικνύονται σημαντικά πλεονεκτήματα, όπως η απλότητα προγραμματισμού σε γλώσσες υψηλού επιπέδου (C/C++), αλλά και το χαμηλό κόστος τους και η ευρεία διαθεσιμότητα διαφορετικών ειδών πλατφόρμας, που επιτρέπουν τη γρήγορη και οικονομική κατασκευή του παιχνιδιού. Η ύπαρξη πολλών βιβλιοθηκών για τη διαχείριση οθονών και εισόδων μειώνει σημαντικά τον χρόνο ανάπτυξης και διευκολύνει την ενσωμάτωση των απαραίτητων περιφερειακών συσκευών. Παρά τα πλεονεκτήματα αυτά, οι μικροελεγκτές παρουσιάζουν και ορισμένους περιορισμούς. Η σειριακή εκτέλεση εργασιών μπορεί να προκαλέσει καθυστερήσεις σε περιπτώσεις απαιτητικής λογικής παιχνιδιού ή γραφικής απεικόνισης, ενώ η απόδοση εξαρτάται άμεσα από τη συχνότητα λειτουργίας του μικροελεγκτή, η οποία συνήθως είναι χαμηλότερη συγκριτικά με αυτή των FPGA. Επιπλέον, οι μικροελεγκτές δυσκολεύονται να υποστηρίξουν οθόνες υψηλής ανάλυσης, περιορίζοντας τις δυνατότητες γραφικής απεικόνισης του παιχνιδιού.

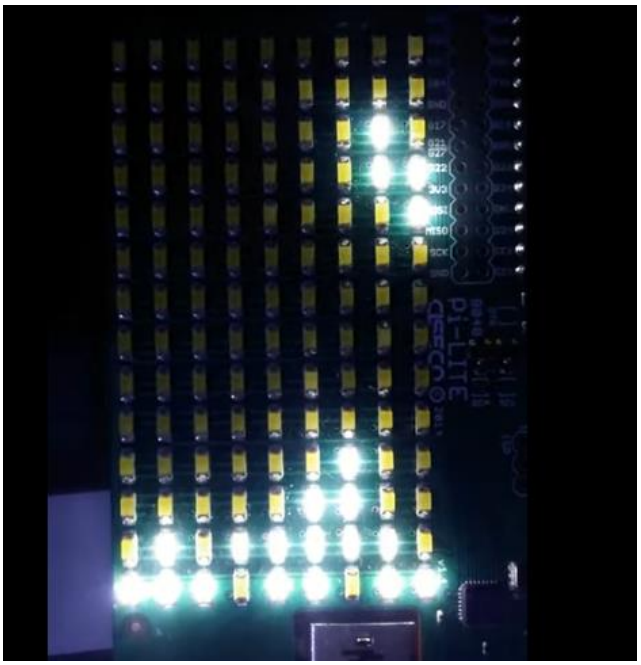
1.3.2 Υλοποίηση Tetris με μικροεπεξεργαστές (Raspberry Pi)

Οι υλοποιήσεις του Tetris σε Raspberry Pi [11] αξιοποιούν τις αυξημένες υπολογιστικές δυνατότητες, την υποστήριξη γλωσσών προγραμματισμού και την ευελιξία σύνδεσης με εξωτερικές οθόνες και LED matrices, επιτρέποντας την ανάπτυξη έργων μεγαλύτερης κλίμακας και πολυπλοκότητας σε σχέση με παραδοσιακούς μικροελεγκτές. Στο έργο “PyTetris” [12], το παιχνίδι υλοποιείται σε Python πάνω σε Raspberry Pi 4, με την απεικόνιση να πραγματοποιείται σε LED matrix 32×64 και τον χειρισμό να γίνεται μέσω arcade κουμπιών, τοποθετημένα σε 3D-εκτυπωμένο περίβλημα, προσφέροντας μια ολοκληρωμένη arcade εμπειρία (Εικόνα 5) .

[4] Η εικόνα είναι από στιγμιότυπο του video από τη διεύθυνση https://projecthub.arduino.cc/RomanSixty/lx-arduino-tetris-708606

Εικόνα 5 Tetris, 432x64 LED matrix[5]

Μια άλλη προσέγγιση παρουσιάζεται στο “Pieces of Pi” [13], όπου υλοποιείται σε Python2 και το Raspberry Pi οδηγεί έναν LED 14x9 x1-bit display, δημιουργώντας έντονα ρετρό χαρακτήρα με χαμηλό ρυθμό ανανέωσης περίπου 5 fps (Εικόνα 6).

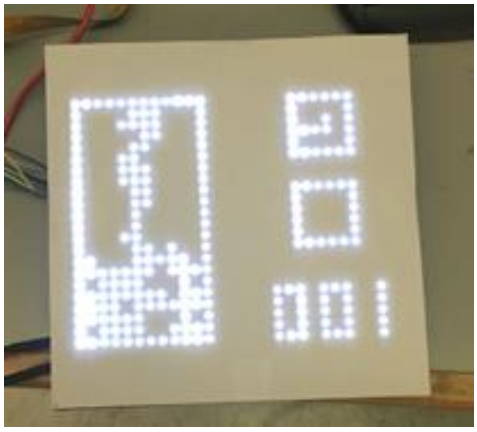


Εικόνα 6 Tetris σε LED 14x9 x1-bit display[6]

[5] Η εικόνα είναι από τη διεύθυνση https://courses.ece.cornell.edu/ece5990/ECE5725_Spring2024_Projects/02%20Monday%20May%2013/06%20Tetris/Monday_Group6_JasonHellerNoahAbramson/index.html

[6] Η εικόνα είναι από στιγμιότυπο του video που βρίσκεται στη διεύθυνση https://piecesofpi.co.uk/tetris-on-leds

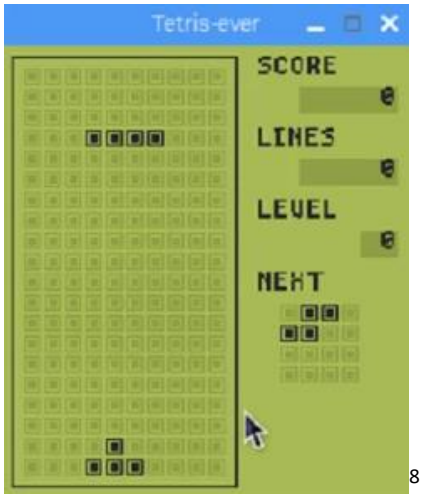
Σε ένα πιο σύνθετο πλαίσιο, ένα ακαδημαϊκό έργο το “LED Tetris” [14] αξιοποιεί ένα Raspberry Pi 3 για τον υπολογισμό της λογικής του παιχνιδιού, γραμμένο σε γλώσσα C, με είσοδο χρήστη 4×4 matrix keypad, ενώ η απεικόνιση γίνεται σε 32×32 RGB LED matrix που ελέγχεται από FPGA (Εικόνα 7).



Εικόνα 7 Tetris σε 32×32 RGB LED matrix[7]

Τέλος, το “Tetris-ever” [15], γραμμένο σε γλώσσα C με χρήση της βιβλιοθήκης SDL2 και με είσοδο από το πληκτρολόγιο, επιτρέπει την εκτέλεση του Tetris σε πλήρες γραφικό περιβάλλον με υψηλή απόδοση και σταθερότητα (Εικόνα 8).

Οι παραπάνω υλοποιήσεις αναδεικνύουν τη μεγάλη ποικιλία επιλογών που προσφέρει το Raspberry Pi καθιστώντας το ιδανική πλατφόρμα για σύγχρονες, πειραματικές και ερευνητικές εκδοχές του Tetris.



[8]

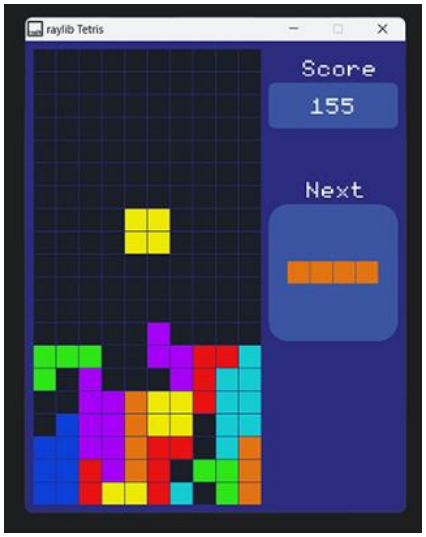
Εικόνα 8 Tetris-ever

[7] Η εικόνα είναι από τη διεύθυνση https://pages.hmc.edu/harris/class/e155/projects16/Linden_Slaats.pdf
[8] Η εικόνα είναι από τη διεύθυνση https://snapcraft.io/install/tetris-ever/raspbian

Η χρήση μικροεπεξεργαστών, όπως πλατφόρμες τύπου Raspberry Pi, προσφέρουν σημαντικά πλεονεκτήματα όπως ισχυρές γραφικές δυνατότητες μέσω βιβλιοθηκών και λειτουργικών συστημάτων ενώ η ανάπτυξη σε γλώσσες υψηλού επιπέδου καθιστά τη διαδικασία υλοποίησης ταχύτερη και πιο ευέλικτη. Επιπλέον, η υποστήριξη πολυμέσων διευκολύνει την ενσωμάτωση ήχου, δικτυακών λειτουργιών ή προηγμένων στοιχείων διεπαφής χρήστη, επεκτείνοντας τις δυνατότητες του κλασικού παιχνιδιού. Ωστόσο, η ύπαρξη ενός πλήρους λειτουργικού συστήματος εισάγει μεγαλύτερες καθυστερήσεις σε σχέση με την υλοποίηση στο υλικό, μειώνοντας την προβλεψιμότητα του χρόνου απόκρισης. Παράλληλα, η υλοποίηση σε μικροεπεξεργαστή επικεντρώνεται κυρίως στο λογισμικό, προσφέροντας λιγότερη εκπαίδευση πάνω στις αρχές σχεδίασης ψηφιακών κυκλωμάτων, ενώ και η κατανάλωση ενέργειας είναι συνήθως μεγαλύτερη σε σύγκριση με μικροελεγκτές ή FPGA για απλές εφαρμογές όπως το Tetris.

1.3.3 Υλοποίηση Tetris σε λογισμικό υπολογιστή

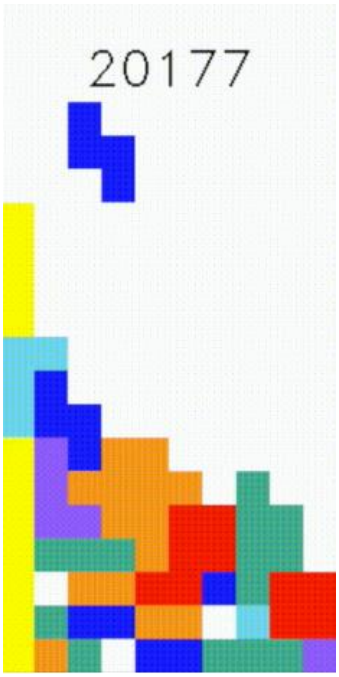
Οι υλοποιήσεις Tetris σε λογισμικό για υπολογιστή παρουσιάζουν εξαιρετική ποικιλία, καλύπτοντας ανάγκες από εκπαίδευση έως έρευνα σε τεχνητή νοημοσύνη. Ένα χαρακτηριστικό παράδειγμα είναι το Tetris σε C++ Tetris Game με χρήση βιβλιοθήκης γραφικών raylib [16], ιδανικό για Windows, MacOS και Linux, το οποίο αξιοποιεί εργαλεία για γραφικά, είσοδο και ήχο, προσφέροντας μια σύγχρονη υλοποίηση (Εικόνα 9).



Εικόνα 9 Raylib-Tetris[9]

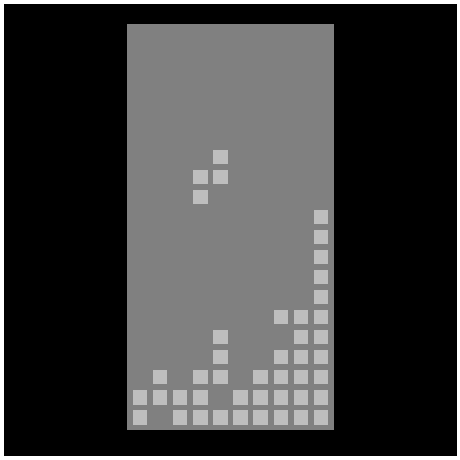
[9] Η εικόνα είναι από τη διεύθυνση https://github.com/educ8s/Cpp-Tetris-Game-with-raylib

Στο άλλο άκρο, το έργο “TetrisAI” [17], γραμμένο σε Python προχωρά υλοποιώντας έναν αυτόνομο παίκτη που μαθαίνει μέσω Deep Q-Learning, διαθέτοντας τόσο διαδραστική έκδοση όσο και αυτόματη εκδοχή στην οποία η τεχνητή νοημοσύνη παίζει από μόνη της (Εικόνα 10).



Εικόνα 10 Tetris AI[10]

Αντίστοιχης ερευνητικής κατεύθυνσης είναι και το Tetris περιβάλλον συμβατό με OpenAI Gym [18], γραμμένο σε Python (Εικόνα 11). Δεν αποτελεί κανονικό παιχνίδι αλλά περιβάλλον έρευνας και προσομοίωσης και είναι κατάλληλο για εκπαίδευση.



Εικόνα 11 Simple Tetris[11]

[10] Η εικόνα είναι από τη διεύθυνση https://github.com/ChesterHuynh/tetrisAI
[11] Η εικόνα είναι από τη διεύθυνση https://github.com/tristanrussell/gym-simpletetris

Τέλος, για εκπαιδευτικούς σκοπούς, υπάρχει και το “Python-Tetris” [19], υλοποιημένο σε Python χωρίς γραφικά, με κώδικα που αναδεικνύει τη βασική λογική του παιχνιδιού και αποτελεί ιδανικό παράδειγμα για αρχάριους προγραμματιστές.

Συνολικά, οι υλοποιήσεις σε PC αναδεικνύουν την ευελιξία του Tetris ως εργαλείο τόσο για ανάπτυξη λογισμικού όσο και για πειραματική έρευνα. Επομένως, η υλοποίηση του Tetris σε υπολογιστή προσφέρει υψηλό επίπεδο ευκολίας και ευελιξίας, καθώς βασίζεται σε εργαλεία ανάπτυξης, χωρίς να απαιτεί γνώση υλικού. Η χρήση ισχυρών γραφικών υποδομών επιτρέπει εύκολη υποστήριξη υψηλής ανάλυσης και animations, ενώ η ποικιλία διαθέσιμων συσκευών εισόδου καθιστά την ενσωμάτωση ελέγχου απλή και διασκεδαστική. Παρ’ όλα αυτά, η υλοποίηση αποκλειστικά σε περιβάλλον υπολογιστή δεν παρέχει εμπειρία στη σχεδίαση υλικού, κάτι που αποτελεί βασικό ζητούμενο σε εργασίες που στοχεύουν στην κατανόηση ψηφιακών κυκλωμάτων. Επιπλέον, η απόδοση του παιχνιδιού εξαρτάται από το λειτουργικό σύστημα χωρίς δυνατότητα βελτιστοποίησης σε επίπεδο κυκλωμάτων, ενώ η συγκεκριμένη επιλογή θεωρείται λιγότερο πρωτότυπη αφού το Tetris έχει υλοποιηθεί πολλές φορές σε λογισμικά υπολογιστών.

1.3.4 Υλοποίηση Tetris με FPGA

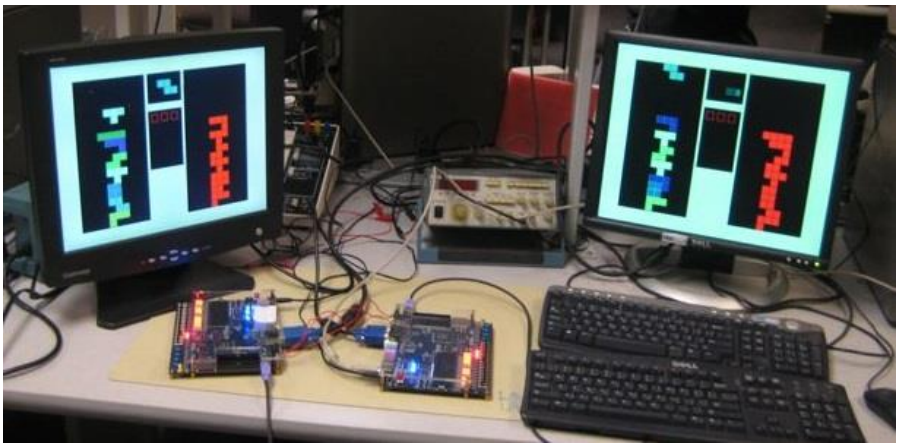
Η υπάρχουσα βιβλιογραφία παρουσιάζει πληθώρα έργων που εξετάζουν την υλοποίηση του Tetris σε FPGA, αναδεικνύοντας τις δυνατότητες των τεχνολογιών ψηφιακής λογικής στη δημιουργία παιχνιδιών με έμφαση στο υλικό. Το έργο ‘’Tetris Video Game via FPGA’’ [20] αποτελεί χαρακτηριστικό παράδειγμα βασικής και ολοκληρωμένης σχεδίασης παιχνιδιού σε FPGA, όπου υλοποιείται σε επεξεργαστή NIOS II με Verilog, με χρήση εισόδου από πληκτρολόγιο, και με έμφαση στη διαμόρφωση της λογικής παιχνιδιού ως πεπερασμένης μηχανής καταστάσεων (FSM), τον έλεγχο σύγκρουσης και την απεικόνιση σε οθόνη VGA.

Αντίστοιχα, το έργο “A fully embedded Tetris game application in VHDL’’ [21] παρουσιάζει μία υλοποίηση σε VHDL στην DE0 Development and Education board με 50MHz on-board clock, είσοδο από buttons της πλακέτας και έξοδο σε VGA. Η πλακέτα ανάπτυξης είναι εξοπλισμένη με τον Altera Cyclone III 3C16, που προσφέρει 15.408 LE, τάση τροφοδοσίας 1,15V–1,25V, και συνολική μνήμη RAM 516.096 bits και 346 pins εισόδου/εξόδου. Έγινε χρήση του λογισμικού Altera Quartus II και για τις προσομοιώσεις

το πρόγραμμα Modelsim. Η υλοποίηση αυτή είναι απλή και αρκετά αποδοτική, χωρίς την ανάγκη χρήσης μνήμης SRAM.

Στα πλαίσια διπλωματικής εργασίας με τίτλο "Implementation of Tetris Game on a FPGA" [22], η υλοποίηση του Tetris χρησιμοποιείται ως ακαδημαϊκή μελέτη για εξερεύνηση προχωρημένων τεχνικών ψηφιακής σχεδίασης, προσφέροντας μια θεωρητική και ερευνητική διάσταση. Η υλοποίηση γίνεται με χρήση της πλακέτας DIGILENT SPARTAN-3 Starter Board Xilinx XC3S200 με 50MHz on-board clock, 4.320 LE, συνολική μνήμη RAM 216.000 bits και 173 pins I/O. Τα buttons της πλακέτας χρησιμοποιήθηκαν για είσοδο και VGA οθόνη για έξοδο.

Το έργο ''Parallel Tetris project'' [23] εστιάζει στην παράλληλη εκτέλεση πολλών στοιχείων που μπορεί να επιτευχθεί μέσω FPGA, υλοποιώντας μηχανισμούς παραλληλίας για τον έλεγχο συγκρούσεων και την ενημέρωση της λογικής του παιχνιδιού (Εικόνα 12). Στόχος ήταν η δημιουργία έκδοσης παιχνιδιού με 2 παίκτες, οι οποίοι θα παίζουν βλέποντας τη δική τους οθόνη και κάθε παίκτης θα είχε τη δυνατότητα να βλέπει την κατάσταση του παιχνιδιού του αντιπάλου του. Για το σκοπό αυτό χρησιμοποιήθηκαν δυο Altera De2 Boards, οποίες επικοινωνούσαν μέσω της σειριακής θύρας. Η είσοδος έγινε από πληκτρολόγιο και η έξοδος σε οθόνη VGA.



Εικόνα 12 Parallel Tetris[12]

[12] Η εικόνα είναι από τη διεύθυνση
https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2008/aj77/aj77/finalreport.html

Τέλος, το “Tetris in Hardware” [24] παρέχει μια υλοποίηση σε Verilog, σε Nexys 3 development board, με μονάδες για χειρισμό του πλέγματος και παραγωγή VGA σημάτων, αποτελώντας άμεση και εύχρηστη βάση αναφοράς για μελλοντικές υλοποιήσεις.

Συνολικά, τα έργα αυτά επιβεβαιώνουν την καταλληλότητα των FPGA για υλοποίηση παιχνιδιών πραγματικού χρόνου, προβάλλοντας πλεονεκτήματα όπως η παράλληλη εκτέλεση διαφορετικών λειτουργιών (έλεγχος εισόδου, ενημέρωση της οθόνης, διαχείριση λογικής του παιχνιδιού), η υψηλή ταχύτητα, η χαμηλή καθυστέρηση και η ακριβής χρονική συμπεριφορά. Η απευθείας διασύνδεση με πρωτόκολλα, όπως VGA ή HDMI μειώνει περαιτέρω τις καθυστερήσεις και προσφέρει καλύτερη κατανόηση της επικοινωνίας με εξωτερικές συσκευές. Παράλληλα, η διαδικασία ανάπτυξης έχει σημαντική εκπαιδευτική αξία, καθώς περιλαμβάνει χρήση γλώσσας HDL (Verilog/VHDL), διαχείριση χρονισμού και σχεδίαση ψηφιακών κυκλωμάτων. Ωστόσο, η ανάπτυξη σε FPGA παρουσιάζει και προκλήσεις. Απαιτεί μεγαλύτερο χρόνο λόγω της πολυπλοκότητας του σχεδιασμού και της ανάγκης εξειδικευμένης γνώσης, ενώ οι διαθέσιμοι λογικοί πόροι (LUTs, flip-flops) είναι πεπερασμένοι και χρειάζονται προσεκτική διαχείριση. Επιπλέον, το κόστος για την προμήθεια πλακετών FPGA και του σχετικού λογισμικού συνήθως υπερβαίνει εκείνο απλών μικροελεγκτών, καθιστώντας την υλοποίηση πιο απαιτητική από πλευράς εξοπλισμού.

1.3.5 Συγκριτική ανάλυση

Η βιβλιογραφία επισημαίνει ότι οι μικροελεγκτές (MCU) και οι μικροεπεξεργαστές (MPU) βασίζονται σε σειριακή εκτέλεση προγραμμάτων, γεγονός που τους καθιστά ιδανικούς για απλές εφαρμογές ελέγχου, αλλά περιορίζει την απόδοσή τους όταν απαιτείται υψηλός βαθμός παραλληλίας ή αυστηρή χρονική προβλεψιμότητα. Στο “Field programmable gate arrays (FPGAs) vs. microcontrollers: What’s the difference?” της IBM [25], μετά από αναλυτική σύγκριση, σημειώνει ότι οι μικροελεγκτές λειτουργούν με εντολές που εκτελούνται διαδοχικά με μη προβλέψιμες χρονικές καθυστερήσεις. Αντίθετα, τα FPGA επιτρέπουν την υλοποίηση ειδικά σχεδιασμένων παράλληλων κυκλωμάτων που εκτελούν όλες τις λειτουργίες ταυτόχρονα. Αυτό προσδίδει στα FPGA μικρές καθυστερήσεις, που είναι σημαντικές σε εφαρμογές πραγματικού χρόνου όπως η παραγωγή γραφικών ή η επεξεργασία εισόδων σε παιχνίδια. Από τα “FPGA vs

Microcontroller Understanding the Key Differences and Use Cases” [26] και "FPGA vs. Microcontroller, what to choose” [27], επισημαίνεται ότι οι μικροελεγκτές είναι σχεδιασμένοι για χαμηλό κόστος, ενεργειακή αποδοτικότητα και ευκολία ανάπτυξης, προσφέροντας έτοιμες βιβλιοθήκες και απλό περιβάλλον προγραμματισμού. Ωστόσο, η ίδια αρχιτεκτονική που τους καθιστά «εύκολους», περιορίζει τις δυνατότητές τους σε πολύπλοκα ψηφιακά συστήματα που απαιτούν ταυτόχρονη λειτουργία πολλών μονάδων.

Συνολικά, τα μικροελεγκτικά συστήματα υπερτερούν σε απλότητα, χαμηλό κόστος, μικρή κατανάλωση και ταχύτατη ανάπτυξη εφαρμογών, καθιστώντας τα ιδανικά για εκπαιδευτικά projects ή έργα χαμηλής πολυπλοκότητας. Για παράδειγμα, η υλοποίηση σε Arduino προσφέρει γρήγορη απόκριση, αλλά οι εξαιρετικά περιορισμένοι πόροι, καθιστούν πρακτικά αδύνατη την υλοποίηση παιχνιδιού με υψηλή ανάλυση ή απαιτητική γραφική απόδοση. Οι περισσότερες υλοποιήσεις επικεντρώνονται στη βασική λειτουργικότητα του παιχνιδιού, θυσιάζοντας ποιότητα εικόνας και ομαλότητα κίνησης λόγω των περιορισμών του μικροελεγκτή. Από την άλλη, η υλοποίηση σε μικροεπεξεργαστές τύπου Raspberry Pi, παρουσιάζει ευκολία ανάπτυξης, διευκολύνει την ανάπτυξη παιχνιδιών, επιτρέποντας χρήση υψηλού επιπέδου γλωσσών προγραμματισμού, έτοιμων βιβλιοθηκών γραφικών και ήχου, αλλά υστερεί σε καθυστέρηση εισόδου και δυνατότητα παράλληλης εκτέλεσης της λογικής του παιχνιδιού. Η βιβλιογραφία δείχνει ότι οι υλοποιήσεις σε FPGA υπερτερούν καθαρά σε real-time απαιτήσεις, σταθερότητα χρονισμού, παράλληλη εκτέλεση, δυνατότητα παραγωγής βίντεο και λογικής παιχνιδιού απευθείας σε υλικό. Αυτά τα χαρακτηριστικά καθιστούν την πλατφόρμα FPGA τεχνικά καταλληλότερη για μια υλοποίηση Tetris, που στόχο έχει να προσομοιώσει την αρχιτεκτονική και την αίσθηση των κλασικών arcade συστημάτων, προσφέροντας μεγαλύτερη πιστότητα, απόδοση και πρωτοτυπία.

Η σύγκριση μεταξύ FPGA και υλοποίησης σε υπολογιστή (CPU) αναδεικνύει θεμελιώδεις διαφορές στη φιλοσοφία επεξεργασίας και στον τρόπο εκτέλεσης του προγράμματος. Σύμφωνα με αναφορές της Reflex CES όπως “Why are FPGAs faster than CPU” [30] και “Why use FPGA instead of CPU” [31], τα FPGA υπερέχουν σημαντικά σε εφαρμογές που απαιτούν παράλληλη εκτέλεση, επειδή η λογική του συστήματος υλοποιείται σε εξειδικευμένο hardware και όχι σειριακά από έναν επεξεργαστή. Η διαφορά αυτή επιτρέπει στα FPGA να είναι ταχύτερα σε εξειδικευμένα tasks και να αποδίδουν χρονικά με ακρίβεια, χωρίς καθυστερήσεις, που συχνά εμφανίζονται σε CPUs λόγω του λειτουργικού συστήματος. Σχετικά με αναφορά που κάνει η LogicMadness στο “FPGAs

vs processor" [32], επιβεβαιώνει ότι τα FPGA έχουν χαμηλότερη κατανάλωση, υπερισχύουν σε επεξεργαστική ταχύτητα για εξειδικευμένα tasks και παρέχουν χαμηλότερη καθυστέρηση σε εργασίες όπως η επεξεργασία σήματος και η ανάλυση δεδομένων σε πραγματικό χρόνο. Από την άλλη, οι επεξεργαστές (CPUs) παραμένουν πιο ευέλικτοι και ισχυροί σε εφαρμογές γενικού σκοπού. Από την FPGAkey στο "Where is FPGA stronger than traditional CPU" [33], τονίζει ότι το κύριο πλεονέκτημα των FPGAs σε data centers είναι η σταθερή και εξαιρετικά χαμηλή καθυστέρηση, κάτι που τα καθιστά κατάλληλα για εργασίες συνεχούς ροής, που απαιτούν υψηλή υπολογιστική ισχύ και για εργασίες με υψηλές απαιτήσεις επικοινωνίας. Τέλος, η μελέτη "Performance Evaluation of FPGA, GPU, and CPU in FIR Filter Implementation for Semiconductor-Based Systems" [34] αποδεικνύει ότι το FPGA είναι 27 φορές ταχύτερο από την CPU, γεγονός που το καθιστά κατάλληλο για εργασίες επεξεργασίας σήματος με χαμηλή καθυστέρηση και παράλληλα με πολύ χαμηλότερη κατανάλωση ενέργειας. Συμπερασματικά, παρόλο που ένας υπολογιστής προσφέρει μεγαλύτερη ισχύ και δυνατότητες πολυπλοκότητας, η FPGA υλοποίηση προσφέρει μοναδικά πλεονεκτήματα σε real-time απαιτήσεις, ακρίβεια χρονισμού και πιστότητα προς τη φιλοσοφία των κλασικών ηλεκτρονικών παιχνιδιών.

Τα συμπεράσματα της συγκριτικής μελέτης δείχνουν ότι, παρόλο που υπάρχουν πολλές υλοποιήσεις ρετρό παιχνιδιών σε μικροελεγκτές και άλλες πλατφόρμες, οι περισσότερες περιορίζονται από το διαθέσιμο υλικό τους, τόσο σε επίπεδο γραφικών όσο και σε δυνατότητες πραγματικού χρόνου. Αντιθέτως, οι υλοποιήσεις που βασίζονται σε FPGA επιτυγχάνουν υψηλότερη απόδοση, με ακριβή χρονισμό, δυνατότητα παράλληλης εκτέλεσης και μεγαλύτερη ευελιξία στον σχεδιασμό του συστήματος εξόδου εικόνας. Η επιλογή, επομένως, μιας FPGA πλατφόρμας για την ανάπτυξη του Tetris δεν αποτελεί μόνο τεχνικά ισχυρή λύση, αλλά και σαφή διαφοροποίηση σε σχέση με τις πιο απλοποιημένες προσεγγίσεις.

Η παρούσα εργασία διαφοροποιείται σημαντικά από άλλες παρόμοιες υλοποιήσεις, καθώς εστιάζει στη δημιουργία ενός συστήματος με απλοποιημένη αλλά πλήρως λειτουργική αρχιτεκτονική. Σε αντίθεση με αρκετές προηγούμενες μελέτες που χρησιμοποιούν για είσοδο πληκτρολόγια, UART, ή ειδικά game controllers, η συγκεκριμένη υλοποίηση ενσωματώνει χειριστήριο και πλήκτρα τύπου arcade, τα οποία συνδέονται απευθείας στα GPIO pins της πλακέτας, εξαλείφοντας την ανάγκη για ενδιάμεσους ελεγκτές, αποκωδικοποιητές ή επιπλέον λογισμικό. Η υλική κατασκευή του arcade χειριστηρίου επιτρέπει την επαναλαμβανόμενη χρήση χωρίς φθορές, κάτι που καθιστά την

αλληλεπίδραση πιο ρεαλιστική και αυθεντική, ιδιαίτερα για ρετρό παιχνίδια που απαιτούν γρήγορες και ακριβείς κινήσεις. Παράλληλα, η έξοδος της εικόνας υλοποιείται αποκλειστικά μέσω HDMI, σε αντίθεση με άλλες υλοποιήσεις που βασίζονται σε VGA, χαμηλότερη ανάλυση ή ακόμη και απλουστευμένη απεικόνιση μέσω LEDs. Η έξοδος σε HDMI επιτρέπει υψηλότερη ανάλυση και καθαρότερη απεικόνιση γραφικών, υποστηρίζοντας σύγχρονες οθόνες και παρέχοντας καλύτερη εμπειρία παιχνιδιού.

Ο συνδυασμός αυτών των στοιχείων, καθιστά την εργασία μια ολοκληρωμένη υλοποίηση, που προσφέρει ρετρό εμπειρία παιχνιδιού με σύγχρονη ποιότητα και αλληλεπίδραση χρήστη, κάτι που απουσιάζει από τις περισσότερες παρόμοιες υλοποιήσεις.

1.3.6 Υλοποίηση άλλων κλασικών παιχνιδιών σε FPGA

Η υλοποίηση ρετρό παιχνιδιών σε FPGA αποτελεί έναν συνδυασμό εκπαίδευσης στην ψηφιακή σχεδίαση και αναβίωσης κλασικών παιχνιδιών. Τα πρώτα βήματα σε αυτόν τον χώρο ξεκινούν με απλές εφαρμογές όπως το FPGA **Pong** σε Basys 3 με Verilog, το οποίο παρέχει AI αντίπαλο, VGA έξοδο και έλεγχο ταχύτητας, αποτελώντας παράλληλα ένα εύχρηστο εκπαιδευτικό tutorial [35]. Στη συνέχεια, η ανάπτυξη παιχνιδιών όπως το **Snake** σε Zybo Zynq-7000 [36] επεκτείνει τις δυνατότητες με VGA, scoring, reset και testbench. Καθώς τα projects γίνονται πιο σύνθετα, τα **Space Invaders** σε Zedboard και Papilio Arcade εισάγουν multiplayer λειτουργικότητα, VGA 640x480 και χρήση αυθεντικών ROMs, αναπαριστώντας πιο πιστά την εμπειρία arcade [38], [39]. Παράλληλα, οι εκδόσεις του **Pac-Man** για NEXYS4 DDR και η Verilog core υλοποίηση σε Xilinx FPGA αναπαράγουν arcade hardware με πλήρη έλεγχο, υποστήριξη ROM και περιφερειακών [37]. Τέλος, υλοποιήσεις όπως το VerilogBoy[40] και το **Game Boy** σε Spartan 6 ή ML505 [41], αναδεικνύουν τις δυνατότητες των FPGA να εκτελούν commercial ROMs με σωστή αναπαραγωγή και υψηλή ανάλυση οθόνης. Μέσα από αυτήν την πορεία, η ανάπτυξη παιχνιδιών αναδεικνύουν τις δυνατότητες των FPGA, όχι μόνο ως εργαλείο εκπαίδευσης, αλλά και ως πλατφόρμα για πιστή αναπαραγωγή και επέκταση κλασικών παιχνιδιών.

1.4 Στόχοι και αναμενόμενα αποτελέσματα

Κεντρικός στόχος της εργασίας είναι η σχεδίαση και υλοποίηση ενός ολοκληρωμένου παιχνιδιού Tetris σε FPGA, με τη χρήση εξωτερικού χειριστηρίου και οθόνης HDMI. Για την επίτευξη αυτού του στόχου τίθενται επιμέρους τεχνικοί, λειτουργικοί και εκπαιδευτικοί στόχοι:

- Κατανόηση εννοιών ψηφιακής σχεδίασης και υλοποίηση ενός πλήρους συστήματος, χρησιμοποιώντας γλώσσες περιγραφής υλικού HDL (Verilog/VHDL).
- Εξοικείωση με τη χρήση FPGA, συμπεριλαμβανομένων της σύνθεσης κυκλωμάτων, του χρονισμού και της διαχείρισης πόρων.
- Σχεδιασμός πεπερασμένων μηχανών καταστάσεων (FSM-Finite State Machines) για τη διαχείριση της λογικής του παιχνιδιού.
- Εφαρμογή μηχανισμού ανίχνευσης συγκρούσεων, ελέγχου πλήρωσης γραμμών και ενημέρωσης της βαθμολογίας.
- Μελέτη και εφαρμογή του πρωτοκόλλου HDMI, συμπεριλαμβανομένων της κωδικοποίησης TMDS, των σημάτων χρονισμού, και της μετάδοσης δεδομένων RGB για απεικόνιση σε υψηλή ανάλυση.
- Εκμάθηση της διασύνδεσης με περιφερειακά εισόδου, όπως κουμπιά ή joystick.
- Πραγματοποίηση προσομοιώσεων και δοκιμών για τον έλεγχο της σωστής λειτουργίας κάθε υποσυστήματος.
- Πειραματική αξιολόγηση του τελικού συστήματος ως προς την ταχύτητα, την απόκριση εισόδου και την ποιότητα απεικόνισης.

Αναμενόμενο αποτέλεσμα είναι ένα πλήρως διαδραστικό παιχνίδι με τα βασικά χαρακτηριστικά του κλασικού Tetris, εκτελούμενο σε πραγματικό χρόνο. Θα εκτελείται αυτόνομα στην πλακέτα FPGA με ακριβή χρονισμό πτώσης και άμεση απόκριση στις εντολές χρήστη χωρίς καθυστερήσεις, και θα απεικονίζει το αποτέλεσμα σε εξωτερική οθόνη μέσω HDMI.

1.5 Δομή εργασίας

Η παρούσα εργασία οργανώνεται σε 6 κεφάλαια, τα οποία παρουσιάζουν το θεωρητικό υπόβαθρο, τη σχεδίαση, την υλοποίηση και τα αποτελέσματα του συστήματος, καθώς και μελλοντικές βελτιώσεις. Αναλυτικότερα, τα κεφάλαια είναι:

- **Κεφάλαιο 1**

Παρουσιάζεται το αντικείμενο και ο σκοπός της εργασίας, τα κίνητρα επιλογής του συγκεκριμένου παιχνιδιού, οι συγκεκριμένοι στόχοι που τέθηκαν καθώς και τα αναμενόμενα αποτελέσματα της υλοποίησης. Επιπλέον, περιλαμβάνεται επισκόπηση της σχετικής ερευνητικής βιβλιογραφίας και σύντομη παρουσίαση συναφών εργασιών, με στόχο να τεκμηριωθεί η αναγκαιότητα και η πρωτοτυπία της παρούσας υλοποίησης. Τέλος, περιγράφεται η δομή της εργασίας, παρέχοντας στον αναγνώστη μια συνοπτική επισκόπηση των επόμενων κεφαλαίων.

- **Κεφάλαιο 2**

Αναλύεται το θεωρητικό υπόβαθρο σχετικά με τον ορισμό, τα χαρακτηριστικά, τις εφαρμογές και τους βασικούς τύπους των FPGA καθώς και την αρχιτεκτονική και την τυπική ροή ανάπτυξης τους. Επιπλέον, παρουσιάζεται ο ρόλος των γλωσσών περιγραφής υλικού HDL, ενώ παράλληλα γίνεται αναφορά σε εργαλεία ανάπτυξης όπως Quartus και ModelSim. Στη συνέχεια, γίνεται αναφορά στα χειριστήρια τύπου arcade, στους τύπους εισόδων, στις τεχνικές αντιμετώπισης αναπήδησης σημάτων και στη χαρτογράφηση των σημάτων ελέγχου. Ακόμη, εξετάζεται η διεπαφή HDMI, αναλύοντας το πρωτόκολλο μετάδοσης, τους απαιτούμενους χρονισμούς και τις προδιαγραφές ανάλυσης που σχετίζονται με την παραγωγή σήματος βίντεο. Τέλος, παρουσιάζεται το παιχνίδι Tetris, με συνοπτικό ιστορικό, βασικά χαρακτηριστικά και γενική περιγραφή των κανόνων του.

- **Κεφάλαιο 3**

Παρουσιάζεται η ανάλυση απαιτήσεων του συστήματος, αναλύοντας τις προδιαγραφές του παιχνιδιού, καθώς και τις απαιτήσεις υλικού, μνήμης, χρονισμού και εισόδου/εξόδου. Στη συνέχεια, γίνεται αναφορά σε παρόμοιες υλοποιήσεις και στον προσδιορισμό των απαιτήσεων τους. Τέλος, γίνεται σύγκριση των δυνατοτήτων και των εργαλείων και αξιολόγηση για την τελική επιλογή πλακέτας.

- **Κεφάλαιο 4**

Περιγράφονται αναλυτικά τα βήματα σχεδίασης και ανάπτυξης του παιχνιδιού Tetris στην επιλεγμένη πλακέτα. Περιλαμβάνει την αρχιτεκτονική του συστήματος, τη διαχείριση των εισόδων και εξόδων, τον σχεδιασμό των λογικών μονάδων και την ολοκλήρωση του παιχνιδιού. Επίσης, αναλύονται οι τεχνικές αποφάσεις, τα χρησιμοποιούμενα εργαλεία ανάπτυξης και τα βήματα προσομοίωσης και ελέγχου.

- **Κεφάλαιο 5**

Παρουσιάζονται τα αποτελέσματα της υλοποίησης, η λειτουργικότητα του παιχνιδιού και η απόδοση της πλακέτας. Αξιολογούνται, συγκεκριμένα, η ταχύτητα εκτέλεσης, η αξιοπιστία του συστήματος και η χρηστικότητα του παιχνιδιού.

- **Κεφάλαιο 6**

Συνοψίζονται τα κύρια συμπεράσματα και προτείνονται δυνατότητες βελτίωσης και πιθανές επεκτάσεις του έργου.

Στο κεφάλαιο που ακολουθεί, αναλύεται το θεωρητικό υπόβαθρο σχετικά με τα FPGA, το χειριστήριο τύπου arcade, τη διεπαφή HDMI και παρουσιάζεται συνοπτικά το παιχνίδι Tetris.

.

2. Θεωρητικό υπόβαθρο

Η υλοποίηση ενός παιχνιδιού σε FPGA απαιτεί την κατανόηση διαφόρων τεχνολογιών και εννοιών που σχετίζονται με τον σχεδιασμό ψηφιακών συστημάτων. Στο παρόν κεφάλαιο παρουσιάζονται τα βασικά θεωρητικά στοιχεία που υποστηρίζουν την ανάπτυξη της εργασίας. Αρχικά γίνεται αναφορά στην τεχνολογία των FPGA και στις δυνατότητες που αυτά προσφέρουν, στη συνέχεια αναλύονται τα χειριστήρια τύπου arcade που αποτελούν το βασικό μέσο εισόδου, καθώς και η διεπαφή HDMI που χρησιμοποιείται για την απεικόνιση σε οθόνη. Τέλος, παρουσιάζονται τα βασικά χαρακτηριστικά και ο τρόπος λειτουργίας του παιχνιδιού Tetris, το οποίο αποτελεί τον πυρήνα της εφαρμογής.

2.1 Εισαγωγή στα FPGA

2.1.1 Τι είναι τα FPGA

Τα Field Programmable Gate Arrays (FPGAs) ή συστοιχία επιτόπια προγραμματιζόμενων πυλών, είναι ψηφιακά ολοκληρωμένα κυκλώματα (Intergrated Circuits - ICs) που περιέχουν προγραμματιζόμενα μπλοκ λογικής (Configurable Logic Blocks – CLBs) και τις μεταξύ τους διαμορφώσιμες διασυνδέσεις [42],[43]. Το μέρος του ονόματος FPGA που αναφέρεται στο “field programmable” αναφέρεται στο ότι ο προγραμματισμός του γίνεται «στο πεδίο» που σημαίνει ότι μπορούν να προγραμματιστούν από τον χρήστη και μετά την παραγωγή τους ανάλογα με τις απαιτήσεις [48]. Η Εικόνα 13 δείχνει ενδεικτικά τη δομή της πλακέτας FPGA, της DE23-Lite της εταιρείας terasic.

Ανάλογα με τον τρόπο υλοποίησής τους, κάποια FPGA μπορεί να προγραμματιστούν μόνο μία φορά (one-time programmable - OTP), ενώ άλλα μπορούν να προγραμματιστούν περισσότερες από μια φορές. Αν μια συσκευή μπορεί να προγραμματιστεί ενώ παραμένει εγκατεστημένη σε ένα ανώτερο σύστημα, αυτό ονομάζεται in-system programmable (ISP) [42].

Τα FPGA, που εμφανίστηκαν για πρώτη φορά στα μέσα της δεκαετίας του 1980, αποτελούν σήμερα μία από τις πιο σημαντικές τεχνολογίες στον ψηφιακό σχεδιασμό χάρη στην ευελιξία και τον επαναπρογραμματισμό τους. Επιτρέπουν την εκτέλεση λειτουργιών σε επίπεδο υλικού με παράλληλη επεξεργασία, προσφέροντας υψηλή ταχύτητα και απόδοση σε σύγκριση με τους παραδοσιακούς μικροεπεξεργαστές. Τα χαρακτηριστικά τους, όπως η χαμηλή πολυπλοκότητα, η δυνατότητα σχεδιασμού μεγάλου όγκου και η

προγραμματιζόμενη λειτουργικότητα, τα καθιστούν χρήσιμα τόσο στην ακαδημαϊκή έρευνα όσο και στη βιομηχανία. Σήμερα χρησιμοποιούνται σε πληθώρα εφαρμογών, όπως τηλεπικοινωνίες, επεξεργασία ψηφιακών σημάτων (DSP), βιομηχανικά συστήματα ελέγχου, ιατρικές συσκευές, αυτοκινητοβιομηχανία, ραντάρ, δορυφορικές επικοινωνίες, κρυπτογράφηση και επεξεργασία εικόνας και βίντεο [44],[45],[46],[47]..

Εικόνα 13 DE23-Lite [13]

2.1.2 Αρχιτεκτονική FPGA

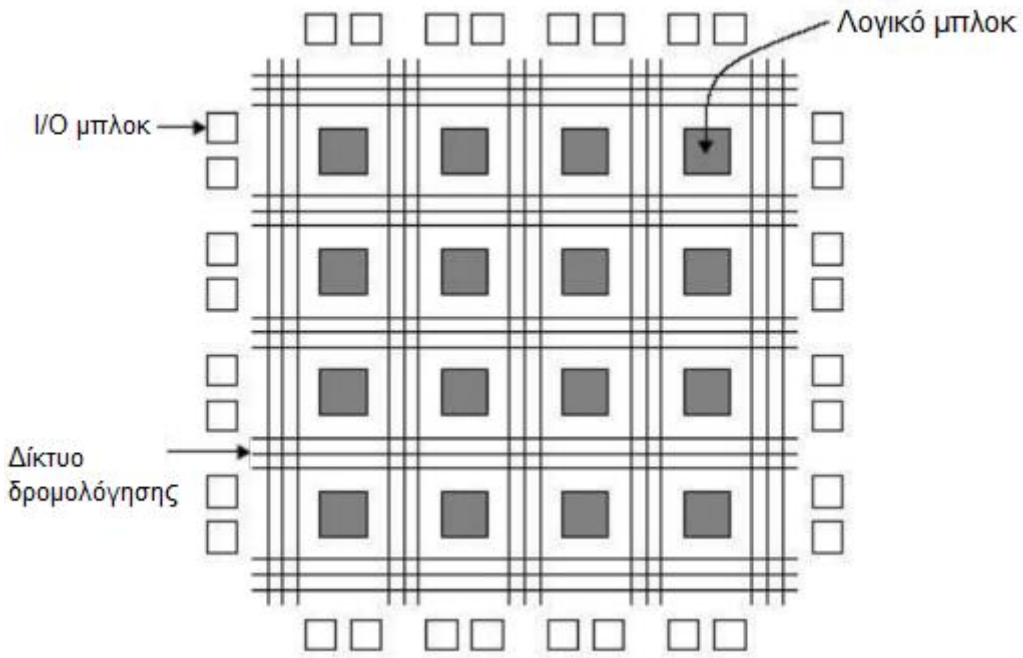
Η αρχιτεκτονική ενός FPGA είναι μια σύνθετη δομή που επιτρέπει την επαναπρογραμματιζόμενη υλοποίηση ψηφιακών κυκλωμάτων. Στον πυρήνα της αποτελείται από τρία βασικά συστατικά [50]:

- **Προγραμματιζόμενα λογικά μπλοκ** (Configurable Logic Blocks - CLBs), που υλοποιούν βασικές λογικές συναρτήσεις.
- **Δίκτυο δρομολόγησης** (routing network), που επιτρέπει τη σύνδεση των μπλοκ μεταξύ τους.
- **Μπλοκ εισόδων/εξόδων** (I/O blocks), που συνδέονται με τα λογικά μπλοκ μέσω του δικτύου διασύνδεσης και πραγματοποιούν συνδέσεις με το εξωτερικό κύκλωμα ή τις περιφερειακές συσκευές.

Σε πιο εξελιγμένα FPGA, προστίθενται εξειδικευμένες μονάδες, όπως μνήμες, μπλοκ Digital Signal Processing -DSP και συστήματα διαχείρισης ρολογιού, τα οποία ενισχύουν σημαντικά την απόδοση και τις δυνατότητες της συσκευής [49].

[13] Η εικόνα είναι από τη διεύθυνση https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=44&No=1383&PartNo=1#contents

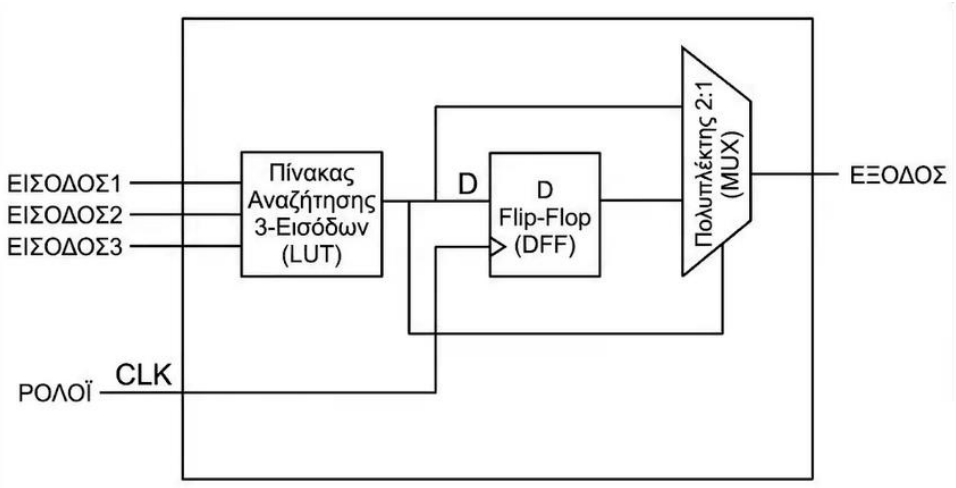
Μια τυπική αρχιτεκτονική παρουσιάζεται στην Εικόνα 14, όπου τα CLBs είναι διατεταγμένα σε ένα δυσδιάστατο πλέγμα (two-dimensional grid) και διασυνδέονται μέσω προγραμματιζόμενων πόρων δρομολόγησης. Τα I/O blocks είναι τοποθετημένα στην περιφέρεια του πλέγματος και επίσης συνδέονται με το προγραμματιζόμενο δίκτυο διασύνδεσης. Αυτή η συμμετρική και επαναλαμβανόμενη διάταξη των λογικών μπλοκ και των διασυνδέσεων επιτρέπει στο FPGA να είναι παραμετροποιήσιμο, αφού το κύκλωμα μπορεί να επαναπρογραμματιστεί για διαφορετικές λειτουργίες. Επιπλέον, η δομή πλέγματος υποστηρίζει παράλληλη επεξεργασία, καθώς πολλά λογικά μπλοκ μπορούν να λειτουργούν ταυτόχρονα, αυξάνοντας σημαντικά την ταχύτητα εκτέλεσης σε σχέση με σειριακά συστήματα.



Εικόνα 14 Αρχιτεκτονική FPGA

Θεμελιώδες δομικό στοιχείο αποτελεί το **προγραμματιζόμενο λογικό μπλοκ (CLB)**, το οποίο συχνά αποκαλείται και Logic Block (LB) ή Logic Element (LE) ή Logic Cell (LC), ανάλογα με τον κατασκευαστή [47]. Κάθε CLB συνήθως αποτελείται από έναν πίνακα αναζήτησης (Lookup Table – LUT), έναν καταχωρητή (D flip-flop) και πολυπλέκτες (Multiplexers) που επιτρέπουν την επιλογή μεταξύ συνδυαστικής ή ακολουθιακής λειτουργίας (Εικόνα 15). Οι LUTs έχουν τη δυνατότητα να υλοποιούν οποιαδήποτε συνδυαστική λογική συνάρτηση με περιορισμένο αριθμό εισόδων (4, 6 ή 8). Ο καταχωρητής χρησιμοποιείται την αποθήκευση πληροφορίας κατάστασης (state) και οι

πολυπλέκτες για να επιλέγουν την έξοδο δεδομένων είτε από τη συνδυαστική είτε από τη ακολουθιακή λογική.



Εικόνα 15 Προγραμματιζόμενο λογικό μπλοκ - CLB

Το **δίκτυο δρομολόγησης** (routing network) σε ένα FPGA επιτρέπει τη μεταφορά σημάτων μεταξύ των λογικών μπλοκ (CLBs), των μνημών, των αριθμητικών μονάδων και των εισόδων/εξόδων της συσκευής. Αποτελείται από ένα πλέγμα οριζόντιων και κάθετων γραμμών, οι οποίες διασταυρώνονται σε κόμβους που περιλαμβάνουν προγραμματιζόμενους διακόπτες (Programmable Interconnect Points – PIPs). Οι διακόπτες αυτοί επιτρέπουν την ενεργοποίηση ή απενεργοποίηση συνδέσεων μεταξύ των γραμμών, σχηματίζοντας έτσι τα μονοπάτια επικοινωνίας ανάμεσα στα διάφορα στοιχεία του FPGA. Στα σημεία όπου το δίκτυο συνδέεται με τα CLBs υπάρχουν προγραμματιζόμενα πλαίσια σύνδεσης (programmable connection boxes), ενώ στις διασταυρώσεις των καναλιών βρίσκονται τα προγραμματιζόμενα πλαίσια μεταγωγής (programmable switch boxes), που καθορίζουν την πορεία κάθε σήματος. Έτσι, τα σήματα μπορούν να μετακινηθούν από ένα CLB σε οποιοδήποτε άλλο σημείο του FPGA, είτε σε τοπικό, είτε σε περιφερειακό, είτε σε καθολικό επίπεδο [42], [50].

Τα **μπλοκ εισόδων/εξόδων** (Input/Output Blocks – I/O Blocks ή IOBs) αποτελούν ένα από τα βασικά δομικά στοιχεία της αρχιτεκτονικής ενός FPGA, καθώς αποτελούν τη σύνδεση μεταξύ του εσωτερικού προγραμματιζόμενου λογικού πυρήνα και του εξωτερικού περιβάλλοντος. Τα I/O blocks βρίσκονται περιμετρικά του ολοκληρωμένου κυκλώματος και συνδέονται με τα λογικά μπλοκ (CLBs) και τα υπόλοιπα υποσυστήματα μέσω του δικτύου δρομολόγησης. Ο κύριος ρόλος τους είναι να προσαρμόζουν ηλεκτρικά, χρονικά και λειτουργικά τα σήματα του FPGA ώστε να είναι συμβατά με εξωτερικά κυκλώματα, αισθητήρες, μνήμες και περιφερειακές συσκευές. Με αυτόν τον τρόπο, τα I/O blocks αποτελούν κρίσιμο κρίκο μεταξύ της εσωτερικής λογικής επεξεργασίας και του πραγματικού φυσικού κόσμου, καθιστώντας τα FPGA ευέλικτα και κατάλληλα για πλήθος εφαρμογών.

Συμπερασματικά, η αρχιτεκτονική των FPGA βασίζεται σε μια ιδιαίτερα ευέλικτη και παραμετροποιήσιμη δομή, η οποία συνδυάζει προγραμματιζόμενα λογικά μπλοκ (CLBs), ιεραρχικά οργανωμένο δίκτυο δρομολόγησης και μπλοκ εισόδων/εξόδων (I/O blocks), επιτρέποντας την υλοποίηση σύνθετων ψηφιακών συστημάτων σε ένα ενιαίο ολοκληρωμένο κύκλωμα. Η στενή αλληλεπίδραση μεταξύ των δομικών αυτών στοιχείων καθιστά δυνατή την προσαρμογή της λειτουργικότητας του FPGA στις εκάστοτε απαιτήσεις της εφαρμογής, τόσο σε επίπεδο λογικής επεξεργασίας όσο και σε επίπεδο επικοινωνίας με το εξωτερικό περιβάλλον.

2.1.3 Αρχιτεκτονική επιλεγμένης πλακέτας ανάπτυξης DE23-Lite

Η DE23-Lite είναι μια πλακέτα ανάπτυξης FPGA της Terasic, σχεδιασμένη για εκπαιδευτικά και ερευνητικά έργα σε ψηφιακή λογική, ενσωματωμένα συστήματα και εφαρμογές επεξεργασίας εικόνας και βίντεο, η οποία έχει επιλεχθεί για την υλοποίηση του συγκεκριμένου έργου. Αποτελεί εξέλιξη της DE10-Lite, προσφέροντας αναβαθμισμένο FPGA, περισσότερα I/O και ενσωματωμένα περιφερειακά, διατηρώντας παράλληλα χαμηλό κόστος και ευκολία στη χρήση [52]. Η αρχιτεκτονική της πλακέτας οργανώνεται γύρω από το Intel Agilex™ 3 FPGA, το οποίο λειτουργεί ως κεντρικός επεξεργαστικός πυρήνας, ενώ πλαισιώνεται από υποσυστήματα μνήμης, χρονισμού, διεπαφών εισόδου/εξόδου και τροφοδοσίας (Εικόνα 16). Συγκεκριμένα αποτελείται από [53]:

- ***FPGA Agilex 3***

Η πλακέτα διαθέτει το FPGA Altera (Intel) Agilex™ 3, μοντέλο A3CZ135BB18AE7S, το οποίο προσφέρει περίπου 135.110 λογικά στοιχεία και 6.89 Mbit εσωτερικής μνήμης τύπου M20K, καθώς και 368 πολλαπλασιαστές. Το Agilex 3 FPGA παρέχει υψηλή λογική πυκνότητα, υποστήριξη PLL για διαχείριση ρολογιών και βελτιωμένη ενεργειακή αποδοτικότητα, καθιστώντας το κατάλληλο για σύνθετες εφαρμογές ψηφιακής λογικής. Η πλακέτα δεν περιλαμβάνει ενσωματωμένο επεξεργαστή, γεγονός που ενισχύει την ευελιξία για εξειδικευμένους ψηφιακούς σχεδιασμούς.

- ***Μνήμη και αποθήκευση***

Η DE23-Lite διαθέτει εξωτερική SDRAM 64 MB σε δίαυλο 32-bit, η οποία χρησιμοποιείται για buffering δεδομένων, αποθήκευση πλαισίων βίντεο και γενική επεξεργασία σημάτων. Επιπλέον, περιλαμβάνει 128 Mbit QSPI Flash, για αποθήκευση του FPGA bitstream και άλλων δεδομένων ή ρυθμίσεων, επιτρέποντας αυτόματη φόρτωση της διαμόρφωσης κατά την εκκίνηση της πλακέτας.

- ***Είσοδοι/Έξοδοι (I/O)***

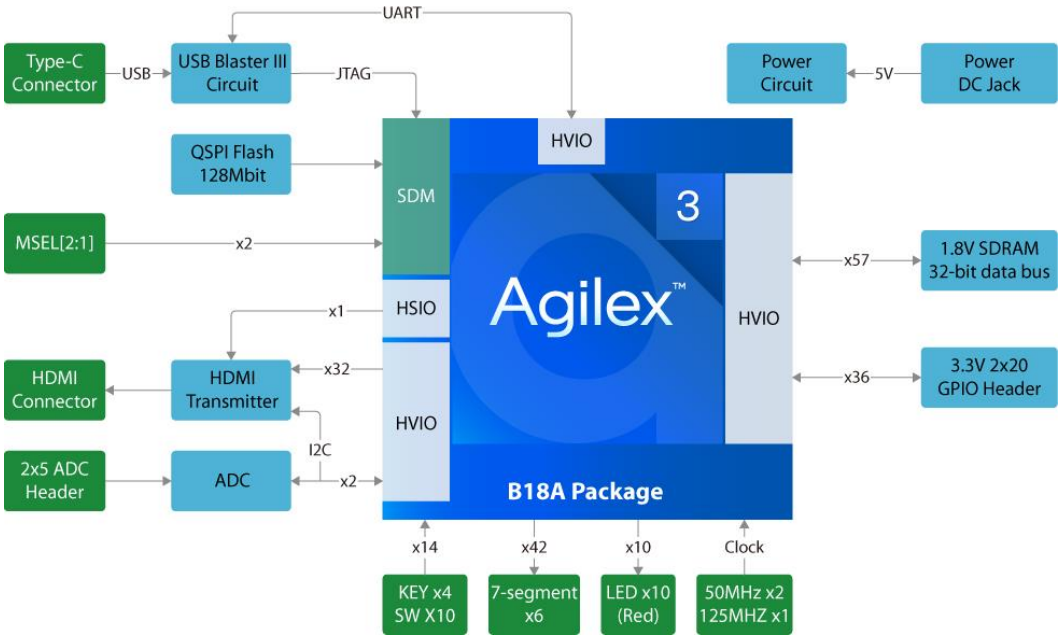
Η πλακέτα προσφέρει πλούσια διασύνδεση I/O, περιλαμβάνοντας δύο headers GPIO 2×20 για γενικές εισόδους/εξόδους, ADC header με 8 κανάλια 12-bit για αισθητήρες, σειριακή επικοινωνία μέσω USB Type-C και υποστήριξη HDMI output μέσω του ADV7513 driver για βίντεο έως Full HD. Επιπλέον, διαθέτει διακόπτες, κουμπιά, LED και 7-segment displays, τα οποία διευκολύνουν τη δοκιμή λογικών κυκλωμάτων και την απεικόνιση αποτελεσμάτων χωρίς εξωτερικό εξοπλισμό.

- ***Τροφοδοσία***

Η τροφοδοσία της DE23-Lite γίνεται είτε μέσω 5 V DC τροφοδοτικού είτε μέσω USB Type-C.

- ***Εργαλεία Ανάπτυξης***

Η πλακέτα ενσωματώνει κύκλωμα USB Blaster III, επιτρέποντας προγραμματισμό και debugging απευθείας από τον υπολογιστή. Η ανάπτυξη λογισμικού και υλοποίηση των σχεδίων μπορεί να γίνει μέσω του Quartus Prime Pro της Intel, για την οποία παρέχεται δωρεάν άδεια με την αγορά της πλακέτας. Τα εργαλεία προσομοίωσης που μπορούν να χρησιμοποιηθούν δωρεάν είναι το ModelSim Intel FPGA Starter Edition καθώς και το Questa FSE που διατίθεται μαζί με το Quartus Prime Pro.



Εικόνα 16 Block diagram DE23-Lite[14]

Συμπερασματικά, η αρχιτεκτονική της πλακέτας DE23-Lite χαρακτηρίζεται από έναν ισορροπημένο και λειτουργικό σχεδιασμό, ο οποίος συνδυάζει τη σύγχρονη τεχνολογία του FPGA Intel Agilex 3 με απαραίτητα υποσυστήματα μνήμης, χρονισμού, εισόδων/εξόδων και τροφοδοσίας, επιτρέποντας την υλοποίηση ευέλικτων και επεκτάσιμων ψηφιακών εφαρμογών. Επιπλέον, η υποστήριξη από τα σύγχρονα εργαλεία ανάπτυξης της Intel και η ύπαρξη ενσωματωμένων διεπαφών καθιστούν την DE23-Lite μια αξιόπιστη και κατάλληλη επιλογή για ακαδημαϊκή χρήση και πειραματική ανάπτυξη εφαρμογών, προσφέροντας ένα ολοκληρωμένο περιβάλλον σχεδίασης.

[14]Η εικόνα είναι από τη διεύθυνση https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=44&No=1383&PartNo=2#contents

2.1.4 Ροή ανάπτυξης σε FPGA

Η ροή ανάπτυξης αποτελεί τη συστηματική διαδικασία μέσω της οποίας μια λειτουργική ιδέα μετατρέπεται σε υλοποιήσιμο ψηφιακό κύκλωμα σε επαναπρογραμματιζόμενη λογική. Πρόκειται για μια επαναληπτική ακολουθία σταδίων που εξασφαλίζει τη λειτουργική ορθότητα, τη χρονική αξιοπιστία και τη βέλτιστη αξιοποίηση των πόρων του FPGA, όπου τα αποτελέσματα του κάθε ενός μπορούν να οδηγήσουν σε ανασχεδιασμό προηγούμενων βημάτων (Εικόνα 17) [54], [55].

Στο πρώτο στάδιο ανάπτυξης σε FPGA, γίνεται ο ορισμός των απαιτήσεων, όπου καθορίζεται τι λειτουργία θα υλοποιεί το σύστημα, ποιες είναι οι είσοδοι και οι έξοδοι, οι απαιτήσεις σε ταχύτητα, κατανάλωση και χρονισμό, καθώς και η επιλογή της κατάλληλης FPGA πλατφόρμας. Στη συνέχεια ακολουθεί ο αρχιτεκτονικός σχεδιασμός, κατά τον οποίο το σύστημα αναλύεται σε επιμέρους λειτουργικά υποσυστήματα, καθορίζονται οι δομές ελέγχου και οι διεπαφές επικοινωνίας, ώστε να δημιουργηθεί ένα αποδοτικό σχέδιο που θα αποτελέσει τη βάση για την περιγραφή υλικού σε HDL.

Το επόμενο στάδιο της ροής ανάπτυξης είναι η περιγραφή του σχεδιασμού. Σε αυτό το στάδιο, οι λειτουργικές απαιτήσεις του συστήματος εκφράζονται μέσω γλωσσών περιγραφής υλικού (Hardware Description Languages – HDL), όπως η VHDL ή η Verilog. Οι γλώσσες αυτές επιτρέπουν την περιγραφή της δομής και της συμπεριφοράς του κυκλώματος σε επίπεδο καταχωρητών (Register Transfer Level – RTL). Η ακρίβεια και η σαφήνεια της περιγραφής σε αυτό το στάδιο είναι κρίσιμη, καθώς επηρεάζει άμεσα τα επόμενα βήματα της ροής.

Μετά την περιγραφή του σχεδιασμού ακολουθεί η λειτουργική προσομοίωση. Σκοπός του σταδίου αυτού είναι η επαλήθευση της λογικής ορθότητας του HDL κώδικα, χωρίς να λαμβάνονται υπόψη φυσικοί περιορισμοί υλοποίησης, όπως καθυστερήσεις ή χρονισμός. Μέσω ειδικών δοκιμών (testbenches), ελέγχεται η συμπεριφορά του κυκλώματος για ένα σύνολο εισόδων και σεναρίων λειτουργίας. Η λειτουργική προσομοίωση θεωρείται θεμελιώδες στάδιο, καθώς επιτρέπει την ανίχνευση και διόρθωση σφαλμάτων σε πρώιμο στάδιο, μειώνοντας σημαντικά το συνολικό κόστος ανάπτυξης.

Αφού επιβεβαιωθεί η λειτουργική ορθότητα του σχεδιασμού, ακολουθεί η σύνθεση. Κατά τη σύνθεση, ο HDL κώδικας μετατρέπεται σε ένα χαμηλότερου επιπέδου λογικό κύκλωμα, γνωστό ως netlist. Η netlist περιγράφει το κύκλωμα σε όρους βασικών λογικών στοιχείων που είναι διαθέσιμα στο FPGA. Στο στάδιο αυτό, τα εργαλεία σύνθεσης

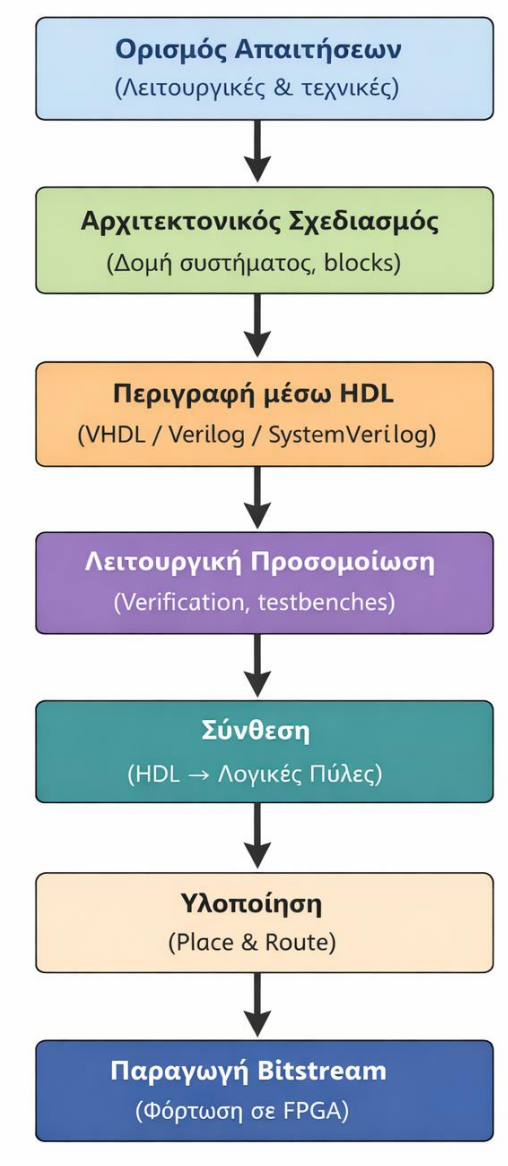
εφαρμόζουν βελτιστοποιήσεις με στόχο τη μείωση της χρήσης πόρων, τη βελτίωση της ταχύτητας λειτουργίας ή τη μείωση της κατανάλωσης ισχύος, σύμφωνα με τις απαιτήσεις του σχεδιασμού.

Το επόμενο στάδιο είναι η υλοποίηση, η οποία περιλαμβάνει τις διαδικασίες της τοποθέτησης και της δρομολόγησης. Σε αυτό το στάδιο, τα λογικά στοιχεία της netlist αντιστοιχίζονται σε συγκεκριμένες φυσικές θέσεις του FPGA και συνδέονται μεταξύ τους μέσω του διαθέσιμου δικτύου διασύνδεσης. Η υλοποίηση επηρεάζει καθοριστικά την απόδοση του κυκλώματος, καθώς οι φυσικές αποστάσεις και οι διαδρομές των σημάτων καθορίζουν τις καθυστερήσεις. Παράλληλα, πραγματοποιείται ανάλυση χρονισμού, ώστε να διασφαλιστεί ότι το κύκλωμα ικανοποιεί τους καθορισμένους χρονικούς περιορισμούς.

Με την επιτυχή ολοκλήρωση της υλοποίησης, παράγεται το αρχείο διαμόρφωσης (bitstream). Το αρχείο αυτό αποτελεί το τελικό προϊόν της ροής ανάπτυξης και περιέχει όλες τις απαραίτητες πληροφορίες για τη διαμόρφωση του FPGA σύμφωνα με τον σχεδιασμό. Είναι εξαρτώμενο από τη συγκεκριμένη αρχιτεκτονική και τον κατασκευαστή του FPGA και δεν μπορεί να χρησιμοποιηθεί σε διαφορετική πλατφόρμα. Η παραγωγή του bitstream σηματοδοτεί την ολοκλήρωση της σχεδίασης σε επίπεδο εργαλείων λογισμικού.

Το τελευταίο στάδιο της ροής ανάπτυξης είναι ο προγραμματισμός και ο έλεγχος στο φυσικό υλικό. Το bitstream φορτώνεται στο FPGA μέσω κατάλληλων διεπαφών και ακολουθεί η αξιολόγηση της λειτουργίας του συστήματος σε πραγματικές συνθήκες. Σε περίπτωση που εντοπιστούν αποκλίσεις από την αναμενόμενη συμπεριφορά, η ροή ανάπτυξης επαναλαμβάνεται, ξεκινώντας από προηγούμενα στάδια.

Συνοψίζοντας, η ροή ανάπτυξης στα FPGA συνιστά μια δομημένη και επαναληπτική διαδικασία, η οποία διασφαλίζει ότι ένας ψηφιακός σχεδιασμός μετατρέπεται με αξιόπιστο τρόπο σε υλοποιήσιμο κύκλωμα. Η κατανόηση και η ορθή εφαρμογή της ροής αυτής αποτελεί προϋπόθεση για την επιτυχή ανάπτυξη σύγχρονων ψηφιακών συστημάτων βασισμένων σε επαναπρογραμματιζόμενη λογική.



Εικόνα 17 Ροή ανάπτυξης σε FPGA

2.1.5 Γλώσσες περιγραφής υλικού (HDL)

Οι γλώσσες περιγραφής υλικού (HDL) είναι εξειδικευμένες γλώσσες προγραμματισμού που χρησιμοποιούνται για την περιγραφή της δομής, της συμπεριφοράς και της χρονικής εξέλιξης ψηφιακών συστημάτων [57]. Σε αντίθεση με τις γλώσσες προγραμματισμού λογισμικού, δεν ακολουθούν σειριακή εκτέλεση εντολών, αλλά περιγράφουν την παράλληλη λειτουργία καταχωρητών, συνδυαστικής λογικής και μηχανών πεπερασμένων καταστάσεων. Η χρήση τους ξεκίνησε πειραματικά από τη δεκαετία του 1950, ενώ η ευρεία υιοθέτηση σημειώθηκε μετά το 1985, αρχικά για προσομοίωση και τεκμηρίωση κυκλωμάτων σε επίπεδο RTL (Register Transfer Level). Με την εισαγωγή της λογικής σύνθεσης στις αρχές της δεκαετίας του 1990, οι HDLs μετατράπηκαν σε βασικό εργαλείο αυτοματοποιημένης σχεδίασης [58], καλύπτοντας πλέον ολόκληρη τη ροή ανάπτυξης, από προσομοίωση και επαλήθευση έως υλοποίηση σε FPGA.

Οι δυο κυρίαρχες γλώσσες HDL στη σχεδίαση FPGA είναι η VHDL (VHSIC Hardware Description Language) και η Verilog. Η VHDL ξεχωρίζει για την αυστηρή σύνταξη και το σύστημα τύπων, καθιστώντας την κατάλληλη για εφαρμογές που απαιτούν υψηλή αξιοπιστία και σαφήνεια, ενώ η Verilog προσφέρει πιο συνοπτική σύνταξη και γρήγορη ανάπτυξη, προτιμώμενη σε εμπορικά έργα [59]. Και οι δύο υποστηρίζουν περιγραφή σε επίπεδο RTL (Register Transfer Level), που αποτελεί το βασικό επίπεδο σχεδίασης FPGA. Επιπλέον, η SystemVerilog, εξέλιξη της Verilog, ενσωματώνει αντικειμενοστραφή στοιχεία και προηγμένα εργαλεία επαλήθευσης [60], βρίσκοντας εφαρμογή σε μεγάλα και πολύπλοκα έργα. Τα τελευταία χρόνια έχουν αναπτυχθεί και γλώσσες υψηλού επιπέδου (High-Level Synthesis – HLS), που επιτρέπουν την περιγραφή υλικού μέσω C/C++ ή OpenCL και τη μετατροπή τους σε HDL [61], διευκολύνοντας την ανάπτυξη. Παρότι οι HLS μέθοδοι απλοποιούν τη διαδικασία, δεν αντικαθιστούν πλήρως τις παραδοσιακές HDL σε εφαρμογές που απαιτούν λεπτομερή έλεγχο του υλικού.

Συνολικά, οι HDLs αποτελούν θεμελιώδες στοιχείο της ροής ανάπτυξης FPGA, λειτουργώντας ως ο συνδετικός κρίκος μεταξύ των θεωρητικών απαιτήσεων και της φυσικής υλοποίησης στο επαναπρογραμματιζόμενο υλικό.

2.1.6 Εργαλεία ανάπτυξης

Τα εργαλεία ανάπτυξης FPGA αποτελούν το βασικό λογισμικό περιβάλλον για τον σχεδιασμό και την υλοποίηση ψηφιακών κυκλωμάτων σε επαναπρογραμματιζόμενη λογική, ενσωματώνοντας όλες τις απαραίτητες λειτουργίες, όπως σύνθεση, υλοποίηση, χρονική ανάλυση, προσομοίωση και προγραμματισμό του FPGA.

Στο χώρο αυτό, υπάρχουν πολυάριθμα εργαλεία που καλύπτουν όλες τις φάσεις του κύκλου ανάπτυξης. Μεταξύ των πιο διαδεδομένων περιλαμβάνονται εργαλεία σχεδιασμού όπως το Quartus Prime Pro, το Quartus Prime Standard και το Vivado Design Suite της Xilinx, εργαλεία προσομοίωσης όπως το ModelSim, το ModelSim Intel FPGA Starter Edition και το Vivado Simulator, εργαλεία τυπικής επαλήθευσης όπως το Questa Formal Simulator (Questa FS) και το Cadence JasperGold, καθώς και πλατφόρμες ανάπτυξης υψηλού επιπέδου όπως το Intel HLS Compiler και το Xilinx Vitis. Στην οικογένεια Quartus περιλαμβάνονται και εξειδικευμένα εργαλεία όπως το SignalTap Logic Analyzer για παρακολούθηση σημάτων σε πραγματικό χρόνο, το TimeQuest Timing Analyzer για ανάλυση χρονισμού, το Power Analyzer για εκτίμηση κατανάλωσης ενέργειας, καθώς και το Platform Designer, για τη σύνθεση και διαχείριση συστημάτων υψηλού επιπέδου. Επιπλέον, υπάρχουν εξειδικευμένα εργαλεία ανάλυσης χρονισμού, δημιουργίας βιβλιοθηκών IP και διαχείρισης έργων, όπως το Synplify Pro, το Libero SoC και το Active-HDL.

Για την υλοποίηση του παρόντος έργου επιλέχθηκαν το Quartus Prime Pro και το ModelSim Intel FPGA Starter Edition, εργαλεία ανάπτυξης που είναι συμβατά με την επιλεγμένη πλακέτα ανάπτυξης και προσφέρουν ολοκληρωμένες δυνατότητες σχεδίασης, προσομοίωσης και αποσφαλμάτωσης.

Το Quartus Prime Pro της Intel [62] αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για τον σχεδιασμό και την υλοποίηση ψηφιακών κυκλωμάτων σε FPGA. Το λογισμικό προσφέρει μια σειρά εργαλείων για τον σχεδιασμό, την τοποθέτηση και δρομολόγηση, καθώς και την ανάλυση χρονισμού. Η πλατφόρμα υποστηρίζει γλώσσες περιγραφής υλικού όπως VHDL και Verilog, ενώ παρέχει ενσωματωμένες βιβλιοθήκες IP για επιτάχυνση της ανάπτυξης σύνθετων κυκλωμάτων. Επιπλέον, διαθέτει δυνατότητες αυτόματης βελτιστοποίησης των σχεδίων, καθιστώντας δυνατή την παραγωγή αποδοτικών FPGA υλοποιήσεων. Η ενσωμάτωση με εργαλεία εξομοίωσης και ανάλυσης σφαλμάτων

διευκολύνει τον εντοπισμό σφαλμάτων σε πρώιμο στάδιο, μειώνοντας τον χρόνο ανάπτυξης.

Το ModelSim Intel FPGA Starter Edition [63] είναι ένα εργαλείο προσομοίωσης και αποσφαλμάτωσης για FPGA., όπου προσφέρει δυνατότητες χρονισμένης προσομοίωσης των σχεδίων VHDL και Verilog, επιτρέποντας την παρακολούθηση των σημάτων σε επίπεδο κύκλου ρολογιού και τον εντοπισμό λειτουργικών σφαλμάτων. Επιπλέον, η ενσωμάτωση με το Quartus Prime Pro επιτρέπει την άμεση μεταφορά των αποτελεσμάτων στην προσομοίωση, διευκολύνοντας τον κύκλο ανάπτυξης FPGA. Αν και πρόκειται για έκδοση περιορισμένης λειτουργικότητας σε σχέση με τις πλήρεις επαγγελματικές εκδόσεις, παρέχει όλα τα βασικά εργαλεία για την εκπαίδευση και τον αρχικό σχεδιασμό ψηφιακών συστημάτων.

Συμπερασματικά, τα εργαλεία ανάπτυξης FPGA αποτελούν αναπόσπαστο μέρος της ροής ανάπτυξης, καθώς αυτοματοποιούν σύνθετες διαδικασίες και επιτρέπουν στον σχεδιαστή να επικεντρωθεί στη λειτουργικότητα και τη βελτιστοποίηση του συστήματος. Η επιλογή του κατάλληλου εργαλείου εξαρτάται άμεσα από την πλατφόρμα FPGA, τις απαιτήσεις της εφαρμογής και το επίπεδο πολυπλοκότητας του σχεδιασμού.

2.2 Χειριστήριο τύπου Arcade

Τα χειριστήρια τύπου Arcade αποτελούν συσκευές εισόδου σχεδιασμένες να αναπαράγουν την εμπειρία των κλασικών παιχνιδιών Arcade, συνδυάζοντας μηχανικά κουμπιά υψηλής αντοχής για τις ενέργειες του χρήστη και μοχλούς (joysticks) για τον προσδιορισμό της κατεύθυνσης στο παιχνίδι. Είναι σχεδιασμένα με τέτοιο τρόπο, ώστε να έχουν μεγαλύτερη ανθεκτικότητα, άμεση και αξιόπιστη απόκριση σε σχέση με τα συμβατικά χειριστήρια παιχνιδιών, εξασφάλιζοντας ακριβή και ευέλικτο ελέγχο κατά τη διάρκεια του παιχνιδιού. Χρησιμοποιούνται ευρέως σε ανακατασκευές παιχνιδομηχανών τύπου Arcade, σε εξομοιωτές παιχνιδιών, καθώς και σε συστήματα ψυχαγωγίας. Παρακάτω ακολουθεί η ιστορική τους αναδρομή, οι τύποι και τα χαρακτηριστικά των εισόδων, η αποθορυβοποίηση, η χαρτογράφηση των σημάτων καθώς και η περιγραφή του επιλεγμένου χειριστηρίου.

2.2.1 Ιστορική αναδρομή

Η εξέλιξη των arcade controllers ξεκινά από τη στρατιωτική και αεροναυτική χρήση των πρώτων joystick στα μέσα του 20ού αιώνα [64] και περνά στη μαζική ψυχαγωγική αξιοποίησή τους τη δεκαετία του 1970 με τις πρώτες arcade μηχανές, όπως το Computer Space και στη συνέχεια την Atari [65], όπου το CX40 (Εικόνα 18) καθιέρωσε το joystick ως βασικό πρότυπο των οικιακών και arcade χειριστηρίων [66].

Εικόνα 18 Atari CX40[15]

Τη δεκαετία του 1980 διαμορφώθηκε ο κλασικός συνδυασμός joystick και κουμπιών [64] σε παιχνίδια όπως το Pac-Man και το Donkey Kong, ενώ η εισαγωγή των twin-stick και fightsticks [67] σε παιχνίδια όπως «Robotron: 2084», και αργότερα του D-pad από τη Nintendo [68], εξειδίκευσε περαιτέρω τον έλεγχο, ιδιαίτερα στα fighting games. Στη δεκαετία του 1990 και έπειτα, η διάδοση των USB arcade sticks επέτρεψε τη χρήση τους σε PC και κονσόλες [70], ενώ αξιόλογες τοπικές πρωτοβουλίες, όπως αυτή των αδελφών Ανερούση στην Ελλάδα [71], συνέβαλαν στη βελτίωση της ανθεκτικότητας και της ποιότητας.

Συνοψίζοντας, η ιστορική εξέλιξη των arcade controllers και ειδικότερα των joystick, αναδεικνύει τη διαχρονική σημασία τους ως βασικές συσκευές εισόδου στον χώρο των ψηφιακών παιχνιδιών. Η ανθεκτικότητα του συνδυασμού joystick και κουμπιών, τόσο σε εργονομικό όσο και σε λειτουργικό επίπεδο, αναδεικνύει ότι οι σχεδιαστικές αρχές των πρώτων arcade συστημάτων αποτέλεσαν θεμέλιο για τον σύγχρονο σχεδιασμό των game controllers. Παρά τις τεχνολογικές εξελίξεις και την εμφάνιση νέων μορφών εισόδου, τα arcade controllers εξακολουθούν να αποτελούν σημείο αναφοράς, κυρίως σε 2D και

[15] Η εικόνα είναι από τη διεύθυνση https://en.wikipedia.org/wiki/Atari_CX40_joystick

ανταγωνιστικά παιχνίδια, επιβεβαιώνοντας τη διαχρονική τους αξία στο πεδίο του gaming και της ακαδημαϊκής έρευνας.

2.2.2 Τύποι Εισόδων

Οι τύποι εισόδων σε ένα χειριστήριο τύπου Arcade αποτελούν τον βασικό μηχανισμό, μέσω του οποίου ο χρήστης αλληλεπιδρά με το σύστημα παιχνιδιού. Οι είσοδοι μεταφέρουν φυσικές ενέργειες, όπως το πάτημα ενός κουμπιού ή τη μετακίνηση ενός μοχλού, σε ηλεκτρικά σήματα που μπορούν να ερμηνευτούν από το υλικό και το λογισμικό του συστήματος. Ανάλογα με τον τρόπο λειτουργίας και το είδος του σήματος που παράγουν, οι είσοδοι των arcade χειριστηρίων διακρίνονται κυρίως σε ψηφιακές και αναλογικές, με τις ψηφιακές να κυριαρχούν στα παραδοσιακά arcade συστήματα λόγω απλότητας και αξιοπιστίας. Κάθε τύπος εισόδου εξυπηρετεί διαφορετικές ανάγκες και παρουσιάζει συγκεκριμένα λειτουργικά και τεχνικά χαρακτηριστικά.

- ***Ψηφιακές είσοδοι***

Οι ψηφιακές είσοδοι βασίζονται στη λογική της δυαδικής κατάστασης και λειτουργούν ως απλοί διακόπτες. Κάθε κουμπί ή κατεύθυνση του joystick αντιστοιχεί σε μία μόνο μεταβλητή κατάστασης, η οποία μπορεί να λάβει είτε ενεργή είτε ανενεργή τιμή. Στην πράξη, όταν ο χρήστης πατά ένα κουμπί, κλείνει ένα ηλεκτρικό κύκλωμα και ο ελεγκτής αναγνωρίζει την είσοδο ως λογικό «1» ή «0», ανάλογα με τη σχεδίαση του κυκλώματος [72]. Η απουσία ενδιάμεσων τιμών καθιστά τις ψηφιακές εισόδους ιδιαίτερα κατάλληλες για παιχνίδια που απαιτούν άμεση απόκριση και ακρίβεια, όπως παιχνίδια δράσης και μάχης.

Στα arcade χειριστήρια, οι ψηφιακές είσοδοι περιλαμβάνουν κυρίως τα κουμπιά δράσης, τα οποία ενεργοποιούν εντολές όπως επίθεση, αναπήδηση ή επιλογή μενού, καθώς και τον μοχλό (joystick) κατευθύνσεων. Ο μοχλός αποτελείται από ανεξάρτητους μηχανικούς διακόπτες για κάθε κατεύθυνση (επάνω, κάτω, αριστερά, δεξιά), επιτρέποντας είτε τεσσάρων (4-way) είτε οκτώ κατευθύνσεων (8-way) λειτουργία. Η επιλογή του τύπου joystick εξαρτάται από το είδος των παιχνιδιών, καθώς τα παιχνίδια παλαιότερης γενιάς απαιτούν συχνά καθορισμένες κατευθύνσεις, ενώ τα σύγχρονα παιχνίδια εκμεταλλεύονται και τις διαγώνιες κινήσεις [73].

- ***Αναλογικές είσοδοι***

Οι αναλογικές είσοδοι διαφέρουν από τις ψηφιακές, καθώς δεν περιορίζονται σε δύο καταστάσεις αλλά παρέχουν ένα συνεχές εύρος τιμών. Αυτό επιτυγχάνεται μέσω στοιχείων όπως τα ποτενσιόμετρα ή αισθητήρες πίεσης, τα οποία μεταβάλλουν την τάση εξόδου ανάλογα με τη θέση ή την πίεση που ασκεί ο χρήστης. Οι αναλογικές είσοδοι χρησιμοποιούνται συχνότερα σε παιχνίδια αγώνων, προσομοιωτές πτήσης ή εφαρμογές όπου απαιτείται βαθμιαίος και πιο φυσικός έλεγχος της κίνησης [73].

Ωστόσο, η χρήση αναλογικών εισόδων συνεπάγεται αυξημένη πολυπλοκότητα, καθώς απαιτείται μετατροπή αναλογικού σήματος σε ψηφιακό μέσω μετατροπέων (ADC), καθώς και πρόσθετη επεξεργασία για τη σταθεροποίηση και το φιλτράρισμα του σήματος. Για τον λόγο αυτό, τα περισσότερα παραδοσιακά arcade χειριστήρια εξακολουθούν να βασίζονται σχεδόν αποκλειστικά σε ψηφιακές εισόδους, οι οποίες προσφέρουν υψηλή αξιοπιστία, απόλυτη ακρίβεια, απλότητα υλοποίησης και ελάχιστη καθυστέρηση απόκρισης.

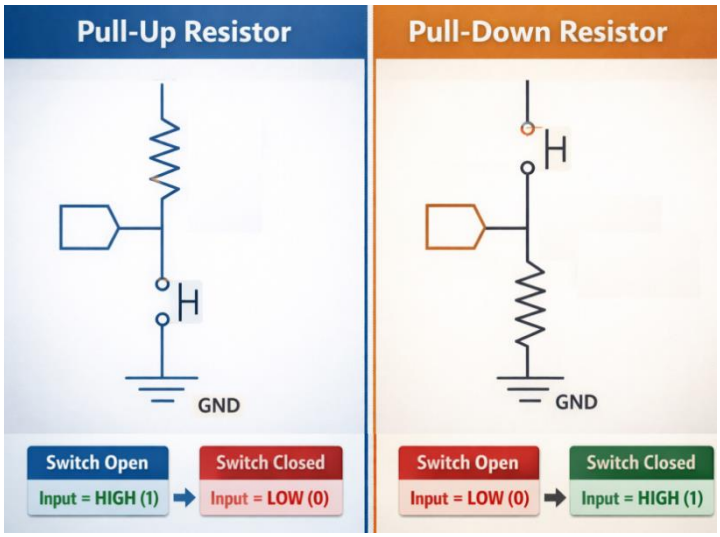
2.2.3 Ηλεκτρικά και λογικά χαρακτηριστικά εισόδων

Τα ηλεκτρικά και λογικά χαρακτηριστικά των εισόδων ενός χειριστηρίου τύπου Arcade καθορίζουν τον τρόπο με τον οποίο οι φυσικές ενέργειες του χρήστη μετατρέπονται σε αναγνωρίσιμα σήματα. Οι είσοδοι βασίζονται κυρίως σε απλά ηλεκτρικά κυκλώματα χαμηλής τάσης, σχεδιασμένα έτσι, ώστε να εξασφαλίζουν σταθερότητα, ταχύτητα απόκρισης και αντοχή στη συνεχή χρήση.

Στην πλειονότητα των arcade συστημάτων, οι είσοδοι υλοποιούνται ως ψηφιακές γραμμές σήματος, οι οποίες συνδέονται σε έναν μικροελεγκτή ή σε έναν encoder εισόδων [74], [75]. Κάθε γραμμή εισόδου αντιστοιχεί σε ένα κουμπί ή σε μία κατεύθυνση του joystick και λειτουργεί ως διακόπτης που είτε συνδέει είτε αποσυνδέει τη γραμμή από τη γείωση του κυκλώματος. Όταν το κουμπί δεν είναι πατημένο, η είσοδος βρίσκεται σε προκαθορισμένη λογική κατάσταση, ενώ όταν πατηθεί, η κατάσταση αυτή αντιστρέφεται, επιτρέποντας στον ελεγκτή να ανιχνεύσει την αλλαγή.

Για τη σταθεροποίηση της λογικής κατάστασης των εισόδων χρησιμοποιούνται αντιστάσεις έλξης (pull-up ή pull-down) (Εικόνα 19) [76]. Συνήθως, εφαρμόζονται εσωτερικές pull-up αντιστάσεις, οι οποίες διατηρούν την είσοδο σε υψηλή λογική στάθμη (HIGH) όταν το κουμπί είναι ανενεργό. Με το πάτημα του κουμπιού, η γραμμή συνδέεται

στη γείωση, προκαλώντας μετάβαση σε χαμηλή λογική στάθμη (LOW). Η τεχνική αυτή μειώνει τον ηλεκτρικό θόρυβο και αποτρέπει την εμφάνιση ασταθών τιμών εισόδου.



Εικόνα 19 Pull-up και pull-down αντιστάσεις

Οι τάσεις λειτουργίας των εισόδων στα arcade χειριστήρια είναι συνήθως χαμηλές, κυμαινόμενες μεταξύ 3,3V και 5V, ανάλογα με τον μικροελεγκτή ή τον encoder που χρησιμοποιείται. Το ρεύμα που διαρρέει τα κυκλώματα εισόδου είναι σχετικά μικρό, γεγονός που συμβάλλει τόσο στην ασφάλεια του χρήστη όσο και στη μακροχρόνια διάρκεια ζωής. Η χαμηλή κατανάλωση ενέργειας καθιστά τα arcade χειριστήρια κατάλληλα για χρήση σε φορητά ή USB-τροφοδοτούμενα συστήματα.

Σε λογικό επίπεδο, οι είσοδοι ερμηνεύονται από τον ελεγκτή ως δυαδικές μεταβλητές, οι οποίες αντιστοιχούν σε συγκεκριμένες εντολές. Η ανάγνωση των εισόδων γίνεται είτε μέσω περιοδικής δειγματοληψίας είτε μέσω μηχανισμών διακοπών, ανάλογα με τη σχεδίαση του συστήματος. Στα περισσότερα arcade συστήματα προτιμάται η δειγματοληψία σε υψηλή συχνότητα, ώστε να επιτυγχάνεται άμεση απόκριση χωρίς αντιληπτή καθυστέρηση από τον χρήστη.

Ιδιαίτερη σημασία έχει η ηλεκτρική αξιοπιστία, καθώς τα arcade κουμπιά και joysticks υπόκεινται σε έντονη και συχνή καταπόνηση. Για τον λόγο αυτό, χρησιμοποιούνται μηχανισμοί υψηλής ποιότητας, όπως μικροδιακόπτες με προκαθορισμένο σημείο ενεργοποίησης και μεγάλη διάρκεια ζωής. Τα στοιχεία αυτά έχουν σχεδιαστεί ώστε να

διατηρούν σταθερά ηλεκτρικά χαρακτηριστικά ακόμα και μετά από εκατομμύρια ενεργοποιήσεις.

Στην περίπτωση αναλογικών εισόδων, τα ηλεκτρικά χαρακτηριστικά διαφοροποιούνται, καθώς η είσοδος δεν αντιστοιχεί σε μία μόνο λογική κατάσταση αλλά σε ένα εύρος τιμών τάσης [74]. Ο ελεγκτής μετατρέπει την αναλογική τάση σε ψηφιακή τιμή μέσω αναλογικού- ψηφιακού μετατροπέα (ADC), επιτρέποντας την ακριβή αποτύπωση της θέσης ή της έντασης της ενέργειας του χρήστη [77]. Παρά τη μεγαλύτερη πολυπλοκότητα, οι αναλογικές είσοδοι χρησιμοποιούνται περιορισμένα στα arcade χειριστήρια, καθώς τα περισσότερα παιχνίδια έχουν σχεδιαστεί με βάση τη δυαδική λογική των ψηφιακών εισόδων.

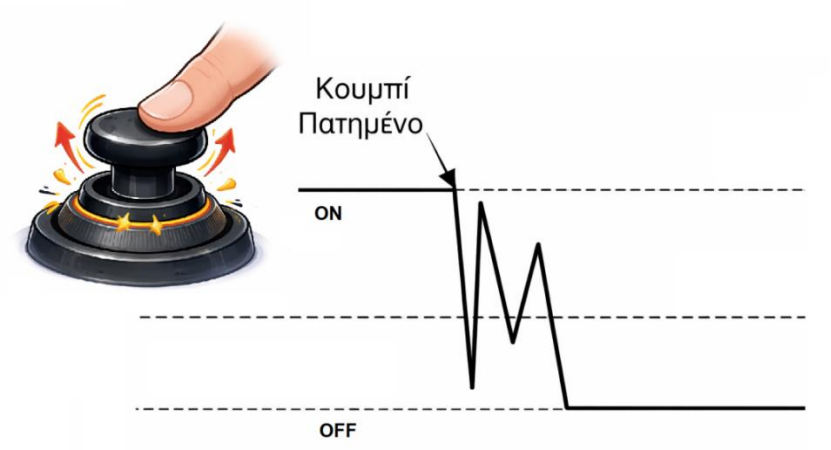
Συνοψίζοντας, τα ηλεκτρικά και λογικά χαρακτηριστικά των εισόδων στα χειριστήρια τύπου Arcade χαρακτηρίζονται από απλότητα, αξιοπιστία και χαμηλές απαιτήσεις ισχύος. Η σωστή επιλογή και υλοποίηση των κυκλωμάτων εισόδου εξασφαλίζει την άμεση απόκριση και τη σταθερή λειτουργία του συστήματος, στοιχεία απαραίτητα για τη διατήρηση της αυθεντικής εμπειρίας arcade.

2.2.4 Αποθορυβοποίηση επαφών (Debouncing)

Η αποθορυβοποίηση (debouncing) αποτελεί τη διαδικασία φιλτραρίσματος των ανεπιθύμητων παροδικών μεταβολών που εμφανίζονται στην έξοδο ενός διακόπτη ή πλήκτρου κατά τη μετάβαση μεταξύ καταστάσεων (ON/OFF). Αποτελεί σημαντικό στάδιο στη διαχείριση των εισόδων ενός χειριστηρίου τύπου Arcade, καθώς σχετίζεται άμεσα με την αξιοπιστία και την ακρίβεια των ενεργειών του χρήστη. Τα arcade χειριστήρια βασίζονται κυρίως σε κουμπιά και μικροδιακόπτες, οι οποίοι, λόγω της φυσικής τους κατασκευής, παρουσιάζουν ανεπιθύμητες ηλεκτρικές μεταβατικές καταστάσεις κατά την ενεργοποίηση και απενεργοποίησή τους. Το φαινόμενο αυτό είναι γνωστό ως «αναπήδηση επαφών διακόπτη» (switch bounce) [78].

Αναλυτικότερα, κατά τη διάρκεια ενός πατήματος, οι μεταλλικές επαφές ενός διακόπτη δεν σταθεροποιούνται άμεσα. Αντίθετα, έρχονται σε επαναλαμβανόμενη επαφή και αποχωρίζονται πολλές φορές μέσα σε χρονικό διάστημα μερικών χιλιοστών του δευτερολέπτου. Το αποτέλεσμα είναι η δημιουργία πολλαπλών γρήγορων μεταβάσεων μεταξύ υψηλού και χαμηλού σήματος (Εικόνα 20). Ένας μικροελεγκτής ή encoder, ο

οποίος διαβάζει τις εισόδους σε υψηλή συχνότητα, ενδέχεται να ερμηνεύσει τις μεταβάσεις αυτές ως πολλαπλά πατήματα αντί για ένα και μοναδικό πάτημα.



Εικόνα 20 Αναπήδηση επαφών διακόπτη (switch bounce)

Η αντιμετώπιση του φαινομένου πραγματοποιείται είτε σε επίπεδο υλικού είτε σε επίπεδο λογισμικού [79], [80]. Σε *επίπεδο υλικού*, η αποθορυβοποίηση σχετίζεται κυρίως με τη φυσική συμπεριφορά των μηχανικών διακοπτών. Χρησιμοποιούνται φίλτρα RC (Resistor-Capacitor) ή λογικές πύλες τύπου Schmitt trigger για την εξομάλυνση των σημάτων [81], ενώ σε πιο σύνθετα συστήματα προτιμώνται εξειδικευμένα ολοκληρωμένα κυκλώματα (ICs) με ρυθμιζόμενες παραμέτρους χρονισμού, που προσφέρουν plug-and-play λύση για πολλαπλούς διακόπτες. Σε *επίπεδο λογισμικού*, η αποθορυβοποίηση εφαρμόζεται κυρίως μέσω χρονικών φίλτρων ή μετρητών που παρακολουθούν τη διάρκεια σταθερής κατάστασης ενός διακόπτη. Η είσοδος θεωρείται έγκυρη μόνο όταν παραμείνει σταθερή για ένα ελάχιστο χρονικό διάστημα (συνήθως περίπου 10 ms), αποτρέποντας έτσι την καταγραφή πολλαπλών ενεργοποιήσεων από ένα μόνο πάτημα [80]. Αυτή η προσέγγιση χρησιμοποιείται ευρέως σε μικροελεγκτές και USB encoders, όπου μπορεί να εκτελείται είτε σε βρόχο (loop) είτε μέσω διακοπών (interrupts).

Συμπερασματικά, η αποθορυβοποίηση αποτελεί θεμελιώδη παράγοντα για την ορθή λειτουργία των arcade χειριστηρίων. Μέσω της κατάλληλης εφαρμογής τεχνικών αποθορυβοποίησης, εξαλείφονται τα ανεπιθύμητα σήματα και διασφαλίζεται ότι κάθε φυσική ενέργεια του χρήστη μεταφράζεται σε μία και μόνο αξιόπιστη λογική εντολή, στοιχείο απαραίτητο για την ακριβή και άμεση απόκριση των παιχνιδιών.

2.2.5 Χαρτογράφηση σημάτων (Signal Mapping)

Η χαρτογράφηση σημάτων (signal mapping) αποτελεί το στάδιο κατά το οποίο τα ηλεκτρικά και λογικά σήματα που παράγονται από τις εισόδους του arcade χειριστηρίου αντιστοιχίζονται σε συγκεκριμένες εντολές λογισμικού. Η διαδικασία αυτή λειτουργεί ως σύνδεση μεταξύ του υλικού και του λογισμικού, επιτρέποντας στο σύστημα παιχνιδιού να ερμηνεύει σωστά τις ενέργειες του χρήστη. Η ορθή χαρτογράφηση επηρεάζει άμεσα τη λειτουργικότητα, την απόκριση και τη συμβατότητα του χειριστηρίου με υλικό και λογισμικό.

Η διαδικασία της χαρτογράφησης σημάτων περιλαμβάνει την αντιστοίχιση κάθε φυσικής εισόδου του χειριστηρίου σε μία λογική εντολή. Η αντιστοίχιση αυτή μπορεί να αφορά είτε πλήκτρα πληκτρολογίου είτε κουμπιά ψηφιακού χειριστηρίου. Σε πολλές περιπτώσεις, η χαρτογράφηση πραγματοποιείται αρχικά στο επίπεδο του υλικού και στη συνέχεια στο επίπεδο του λογισμικού. Αυτή η προσέγγιση προσφέρει ευελιξία, καθώς επιτρέπει την προσαρμογή της λειτουργίας του χειριστηρίου σε διαφορετικά περιβάλλοντα και εφαρμογές.

Σε *επίπεδο υλικού*, η χαρτογράφηση ξεκινά από τον encoder ή τον μικροελεγκτή, ο οποίος λειτουργεί ως ενδιάμεσο επίπεδο επικοινωνίας μεταξύ του φυσικού υλικού εισόδου και του υπολογιστικού συστήματος. Ο encoder δέχεται τα σήματα από τους μηχανικούς διακόπτες, τα οποία είναι απλές μεταβολές κατάστασης (ανοιχτό ή κλειστό κύκλωμα), και τα μεταφράζει σε ψηφιακές εντολές. Η διαδικασία αυτή περιλαμβάνει τη δειγματοληψία των εισόδων, την αποθορυβοποίηση και την κωδικοποίηση της κατάστασης κάθε διακόπτη.

Στις σύγχρονες υλοποιήσεις, η επικοινωνία αυτή πραγματοποιείται μέσω του προτύπου USB Human Interface Device (HID) [82]. Το πρότυπο αυτό έχει σχεδιαστεί ώστε να υποστηρίζει συσκευές εισόδου, όπως πληκτρολόγια, ποντίκια και χειριστήρια, παρέχοντας έναν τυποποιημένο τρόπο επικοινωνίας με το λειτουργικό σύστημα. Ένα βασικό πλεονέκτημα του USB HID είναι η δυνατότητα αυτόματης αναγνώρισης της συσκευής (plug-and-play), χωρίς την ανάγκη εγκατάστασης εξειδικευμένων οδηγών. Το λειτουργικό σύστημα αναγνωρίζει τον encoder ως γενική συσκευή εισόδου και μπορεί άμεσα να λάβει και να επεξεργαστεί τα δεδομένα που αποστέλλονται.

Σε *επίπεδο λογισμικού*, η χαρτογράφηση ολοκληρώνεται μέσω της αντιστοίχισης των γεγονότων εισόδου σε συγκεκριμένες λειτουργίες του παιχνιδιού, όπως η κίνηση ενός

χαρακτήρα ή η εκτέλεση μίας ενέργειας. Στην περίπτωση εφαρμογών εξομοίωσης arcade συστημάτων [83], το λογισμικό παρέχει ένα επίπεδο αντιστοίχισης (mapping layer), το οποίο επιτρέπει στον χρήστη να καθορίσει ποιο κουμπί ή κατεύθυνση αντιστοιχεί σε συγκεκριμένη λειτουργία του παιχνιδιού. Η ύπαρξη λογισμικού επιπέδου χαρτογράφησης επιτρέπει την προσαρμογή του ίδιου χειριστηρίου σε διαφορετικά παιχνίδια ή ακόμα και σε διαφορετικές πλατφόρμες.

Συνολικά, η χαρτογράφηση σημάτων αποτελεί καθοριστικό παράγοντα για τη λειτουργικότητα ενός arcade χειριστηρίου. Μέσω αυτής, εξασφαλίζεται η ορθή επικοινωνία μεταξύ υλικού και λογισμικού, η συμβατότητα με διαφορετικά συστήματα και η παροχή μιας αξιόπιστης εμπειρίας χειρισμού.

2.2.6 Περιγραφή και ανάλυση επιλεγμένου χειριστηρίου τύπου arcade

Το επιλεγμένο χειριστήριο τύπου arcade είναι το Haitronic 1-Player Arcade Game Kit (HS1050) (Εικόνα 21), ένα ολοκληρωμένο arcade-style joystick kit το οποίο παρέχει τα βασικά στοιχεία χειρισμού για arcade εφαρμογές. Το συγκεκριμένο kit αποτελεί μια οικονομική και ευέλικτη λύση για DIY arcade projects, προσφέροντας μια βασική πλατφόρμα εισόδων με joystick και κουμπιά.

Το kit περιλαμβάνει περιλαμβάνει:

- ***Joystick***

  Ένας μοχλός με 5 pin σύνδεση για την ανίχνευση των τεσσάρων βασικών κατευθύνσεων (επάνω / κάτω / αριστερά / δεξιά).

- ***Κουμπιά***

  8 κουμπιά διαμέτρου 30 mm και 2 κουμπιά 24 mm, τα οποία χρησιμοποιούνται για εντολές δράσης (π.χ. fire, jump).

- ***Καλώδια***

  Παρέχονται συνολικά 13 καλώδια για σύνδεση των κουμπιών με τον πίνακα ελέγχου, καθώς και 5-pin καλώδιο για σύνδεση του joystick.

- ***USB encoder board***

  Το USB encoder board που περιλαμβάνεται στη συσκευασία επιτρέπει τη μετατροπή των αναλογικών μετακινήσεων και των πατημάτων των κουμπιών σε ψηφιακά σήματα, τα οποία κατά κανόνα αποστέλλονται ως HID events όταν συνδέονται σε υπολογιστή ή άλλη συσκευή μέσω USB.

Εικόνα 21 Arcade joystick

Σημαντικό πλεονέκτημα του kit είναι ότι η σύνδεση με την πλακέτα ανάπτυξης μπορεί να πραγματοποιηθεί και χωρίς τη χρήση του USB encoder board. Σε αυτή την περίπτωση, τα σήματα των microswitches των κουμπιών και του joystick μπορούν να οδηγηθούν απευθείας στις ψηφιακές εισόδους της πλακέτας, επιτρέποντας άμεσο έλεγχο και μεγαλύτερη ευελιξία στον σχεδιασμό της λογικής επεξεργασίας των εισόδων.

Ο μοχλός λειτουργεί με έναν απλό μηχανισμό που βασίζεται σε μικροδιακόπτες (microswitches) για την ανίχνευση των κατευθύνσεων. Κάθε μια από τις τέσσερις κατευθύνσεις ενεργοποιεί έναν ανεξάρτητο διακόπτη, επιτρέποντας τη μετάδοση ψηφιακών διακοπών (on/off) στον πίνακα ελέγχου. Παρόμοια διάταξη χρησιμοποιεί και κάθε κουμπί, το οποίο όταν πατιέται, κλείνει ένα κύκλωμα και στέλνει ένα καθαρό ψηφιακό σήμα. Μέσω της καλωδίωσης, τα σήματα μπορούν είτε να καταλήγουν στον encoder board είτε απευθείας στο κύκλωμα ελέγχου.

Συμπερασματικά, το Haitronic HS1050 αποτελεί μια αξιόπιστη και οικονομική λύση εισόδου για arcade εφαρμογές, προσφέροντας απλή ηλεκτρική διασύνδεση, σαφή

ψηφιακά σήματα και ευελιξία στη χρήση τόσο με USB όσο και με απευθείας σύνδεση σε πλακέτες ανάπτυξης.

2.3 Διεπαφή HDMI

Η διεπαφή HDMI (High-Definition Multimedia Interface) αποτελεί ένα από τα καθιερωμένα ψηφιακά πρότυπα για τη μεταφορά μη συμπιεσμένων σημάτων εικόνας και ήχου υψηλής ευκρίνειας μεταξύ ηλεκτρονικών συσκευών. Η σχεδίασή της έγινε με σκοπό τη μεταφορά ψηφιακού σήματος εικόνας και ήχου μέσω ενός και μόνο καλωδίου, αντικαθιστώντας παλαιότερες αναλογικές και ψηφιακές διεπαφές. Η διεπαφή HDMI (Εικόνα 22) χρησιμοποιείται ευρέως σε καταναλωτικές και επαγγελματικές συσκευές, προσφέροντας υψηλή αξιοπιστία, συμβατότητα και απλότητα στη σύνδεση [84].

Η τεχνολογία HDMI παρουσιάστηκε επίσημα το 2002 και αναπτύχθηκε με στόχο τη δημιουργία ενός κοινού ψηφιακού προτύπου για τη μετάδοση ήχου και εικόνας. Σε αντίθεση με παλαιότερες αναλογικές διεπαφές, όπως το VGA ή το SCART, το HDMI μπορεί να μεταδώσει τα δεδομένα σε ψηφιακή μορφή, εξασφαλίζοντας αναλλοίωτη ποιότητα σήματος από την πηγή έως τον δέκτη. Κατά τη διάρκεια των ετών, το HDMI εξελίχθηκε μέσω διαδοχικών εκδόσεων, με κάθε νέα έκδοση να εισάγει αυξημένο εύρος ζώνης, υποστήριξη υψηλότερων αναλύσεων και νέες λειτουργίες, όπως τρισδιάστατη απεικόνιση, υψηλό δυναμικό εύρος και βελτιωμένους μηχανισμούς συγχρονισμού ήχου και εικόνας [84], [85].

Ένα από τα βασικά πλεονεκτήματα της διεπαφής HDMI είναι η δυνατότητα ταυτόχρονης μεταφοράς εικόνας, πολλαπλών καναλιών ήχου και βοηθητικών δεδομένων, όπως πληροφορίες συγχρονισμού και ελέγχου συσκευών, χωρίς απώλειες ποιότητας, καθιστώντας το αποδοτικό για σύγχρονες εφαρμογές πολυμέσων. Με την εξέλιξη των εκδόσεών της, προσαρμόστηκε στις αυξανόμενες απαιτήσεις ανάλυσης, ρυθμού καρέ και βάθους χρώματος [84].

Εικόνα 22 Καλώδιο HDMI[16]

Η διεπαφή HDMI υποστηρίζει διαφορετικούς τύπους συνδέσεων [86] [87], οι οποίοι διαφέρουν ως προς το μέγεθος και τη χρήση, με βασικούς τους Type A (Standard HDMI) για τηλεοράσεις και επαγγελματικό εξοπλισμό, Type B (Dual-Link) για υψηλές αναλύσεις, Type C (Mini HDMI) για φορητές συσκευές, Type D (Micro HDMI) για πολύ μικρές συσκευές και Type E για αυτοκινητιστικές εφαρμογές με αυξημένη μηχανική αντοχή. Κάθε τύπος εξασφαλίζει αξιόπιστη μετάδοση ψηφιακού σήματος και η επιλογή εξαρτάται από τις απαιτήσεις χώρου και απόδοσης του συστήματος. Στο παρόν έργο επιλέχθηκε ο τύπος Type A, καθώς αποτελεί το πιο διαδεδομένο και συμβατό πρότυπο για συστήματα απεικόνισης.

2.3.1 Αρχιτεκτονική και λειτουργία HDMI

Το HDMI αποτελεί μια ψηφιακή διεπαφή πολυμέσων που συνδυάζει τη μετάδοση εικόνας, ήχου και βοηθητικών δεδομένων σε ένα ενιαίο σύστημα, βασιζόμενη σε αρχιτεκτονική τύπου πηγή–δέκτης (source–sink architecture) [91]. Το καλώδιο HDMI μεταφέρει τρία ή τέσσερα, ανάλογα με την έκδοση, διαφορικά ζεύγη για σήματα Transition-Minimized Differential Signaling (TMDS) για εικόνα, ήχο και βοηθητικά δεδομένα, ένα κανάλι ρολογιού για συγχρονισμό, καθώς και κανάλια για έλεγχο και διαμόρφωση, εξασφαλίζοντας ολοκληρωμένη επικοινωνία χωρίς την ανάγκη πολλών καλωδίων [92]. Η μετάδοση χρησιμοποιεί την τεχνολογία TMDS [93],[94] με κωδικοποίηση 8b/10b ή 4b/10b, μειώνοντας τις ηλεκτρομαγνητικές παρεμβολές και διασφαλίζοντας την ακεραιότητα των δεδομένων.

Η αρχιτεκτονική μπορεί να αναλυθεί λειτουργικά σε διακριτά επίπεδα, τα οποία συνεργάζονται για την αξιόπιστη μετάδοση ψηφιακού οπτικοακουστικού περιεχομένου,

[16] Η εικόνα είναι από τη διεύθυνση https://electronics.howstuffworks.com/hdmi.htm

όπως το φυσικό επίπεδο (physical layer), το επίπεδο μετάδοσης δεδομένων (link layer) και το επίπεδο ελέγχου και διαχείρισης μέσω των βοηθητικών καναλιών επικοινωνίας, επιτρέποντας έτσι την αξιόπιστη μετάδοση υψηλής ποιότητας οπτικοακουστικών σημάτων [91]. Το φυσικό επίπεδο είναι υπεύθυνο για τη φυσική και ηλεκτρική μετάδοση των σημάτων, παρέχοντας τα απαραίτητα ζεύγη δεδομένων και κανάλι ρολογιού. Το επίπεδο μετάδοσης δεδομένων διαχειρίζεται την κωδικοποίηση, τη δομή και τον συγχρονισμό των πακέτων βίντεο, ήχου και βοηθητικών δεδομένων, ενώ το επίπεδο ελέγχου και διαχείρισης υποστηρίζει λειτουργίες όπως το DDC, το CEC, το HDCP, το ARC και το HPD για επικοινωνία, έλεγχο, ασφάλεια και διαλειτουργικότητα των συσκευών.

Συμπερασματικά, η συνολική λειτουργία του HDMI βασίζεται στη στενή συνεργασία των παραπάνω επιπέδων. Το φυσικό επίπεδο παρέχει το αξιόπιστο μέσο μετάδοσης, το επίπεδο μετάδοσης δεδομένων εξασφαλίζει τη σωστή κωδικοποίηση και τον συγχρονισμό του οπτικοακουστικού περιεχομένου, ενώ το επίπεδο ελέγχου και διαχείρισης επιτρέπει την προσαρμογή, την ασφάλεια και τον έλεγχο της σύνδεσης. Η σωστή αρχιτεκτονική και το αξιόπιστο πρωτόκολλο μετάδοσης αποτελούν τη βάση για την υψηλή ποιότητα εικόνας και ήχου της HDMI, καθώς και για τη σταθερότητα και τη διαλειτουργικότητα των συνδεδεμένων συσκευών.

2.3.2 Διεπαφή HDMI στην πλακέτα DE23-Lite

Η αναπτυξιακή πλακέτα DE23-Lite διαθέτει ενσωματωμένο HDMI transmitter (ADV7513), ο οποίος αναλαμβάνει την πλήρη υλοποίηση της διεπαφής HDMI σε επίπεδο μετάδοσης και πρωτοκόλλου. Λειτουργεί ως ενδιάμεσο υποσύστημα, το οποίο μεταφράζει τα παράλληλα ψηφιακά σήματα εικόνας και συγχρονισμού σε συμβατό σήμα HDMI, σύμφωνα με τις προδιαγραφές του προτύπου. Έτσι, το FPGA αντιμετωπίζει τη διεπαφή HDMI ως μια απλοποιημένη διεπαφή εξόδου βίντεο, παρόμοια με τη λογική λειτουργία παλαιότερων διεπαφών όπως το VGA, αλλά με ψηφιακή μορφή μετάδοσης. Επομένως, δεν απαιτείται κωδικοποίηση TMDS, αφού ο HDMI transmitter αναλαμβάνει να μετατρέψει τα παραγόμενα, από το FPGA, ψηφιακά σήματα βίντεο σε TMDS, επιτρέποντας την ανάπτυξη εφαρμογών απεικόνισης με μειωμένες απαιτήσεις σχεδιασμού.

Σημαντικό πλεονέκτημα της πλακέτας DE23-Lite αποτελεί το γεγονός ότι η κατασκευάστρια εταιρεία παρέχει μέσω της επίσημης ιστοσελίδας της έτοιμα παραδείγματα αναφοράς (reference designs) για την έξοδο εικόνας σε οθόνη μέσω HDMI

[99]. Τα παραδείγματα αυτά περιλαμβάνουν πλήρως λειτουργικές υλοποιήσεις παραγωγής χρονισμών εικόνας, δημιουργίας pixel δεδομένων και οδήγησης του ενσωματωμένου HDMI transmitter, προσφέροντας ένα σημείο αναφοράς ορθής υλοποίησης, βασισμένο στις προδιαγραφές της πλακέτας και του ενσωματωμένου HDMI transmitter.

Συμπερασματικά, η πλακέτα DE23-Lite με ενσωματωμένο HDMI transmitter απλοποιεί την παραγωγή ψηφιακής εικόνας, καθώς το FPGA παρέχει μόνο τα σήματα pixel και συγχρονισμού, ενώ η φυσική μετάδοση αναλαμβάνεται από το ενσωματωμένο υλικό. Η εταιρεία παρέχοντας παραδείγματα αναφοράς για έξοδο εικόνας μέσω HDMI, διευκολύνει την ανάπτυξη εξασφαλίζοντας συμβατότητα. Η προσέγγιση αυτή επιτρέπει τη μελέτη των βασικών αρχών απεικόνισης και χρονισμού χωρίς να απαιτείται ενασχόληση με το πρωτόκολλο HDMI, προσφέροντας μια αξιόπιστη βάση για την υλοποίηση του συστήματος.

2.4 Tetris game

2.4.1 Γενική περιγραφή

Το Tetris δημιουργήθηκε το 1984 από τον Ρώσο μηχανικό υπολογιστών Αλεξέι Πατζίτνοφ (Alexey Pajitnov) στη Μόσχα εμπνευσμένος από ένα παιχνίδι που είχε, το Pentominoes [100], [101] . Η ονομασία του παιχνιδιού προήλθε από το ελληνικό πρόθημα «τέτρα», καθώς κι από την αντισφαίριση, όπου ήταν και το αγαπημένο του άθλημα [106]. Η λογική του βασίζεται στην πτώση tetrominoes (σχημάτων αποτελούμενων από 4 τετράγωνα), τα οποία ο παίκτης μπορεί να μετακινεί δεξιά ή αριστερά, να περιστρέφει ή και να επιταχύνει την πτώσης τους ώστε να σχηματίζει οριζόντιες γραμμές χωρίς κενά. Όταν σχηματιστεί μια ολοκληρωμένη γραμμή τότε εξαφανίζεται και τα υπόλοιπα σχήματα πάνω από αυτή πέφτουν ένα επίπεδο κάτω. Το παιχνίδι συνεχίζεται με αυξανόμενη ταχύτητα, και αν το πλέγμα γεμίσει, τότε δεν υπάρχει χώρος για νέο σχήμα και το παιχνίδι τελειώνει.

Ο Αλεξέι δημιούργησε το πρώτο πρωτότυπο του παιχνιδιού για τον υπολογιστή Electronica 60 (Εικόνα 23) και λίγες μέρες μετά υλοποιήθηκε η πρώτη έκδοση του Tetris για MS DOS [102].

Εικόνα 23 Το Tetris στον Electronica 60

Το παιχνίδι αυτό έγινε το πρώτο βιντεοπαιχνίδι της Σοβιετικής Ένωσης που κυκλοφόρησε στις ΗΠΑ, όπου διανεμήθηκε από την Spectrum HoloByte για το Commodore 64 και τους υπολογιστές IBM. Έκτοτε, έχει κυκλοφορήσει σε μηχανές τύπου arcade και στη φορητή κονσόλα Game Boy το 1989, όπου και έγινε ένα από τα δημοφιλέστερα και κλασικότερα παιχνίδια όλων των εποχών [106]. Πλέον, είναι διαθέσιμο σχεδόν σε οποιαδήποτε κονσόλα και λειτουργικό σύστημα υπολογιστή, καθώς σε φορητές συσκευές αναπαραγωγής πολυμέσων και σε κινητά τηλέφωνα.

Το Tetris αποτελεί ένα από τα πιο αναγνωρίσιμα βιντεοπαιχνίδια παγκοσμίως. Είναι ένα παιχνίδι συνδυαστικής σκέψης, στρατηγικής και ταχύτητας αντίδρασης με εύκολους κανόνες στην κατανόηση και απαιτώντας για την εκτέλεσή του υψηλό επίπεδο συγκέντρωσης και δεξιοτήτων. Μετά από μελέτες για την επίδρασή του σε επίπεδο ψυχολογίας και νευροεπιστήμης, φαίνεται ότι η ενασχόληση με το παιχνίδι μπορεί να ενισχύσει τη χωρική αντίληψη και την οπτικο-κινητικό συντονισμό [103],[104],[105].

2.4.2 Τρόπος λειτουργίας

Η λειτουργία του παιχνιδιού στηρίζεται σε απλούς κανόνες και ο παίκτης για να μεγιστοποιήσει το σκορ του, πρέπει να σκέφτεται γρήγορα και στρατηγικά ώστε αποφύγει το «γέμισμα» του πλέγματος, που οδηγεί στο τέλος του παιχνιδιού.

Συγκεκριμένα, αποτελείται από ένα κατακόρυφο πλέγμα, το οποίο συνήθως έχει 10 στήλες και 20 γραμμές. Από την κορυφή του πλέγματος πέφτουν σχήματα, γνωστά ως τετραμίνια (tetriminoes). Κάθε tetromino αποτελείται από τέσσερα τετράγωνα συνδεδεμένα σε διαφορετικές διαμορφώσεις, σχηματίζοντας επτά βασικούς τύπους: I, O,

T, S, Z, J και L (Εικόνα 24). Ο παίκτης μπορεί να το μετακινήσει αριστερά ή δεξιά ή να το περιστρέψει κατά 90 μοίρες, και να το τοποθετήσει έτσι ώστε να σχηματίσει πλήρη οριζόντια γραμμή χωρίς κενά. Όταν μια γραμμή συμπληρωθεί πλήρως, εξαφανίζεται και τα τουβλάκια πάνω από αυτήν πέφτουν προς τα κάτω. Το παιχνίδι με την πάροδο του χρόνου αυξάνεται σε δυσκολία, καθώς αυξάνεται η ταχύτητα με την οποία πέφτουν τα τετραμίνια. Αν ο παίκτης δεν καταφέρνει να διαγράψει αρκετές γραμμές ώστε να δημιουργήσει χώρο, τότε τα τετραμίνια συσσωρεύονται μέχρι την κορυφή του πλέγματος και το παιχνίδι τελειώνει.

Εικόνα 24 Βασικοί τύποι "tetrominoes"

Σημαντικά στοιχεία στρατηγικής στο Tetris αποτελούν η χωρική αντίληψη, η πρόβλεψη και η ταχύτητα αντίδρασης, όπου ο συνδυασμός αυτών μεγιστοποιεί τη διάρκεια του παιχνιδιού και το σκορ. Σε πολλές εκδόσεις του παιχνιδιού, εμφανίζεται σε προεπισκόπηση το επόμενο tetromino, όπου δίνει τη δυνατότητα στον παίκτη για καλύτερο σχεδιασμό ώστε να δημιουργήσει συνδυασμούς για πολλαπλές διαγραφές γραμμών. Η συνεχής ανάγκη για λήψη αποφάσεων υπό πίεση κάνει το Tetris ιδιαίτερα απαιτητικό σε επίπεδο δεξιοτήτων και συγκέντρωσης.

Συμπερασματικά, το Tetris έχει απλούς και ξεκάθαρους κανόνες που το καθιστούν εύκολα κατανοητό και η επιτυχία του βασίζεται σε μια πολυδιάστατη στρατηγική εξαρτώμενη από την ικανότητα του παίκτη να προσαρμόζεται στις δυναμικές συνθήκες του παιχνιδιού.

2.5 Συμπεράσματα

Η ανάλυση του θεωρητικού υπόβαθρου ανέδειξε τη σημασία των FPGA ως ευέλικτων και ισχυρών πλατφορμών για την υλοποίηση ψηφιακών συστημάτων, με δυνατότητα παραμετροποίησης, υποστήριξη γλωσσών περιγραφής υλικού (HDL) και χρήση εργαλείων ανάπτυξης για αξιόπιστες εφαρμογές. Η κατανόηση της αρχιτεκτονικής και

της ροής ανάπτυξης εφαρμογών επιτρέπει τη μετατροπή των ιδεών σε λειτουργικά ψηφιακά συστήματα.

Παράλληλα, η μελέτη των χειριστηρίων τύπου arcade ανέδειξε τη σημασία των ηλεκτρικών και λογικών χαρακτηριστικών εισόδων, όπως οι ψηφιακές γραμμές, οι αντιστάσεις έλξης και η αποθορυβοποίηση, για την αξιόπιστη καταγραφή των ενεργειών του χρήστη. Στη συνέχεια, η ανάλυση της αρχιτεκτονικής HDMI και της λειτουργίας της διασφαλίζει την ακριβή μετάδοση δεδομένων βίντεο από το FPGA προς την οθόνη, αποτελώντας κρίσιμο στοιχείο για εφαρμογές οπτικής απεικόνισης, όπως τα παιχνίδια.

Τέλος, η περιγραφή του παιχνιδιού Tetris έδειξε πώς η ενσωμάτωση FPGA, χειριστηρίου και οπτικής διεπαφής μπορεί να εξασφαλίσει άμεση και αξιόπιστη καταγραφή των ενεργειών του χρήστη. Παράλληλα, η πολυδιάστατη στρατηγική που απαιτεί το παιχνίδι, σε συνδυασμό με την ανάγκη για γρήγορες αποφάσεις και σωστό σχεδιασμό, αναδεικνύει τη σημασία της συγχρονισμένης λειτουργίας των ψηφιακών συστημάτων σε πραγματικό χρόνο.

Στη συνέχεια, ακολουθεί το επόμενο κεφάλαιο, όπου παρουσιάζεται η ανάλυση των απαιτήσεων του συστήματος, εξετάζοντας τις λειτουργικές προδιαγραφές του παιχνιδιού, τις απαιτήσεις υλικού και τα κριτήρια επιλογής της κατάλληλης FPGA πλακέτας, έτσι ώστε να διασφαλιστεί η σωστή υλοποίηση και λειτουργία του συστήματος

3. Ανάλυση απαιτήσεων συστήματος

Η ανάλυση απαιτήσεων αποτελεί θεμελιώδες στάδιο στη διαδικασία σχεδίασης και υλοποίησης ενός ψηφιακού συστήματος, καθώς καθορίζει τις λειτουργίες που πρέπει να υποστηρίζει το σύστημα, τις προδιαγραφές του περιβάλλοντος υλοποίησης και τους περιορισμούς που πρέπει να ληφθούν υπόψη. Παρακάτω, ακολουθούν η ανάλυση των προδιαγραφών του παιχνιδιού, η ανάλυση απαιτήσεων υλικού καθώς και τα κριτήρια επιλογής της FPGA πλακέτας.

3.1 Προδιαγραφές παιχνιδιού

Βασική προϋπόθεση για μια επιτυχημένη υλοποίηση αποτελεί η σαφής αποτύπωση των κανόνων, των μηχανισμών που καθορίζουν την συμπεριφορά των στοιχείων του παιχνιδιού, την αλληλουχία ενεργειών του παίκτη και τον χρονισμό των λειτουργιών. Η κατανόηση αυτών των προδιαγραφών, αποτελεί τη βάση για τη σχεδίαση της μηχανής καταστάσεων (FSM), τον καθορισμό των χρονικών παραμέτρων, καθώς και τη διασφάλιση ορθής και αξιόπιστης λειτουργίας σε πραγματικό χρόνο.

Στην παρούσα υλοποίηση, οι κανόνες του Tetris θα απλοποιηθούν ώστε να ανταποκρίνονται στις ελάχιστες απαιτήσεις για τη λειτουργία του παιχνιδιού. Αυτό σημαίνει ότι το παιχνίδι θα περιλαμβάνει βασική λογική τοποθέτησης και περιστροφής των tetrominoes, χωρίς πολύπλοκα γραφικά, επίπεδα ή πρόσθετες λειτουργίες όπως εμφάνιση επόμενου κομματιού και εμφάνιση σκορ. Η προσαρμογή αυτή στοχεύει στη διατήρηση της βασικής εμπειρίας του Tetris με ελάχιστη χρήση πόρων και απλή υλοποίηση, κατάλληλη για πλατφόρμες FPGA με περιορισμένη μνήμη και επεξεργαστική ισχύ. Παρακάτω, παρουσιάζονται η περιγραφή του παιχνιδιού και οι βασικοί κανόνες, η ανάλυση λειτουργίας και μηχανισμών και οι βασικές λειτουργικές απαιτήσεις χρονισμού.

3.1.1 Περιγραφή του παιχνιδιού και βασικοί κανόνες

Το Tetris είναι ένα παιχνίδι πάζλ που αποτελείται από ένα ορθογώνιο πλέγμα και από την κορυφή του εμφανίζονται σχήματα τα οποία έχουν καθοδική πορεία. Τα σχήματα, γνωστά ως tetriminoes, αποτελούνται από τετράγωνα και καθώς πέφτουν, τοποθετούνται ώστε να σχηματίσουν πλήρεις οριζόντιες γραμμές. Όταν συμπληρωθεί μια γραμμή, αυτή διαγράφεται, τα πάνω τετράγωνα από αυτή κατεβαίνουν και έτσι δημιουργείται χώρος στο

πλέγμα. Στόχος είναι η διαγραφή όσο το δυνατόν περισσότερων γραμμών. Το παιχνίδι τερματίζει όταν δεν υπάρχει χώρος να κατέβει νέο σχήμα από την κορυφή του πλέγματος [106].

Αναλυτικότερα:

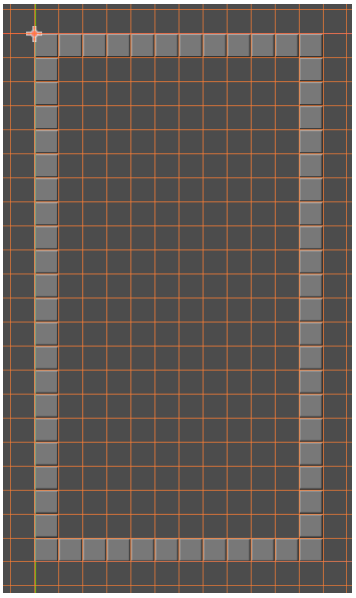
- Γίνεται έναρξη του παιχνιδιού από τον παίκτη.
- Εμφανίζεται το πλέγμα κενό και από την κορυφή πέφτει με σταθερό ρυθμό ένα σχήμα.
- Ο παίκτης έχει τη δυνατότητα να το μετακινήσει δεξιά ή αριστερά, να το περιστρέψει κατά 90 μοίρες και να το μετακινήσει γρήγορα προς τα κάτω.
- Καθώς το σχήμα παίρνει την τελική του θέση στο πλέγμα, αν έχει σχηματιστεί μια ή και περισσότερες πλήρεις οριζόντιες γραμμές, αυτές εξαφανίζονται και ότι υπάρχει πάνω από αυτές πέφτει προς τα κάτω. Στη συνέχεια, εμφανίζεται νέο σχήμα από την κορυφή.
- Η διαδικασία αυτή επαναλαμβάνεται έως ότου το νέο σχήμα που θα εμφανιστεί δεν έχει χώρο και ακουμπήσει τα προηγούμενα σχήματα που έχουν φτάσει στην κορυφή.
- Το παιχνίδι τερματίζει.

3.1.2 Ανάλυση λειτουργίας και μηχανισμών παιχνιδιού

Η ανάλυση που ακολουθεί περιλαμβάνει τη δομή του παιχνιδοχώρου, τους κανόνες κίνησης και περιστροφής των Tetrominoes, τη διαχείριση σύγκρουσης, τον μηχανισμό ολοκλήρωσης γραμμών, τις συνθήκες τερματισμού, ώστε να παρέχει ένα πλήρες πλαίσιο για τη σχεδίαση της λογικής του παιχνιδιού [4].

- ***Δομή παιχνιδοχώρου***

Ο χώρος του παιχνιδιού αποτελείται από ένα σταθερό πλέγμα διαστάσεων 10 στηλών επί 20 γραμμών. Κάθε τετράγωνο του πλέγματος μπορεί να είναι είτε κενό είτε κατειλημμένο από κάποιο κομμάτι. Στην Εικόνα 25 φαίνεται ένα παράδειγμα παιχνιδοχώρου.



Εικόνα 25 Πλέγμα διαστάσεων 10x20[17]

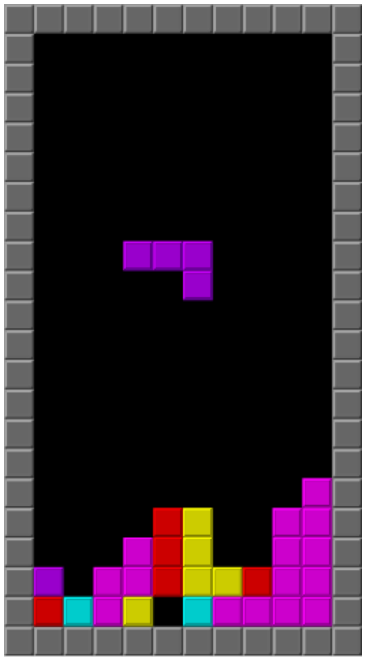
- ***Tetrominoes***

Τα Tetrominoes, τα οποία είναι τα δομικά στοιχεία του παιχνιδιού, αποτελούνται από τέσσερα τετράγωνα, (Εικόνα 24), συνθέτοντας 7 διαφορετικούς τύπους: I, O, T, S, Z, J, L. Κάθε Tetromino μπορεί να περιστραφεί σε τέσσερις πιθανές καταστάσεις (0°, 90°, 180°, 270°). Η αρχική τοποθέτηση κάθε στοιχείου γίνεται στην κορυφή του πλέγματος, στη μέση των στηλών, το οποίο κατεβαίνει με σταθερό ρυθμό μέχρι να ακουμπήσει στη βάση του πλέγματος ή σε κάποιο άλλο στοιχείο. Η εμφάνιση κάθε Tetromino είναι τυχαία και η επιλογή γίνεται από γεννήτρια (random number generator).

- ***Κίνηση Tetromino***

Οι κανόνες κίνησης ορίζουν τις επιτρεπόμενες ενέργειες του παίκτη. Κάθε Tetromino μπορεί να μετακινηθεί δεξιά ή αριστερά κατά μία στήλη και να περιστραφεί κατά 90 μοίρες δεξιόστροφα (εκτός από το O), έτσι ώστε η νέα θέση να είναι έγκυρη και να μην συγκρούεται με άλλα τετράγωνα ή τα όρια του πλέγματος. Η κίνηση προς τα κάτω προκαλεί επιτάχυνση πτώσης των Tetrominoes, αλλιώς κατεβαίνουν με σταθερή ταχύτητα. Στην Εικόνα 26 φαίνεται πώς κατεβαίνει ένα tetromino, για να συναντήσει τα υπόλοιπα, στο κάτω μέρος του πλέγματος.

[17] Η εικόνα είναι από τη διεύθυνση https://dev.to/xzzz3/tetris-development-2-intro-to-godot-and-basic-tetris-setup-4m4h



Εικόνα 26 Στιγμιότυπο-1 παιχνιδιού Tetris[18]

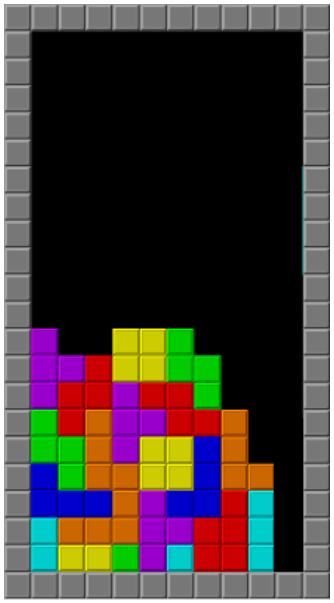
- ***Ανίχνευση σύγκρουσης***

Η λογική σύγκρουσης εξασφαλίζει ότι τα Tetrominoes δεν μπορούν να καλύψουν τα ήδη υπάρχοντα τετράγωνα και να υπερβούν τα όρια του πλέγματος. Όταν ένα Tetromino φτάσει σε μη επιτρεπτή θέση ή έρθει σε επαφή με άλλο κομμάτι, ενεργοποιείται η διαδικασία κλειδώματος, όπου το κομμάτι γίνεται μέρος του πλέγματος. Στην Εικόνα 27 φαίνεται όταν το tetromino O, έχει πάρει την τελική του θέση στο σύνολο του πλέγματος.

- ***Έλεγχος πληρότητας και διαγραφή γραμμών***

Όταν μια οριζόντια γραμμή σχηματιστεί πλήρως, τότε διαγράφεται, και τα τετράγωνα που βρίσκονται πάνω της μετακινούνται προς τα κάτω. Ταυτόχρονα μπορούν να διαγραφούν έως και 4 γραμμές

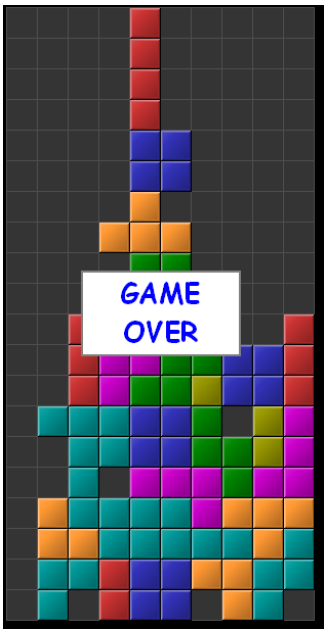
[18] Η εικόνα είναι από τη διεύθυνση https://sco.wikipedia.org/wiki/Tetris



Εικόνα 27 Στιγμιότυπο-2 παιχνιδιού Tetris

.*Τερματισμός παιχνιδιού*

Το παιχνίδι τερματίζει όταν ένα νέο Tetromino δεν μπορεί να τοποθετηθεί στην κορυφή του πλέγματος, λόγω ύπαρξης κομματιών στην κορυφή του πλέγματος. Στην περίπτωση αυτή, ενεργοποιείται η διαδικασία τερματισμού και εμφανίζεται η δυνατότητα επανεκκίνησης του παιχνιδιού , η οποία επαναφέρει το πλέγμα στην αρχική κατάσταση. Η Εικόνα 28 δείχνει τη λήξη του παιχνιδιού λόγω πληρότητας του πλέγματος.



Εικόνα 28 Τερματισμός παιχνιδιού Tetris

3.1.3 Λειτουργικές απαιτήσεις χρονισμού

Οι απαιτήσεις χρονισμού καθορίζουν τον τρόπο με τον οποίο το παιχνίδι ανταποκρίνεται στις ενέργειες του παίκτη και στο περιβάλλον του, διασφαλίζοντας τη σωστή λειτουργία και την ομαλή απεικόνιση των γεγονότων στην οθόνη. Στο Tetris, ο χρονισμός περιλαμβάνει τη συχνότητα ενημέρωσης της λογικής του παιχνιδιού, το χρόνο εμφάνισης του νέου Tetromino, την ταχύτητα πτώσης αυτών, την καθυστέρηση μετακίνησης και περιστροφής , καθώς και τις χρονικές παραμέτρους για τον μηχανισμό κλειδώματος των σχημάτων και εξαφάνισης των γραμμών. Η προσεκτική ρύθμιση αυτών των παραμέτρων διασφαλίζει μια ομαλή εμπειρία παιχνιδιού, όπου η ανταπόκριση των κομματιών και ο βαθμός δυσκολίας παραμένουν σε ισορροπία, επιτρέποντας στον παίκτη να έχει πλήρη έλεγχο.

Η επιλογή των χρονικών παραμέτρων βασίζεται σε ανάλυση και δεδομένα παρόμοιων υλοποιήσεων, ώστε να διασφαλιστεί η πληρότητα και η αξιοπιστία της λειτουργίας του παιχνιδιού. Οι κύριες παράμετροι που χρησιμοποιούνται συνήθως περιλαμβάνουν το DAS (Delayed Auto Shift), συνήθως 150–200 ms για την αρχική καθυστέρηση μετακίνησης, το ARR (Auto Repeat Rate) περίπου 50 ms για την επανάληψη μετακινήσεων, το lock delay 300–500 ms, την ταχύτητα πτώσης των Tetrominoes που ξεκινά από 500 ms ανά βήμα στο επίπεδο 0 και μειώνεται προοδευτικά σε υψηλά επίπεδα, καθώς και τα animation των γραμμών διάρκειας 150–250 ms. Οι τιμές αυτές προέρχονται από την “Tetris Guideline” [4], το “Tetris The Grand Master” [107], το “Tetris (NES)” [108] και το “Four-tris” [109] και έχουν δοκιμαστεί σε πλήθος υλοποιήσεων για ομαλή απόκριση. Επιπλέον, παρατηρούνται παραλλαγές στις τιμές DAS/ARR ανάλογα με τις εκδόσεις των παιχνιδιών, όπως στο “Taming Tetris timing” [110], (DAS ~183 ms, ARR ~83 ms) ή στο “Heboris Unofficial Expansion” [111], όπου όλες οι παραπάνω παράμετροι μπορούν να παραμετροποιηθούν.

Εκτός από τις βασικές παραμέτρους, ορισμένες υλοποιήσεις χρησιμοποιούν και επιπλέον ρυθμίσεις χρονισμού για μεγαλύτερη ακρίβεια και έλεγχο, όπου επεξηγούνται συνολικά στον Πίνακα 1. Τέτοιες περιλαμβάνουν το ARE (Entry / Spawn Delay), που στην “Tetris Design Guideline” [112] το καθορίζει στα 200 ms, και τη μέτρηση "ground time", όπου στο έργο “tetrs – Tetromino Game Engine + Terminal Application” [113], το κομμάτι κλειδώνει, αν υπερβεί στο έδαφος το όριο των 2,25sec. Τέλος, γίνεται αναφορά για soft/hard drop στις υλοποιήσεις“Tetris The Grand Master 4 Absolute Eye” [114], “Tetris

(NES)” [108] που ορίζει soft drop ~16.7ms , στην “Four-tris” [109] όπου την ορίζει στα 32ms και στο “TGM Guide” [115] με τιμές ~16,7–33 ms.

Ο Πίνακας 2 παρουσιάζει τις χρονικές παραμέτρους που επιλέχθηκαν για την υλοποίηση του παιχνιδιού. Οι τιμές εκφράζονται σε milliseconds (ms) και αποτελούν αποτέλεσμα μελέτης προηγούμενων υλοποιήσεων και έχοντας ως στόχο την ομαλή, σταθερή και ευχάριστη λειτουργία του παιχνιδιού. Η επιλογή τους διασφαλίζει ότι η απόκριση των κομματιών, η ταχύτητα πτώσης και οι καθυστερήσεις στις μεταβάσεις θα διατηρούνται σε επίπεδα που ανταποκρίνονται στον κλασικό τρόπο παιχνιδιού, απορρίπτοντας πολύ μικρές ή πολύ μεγάλες τιμές που είτε θα δυσκολεύουν τον παίκτη, είτε θα μειώνουν την ένταση και το ρυθμό του παιχνιδιού.

Παράμετροι	Επεξήγηση
DAS (Delayed Auto Shift)	Αρχική καθυστέρηση πριν ξεκινήσει η συνεχής μετακίνηση κομματιού (δεξιά ή αριστερά). Επιτρέπει μονές κινήσεις με ακρίβεια
ARR (Auto Repeat Rate)	Ρυθμός συνεχούς μετακίνησης μετά το DAS. Ελέγχει την ταχύτητα επαναλαμβανόμενων κινήσεων.
ARE(Appearance Delay)	Χρόνος μεταξύ κλειδώματος προηγούμενου κομματιού και εμφάνισης νέου. Διατηρεί ομαλή ροή και αποφεύγει τη συμφόρηση
Falling speed	Ταχύτητα πτώσης κομματιών
Lock Delay	Χρόνος καθυστέρησης πριν το κομμάτι κλειδώσει όταν αγγίζει το έδαφος ή άλλο κομμάτι. Επιτρέπει περιστροφή/μετακίνηση για μικρές διορθώσεις
Line Clear Delay	Διάρκεια animation κατά το καθάρισμα γραμμής. Παρέχει οπτική ανατροφοδότηση στον παίκτη
Ground Time	Ο χρόνος που ένα κομμάτι παραμένει στο έδαφος πριν ξεκινήσει η διαδικασία του κλειδώματος. Συνήθως περιλαμβάνει το Lock Delay
Soft Drop	Επιτάχυνση πτώσης κομματιού ενώ πατιέται το κουμπί
Hard Drop	Άμεση πτώση και κλείδωμα κομματιού, χωρίς lock delay

Πίνακας 1 Επεξήγηση χρονικών παραμέτρων παιχνιδιού

Χρονικοί Παράμετροι	Τιμή σε ms
DAS	150
ARR	50
ARE	200
Falling Speed	500
Lock Delay	400
Line Clear Delay	200
Ground Time	1500
Soft/Hard Drop	30

Πίνακας 2 Επιθυμητές τιμές χρονικών παραμέτρων

Συμπερασματικά, η ανάλυση των κανόνων, των μηχανισμών και των χρονικών απαιτήσεων του Tetris επιτρέπει τη δημιουργία μιας απλοποιημένης και λειτουργικά ολοκληρωμένης υλοποίησης. Με τον καθορισμό της δομής του παιχνιδοχώρου, της συμπεριφοράς των Tetrominoes και των συνθηκών σύγκρουσης και τερματισμού, δημιουργείται ένα σταθερό πλαίσιο λειτουργίας που εξασφαλίζει προβλεψιμότητα και ομαλή αλληλεπίδραση. Παράλληλα, η επιλογή τεκμηριωμένων χρονικών παραμέτρων, βασισμένων σε καθιερωμένες υλοποιήσεις και πρότυπα, εγγυάται ότι ο ρυθμός του παιχνιδιού παραμένει ισορροπημένος και ευχάριστος για τον παίκτη. Επομένως, το τελικό σύστημα εξασφαλίζει αξιόπιστη λειτουργία και αποτελεσματικότητα.

3.2 Προσδιορισμός απαιτήσεων υλικού

Ο προσδιορισμός των απαιτήσεων υλικού αποτελεί θεμελιώδη βήμα σε οποιοδήποτε έργο ανάπτυξης ενσωματωμένων συστημάτων ή ψηφιακών εφαρμογών. Μέσω αυτής της διαδικασίας, καθορίζονται οι τεχνικές προδιαγραφές που πρέπει να πληροί η πλατφόρμα, ώστε να υποστηρίζει σωστά τις λειτουργίες της εφαρμογής, να ανταποκρίνεται στις επιδόσεις που απαιτούνται και να διασφαλίζεται η σταθερότητα και αξιοπιστία της λειτουργίας της. Οι απαιτήσεις υλικού προκύπτουν τόσο από τις λειτουργικές ανάγκες του συστήματος, όπως η επεξεργαστική ισχύς, η μνήμη και οι δυνατότητες εισόδου-εξόδου, όσο και από παράγοντες, όπως η συμβατότητα με εξωτερικά περιφερειακά, η τροφοδοσία, η προστασία από σφάλματα και η δυνατότητα μελλοντικών επεκτάσεων. Αξίζει να αναφερθεί ότι, σύμφωνα με βιβλιογραφικές αναφορές, υπάρχουν γενικά διάφορες μεθοδολογίες εκτίμησης πόρων, πριν τη σύνθεση και υλοποίηση των έργων, με αποτέλεσμα τη μείωση του χρόνου ανάπτυξης και βελτίωση της αξιοπιστίας του τελικού συστήματος [116], [117].

3.2.1 Ανάλυση απαιτήσεων υλικού και περιφερειακών συσκευών

Για την επιτυχή υλοποίηση του συστήματος, είναι απαραίτητη η προσεκτική επιλογή του υλικού και των περιφερειακών συσκευών που θα υποστηρίξουν όλες τις επιθυμητές λειτουργίες του παιχνιδιού. Στην ανάλυση που ακολουθεί, παρουσιάζονται τα βασικά στοιχεία, οι απαιτήσεις τους και οι παράμετροι που πρέπει να πληρούν, προκειμένου να διασφαλιστεί η ορθή λειτουργία, η σταθερότητα και η αξιοπιστία του συστήματος. Αναλυτικότερα:

- ***Πλακέτα ανάπτυξης FPGA***

Η επιλογή της κατάλληλης πλακέτας FPGA αποτελεί σημαντικό παράγοντα για την επιτυχή υλοποίηση ενσωματωμένων συστημάτων, καθώς καθορίζει τις δυνατότητες, τους περιορισμούς και τη συνολική απόδοση της εφαρμογής. Σχετικά με τη συγκεκριμένη υλοποίηση, το FPGA πρέπει να διαθέτει επαρκείς υπολογιστικούς πόρους (LUTs, flip-flops, block RAM), ώστε να υποστηρίζει όλες τις λειτουργίες του παιχνιδιού και πιθανές μελλοντικές επεκτάσεις. Συγκεκριμένα, τα *lookup tables (LUTs)* θα χρησιμοποιηθούν για την υλοποίηση της λογικής διαχείρισης του πλέγματος του παιχνιδιού, των μηχανών

καταστάσεων και των μονάδων σύγκρουσης των tetrominoes, καθώς και για την παραγωγή των γραφικών και των σημάτων HDMI. Τα *flip-flops (FFs)* είναι απαραίτητα για την αποθήκευση της κατάστασης των tetrominoes, των χρονομετρητών πτώσης και των εισόδων του χειριστηρίου, ενώ η *block RAM (BRAM)* θα χρησιμοποιηθεί για την αποθήκευση του πλέγματος του παιχνιδιού και των προσωρινών γραφικών.

Επιπλέον, για την ορθή απεικόνιση του παιχνιδιού στην οθόνη, το FPGA πρέπει να ενσωματώνει υψηλής συχνότητας *κύριο ρολόι* (system clock) για τη λογική του παιχνιδιού και να υποστηρίζει την παραγωγή κατάλληλων ρολογιών για το HDMI και για εσωτερικούς χρονιστές.

Σημαντική είναι η ύπαρξη, ικανών σε αριθμό, *pins I/O*, ώστε να μπορεί να συνδεθεί απευθείας το χειριστήριο και τα πιθανά buttons, αλλά και να υπάρχουν και άλλα διαθέσιμα για μελλοντικές επεκτάσεις. Τέλος, θα πρέπει να διαθέτει ενσωματωμένη *θύρα HDMI* ή διάταξη TMDS συμβατή με HDMI και δυνατότητα παραγωγής σήματος με χρονισμό που απαιτεί το HDMI.

- ***Χειρισμός Εισόδου μέσω Arcade Joystick/buttons***

Η επιλογή του *arcade joystick* και των *buttons* πρέπει να βασίζεται στη συμβατότητα με τα GPIO της πλακέτας FPGA και στη δυνατότητα αξιόπιστης μετάδοσης ψηφιακών σημάτων. Το joystick πρέπει να παρέχει διακριτές ψηφιακές εξόδους για κάθε εντολή του χρήστη, όπως κίνηση αριστερά και δεξιά, επιτάχυνση πτώσης, περιστροφή και εκκίνηση/παύση του παιχνιδιού. Επιπλέον, πρέπει να συνδέεται απευθείας στα pins της FPGA πλακέτας, χωρίς τη χρήση ενδιάμεσου μικροελεγκτή, και να υπάρχει δυνατότητα εφαρμογής pull-up ή pull-down αντιστάσεων για τη σταθερότητα των σημάτων. Έτσι, η σύνδεση του χειριστηρίου γίνεται χωρίς τη χρήση ψηφιακών πρωτοκόλλων και κάθε button ή κίνηση του χειριστηρίου αντιστοιχεί σε ένα ξεχωριστό ψηφιακό σήμα, αντιπροσωπεύοντας διαφορετική λειτουργία. Τα σήματα αυτά μεταδίδουν λογικό επίπεδο 1/0, ανάλογα με το αν το κουμπί είναι πατημένο ή όχι, δημιουργώντας τη βασική είσοδο ψηφιακού bit. Επιπλέον, απαιτείται υλοποίηση αποθορυβοποίησης (debouncing) , ώστε να αποφεύγονται ανεπιθύμητες πολλαπλές ενεργοποιήσεις κατά το πάτημα των κουμπιών.

- ***Έξοδος σε οθόνη***

Η επιλογή της *οθόνης για έξοδο μέσω HDMI* απαιτεί την αξιολόγηση τόσο της ανάλυσης και των χρονισμών όσο και της συμβατότητας με τις δυνατότητες του FPGA. Η οθόνη πρέπει να υποστηρίζει τις αναλύσεις και το ρυθμό ανανέωσης που μπορεί να παράγει η πλακέτα, για ομαλή απεικόνιση χωρίς παραμορφώσεις. Απαιτείται, επίσης, κωδικοποίηση σε μορφή TMDS, είτε μέσω ενσωματωμένου στο FPGA κυκλώματος είτε μέσω εξωτερικού HDMI transmitter. Επιπλέον, η οθόνη που θα επιλεχθεί, πρέπει να έχει χαμηλή καθυστέρηση και να μπορεί να δεχτεί συνεχές σήμα από το FPGA, χωρίς buffering που θα επηρεάζει την απόκριση του παιχνιδιού.

- ***Τροφοδοσία και Συνδέσεις***

Το σύστημα απαιτεί σταθερή τροφοδοσία για την FPGA πλακέτα και τα περιφερειακά της, σύμφωνα με τις προδιαγραφές τους. Οι συνδέσεις περιλαμβάνουν φυσικά καλώδια και θύρες για επικοινωνία μεταξύ των συσκευών, όπως HDMI καλώδιο για την οθόνη και καλώδια σύνδεσης τύπου dupont για joystick και buttons.

- ***Εργαλεία Ανάπτυξης***

Εκτός από τις υλικές απαιτήσεις, η ανάπτυξη του συστήματος απαιτεί κατάλληλα εργαλεία και λογισμικά που υποστηρίζουν τον πλήρη κύκλο σχεδίασης, από τον κώδικα HDL έως τη μεταφόρτωση στο υλικό. Η επιλογή του FPGA πρέπει να συνοδεύεται από εργαλεία ανάπτυξης, όπως *Quartus* για Intel/Altera [118] ή *Vivado* για Xilinx [120], επιτρέποντας τον σχεδιασμό ψηφιακών κυκλωμάτων σε γλώσσες περιγραφής υλικού όπως VHDL και Verilog. Ένα εργαλείο προσομοίωσης, όπως το *ModelSim* [119], επιτρέπει τη προσομοίωση συμπεριφοράς και χρονισμού, διευκολύνοντας την ανίχνευση σφαλμάτων πριν την υλοποίηση στο υλικό. Ένας σημαντικός παράγοντας στην επιλογή των εργαλείων ανάπτυξης είναι και το κόστος των πλήρων εκδόσεων. Οι εμπορικές εκδόσεις τόσο του Quartus όσο και του ModelSim μπορεί να είναι ιδιαίτερα δαπανηρές, ωστόσο οι δωρεάν εκδόσεις (π.χ. Quartus Prime Lite και ModelSim Intel FPGA Starter Edition) παρέχουν όλες τις βασικές λειτουργίες που απαιτούνται για τη συγκεκριμένη υλοποίηση. Η επιλογή δωρεάν εργαλείων καθιστά την ανάπτυξη πιο προσιτή, χωρίς να περιορίζει τις δυνατότητες για σχεδίαση, σύνθεση ή προσομοίωση.

Ο συνολικός καθορισμός των απαιτήσεων υλικού αποτελεί βασικό βήμα για την επιτυχημένη υλοποίηση της εφαρμογής, καθώς εξασφαλίζει ότι η επιλεγμένη πλακέτα ανάπτυξης μπορεί να υποστηρίξει όλες τις απαραίτητες λειτουργίες με σταθερότητα, αξιοπιστία και επεκτασιμότητα. Μέσα από την επιλογή κατάλληλης FPGA πλακέτας, αξιόπιστου συστήματος εισόδου, συμβατής οθόνης και σταθερής τροφοδοσίας, διαμορφώνεται ένα ολοκληρωμένο τεχνικό πλαίσιο που επιτρέπει την ομαλή λειτουργία του παιχνιδιού και την ορθή αναπαράσταση των γραφικών. Παράλληλα, η χρήση κατάλληλων εργαλείων ανάπτυξης και προσομοίωσης εξασφαλίζει έναν αποδοτικό κύκλο σχεδίασης. Στο επόμενο στάδιο θα ακολουθήσει αναζήτηση παρόμοιων υλοποιήσεων, ώστε να προσδιοριστούν κατά προσέγγιση οι ελάχιστες απαιτήσεις για την συγκεκριμένη υλοποίηση.

3.2.2 Έρευνα παρόμοιων υλοποιήσεων και αξιολόγηση απαιτήσεων υλικού

Η υλοποίηση του κλασικού παιχνιδιού Tetris σε FPGA έχει αποτελέσει αντικείμενο μελέτης σε πολλές ακαδημαϊκές εργασίες και ανοικτές υλοποιήσεις, παρέχοντας κρίσιμα στοιχεία για τον υπολογισμό των απαιτούμενων πόρων και την οργάνωση του συστήματος. Ενδεικτικά, μπορούν να αναφερθούν ορισμένες υλοποιήσεις, οι οποίες έδωσαν μια ξεκάθαρη εικόνα για τις απαιτήσεις του συστήματος.

Η ακαδημαϊκή εργασία με τίτλο “A fully embedded Tetris game application in VHDL’’ [21] παρουσιάζει μία υλοποίηση σε VHDL με χρήση του λογισμικού Altera Quartus II και Modelsim. Η πλακέτα ανάπτυξης που χρησιμοποιήθηκε ήταν η DE0 board με 50MHz on-board clock, 15.408 Logic Elements, συνολικά 516.096 bits RAM, δηλαδή ~64 KB συνολικά block RAM. Η διαχείριση μνήμης έγινε μέσω tile-mapped αναπαράστασης, μειώνοντας σημαντικά την ανάγκη για μεγάλο frame buffer, όπως επίσης δεν έγινε χρήση της εξωτερικής SRAM. Για VGA out, χρησιμοποιήθηκε system clock 50 MHz clock, από το οποίο παράγεται pixel clock 25 MHz για 640×480 60Hz.

Επιπλέον, η υλοποίηση “primiano/tetris-vhdl” [121], επιβεβαιώνει ότι το Tetris μπορεί να υλοποιηθεί σε “entry-level” FPGA boards χωρίς χρήση CPU, χρησιμοποιώντας αρχιτεκτονική υλικού με modules για control, datapath, view, timing. Η πλακέτα που χρησιμοποιείται είναι η Altera DE0 [122], με 20.000 LEs, 8 MB SDRAM, 512 KB SRAM, ενώ το σύστημα τροφοδοτείται από system clock 50MHz, με pixel clock 25 MHz για VGA out.

Στη συνέχεια, η εργασία με τίτλο “Implementation of Tetris Game on a FPGA” [22], αναφέρεται σε μια υλοποίηση Tetris σε FPGA χρησιμοποιώντας VHDL, με modular αρχιτεκτονική που περιλαμβάνει modules για τη λογική του παιχνιδιού, εισόδους από switches και οπτική έξοδο μέσω VGA. Γίνεται χρήση της πλακέτας DIGILENT SPARTAN-3 Starter Board Xilinx XC3S200 με 50MHz on-board clock, 4.320 LE, μνήμη RAM 216.000 bits. Το λογισμικό που χρησιμοποιήθηκε για προγραμματισμό και σύνθεση ήταν το Xilinx ISE.

Τέλος, η υλοποίηση “Tetris2Players” [123] σε FPGA αποτελεί ένα πλήρες σύστημα δύο παικτών με modular αρχιτεκτονική, περιλαμβάνοντας μονάδες για την απεικόνιση του φόντου, των αντικειμένων και γραμμές προσωρινής αποθήκευσης για την εμφάνιση των τετραγώνων σε οθόνη VGA με pixel clock 25 MHz. Έγινε χρήση πλακέτας με 3.525 LUTs, 22 block RAM, 4 DSP και system clock 99.5 MHz, ενώ ο graphic accelerator τρέχει στα 134 MHz. Το λογισμικό ανάπτυξης που χρησιμοποιήθηκε ήταν το Xilinx ISE.

Συμπερασματικά, η επιλογή FPGA με μεσαία χωρητικότητα παρέχει πλήρη λειτουργικότητα, , HDMI ή VGA output, buffer, και πιθανή επέκταση σε περισσότερα χαρακτηριστικά. Ταυτόχρονα, οι παρατηρήσεις από όχι τόσο απαιτητικές υλοποιήσεις, δείχνουν ότι για ένα βασικό Tetris, αρκεί ένα μικρότερης κλίμακας FPGA, γεγονός που επιτρέπει ευελιξία στην επιλογή υλικού ανάλογα με τις απαιτήσεις. Η συνδυαστική αξιολόγηση των διαφορετικών προσεγγίσεων, με αναφορά σε LUTs, BRAM, DSP, system clock και pixel clock, τεκμηριώνει πλήρως τις επιλογές σχεδίασης και τις απαιτήσεις των πόρων.

3.2.3 Επιθυμητές προδιαγραφές συστήματος

Με βάση τη μελέτη παρόμοιων υλοποιήσεων FPGA Tetris και την αξιολόγηση των απαιτούμενων πόρων, γίνεται προσδιορισμός των επιθυμητών προδιαγραφών με στόχο τη διασφάλιση αξιόπιστης λειτουργίας του παιχνιδιού, ομαλής απεικόνισης βίντεο και αποτελεσματικής αλληλεπίδρασης με τον χρήστη. Μια σύντομη αναφορά φαίνεται στον Πίνακα 3.

Η υλοποίηση του παιχνιδιού στο FPGA απαιτεί τη χρήση ικανών σε αριθμό λογικών στοιχείων (logic elements), που θα χρησιμοποιηθούν για την υλοποίηση των timers, της λογικής των κινήσεων των τετραγωνιδίων, των μετρητών και της διαχείρισης του video output. Επιπλέον, η BRAM χρησιμοποιείται για την αποθήκευση του πλέγματος του

παιχνιδιού και άλλων βοηθητικών δομών. Για μια απλή σχετικά υλοποίηση εκτιμάται ότι η χρήση των πόρων θα είναι 5.000-10.000 LEs και 64–128 KB BRAM

Το βασικό ρολόι του συστήματος (system clock) ορίζεται στα 50 MHz, το οποίο παρέχεται συνήθως από τις περισσότερες FPGA πλακέτες ανάπτυξης. Από το βασικό ρολόι παράγονται υπο-ρολόγια μέσω PLL, τα οποία χρησιμοποιούνται τόσο για τον έλεγχο των χρονισμών του παιχνιδιού, όπως DAS, ARR, lock, ARE, gravity και line clearing, όσο και για τη δημιουργία του pixel clock που απαιτείται για την έξοδο βίντεο. Για παράδειγμα, σε ανάλυση VGA 640×480 60 Hz απαιτείται pixel clock περίπου 25 MHz. Η χρήση PLL είναι κρίσιμη ώστε να παραχθεί σταθερό pixel clock για την ορθή λειτουργία του video output.

Επιπλέον, απαιτείται επαρκής αριθμός I/O pins για τη σύνδεση των περιφερειακών συσκευών. Συγκεκριμένα, τα κουμπιά του arcade χειριστηρίου και το joystick χρειάζονται συνήθως 5–10 pins, ανάλογα με τον αριθμό των λειτουργιών που υποστηρίζονται (π.χ. αριστερά/δεξιά, κάτω, περιστροφή, start). Επιπλέον, η HDMI έξοδος χρησιμοποιεί ζεύγη διαφορικών σημάτων για τη μετάδοση των δεδομένων και του pixel clock (συνήθως 4 ζεύγη) και μερικά pins για I^2C επικοινωνία με τον HDMI transmitter. Επομένως, ένας καλός αριθμός I/O pins είναι 20-40 λαμβάνοντας υπόψη και μελλοντικές προεκτάσεις.

Όσον αφορά το video output, η υλοποίηση θα υποστηρίζει HDMI 60 Hz με ανάλυση τουλάχιστον 640×480. Η επιλογή της ανάλυσης καθορίζει τον απαιτούμενο pixel clock και τη μνήμη για την αποθήκευση των pixels. Για το HDMI απαιτείται TMDS encoding, είτε μέσω έτοιμου IP core είτε μέσω εξωτερικού HDMI transmitter, ενώ η σωστή αρχικοποίηση μέσω I^2C/EDID διασφαλίζει την επικοινωνία με την οθόνη.

Τέλος, η τροφοδοσία του συστήματος πρέπει να παρέχεται από 5V power supply, το οποίο είναι τυπικό για τις περισσότερες αναπτυξιακές πλακέτες και εξασφαλίζει σταθερή λειτουργία όλων των υποσυστημάτων, συμπεριλαμβανομένων των FPGA, των θυρών I/O και των περιφερειακών συσκευών, χωρίς να προκαλεί υπερφόρτωση ή αστάθεια στην τάση λειτουργίας.

Παράμετροι	Προδιαγραφή
Logic Elements	5.000-10.000 LEs
BRAM	64-128 KB
System clock	50 MHz
Pixel clock	25 MHz VGA 640×480 60Hz
I/O Pins	20-40
Input controls	joystick/buttons
Video output	HDMI 60 Hz, 640×480
Power Supply	5V USB/adapter

Πίνακας 3 Επιθυμητές προδιαγραφές υλικού

Συμπερασματικά, ο συνολικός αριθμός LEs κυμαίνεται σε επίπεδα της τάξης των 5.000 έως 10.000 και system clock στα 50MHz. Η ενσωματωμένη μνήμη BRAM χρησιμοποιείται για την αποθήκευση της διάταξης του παιχνιδιού, των γραμματοσειρών και άλλων βοηθητικών δομών, με τις συνολικές απαιτήσεις να είναι στα 64-128 KB. Επιπλέον, απαιτείται ένας μικρός αριθμός εισόδων/εξόδων, κατά προσέγγιση 20-40, με χειρισμό εισόδου από joystick/buttons και έξοδο σε οθόνη HDMI. Η τροφοδοσία του συστήματος προτείνεται στα 5V, εξασφαλίζοντας σταθερή λειτουργία όλων των υποσυστημάτων. Με βάση αυτές τις προδιαγραφές, το επόμενο βήμα αφορά τη διαδικασία επιλογής κατάλληλης FPGA πλακέτας, ώστε να καλύπτονται όλες οι απαιτήσεις του έργου σε πόρους, μνήμη και I/O.

3.3 Μεθοδολογία επιλογής πλακέτας ανάπτυξης FPGA

Η επιλογή του κατάλληλου υλικού αποτελεί κρίσιμο στάδιο στην ανάπτυξη ενός ενσωματωμένου συστήματος βασισμένου σε FPGA. Για τις ανάγκες του έργου, που αφορά την υλοποίηση του παιχνιδιού Tetris με είσοδο από arcade joystick και έξοδο εικόνας μέσω HDMI, εφαρμόστηκαν τεχνικές αξιολόγησης με στόχο την επιλογή

πλακέτας που να εξασφαλίζει αξιοπιστία, επεκτασιμότητα και επαρκή υπολογιστική ισχύ, ενώ ταυτόχρονα να υποστηρίζει όλες τις τεχνικές απαιτήσεις του συστήματος.

Η διαδικασία επιλογής της κατάλληλης πλακέτας ανάπτυξης βασίστηκε σε αναζήτηση και αξιολόγηση διαθέσιμων αναπτυξιακών πλακετών που μπορούν να καλύψουν τις απαιτήσεις του έργου. Αρχικά, έγινε καθορισμός των κριτηρίων επιλογής, ώστε η έρευνα να είναι στοχευμένη και να αποκλείει λύσεις που δεν πληρούν τις τεχνικές και λειτουργικές ανάγκες του έργου. Τα βασικά κριτήρια που τέθηκαν περιλαμβάνουν την ύπαρξη θύρας HDMI ή δυνατότητα εύκολης ενσωμάτωσης εξόδου υψηλής ανάλυσης, την επάρκεια των διαθέσιμων πόρων του FPGA, την αξιοπιστία του κατασκευαστή, την ευκολία προμήθειας, καθώς και το συνολικό κόστος της πλατφόρμας.

Στο πρώτο στάδιο της αναζήτησης πραγματοποιήθηκε έρευνα των προϊόντων που παρέχουν οι κυριότεροι κατασκευαστές FPGA εκπαιδευτικών και ερευνητικών πλακετών, όπως η Terasic (Intel/Altera-based boards) [124] και η Digilent (Xilinx-based boards) [125]. Η επιλογή τους έγινε λόγω της διεθνούς αναγνώρισης, της τεχνικής τεκμηρίωσης που παρέχουν και της μεγάλης κοινότητας χρηστών που υποστηρίζει το υλικό τους.

Ακολούθως, προτεραιότητα είχαν οι πλακέτες που διέθεταν ενσωματωμένη θύρα HDMI ή υποστήριξη για HDMI transmitter (όπως ADV7513), καθώς η υλοποίηση προϋποθέτει απεικόνιση σε σύγχρονη οθόνη υψηλής ανάλυσης. Συστήματα που παρείχαν αποκλειστικά VGA έξοδο ή απαιτούσαν πρόσθετες επεκτάσεις για HDMI δεν απορρίφθηκαν άμεσα, αλλά θεωρήθηκαν λιγότερο πρακτικές λόγω αυξημένης πολυπλοκότητας και επιπλέον κόστους. Παράλληλα εξετάστηκε η διαθέσιμη μνήμη, που είναι απαραίτητη για την αποθήκευση γραφικών δεδομένων ή framebuffer, καθώς και η ποσότητα των διαθέσιμων GPIO pins, ώστε να εξασφαλίζεται η απευθείας σύνδεση του arcade χειριστηρίου.

Επιπλέον, ιδιαίτερη σημασία δόθηκε στο κόστος των πλακετών. Οι πλακέτες έπρεπε να είναι οικονομικά προσιτές, με δωρεάν ή δοκιμαστικές εκδόσεις των εργαλείων ανάπτυξης (Quartus Prime ή Xilinx Vivado). Απορρίφθηκαν αυτές που απαιτούσαν ακριβές επαγγελματικές άδειες λογισμικού ή ειδικό προγραμματιστικό εξοπλισμό. Η ύπαρξη δωρεάν υποστήριξης ήταν επίσης σημαντικός παράγοντας, καθώς μειώνει τον χρόνο ανάπτυξης και οδηγεί σε πιο ασφαλή υλοποίηση.

Τέλος, η επιλογή επηρεάστηκε και από τη διαθεσιμότητα της πλακέτας στην αγορά, καθώς η υλοποίηση έπρεπε να ολοκληρωθεί εντός συγκεκριμένης προθεσμίας. Για τον λόγο αυτό, εξετάστηκαν αποκλειστικά πλακέτες που βρίσκονται σε συνεχή παραγωγή και

είναι άμεσα διαθέσιμες από επίσημους διανομείς. Με βάση όλα τα παραπάνω, διαμορφώθηκε μια τελική ομάδα υποψήφιων πλακετών, οι οποίες συγκρίθηκαν ως προς τα τεχνικά χαρακτηριστικά, την καταλληλότητά τους για υλοποίηση παιχνιδιού, καθώς και το κόστος σε σχέση με την απόδοσή τους.

3.3.1 Αξιολόγηση υποψήφιων πλακετών και τελική επιλογή

Η διαδικασία επιλογής της κατάλληλης πλακέτας ανάπτυξης ξεκίνησε με την αξιολόγηση πλακετών της εταιρείας Terasic και της εταιρείας Digilent, δεδομένης της εκτεταμένης χρήσης τους σε πανεπιστημιακές και ερευνητικές εφαρμογές. Έτσι στις υποψήφιες επιλογές συμπεριλήφθηκαν οι DE10-Lite [126], DE10-Nano[127], DE23-Lite [128] και Arty A7-100T [129] (Πίνακας 4), οι οποίες εξετάστηκαν ως προς τις τεχνικές δυνατότητες και τη συμβατότητά τους με τις απαιτήσεις του συστήματος.

Η DE10-Lite εξετάστηκε, αρχικά, ως οικονομική λύση, με ικανούς πόρους για βασικά συστήματα λογικής. Ωστόσο, η απουσία ενσωματωμένης θύρας HDMI και η περιορισμένη εξωτερική μνήμη την καθιστούν λιγότερο κατάλληλη για ένα σύστημα απαιτήσεων πραγματικού χρόνου με επεξεργασία και απεικόνιση γραφικών. Αντίθετα, οι DE10-Nano και DE23-Lite προσφέρουν σημαντικά πλεονεκτήματα ως προς την ύπαρξη ενσωματωμένης εξόδου HDMI, την υποστήριξη εξωτερικής μνήμης (DDR3 ή SDRAM), το μεγάλο εύρος GPIO connectors για απευθείας σύνδεση του joystick και τη διάθεση υψηλών πόρων και DSP blocks. Και οι τρεις συνοδεύονται από αξιόπιστα εργαλεία ανάπτυξης (Intel Quartus/Modelsim).

Η Arty A7-100T προσφέρει υψηλούς πόρους, αξιόπιστα εργαλεία ανάπτυξης μέσω του Vivado, καθώς και εκτεταμένες επιλογές I/O. Στα μειονεκτήματα είναι η μη διάθεση ενσωματωμένης θύρας HDMI, γεγονός που επιβάλλει τη χρήση πρόσθετου HDMI μετατροπέα για την υλοποίηση σήματος TMDS. Αυτό αυξάνει τον βαθμό πολυπλοκότητας της ανάπτυξης και επηρεάζει το τελικό κόστος.

Επιπλέον, σχετικά με τη διαθεσιμότητα των παραπάνω πλακετών, την περίοδο αναζήτησης ήταν όλες διαθέσιμες εκτός από την DE23-Lite, όπου χρειαζόταν παραγγελία αλλά με σχετικά σύντομο χρόνο παράδοσης. Όσον αφορά το κόστος, η εταιρεία Terasic εφαρμόζει σε αυτές τις πλακέτες ειδικές τιμές (academic price) για πανεπιστημιακούς και εκπαιδευτικούς φορείς, ενώ η Arty A7-100T είχε, συγκριτικά, το μεγαλύτερο κόστος.

Δεδομένου ότι η εφαρμογή απαιτεί σταθερή και ποιοτική έξοδο HDMI, απευθείας σύνδεση arcade joystick χωρίς εξειδικευμένα πρωτόκολλα, καθώς και επαρκείς πόρους για υλοποίηση γραφικού υποσυστήματος, επιλέθχηκε η πλακέτα *Terasic DE23-Lite*. Η DE23-Lite διαθέτει υψηλούς πόρους και ενσωματωμένο HDMI transmitter, γεγονός που επιτρέπει την απευθείας δημιουργία σήματος υψηλής ανάλυσης. Αυτό μειώνει σημαντικά την πολυπλοκότητα της υλοποίησης και εξασφαλίζει αξιόπιστη λειτουργία. Επιπλέον, η παρουσία 64 MB SDRAM επιτρέπει την υλοποίηση framebuffer και άλλων γραφικών δομών που απαιτούνται σε ένα παιχνίδι τύπου Tetris. Η ύπαρξη μεγάλου αριθμού GPIO pins επιτρέπει την απευθείας σύνδεση του χειριστηρίου arcade μέσω ψηφιακών γραμμών χωρίς την ανάγκη πρωτοκόλλου επικοινωνίας, διευκολύνοντας την ανάπτυξη και πιθανές μελλοντικές επεκτάσεις. Τέλος, είναι διαθέσιμη με σύντομο χρόνο αναμονής (την περίοδο αναζήτησης) και τα εργαλεία ανάπτυξης (Intel Quartus Pro, ModelSim Starter Edition) προσφέρονται σε δωρεάν έκδοση. Λαμβάνοντας υπόψη όλα τα παραπάνω, η DE23-Lite αξιολογήθηκε ως η πλέον κατάλληλη για τις ανάγκες της παρούσας εργασίας αλλά και για μελλοντικές επεκτάσεις και βελτιστοποιήσεις.

Πλακέτα	Πόροι	HDMI output	Εργαλεία Ανάπτυξης	Κόστος	Διαθεσιμότητα
DE10-Lite (Terasic)	- Intel MAX 10 - 50.000LEs - 1.638Kbit Memory - 4 PLLs - 64MB SDRAM - 2x20 GPIO Connector - 5V DC input	OXI - Μόνο VGA onboard - Για HDMI χρειάζεται εξωτερική λύση	Intel/Quartus Prime Lite ModelSim Starter Edition	140EUR (Academic 83EUR)	Άμεση
DE10-Nano (Terasic)	- Intel/Altera Cyclone V SoC (800MHz Dual-core ARM Cortex-A9 processor) - 110.000LEs - Three 50MHz clock sources from the clock generator - 1GB SDRAM - 2x40 GPIO Connector - 5V DC input	NAI -HDMI TX (DVI/ HDCP)	Intel/Quartus ModelSim Starter Edition	225EUR (Academic 190EUR)	Άμεση
DE23-Lite (Terasic)	- Intel Altera Agilex3 - 135.110LEs - 6.89 Mbit M20K - 4 IO PLL, 8 Fabric I/O PLL - 64MB SDRAM - 2x20 GPIO Connector - 5V DC input	NAI - HDMI Output with Audio	Intel/Quartus Pro ModelSim Starter Edition	169EUR (Academic 133EUR)	Με παραγγελία
Arty A7-100T (Digilent)	- Artix-7 XC7A100TCSG324-1 - 101.440 LEs - 240 DSP Slices - PMOD, general I/O - 4860 Kbits Memory	OXI - Χρειάζεται εξωτερικό μετατροπέα για HDMI	Xilinx/ Vivado Design Vivado Simulator	258EUR	Άμεση

Πίνακας 4 Συγκριτικός πίνακας πλακετών FPGA

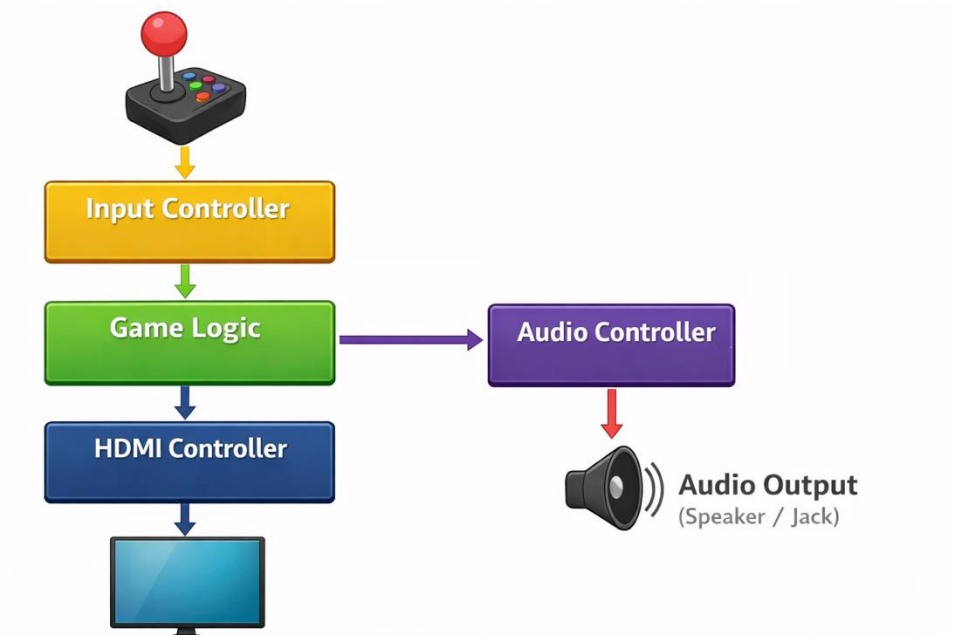
4. Σχεδίαση συστήματος και ανάπτυξη έργου

Η σχεδίαση του συστήματος αποτελεί σημαντικό στάδιο στην ανάπτυξη του παιχνιδιού Tetris σε FPGA, καθώς καθορίζει τη δομή, τη λειτουργικότητα και την απόδοση της εφαρμογής. Στο παρόν κεφάλαιο παρουσιάζεται η προσέγγιση που ακολουθήθηκε, με σαφή διαχωρισμό υποσυστημάτων, ώστε η κάθε μονάδα να μπορεί να αναπτυχθεί, επαληθευτεί και ενσωματωθεί ανεξάρτητα. Η ανάλυση καλύπτει τόσο τις απαιτήσεις εισόδων και εξόδων όσο και τη λογική διαχείρισης των δεδομένων, με στόχο τη δημιουργία ενός επεκτάσιμου και αποδοτικού συστήματος που μπορεί να υλοποιηθεί με ασφάλεια και ακρίβεια σε πραγματικό χρόνο. Παρακάτω, παρουσιάζονται η αρχιτεκτονική, η αναλυτική παρουσίαση των υποσυστημάτων, ο σχεδιασμός και η τελική υλοποίηση του συστήματος σε HDL.

4.1 Αρχιτεκτονική συστήματος

Η αρχιτεκτονική του συστήματος καθορίζει τη συνολική δομή και περιγράφει τον τρόπο με τον οποίο τα επιμέρους υποσυστήματα συνεργάζονται για την επίτευξη της λειτουργικότητας του παιχνιδιού Tetris σε FPGA. Στο πλαίσιο της παρούσας εργασίας, ο σχεδιασμός της αρχιτεκτονικής βασίστηκε στις απαιτήσεις της εφαρμογής και σχεδιάστηκε έτσι ώστε να υποστηρίζει την αξιόπιστη και συγχρονισμένη επεξεργασία πολλαπλών λειτουργιών σε πραγματικό χρόνο, όπως η ανάγνωση των εισόδων από το χειριστήριο τύπου arcade, η εκτέλεση της λογικής του παιχνιδιού, η έξοδος εικόνας μέσω διεπαφής HDMI και η παραγωγή ήχου. Για τον σκοπό αυτό, το σύστημα διασπάται σε ξεχωριστά και λειτουργικά ανεξάρτητα υποσυστήματα, καθένα από τα οποία έχει συγκεκριμένο ρόλο αλλά επικοινωνεί και με τα υπόλοιπα υποσυστήματα. Η προσέγγιση αυτή διευκολύνει την ανάλυση, τη σχεδίαση και την επαλήθευση του συστήματος, ενώ επιτρέπει την απομόνωση σφαλμάτων καθώς και τη μελλοντική επέκταση της εφαρμογής με πρόσθετες λειτουργίες.

Το block diagram της αρχιτεκτονικής του συστήματος (Εικόνα 29), απεικονίζει τη λογική οργάνωση των βασικών υποσυστημάτων που συνεργάζονται για την υλοποίηση του παιχνιδιού Tetris σε FPGA. Η σχεδίαση ακολουθεί κατεύθυνση ροής από τις εισόδους του χρήστη προς τις μονάδες επεξεργασίας και, στη συνέχεια, προς τις μονάδες εξόδου εικόνας και ήχου.



Εικόνα 29 Block Diagram συστήματος

Στην ανώτερη βαθμίδα του διαγράμματος βρίσκεται το χειριστήριο τύπου arcade (joystick, buttons), όπου τα σήματα που παράγονται από τις επιλογές του χρήστη, μεταφέρονται στο υποσύστημα εισόδου (Input Controller). Η μονάδα αυτή διασφαλίζει ότι οι εντολές του χρήστη μετατρέπονται σε έγκυρα και χρονικά σταθερά σήματα ελέγχου πριν προωθηθούν στο υποσύστημα ελέγχου του παιχνιδιού .

Στη συνέχεια, το υποσύστημα ελέγχου παιχνιδιού (Game Logic) αποτελεί το κεντρικό στοιχείο της αρχιτεκτονικής και είναι υπεύθυνο για την υλοποίηση των κανόνων του παιχνιδιού. Σε αυτό πραγματοποιείται η διαχείριση της κίνησης και περιστροφής των tetrominoes, ο έλεγχος συγκρούσεων, η ανίχνευση πλήρων γραμμών και η ενημέρωση της κατάστασης του παιχνιδιού.

Το υποσύστημα απεικόνισης HDMI (HDMI Controller) λαμβάνει δεδομένα από το υποσύστημα ελέγχου παιχνιδιού και είναι υπεύθυνο για τη δημιουργία των σημάτων χρονισμού και συγχρονισμού που απαιτούνται για την απεικόνιση σε ψηφιακή οθόνη, εξασφαλίζοντας σταθερή και ομαλή έξοδο εικόνας μέσω της διεπαφής HDMI.

Παράλληλα, υπάρχει το υποσύστημα ήχου (Audio Controller), το οποίο ενεργοποιείται από γεγονότα που παράγονται στο υποσύστημα ελέγχου παιχνιδιού. Το υποσύστημα ήχου λαμβάνει τα σήματα και τα μεταφράζει σε εντολές, όπου παράγεται ο ήχος μέσω τεχνικών όπως PWM ή απλή ψηφιακή σύνθεση. Η έξοδος του υποσυστήματος οδηγείται σε ηχείο ή έξοδο ακουστικών, χωρίς να επηρεάζεται η λειτουργία της απεικόνισης.

Επομένως, η επιλεγμένη αρχιτεκτονική εξασφαλίζει προβλέψιμη συμπεριφορά, χαμηλή καθυστέρηση και αξιόπιστη λειτουργία σε πραγματικό χρόνο, δίνοντας τη δυνατότητα για μελλοντικές επεκτάσεις ή υλοποίηση άλλων ρετρό παιχνιδιών στο ίδιο πλαίσιο.

Στις επόμενες ενότητες ακολουθεί αναλυτική παρουσίαση κάθε υποσυστήματος, αναλύοντας τη συνολική λειτουργική δομή τους και τον τρόπο διασύνδεσής τους.

4.1.1 Υποσύστημα εισόδου (Input Controller)

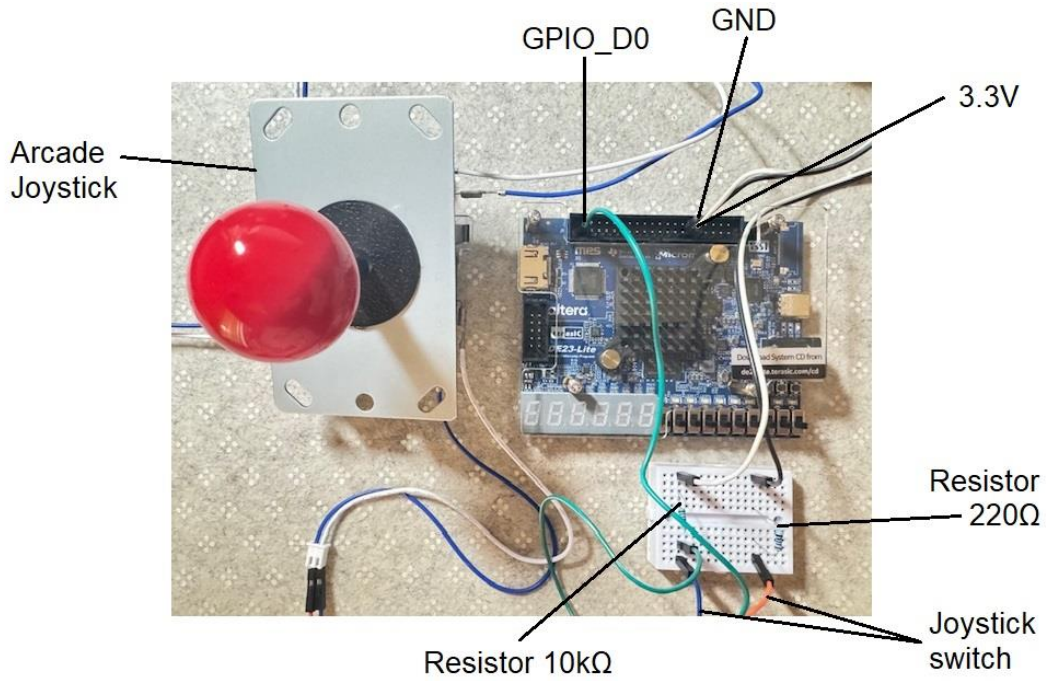
Το υποσύστημα εισόδου (Input Controller) είναι υπεύθυνο για την ανίχνευση των ενεργοποιήσεων που προέρχονται από το χειριστήριο τύπου arcade, το οποίο περιλαμβάνει ένα μοχλό κατεύθυνσης τεσσάρων κατευθύνσεων και ένα σύνολο κουμπιών. Η λειτουργία του υποσυστήματος αυτού είναι καθοριστική, καθώς αναλαμβάνει να ανιχνεύσει και να ερμηνεύσει τα σήματα από τις φυσικές εισόδους, μετατρέποντας τα σε λογικές εντολές για το υποσύστημα ελέγχου του παιχνιδιού.

Το υποσύστημα εισόδου αποτελείται από δύο κύρια επίπεδα: το υλικό (hardware) και την υλοποίηση σε γλώσσα HDL. Σε επίπεδο υλικού, οι φυσικές είσοδοι του χειριστηρίου, δηλαδή ο μοχλός κατεύθυνσης τεσσάρων κατευθύνσεων και τα κουμπιά, συνδέονται απευθείας με τα pins του FPGA μέσω pull-up αντιστάσεων, ώστε να διασφαλίζεται η σταθερή ανάγνωση των ψηφιακών σημάτων. Επίσης, μπορούν να χρησιμοποιηθούν προστατευτικές αντιστάσεις μεταξύ του διακόπτη και της γείωσης (GND), για την αποφυγή υπερτάσεων ή βραχυκυκλωμάτων και την προστασία των εισόδων του FPGA. Κάθε κατεύθυνση του μοχλού και κάθε κουμπί αντιστοιχεί σε ένα συγκεκριμένο pin εισόδου, δημιουργώντας έναν άμεσο και αξιόπιστο τρόπο επικοινωνίας με το ψηφιακό περιβάλλον της πλακέτας.

Σε επίπεδο HDL υλοποίησης, στόχος είναι η μετατροπή των ασύγχρονων σημάτων που προέρχονται από το χειριστήριο σε αξιόπιστες, συγχρονισμένες λογικές εντολές για το σύστημα. Η υλοποίηση περιλαμβάνει στάδια συγχρονισμού των εισόδων με το ρολόι του FPGA, μηχανισμούς αποθορυβοποίησης για την αντιμετώπιση της αναπήδησης των επαφών των διακοπτών και λογική αποκωδικοποίησης που αντιστοιχίζει τις φυσικές ενεργοποιήσεις σε λειτουργικές εντολές. Όλα τα παραγόμενα σήματα εξόδου είναι πλήρως συγχρονισμένα και διαμορφωμένα, ώστε να μπορούν να χρησιμοποιηθούν άμεσα από το υποσύστημα ελέγχου του παιχνιδιού, εξασφαλίζοντας ομαλή και προβλέψιμη αλληλεπίδραση μεταξύ χρήστη και ψηφιακού συστήματος.

Στο πλαίσιο του σχεδιασμού και επαλήθευσης του υποσυστήματος εισόδου, υλοποιήθηκαν δοκιμαστικές εφαρμογές ανεξάρτητες από τη λογική του παιχνιδιού. Η επιλογή αυτή αποσκοπούσε στον απομονωμένο έλεγχο της ορθής λειτουργίας του υποσυστήματος, χωρίς την επιβάρυνση της πολυπλοκότητας που διαθέτει η πλήρης υλοποίηση. Η δοκιμαστική εφαρμογή περιελάμβανε, αρχικά, την προς τα πάνω χρήση της κίνησης του μοχλού, όπου αυτή θα επιβεβαιωνόταν με το άναμμα ενός led της πλακέτας. Σε περίπτωση επιτυχούς λειτουργίας της συγκεκριμένης δοκιμής, η ίδια μεθοδολογία θα μπορούσε να επεκταθεί με παρόμοιο τρόπο και για τις υπόλοιπες κατευθύνσεις του μοχλού.

Σχετικά με τη φυσική σύνδεση του χειριστηρίου, αυτή έγινε απευθείας με την πλακέτα, παρακάμπτοντας τον ενσωματωμένο USB HID encoder, ώστε να επιτευχθεί άμεση και ακριβής ανάγνωση των ψηφιακών σημάτων χωρίς την καθυστέρηση και την πολυπλοκότητα του πρωτοκόλλου USB. Η συνδεσμολογία που υλοποιήθηκε (Εικόνα 30) αφορά τη σύνδεση ενός μόνο switch του χειριστηρίου που αφορά την προς τα πάνω κίνηση του μοχλού, με τη βοήθεια ενός breadboard, μερικών καλωδίων τύπου dupont και αντιστάσεων. Συγκεκριμένα, το ένα άκρο του διακόπτη (switch) του joystick συνδέθηκε σε pin του expansion header της πλακέτας (GPIO_D0), με τη μεσολάβηση pull-up αντίστασης, η οποία, μέσω καλωδίου, συνδέθηκε στην τάση τροφοδοσίας VCC (3.3V). Η συγκεκριμένη επιλογή διασφαλίζει ότι η είσοδος διατηρείται σε λογικό επίπεδο υψηλής τάσης όταν ο διακόπτης βρίσκεται σε ανενεργή κατάσταση, ενώ μεταβαίνει σε χαμηλό λογικό επίπεδο κατά την ενεργοποίηση. Στη συνέχεια, το άλλο άκρο του switch συνδέθηκε, μέσω προστατευτικής αντίστασης, στη γείωση (GND) της πλακέτας.

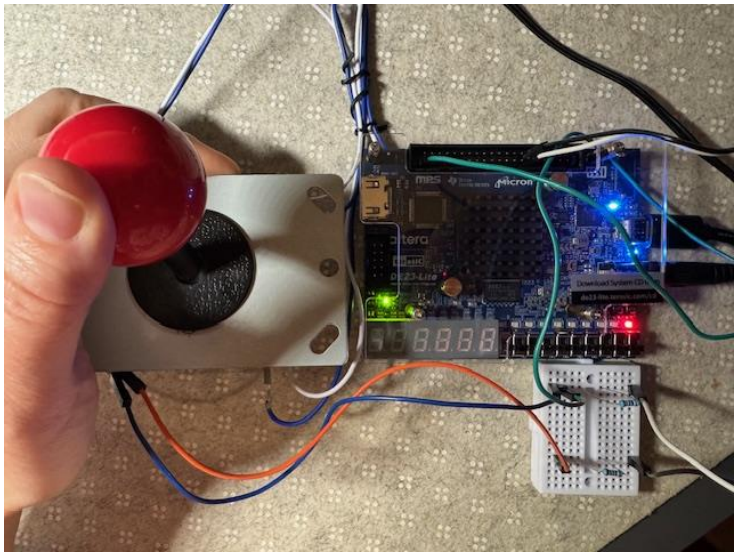


Εικόνα 30 Συνδεσμολογία χειριστηρίου με την πλακέτα DE23-Lite για ένα switch του joystick

Η επιλογή των αντιστάσεων μπορεί να τεκμηριωθεί βάσει των ηλεκτρικών χαρακτηριστικών που παρέχονται στο επίσημο εγχειρίδιο χρήσης. Οι ακροδέκτες (GPIO) λειτουργούν στα 3.3V, ενώ το Voltage and Maximum Current Limit of Expansion Header αναφέρει μέγιστο επιτρεπτό ρεύμα 1.5A συνολικά για το σύνολο των ακροδεκτών. Για την είσοδο του χειριστηρίου, επιλέγεται pull-up αντίσταση, ώστε το ρεύμα που διαρρέει κάθε είσοδο, όταν ο διακόπτης ενεργοποιείται να είναι ιδιαίτερα μικρό. Υπολογίζοντας, με βάση τον νόμο του Ohm, και θεωρώντας επιθυμητό ρεύμα περίπου 0,33 mA, η τιμή της pull-up αντίστασης που προκύπτει είναι: R = V/I = 3.3V/0.00033A= 10kΩ.

Επιπλέον, η χρήση προστατευτικής αντίστασης σε σειρά με τον διακόπτη και τη γείωση περιορίζει το στιγμιαίο ρεύμα σε περιπτώσεις λανθασμένης σύνδεσης ή βραχυκυκλώματος, λειτουργώντας ως στοιχειώδης μηχανισμός προστασίας των ακροδεκτών. Υπολογίζοντας, με τον νόμο του Ohm και θεωρώντας ως ασφαλές μέγιστο ρεύμα για ένα pin περίπου 15 mA, η τιμή της σειράς αντίστασης που προκύπτει είναι : R = V/I = 3.3V/0,015A= 220Ω. Συνεπώς, οι επιλεγμένες τιμές αντιστάσεων είναι πλήρως συμβατές με τις ηλεκτρικές προδιαγραφές της πλακέτας DE23-lite.

Μετά την ολοκλήρωση της φυσικής συνδεσμολογίας του χειριστηρίου, η λειτουργικότητα του υποσυστήματος εισόδου υλοποιήθηκε σε επίπεδο HDL, με στόχο την αξιόπιστη ανίχνευση και αποκωδικοποίηση των σημάτων που προέρχονται από το χειριστήριο. Ως αποτέλεσμα, κατά τον έλεγχο της λειτουργίας του συστήματος, το LED της πλακέτας άναψε με επιτυχία (Εικόνα 31), όταν ο μοχλός του χειριστηρίου μετακινήθηκε προς τα πάνω, επιβεβαιώνοντας την ορθή λήψη και επεξεργασία του αντίστοιχου σήματος εισόδου.



Εικόνα 31 Ενεργοποίηση led κατά τη μετακίνηση του μοχλού προς τα πάνω

Παρακάτω, ακολουθεί ο κώδικας υλοποίησης σε Verilog που χρησιμοποιήθηκε για την επαλήθευση της ορθής λειτουργίας του υποσυστήματος εισόδου:

```
module joytest(
input CLOCK0_50,
input [0:0] KEY,
output [9:0]   LEDR, //LED is Low-Active
input  [0:0]   GPIO_D
);

assign LEDR[9:1] = 9'b111111111;
assign LEDR[0:0] = oneled;

wire clk_50;
wire joyup;
reg oneled;
reg gpio_prev;
assign joyup = GPIO_D;

pll u_pll(
```

```
.refclk(CLOCK0_50),
.rst(~KEY),
.outclk_0(clk_50)
);

always @(posedge clk_50) begin
  // store previous GPIO value
  gpio_prev <= GPIO_D;

  // falling edge: 1 -> 0
  if (gpio_prev == 1'b1 && GPIO_D == 1'b0)
    oneled <= 1'b0;  // LED ON (active-low)

  // rising edge: 0 -> 1
  else if (gpio_prev == 1'b0 && GPIO_D == 1'b1)
    oneled <= 1'b1;  // LED OFF (active-low)
end
endmodule
```

Ο παραπάνω κώδικας, στοχεύει στον έλεγχο ενός LED μέσω ενός εξωτερικού GPIO σήματος, που προέρχεται από το χειριστήριο. Το κύριο module με όνομα joytest δέχεται ως είσοδο το εξωτερικό ρολόι της πλακέτας στα 50 MHz (CLOCK0_50), ένα πλήκτρο reset (KEY[0]) και ένα ψηφιακό σήμα εισόδου από το χειριστήριο (GPIO_D[0]), ενώ τα LEDR[9:0] είναι έξοδοι LED που υλοποιούνται ως active-low. Τα LEDR[9:1] ανατίθενται σταθερά στην τιμή 1 με την εντολή assign LEDR[9:1] = 9'b111111111;, διατηρώντας τα σβηστά, ενώ το LEDR[0] ελέγχεται από το σήμα oneled, το οποίο μεταβάλλεται δυναμικά ανάλογα με την κατάσταση του GPIO.

Η παραγωγή του εσωτερικού ρολογιού γίνεται μέσω ενός PLL (pll u_pll), το οποίο λαμβάνει ως είσοδο το κύριο ρολόι CLOCK0_50 και ένα σήμα reset ~KEY, και παρέχει ως έξοδο το clk_50. Αυτό το εσωτερικό ρολόι χρησιμοποιείται για τον συγχρονισμό όλων των λειτουργιών του module, εξασφαλίζοντας ότι οι αλλαγές στα registers πραγματοποιούνται σε ανερχόμενα μέτωπα ρολογιού.

Η κύρια λειτουργία του module βασίζεται σε ένα always @(posedge clk_50) μπλοκ, στο οποίο αποθηκεύεται η προηγούμενη κατάσταση της εισόδου GPIO_D σε ένα register (gpio_prev) και συγκρίνεται με την τρέχουσα τιμή της. Η λογική ανιχνεύει μεταβάσεις του σήματος, δηλαδή στην περίπτωση που η είσοδος αλλάξει από υψηλή σε χαμηλή τιμή (falling edge), το LEDR[0] ανάβει (active-low), ενώ στην αντίθετη περίπτωση (rising edge) σβήνει. Με αυτόν τον τρόπο, το σύστημα ανιχνεύει καθαρές μεταβολές της εισόδου

και αντιστοιχεί την κατάσταση του LED στην κατεύθυνση της μετάβασης, αποφεύγοντας ασυγχρονισμένα ή ενδιάμεσα σήματα. Το PLL και το μπλοκ συγχρονισμού εξασφαλίζουν ότι όλες οι αλλαγές της κατάστασης πραγματοποιούνται σε συγκεκριμένα ανερχόμενα μέτωπα του ρολογιού, παρέχοντας σταθερότητα και ακρίβεια στη λειτουργία του κυκλώματος.

Σε συνέχεια του προηγούμενου κώδικα, κρίθηκε σημαντικό να προστεθεί μηχανισμός debounce με διάρκεια 10ms στην είσοδο του GPIO, προκειμένου να εξαλειφθεί το φαινόμενο του μηχανικού "θορύβου" που εμφανίζεται κατά την πίεση ή την απελευθέρωση του switch του χειριστηρίου. Η προσθήκη αυτή είναι αναγκαία καθώς τα μηχανικά κουμπιά δεν αλλάζουν κατάσταση καθαρά από 0 σε 1 ή από 1 σε 0, αλλά παρουσιάζουν ταχεία εναλλαγή της τάσης μέσα σε μερικά milliseconds, γεγονός που μπορεί να προκαλέσει πολλαπλές ψευδο-μεταβάσεις και να οδηγήσει σε ανεπιθύμητο τρεμόπαιγμα των LED. Η επιλογή του χρόνου 10ms θεωρείται συντηρητική και ασφαλής για τα περισσότερα μηχανικά κουμπιά, εξασφαλίζοντας ότι μόνο οι σταθερές αλλαγές κατάστασης θεωρούνται έγκυρες, ενώ ταυτόχρονα διατηρείται ικανοποιητική ταχύτητα απόκρισης του συστήματος. Παρακάτω ακολουθεί ο τροποποιημένος κώδικας:

```
module joytest(
input CLOCK0_50,
input [0:0] KEY,
output [9:0]   LEDR, //LED is Low-Active
input  [0:0]   GPIO_D
);

assign LEDR[9:1] = 9'b111111111;
assign LEDR[0]   = oneled;

wire clk_50;
reg oneled;
reg gpio_prev;
reg gpio_sync;
reg [19:0] debounce_cnt;
reg gpio_db;   // debounced GPIO


pll u_pll(
.refclk(CLOCK0_50),
.rst(~KEY),
.outclk_0(clk_50)
);

// Debounce + Synchronization (10ms @50MHz)
```

```
always @(posedge clk_50) begin
   gpio_sync <= GPIO_D;  // συγχρονισμός εισόδου

   if (gpio_sync == gpio_db) begin
      debounce_cnt <= 20'd0;  // σταθερό -> μηδενίζουμε μετρητή
   end else begin
      debounce_cnt <= debounce_cnt + 1'b1;
      if (debounce_cnt == 20'd500_000) begin  // 10ms
         gpio_db <= gpio_sync;  // αποδοχή νέας τιμής
         debounce_cnt <= 20'd0;
      end
   end
end

// Edge detection στο debounced σήμα
always @(posedge clk_50) begin
   gpio_prev <= gpio_db;

   if (gpio_prev == 1'b1 && gpio_db == 1'b0)
      oneled <= 1'b0;  // LED ON
   else if (gpio_prev == 1'b0 && gpio_db == 1'b1)
      oneled <= 1'b1;  // LED OFF
end
endmodule
```

Στον τροποποιημένο κώδικα, το σύστημα εισόδου GPIO_D περνάει πρώτα από μηχανισμό debounce διάρκειας 10ms προκειμένου να εξασφαλιστεί ότι οι αλλαγές της εισόδου είναι σταθερές και αξιόπιστες. Αρχικά, η ασύγχρονη είσοδος GPIO_D συγχρονίζεται με το εσωτερικό ρολόι clk_50 μέσω του register gpio_sync, εξασφαλίζοντας ότι οι επόμενες μεταβολές εξετάζονται σε καθορισμένα χρονικά σημεία. Στη συνέχεια, χρησιμοποιείται ένας μετρητής debounce_cnt ο οποίος αυξάνεται κάθε κύκλο ρολογιού όταν η τρέχουσα συγχρονισμένη τιμή διαφέρει από την τελευταία αποδεκτή τιμή gpio_db. Όταν η τιμή παραμένει σταθερή για 10ms (δηλαδή 50.000.000Hz × 0.01s = 500.000 κύκλους ρολογιού σε 50MHz), η νέα τιμή θεωρείται έγκυρη και ενημερώνεται το register gpio_db. Στη συνέχεια, χρησιμοποιείται ανίχνευση αλλαγών (rising/falling edge) στο debounced σήμα για να ελέγχεται το LEDR[0]: όταν το σήμα πηγαίνει από 1 σε 0 το LED ανάβει (active-low) και όταν πηγαίνει από 0 σε 1 το LED σβήνει. Όλα τα υπόλοιπα LED παραμένουν σβηστά, έτσι το κύκλωμα μετατρέπει τις καθαρές μεταβάσεις του κουμπιού σε αξιόπιστο άναμμα/σβήσιμο του LED.

Η συγκεκριμένη δοκιμαστική υλοποίηση επέτρεψε τον έλεγχο κρίσιμων παραμέτρων, όπως η ορθότητα της ηλεκτρικής σύνδεσης, η απουσία ψευδών ενεργοποιήσεων, η επάρκεια των αντιστάσεων pull-up και προστασίας, καθώς και η λειτουργική συνέπεια

μεταξύ φυσικής εισόδου και οπτικής ανατροφοδότησης. Με την ολοκλήρωση της δοκιμής, κατέστη δυνατή η τεκμηριωμένη μετάβαση στη σχεδίαση της πλήρους μονάδας εισόδου για το παιχνίδι Tetris, με αυξημένη βεβαιότητα ως προς την αξιοπιστία του υποσυστήματος.

4.1.2 Υποσύστημα απεικόνισης HDMI (HDMI Controller)

Το υποσύστημα απεικόνισης αποτελεί κρίσιμο τμήμα του συστήματος, καθώς διασφαλίζει την οπτική αναπαράσταση του παιχνιδιού και τη διασύνδεση με εξωτερική οθόνη μέσω της διεπαφής HDMI. Στόχος του είναι να μεταφράζει την κατάσταση του πλέγματος και των tetrominoes σε εικόνα υψηλής ποιότητας με σταθερό ρυθμό ανανέωσης. Η οπτική ανατροφοδότηση που παρέχει είναι κρίσιμη για την εμπειρία του παίκτη, καθώς επιτρέπει την αντίληψη της θέσης των κομματιών στο πλέγμα και την εξέλιξη του παιχνιδιού σε πραγματικό χρόνο.

Η ενσωματωμένη ύπαρξη HDMI transmitter στην πλακέτα ανάπτυξης επιτρέπει την απευθείας ψηφιακή μετάδοση εικόνας υψηλής ποιότητας, εξασφαλίζοντας σταθερότητα, ακρίβεια χρωμάτων και ευρεία συμβατότητα με σύγχρονες οθόνες. Επιπλέον, συμβάλλει στη μείωση της πολυπλοκότητας του σχεδιασμού του υποσυστήματος απεικόνισης, αφού αναλαμβάνει την κωδικοποίηση TMDS και την οδήγηση των διαφορικών σημάτων HDMI. Τέλος, επιτρέπει την αξιοποίηση έτοιμων σχεδίων αναφοράς και επιταχύνει τη διαδικασία ανάπτυξης, καθιστώντας το υποσύστημα απεικόνισης πιο αξιόπιστο, επεκτάσιμο και κατάλληλο για εφαρμογές πραγματικού χρόνου.

Η διαδικασία απεικόνισης ξεκινά με τη μετατροπή του πλέγματος σε συντεταγμένες οθόνης. Κάθε κελί του πλέγματος αντιστοιχεί σε ένα προκαθορισμένο σύνολο pixel στην οθόνη και τα τετραγωνάκια που απαρτίζουν τα tetrominoes, ενημερώνονται συνεχώς με βάση τη θέση τους. Αυτή η αναπαράσταση περιλαμβάνει τόσο τα τετραγωνάκια που κινούνται όσο και αυτά που έχουν ήδη «κλειδώσει», ώστε η εικόνα να αντανακλά με ακρίβεια την τρέχουσα κατάσταση του παιχνιδιού.

Αν και η απεικόνιση πραγματοποιείται σε ασπρόμαυρη μορφή, το ψηφιακό πρότυπο HDMI εξακολουθεί να βασίζεται στη μετάδοση σημάτων RGB. Στη συγκεκριμένη υλοποίηση, τα τρία κανάλια κόκκινου (R), πράσινου (G) και μπλε (B) δεν μεταφέρουν ανεξάρτητη χρωματική πληροφορία, αλλά λαμβάνουν την ίδια τιμή έντασης για κάθε pixel, με αποτέλεσμα η οθόνη να εμφανίζει αποκλειστικά μαύρο και λευκό. Τα ψηφιακά

δεδομένα RGB συγχρονίζονται με τα σήματα χρονισμού Horizontal Sync (HSYNC) και Vertical Sync (VSYNC), τα οποία καθορίζουν την αρχή και το τέλος κάθε γραμμής και κάθε πλήρους καρέ αντίστοιχα, ενώ το σήμα Active Video υποδεικνύει τις χρονικές περιόδους κατά τις οποίες τα δεδομένα RGB είναι έγκυρα και αντιστοιχούν σε ορατά pixel. Με αυτόν τον τρόπο, τα σήματα συγχρονισμού διασφαλίζουν τη σωστή χωρική και χρονική τοποθέτηση των pixel στην οθόνη, χωρίς να επηρεάζουν το περιεχόμενο της εικόνας. Τα pixel, υπολογίζονται σε ρυθμό pixel clock 148.5 MHz για ανάλυση 1920×1080 60Hz και μεταφέρονται απευθείας στον ενσωματωμένο HDMI transmitter, ο οποίος αναλαμβάνει την κωδικοποίηση TMDS και τη ψηφιακή μετάδοση στην οθόνη. Με αυτόν τον τρόπο διασφαλίζεται ότι το τελικό καρέ απεικονίζεται σε πραγματικό χρόνο με σταθερό συγχρονισμό και ομαλή ανανέωση, παρέχοντας στον παίκτη ακριβή οπτική ανατροφοδότηση για την κατάσταση του παιχνιδιού.

Συνοψίζοντας, το υποσύστημα απεικόνισης μέσω HDMI με ενσωματωμένο transmitter, σχεδιάστηκε ώστε να παρέχει αξιόπιστη και ευέλικτη ψηφιακή έξοδο εικόνας, αξιοποιώντας πλήρως τις δυνατότητες της πλακέτας. Η αρχιτεκτονική του, ο παραλληλισμός των λειτουργιών και η ενσωματωμένη κωδικοποίηση TMDS διασφαλίζουν ότι το σύστημα μπορεί να υποστηρίξει σταθερή αναπαράσταση εικόνας υψηλής ποιότητας με μειωμένη πολυπλοκότητα σχεδίασης.

Σημαντικό είναι να αναφερθεί, ότι η πλακέτα DE23-Lite συνοδεύεται από ένα πακέτο παραδειγμάτων (demonstrations), το οποίο διατίθεται στην επίσημη ιστοσελίδα της Terasic, στο DE23-Lite System CD [128]. Το πακέτο αυτό περιλαμβάνει πλήρεις υλοποιήσεις καθώς και όλα τα απαραίτητα αρχεία για την αξιολόγηση των βασικών περιφερειακών της πλακέτας, όπως SDRAM, UART, ADC και τη διεπαφή HDMI. Αυτά τα παραδείγματα λειτουργούν ως έτοιμες υλοποιήσεις HDMI controller, παρέχοντας έτοιμα RTL δομές που ενσωματώνουν τόσο τα σήματα εικόνας όσο και τη σωστή παραμετροποίηση της εξόδου HDMI, συμπεριλαμβανομένης της αρχικοποίησης του HDMI transmitter μέσω I2C και της παραγωγής χρωμάτων/χρονισμών για απεικόνιση στην οθόνη. Η διαθεσιμότητα αυτών των υλοποιήσεων προσφέρει στον χρήστη ένα δοκιμασμένο και αξιόπιστο σημείο εκκίνησης για την ανάπτυξη και προσαρμογή εξειδικευμένων HDMI εφαρμογών στην πλακέτα, περιορίζοντας σημαντικά την πολυπλοκότητα του σχεδιασμού, καθώς δεν απαιτείται η εκ νέου υλοποίηση των βασικών ρυθμίσεων του πρωτοκόλλου μετάδοσης και των κρίσιμων χρονικών παραμέτρων της εικόνας.

Για τις δοκιμαστικές υλοποιήσεις έγινε χρήση του αρχείου HDMI_out_RTL, όπου περιλαμβάνει το project golden_top (Παράρτημα Α), το οποίο παράγει ένα VGA pattern με τέσσερις περιοχές όπου εναλλάσσονται τα χρώματα κόκκινο, πράσινο, μπλε και γκρι.

Αναλυτικότερα, το module golden_top αρχικοποιεί και συντονίζει όλα τα περιφερειακά του συστήματος, συμπεριλαμβανομένων HDMI εξόδου βίντεο και ήχου, SDRAM, LED, 7-segment οθονών, και UART/I2C. Χρησιμοποιεί PLLs για να παράγει σταθερά ρολόγια βίντεο (148,5 MHz για HDMI) και ήχου (12,288 MHz για DAC), ενώ περιλαμβάνει reset μέσω ResetRelease. Το HDMI σήμα παράγεται από το submodule vpg, που δημιουργεί RGB pattern, HSYNC, VSYNC και DE για εμφάνιση στην οθόνη, τα οποία συνδέονται απευθείας στις εξόδους HDMI. Παράλληλα, το module διαμορφώνει και στέλνει ήχο μέσω I2S, χρησιμοποιώντας το AUDIO_DAC και συνδέοντας τα σήματα AUD_BCLK, AUD_LRCLK και AUD_DATA στα αντίστοιχα HDMI pins. Ο ήχος ενεργοποιείται κάνοντας χρήση του KEY3 της πλακέτας. Τα LEDs χρησιμοποιούνται για ένδειξη κατάστασης PLLs και HDMI readiness, ενώ τα HEX displays και τα πλήκτρα παραμένουν απενεργοποιημένα ή χαμηλής προτεραιότητας. Το I2C χρησιμοποιείται για την αρχικοποίηση του HDMI και πιθανώς άλλων περιφερειακών.

Το module vga_generator παράγει τα σήματα VGA/HDMI: HSYNC, VSYNC, Display Enable και RGB για κάθε pixel, χρησιμοποιώντας παραμέτρους οριζόντιου και κάθετου συγχρονισμού και ενεργών περιοχών (h_total, h_sync, h_start, h_end, v_total, v_sync, v_start, v_end), καθώς και τέσσερις προγραμματιζόμενες ενεργές γραμμές (v_active_14, v_active_24, v_active_34) για εναλλαγή χρωμάτων. Οριζόντιοι και κάθετοι μετρητές (h_count, v_count) παρακολουθούν τη θέση των pixels, ενώ τα σήματα h_act και v_act καθορίζουν πότε βρισκόμαστε στην ενεργή περιοχή. Το Display Enable ενεργοποιείται μόνο όταν βρισκόμαστε σε ενεργή περιοχή, εξασφαλίζοντας σωστή εμφάνιση. Οι χρωματικές γραμμές δημιουργούνται με ένα απλό pattern generator, όπου οι τέσσερις περιοχές εναλλάσσουν τα χρώματα κόκκινο, πράσινο, μπλε ή γκρι ανάλογα με τη θέση κάθε pixel (pixel_x) και την ενεργή γραμμή, ενώ τα περιγράμματα των ενεργών περιοχών εμφανίζονται λευκά για οπτική διάκριση.

Το module vpg δέχεται ένα ρολόι 50 MHz και ένα σήμα reset, και παράγει σήματα VPG pixel clock (vpg_pclk), HSYNC, VSYNC, Display Enable και RGB για την οθόνη. Χρησιμοποιεί τις παραμέτρους χρονισμού οθόνης (h_total, h_sync, h_start, h_end, v_total, v_sync, v_start, v_end) και τέσσερις ενεργές γραμμές (v_active_14, v_active_24, v_active_34) για να ελέγχει την ενεργή περιοχή και τα χρώματα στο κάθε καρέ, τα οποία

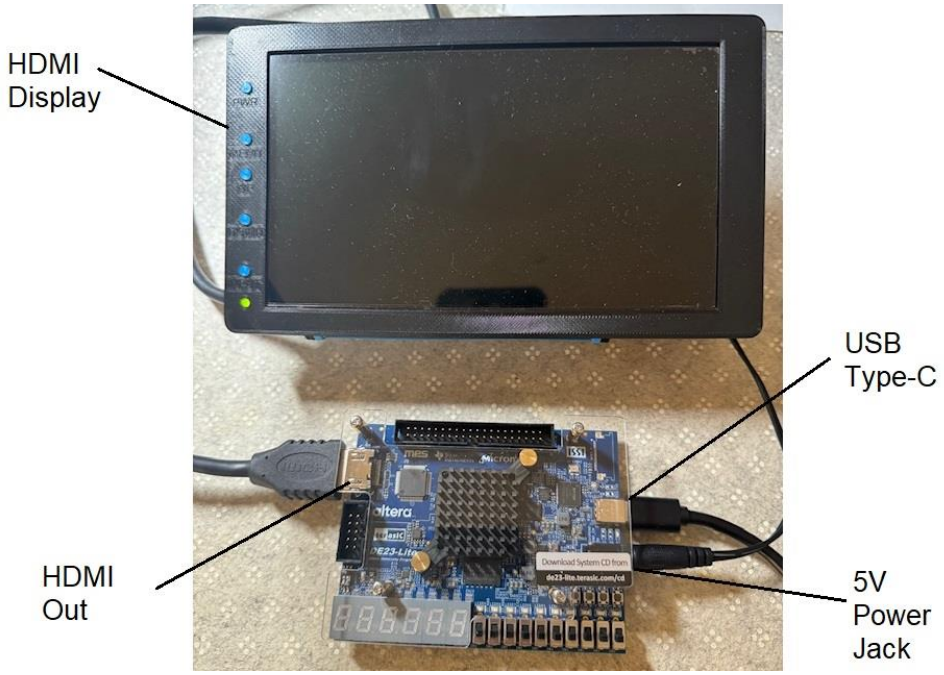
περνούν στο sub-module vga_generator. Οι τιμές αυτές ορίζονται ανάλογα με την ανάλυση (π.χ. 1920x1080 60Hz), ενώ υπάρχει και σχολιασμένο υποστηριζόμενο εύρος για άλλες αναλύσεις όπως 1280x720, 1024x768 ή 1600x1200. Το module διασφαλίζει ότι το pixel clock εξάγεται στο vpg_pclk_out, ενώ το vga_generator αναλαμβάνει τον υπολογισμό των συγχρονιστικών σημάτων και την παραγωγή χρωματικών patterns ανά pixel.

Στο σύστημα αυτό τα αρχεία χωρίζονται λειτουργικά σε δύο φακέλους, τον φάκελο V (Clocking & System Control) και τον φάκελο V_HDMI (Video & Audio μέσω HDMI). Στον πρώτο, περιλαμβάνονται τα αρχεία που αφορούν τη δημιουργία και διαχείριση ρολογιών και reset, όπως το AUDIO_PLL που παράγει το κατάλληλο ρολόι για την έξοδο ήχου μέσω HDMI, το VEDIO_PLL που δημιουργεί το pixel clock και τα ρολόγια συγχρονισμού που απαιτούνται για την έξοδο εικόνας VGA/HDMI, και το ResetRelease που εξασφαλίζει ότι το σύστημα βγαίνει από reset μόνο όταν τα PLL έχουν κλειδώσει και τα ρολόγια είναι σταθερά. Επιπλέον, το CLOCKMEM.v λειτουργεί ως διαιρέτης ρολογιών για οπτική ένδειξη της λειτουργίας τους μέσω LEDs, και το FRE_MEASURE.v μετρά τη συχνότητα ενός ρολογιού υπολογίζοντας πόσοι παλμοί εμφανίζονται μέσα σε ένα δευτερόλεπτο.

Ο δεύτερος φάκελος περιλαμβάνει όλα τα modules που σχετίζονται με την παραγωγή και τη διαμόρφωση σήματος εικόνας και ήχου μέσω HDMI. Το vga_generator.v δημιουργεί τα γραφικά patterns που εμφανίζονται στην οθόνη, ενώ το vpg.v λειτουργεί ως Video Pattern Generator που ορίζει τα σωστά HDMI timings και συνδέει το pattern generator με το υπόλοιπο σύστημα. Για τον ήχο, το AUDIO_DAC.v παράγει ψηφιακό ήχο και τον προωθεί προς το HDMI interface, ενώ τα αρχεία I2C_Controller.v, I2C_AV_Config.v, I2C_HDMI_Config.v και I2C_WRITE_WDATA.v υλοποιούν την επικοινωνία μέσω I2C για τη ρύθμιση των audio/video codecs και του HDMI transmitter. Συνολικά, τα αρχεία αυτά συνεργάζονται ώστε η FPGA να παράγει πλήρως συγχρονισμένο σήμα εικόνας και ήχου μέσω HDMI, με σωστή αρχικοποίηση, χρονισμό και έλεγχο περιφερειακών.

Στη συνέχεια, για την επαλήθευση του υποσυστήματος απεικόνισης υλοποιήθηκε δοκιμαστική εφαρμογή με στόχο τον έλεγχο της ορθής λειτουργίας της εξόδου εικόνας σε οθόνη μέσω διεπαφής HDMI. Η υλοποίηση αυτή σχεδιάστηκε ανεξάρτητα από τη λογική του παιχνιδιού, ώστε να είναι δυνατή η αξιολόγηση της λειτουργίας του υποσυστήματος απεικόνισης χωρίς την επίδραση πρόσθετων λειτουργικών παραμέτρων της συνολικής αρχιτεκτονικής του παιχνιδιού.

Η δοκιμαστική διαδικασία ξεκίνησε με την φυσική προετοιμασία, η οποία αφορούσε τη σύνδεση της πλακέτας DE23-Lite με μια εξωτερική οθόνη μέσω της ενσωματωμένης θύρας HDMI. Η πλακέτα ανάπτυξης διαθέτει ενσωματωμένο HDMI transmitter, επιτρέποντας απευθείας διασύνδεση με σύγχρονες οθόνες χωρίς επιπρόσθετα κυκλώματα μετατροπής. Η διασύνδεση πραγματοποιήθηκε με καλώδιο HDMI και έγιναν οι απαραίτητες συνδέσεις με τον υπολογιστή μέσω της θύρας Type-C της πλακέτας και τροφοδοσίας μέσω του 5V power jack, ώστε να εξασφαλιστεί ασφαλής και σταθερή λειτουργία του υποσυστήματος (Εικόνα 32).



Εικόνα 32 Σύνδεση οθόνης HDMI με την πλακέτα DE23-Lite

Για τη διευκόλυνση της υλοποίησης, αξιοποιήθηκαν έτοιμα παραδείγματα αναφοράς, τα οποία διατίθενται στην επίσημη ιστοσελίδα του κατασκευαστή της πλακέτας. Πάνω στον έτοιμο κώδικα αναφοράς πραγματοποιήθηκαν επεκτάσεις, με σκοπό την εμφάνιση ενός τετραγώνου στο κέντρο της οθόνης. Η επιλογή του συγκεκριμένου σχήματος κρίθηκε κατάλληλη, καθώς επιτρέπει την άμεση και σαφή οπτική επαλήθευση της σωστής λειτουργίας του συστήματος απεικόνισης, χωρίς να απαιτείται σύνθετη διαχείριση γραφικών.

Η ανάπτυξη και η διαχείριση του κώδικα υλοποιήθηκαν με τη χρήση του λογισμικού Quartus Prime Pro, το οποίο αποτέλεσε το βασικό περιβάλλον ανάπτυξης για τη συγγραφή, τη σύνθεση και τη μεταφόρτωση του παραγόμενου αρχείου διαμόρφωσης στην

πλακέτα ανάπτυξης. Η τροποποίηση του κώδικα έγινε στο αρχείο golden_top.v, με προσθήκη της δήλωσης GPIO ως «inout [35: 0] GPIO_D» στο module golden_top και στη συνέχεια έγινε τροποποίηση του κώδικα του αρχείου vpg.v, όπως φαίνεται παρακάτω:

```
module vpg (
  input          vpg_pclk,
  input          reset_n,
  output reg       vpg_de,
  output reg       vpg_hs,
  output reg       vpg_vs,
  output          vpg_pclk_out,
  output reg [7:0]  vpg_r,
  output reg [7:0]  vpg_g,
  output reg [7:0]  vpg_b
);

assign vpg_pclk_out = vpg_pclk;

//=======================================================
//  1080p timing parameters (Terasic default)
//=======================================================
localparam H_ACTIVE = 1920;
localparam H_FP    = 88;
localparam H_SYNC  = 44;
localparam H_BP    = 148;
localparam H_TOTAL  = H_ACTIVE + H_FP + H_SYNC + H_BP;

localparam V_ACTIVE = 1080;
localparam V_FP    = 4;
localparam V_SYNC  = 5;
localparam V_BP    = 36;
localparam V_TOTAL  = V_ACTIVE + V_FP + V_SYNC + V_BP;

//=======================================================
//  Counters
//=======================================================
reg [11:0] h_cnt;
reg [11:0] v_cnt;

//=======================================================
//  Square parameters
//=======================================================
localparam SQUARE_SIZE = 100;

reg [11:0] square_x;
reg [11:0] square_y;

//=======================================================
//  Horizontal counter
//=======================================================
```

```
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      h_cnt <= 12'd0;
   end else begin
      if (h_cnt == H_TOTAL-1)
         h_cnt <= 12'd0;
      else
         h_cnt <= h_cnt + 12'd1;
   end
end

//=========================================================
//  Vertical counter
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      v_cnt <= 12'd0;
   end else if (h_cnt == H_TOTAL-1) begin
      if (v_cnt == V_TOTAL-1)
         v_cnt <= 12'd0;
      else
         v_cnt <= v_cnt + 12'd1;
   end
end

//=========================================================
//  Sync signals
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      vpg_hs <= 1'b1;
      vpg_vs <= 1'b1;
   end else begin
      vpg_hs <= ~((h_cnt >= H_ACTIVE + H_FP) &&
                  (h_cnt <  H_ACTIVE + H_FP + H_SYNC));

      vpg_vs <= ~((v_cnt >= V_ACTIVE + V_FP) &&
                  (v_cnt <  V_ACTIVE + V_FP + V_SYNC));
   end
end

//=========================================================
//  Display enable
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n)
      vpg_de <= 1'b0;
   else
      vpg_de <= (h_cnt < H_ACTIVE) && (v_cnt < V_ACTIVE);
end
```

```
//=====================================================
//  RGB generation
//=====================================================
always @(posedge vpg_pclk or negedge reset_n) begin
  if (!reset_n) begin
    vpg_r <= 8'h00; //Όλα τα πίξελ μαύρα
    vpg_g <= 8'h00;
    vpg_b <= 8'h00;
    square_x <= (H_ACTIVE - SQUARE_SIZE) / 2; //τετράγωνο στο κέντρο της οθόνης
    square_y <= (V_ACTIVE - SQUARE_SIZE) / 2;
  end else if (vpg_de && //όταν είμαστε μέσα στο τετράγωνο: άσπρα πίξελ
          h_cnt >= square_x &&
          h_cnt <  square_x + SQUARE_SIZE &&
          v_cnt >= square_y &&
          v_cnt <  square_y + SQUARE_SIZE) begin
    vpg_r <= 8'hFF;
    vpg_g <= 8'hFF;
    vpg_b <= 8'hFF;
  end else begin //αλλιώς μαύρα
    vpg_r <= 8'h00;
    vpg_g <= 8'h00;
    vpg_b <= 8'h00;
  end
end

endmodule
```

Ο συγκεκριμένος κώδικας υλοποιεί ένα Video Pattern Generator (VPG) για οθόνες 1080p, που παράγει σήματα HSYNC, VSYNC, Display Enable και RGB για κάθε pixel. Χρησιμοποιεί οριζόντιο και κάθετο μετρητή για να καθορίζει τη θέση κάθε pixel στην οθόνη και υπολογίζει πότε τα σήματα συγχρονισμού πρέπει να είναι ενεργά με βάση τις παραμέτρους sync/porch της 1080p οθόνης. Το σήμα Display Enable ενεργοποιείται μόνο στην ενεργή περιοχή της οθόνης, ενώ η RGB έξοδος παράγει ένα λευκό τετράγωνο 100x100 pixels στο κέντρο, με όλα τα υπόλοιπα pixels μαύρα. Εμφανίζεται, επομένως, ένα λευκό τετράγωνο σε μαύρο φόντο (Εικόνα 33), ενώ τα σήματα HSYNC, VSYNC και Display Enable εξασφαλίζουν τον σωστό συγχρονισμό της εικόνας στην οθόνη.

Εικόνα 33 Στιγμιότυπο εμφάνισης τετραγώνου στην οθόνη HDMI

Συμπερασματικά, η επιτυχής εμφάνιση του τετραγώνου στο κέντρο της οθόνης επιβεβαίωσε την ορθή λειτουργία του HDMI υποσυστήματος, τη σωστή αξιοποίηση των έτοιμων παραδειγμάτων και τη δυνατότητα επέκτασης του υπάρχοντος κώδικα και για πιο σύνθετα γραφικά.

Σε επόμενο στάδιο, η δοκιμαστική υλοποίηση τροποποιήθηκε με σκοπό την αξιολόγηση της αλληλεπίδρασης με το υποσύστημα εισόδου. Συγκεκριμένα, προστέθηκε η δυνατότητα μετακίνησης του τετραγώνου στην οθόνη χρησιμοποιώντας το μοχλό του χειριστηρίου, έτσι ώστε μετακινώντας τον προς τα επάνω, να μετακινείται το τετράγωνο προς την ίδια κατεύθυνση. Η κίνηση πραγματοποιήθηκε σε διακριτά βήματα, συγχρονισμένα με το ρολόι του συστήματος, επιτρέποντας την άμεση οπτική επιβεβαίωση της σωστής λειτουργίας της εισόδου (Εικόνα 34).

Εικόνα 34 Στιγμιότυπο μετακίνησης τετραγώνου μέσω του χειριστηρίου

Σχετικά με την HDL υλοποίηση, η τροποποίηση του κώδικα έγινε στο αρχείο vpg.v της εμφάνισης του τετραγώνου, όπως φαίνεται παρακάτω:

```
module vpg (
  input          vpg_pclk,
  input          reset_n,

  input    [0:0]  JOY,      // <<< ΠΡΟΣΘΗΚΗ

  output reg       vpg_de,
  output reg       vpg_hs,
  output reg       vpg_vs,
  output          vpg_pclk_out,
  output reg [7:0]  vpg_r,
  output reg [7:0]  vpg_g,
  output reg [7:0]  vpg_b
);

assign vpg_pclk_out = vpg_pclk;

//=======================================================
//  1080p timing parameters (Terasic default)
//=======================================================
localparam H_ACTIVE = 1920;
localparam H_FP    = 88;
localparam H_SYNC  = 44;
localparam H_BP    = 148;
localparam H_TOTAL  = H_ACTIVE + H_FP + H_SYNC + H_BP;

localparam V_ACTIVE = 1080;
localparam V_FP    = 4;
localparam V_SYNC  = 5;
localparam V_BP    = 36;
localparam V_TOTAL  = V_ACTIVE + V_FP + V_SYNC + V_BP;

//=======================================================
//  Counters
//=======================================================
reg [11:0] h_cnt;
reg [11:0] v_cnt;

//=======================================================
//  Square parameters
//=======================================================
localparam SQUARE_SIZE = 100;
localparam MOVE_STEP   = 10;

reg [11:0] square_x;
reg [11:0] square_y;

//=======================================================
```

```
// Horizontal counter
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      h_cnt <= 12'd0;
   end else begin
      if (h_cnt == H_TOTAL-1)
         h_cnt <= 12'd0;
      else
         h_cnt <= h_cnt + 12'd1;
   end
end

//=========================================================
// Vertical counter
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      v_cnt <= 12'd0;
   end else if (h_cnt == H_TOTAL-1) begin
      if (v_cnt == V_TOTAL-1)
         v_cnt <= 12'd0;
      else
         v_cnt <= v_cnt + 12'd1;
   end
end

//=========================================================
// Sync signals
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      vpg_hs <= 1'b1;
      vpg_vs <= 1'b1;
   end else begin
      vpg_hs <= ~((h_cnt >= H_ACTIVE + H_FP) &&
            (h_cnt <  H_ACTIVE + H_FP + H_SYNC));

      vpg_vs <= ~((v_cnt >= V_ACTIVE + V_FP) &&
            (v_cnt <  V_ACTIVE + V_FP + V_SYNC));
   end
end

//=========================================================
// Display enable
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n)
      vpg_de <= 1'b0;
   else
      vpg_de <= (h_cnt < H_ACTIVE) && (v_cnt < V_ACTIVE);
```

```
end

//=======================================================
//  Square movement (VSYNC-based, rock solid)
//=======================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      square_x <= (H_ACTIVE - SQUARE_SIZE) / 2;
      square_y <= (V_ACTIVE - SQUARE_SIZE) / 2;

   end else if (h_cnt == 0 && v_cnt == 0) begin
      // once per frame (~60Hz)
      if (JOY[0] == 1'b0) begin
         if (square_y >= MOVE_STEP)
            square_y <= square_y - MOVE_STEP;
      end
           end
end

//=======================================================
//  RGB generation
//=======================================================
always @(posedge vpg_pclk or negedge reset_n) begin
   if (!reset_n) begin
      vpg_r <= 8'h00;
      vpg_g <= 8'h00;
      vpg_b <= 8'h00;
   end else if (vpg_de &&
            h_cnt >= square_x &&
            h_cnt <  square_x + SQUARE_SIZE &&
            v_cnt >= square_y &&
            v_cnt <  square_y + SQUARE_SIZE) begin
      vpg_r <= 8'hFF;
      vpg_g <= 8'hFF;
      vpg_b <= 8'hFF;
   end else begin
      vpg_r <= 8'h00;
      vpg_g <= 8'h00;
      vpg_b <= 8'h00;
   end
end

endmodule
```

Η τροποποίηση του κώδικα, σε συνέχεια του προηγούμενου, προσθέτει τη μετακίνηση του λευκού τετραγώνου στην οθόνη, ελεγχόμενη από το σήμα JOY[0], προερχόμενο από το μοχλό του χειριστηρίου. Οι συντεταγμένες του τετραγώνου (square_x και square_y) αρχικοποιούνται στο κέντρο της οθόνης και ενημερώνονται μία φορά ανά καρέ όταν h_cnt

= 0 και v_cnt = 0, εξασφαλίζοντας ομαλή και συγχρονισμένη κίνηση (60Hz). Αν το JOY[0] είναι ενεργό, το τετράγωνο μετακινείται προς τα πάνω κατά MOVE_STEP pixels, χωρίς να βγει εκτός οθόνης. Ταυτόχρονα, τα σήματα HSYNC, VSYNC, Display Enable συνεχίζουν να ελέγχουν το συγχρονισμό και την ενεργή περιοχή της οθόνης, ενώ η RGB έξοδος παραμένει λευκή για το τετράγωνο και μαύρη για τα υπόλοιπα pixels, δημιουργώντας έτσι ένα κινούμενο τετράγωνο πάνω σε μαύρο φόντο.

Η επιτυχημένη εμφάνιση και μετακίνηση του τετραγώνου επιβεβαιώνει τη σωστή λειτουργία τόσο του υποσυστήματος απεικόνισης όσο και της ενσωμάτωσης του υποσυστήματος εισόδου, επιβεβαιώνοντας την ορθή συνεργασία των υποσυστημάτων.

4.1.3 Υποσύστημα ήχου (Audio Controller)

Το υποσύστημα ήχου αναλαμβάνει την παραγωγή ακουστικών ενδείξεων που συνοδεύουν τη δράση του παιχνιδιού, όπως η πτώση ή η περιστροφή των τετραγώνων, η ολοκλήρωση γραμμών ή η εμφάνιση ειδικών γεγονότων. Στόχος του είναι να ενισχύσει την εμπειρία του χρήστη, παρέχοντας άμεση ανατροφοδότηση για τις ενέργειές του και δημιουργώντας μια πιο ρεαλιστική και ελκυστική αλληλεπίδραση με το παιχνίδι.

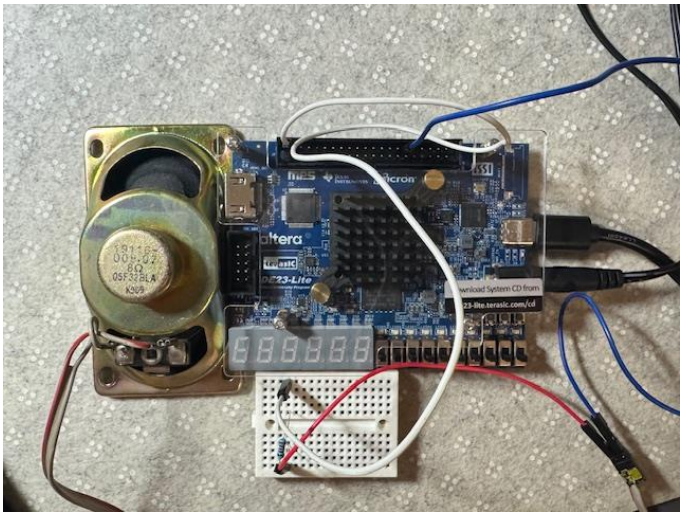
Σε επίπεδο υλικού, το υποσύστημα ήχου αποτελείται συνήθως από ένα μικρό ηχείο, που συνδέεται σε ένα ακροδέκτη GPIO της πλακέτας και μια προστατευτική αντίσταση. Η HDL υλοποίηση του υποσυστήματος ήχου περιλαμβάνει συνήθως μια μικρή γεννήτρια συχνότητας που παράγει κύματα τετραγωνικού σήματος στην επιθυμητή συχνότητα. Η παραγωγή του σήματος μπορεί να συγχρονίζεται με το κύριο ρολόι του FPGA, διασφαλίζοντας σταθερή και αξιόπιστη αναπαραγωγή ήχου. Παράλληλα, η λογική μπορεί να υποστηρίζει πολλαπλές εντολές ήχου ταυτόχρονα, ρυθμίζοντας τη διάρκεια ή την ένταση κάθε τόνου, ώστε να αναπαράγονται διαφορετικά ηχητικά εφέ ανάλογα με τη δράση στο παιχνίδι.

Τέλος, η ενσωμάτωση του υποσυστήματος ήχου σε επίπεδο αρχιτεκτονικής ενισχύει τη συνολική εμπειρία χρήστη, χωρίς να προσθέτει σημαντική πολυπλοκότητα στο κύριο σύστημα. Η διακριτή δομή του επιτρέπει ανεξάρτητη ανάπτυξη, δοκιμή και επαλήθευση, χωρίς να αυξάνει τις απαιτήσεις των πόρων.

Αντίστοιχα με τα παραπάνω υποσυστήματα, για την επαλήθευση του υποσυστήματος ήχου, υλοποιήθηκε δοκιμαστική εφαρμογή, που αποσκοπούσε στον απομονωμένο έλεγχο της ορθής λειτουργίας του, χωρίς την επιβάρυνση της πολυπλοκότητας της

πλήρους υλοποίησης. Η δοκιμαστική εφαρμογή αφορούσε την παραγωγή ήχου από ένα μικρό ηχείο (8Ω), το οποίο είχε συνδεθεί με την πλακέτα ανάπτυξης.

Η συνδεσμολογία έγινε με τη βοήθεια ενός breadboard, ενός καλωδίου τύπου dupont και περιελάμβανε την τροφοδοσία (3.3V) του ηχείου από την πλακέτα, μέσω του ακροδέκτη της πλακέτας (GPIO_D1), παρεμβάλλοντας μια προστατευτική αντίσταση 220Ω (Εικόνα 35). Με αυτόν τον τρόπο, εξασφαλίζεται ότι το μέγιστο ρεύμα δεν ξεπερνά τα επιτρεπτά όρια του pin, διατηρώντας έτσι την ασφάλεια και την αξιοπιστία του κυκλώματος.



Εικόνα 35 Συνδεσμολογία ηχείου με την πλακέτα ανάπτυξης

Αφού ολοκληρώθηκε η περιγραφή της συνδεσμολογίας του ηχείου με την πλακέτα, ο παρακάτω κώδικας σε Verilog παρουσιάζει τη δοκιμαστική υλοποίηση της παραγωγής ήχου. Χρησιμοποιείται μια γεννήτρια συχνότητας, η οποία υλοποιείται μέσω της μονάδας PLL, η οποία παρέχει σταθερό ρολόι για τον μετρητή pwm, δημιουργώντας τον ακουστικό τόνο στο ηχείο.

```
module sound(
input CLOCK0_50,
input [0:0] KEY,
output [9:0] LEDR, //LED is Low-Active
output [1:1] GPIO_D
);

reg [16:0] pwm_cnt;   // 8-bit counter for PWM
wire clk_50;
reg speaker;

pll u_pll(
.refclk(CLOCK0_50),
.rst(~KEY),
.outclk_0(clk_50)
);

//----LED off device
assign LEDR[9:0] = 10'b1111111111;
assign GPIO_D[1] = speaker;

always @(posedge clk_50 or negedge KEY[0]) begin
  if (!KEY[0]) begin
     pwm_cnt <= 0;
     speaker <= 0;
                if (pwm_cnt >= 1493) begin //50mHz / 1493 = 33488 Νότα C11
                        pwm_cnt <= 0;
                        speaker <= ~speaker;  // toggle output
                end else begin
                        pwm_cnt <= pwm_cnt + 1;
                end
  end
end

endmodule
```

Ο παραπάνω κώδικας περιγράφει ένα δοκιμαστικό υποσύστημα παραγωγής ήχου, υλοποιημένο σε γλώσσα Verilog. Η σχεδίαση περιλαμβάνει βασικά στοιχεία ελέγχου συχνότητας και PWM (Pulse Width Modulation) για την παραγωγή ακουστικών σημάτων μέσω ενός ηχείου συνδεδεμένου σε έξοδο GPIO. Το κύριο χαρακτηριστικό της δοκιμαστικής υλοποίησης είναι η παραγωγή ενός μονοφωνικού ψηφιακού σήματος ήχου με ελεγχόμενη συχνότητα, ώστε να είναι δυνατή η αναπαραγωγή συγκεκριμένων μουσικών νότων.

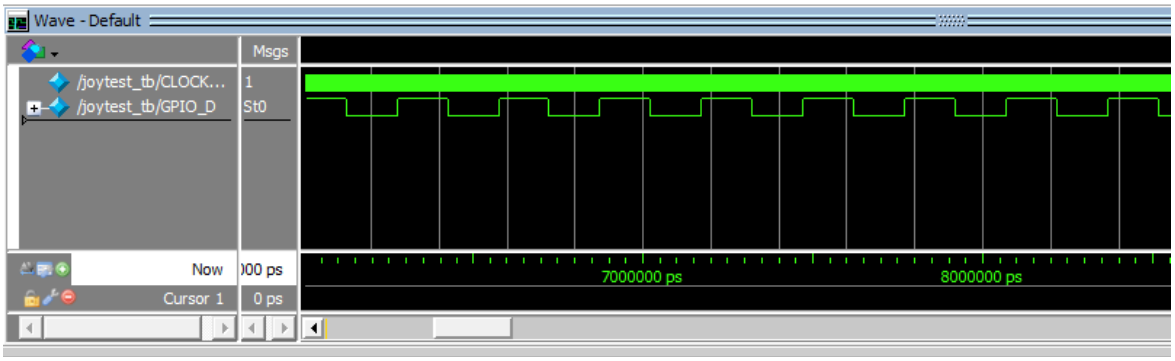
Αποτελείται από το module sound, όπου διαθέτει ως εισόδους το ρολόι συστήματος CLOCK0_50 και ένα διακόπτη (KEY[0:0]) για έλεγχο επανεκκίνησης. Ως έξοδοι

χρησιμοποιούνται τα 10 LED LEDR[9:0], που στη δοκιμαστική υλοποίηση είναι μόνιμα σβηστά, και το GPIO_D[1], που συνδέεται στο ηχείο και μεταφέρει το ψηφιακό ηχητικό σήμα.

Η υλοποίηση περιλαμβάνει έναν PLL (pll u_pll) για τη μετατροπή του εξωτερικού ρολογιού 50 MHz σε ένα σταθερό εσωτερικό ρολόι clk_50, το οποίο χρησιμοποιείται για συγχρονισμό της διαδικασίας παραγωγής PWM. Ο PLL επιτρέπει τη σταθεροποίηση και ακριβή χρονισμό των σημάτων, γεγονός κρίσιμο για την παραγωγή ηχητικών συχνοτήτων που απαιτούν συγκεκριμένη ακρίβεια. Η παραγωγή ήχου βασίζεται σε ένα μετρητή pwm_cnt [16:0] και ένα καταχωρητή speaker. Το ψηφιακό ηχητικό σήμα δημιουργείται με εναλλαγή της εξόδου σε προκαθορισμένα χρονικά διαστήματα. Ο μετρητής αυξάνεται σε κάθε θετική ακμή του ρολογιού clk_50, και όταν φτάνει στο όριο 1493, επανεκκινεί από μηδέν και αναστρέφει την κατάσταση του speaker. Η τιμή αυτή αντιστοιχεί σε συχνότητα περίπου 33,488 kHz, η οποία στο πλαίσιο της δοκιμαστικής υλοποίησης έχει επιλεγεί για την παραγωγή μιας υψηλής νότας (C11), παρέχοντας μια ένδειξη ότι το σύστημα μπορεί να παράγει συγκεκριμένες μουσικές συχνότητες.

Η επιλογή του PWM με απλή εναλλαγή (toggle) της εξόδου προσφέρει έναν εύκολο τρόπο για παραγωγή ηχητικών σημάτων χωρίς τη χρήση αναλογικών κυκλωμάτων. Επιπλέον, ο κώδικας περιλαμβάνει μηχανισμό επανεκκίνησης μέσω του KEY[0], που διασφαλίζει την ασφαλή αρχικοποίηση των καταχωρητών και την αποφυγή ανεπιθύμητων παλμών κατά την ενεργοποίηση του συστήματος.

Η λειτουργία του υποσυστήματος ήχου θεωρήθηκε επιτυχημένη, καθώς κατά τη λειτουργία του στην πλακέτα, παρατηρήθηκε η παραγωγή ακουστικού σήματος από το συνδεδεμένο ηχείο. Η υλοποίηση αυτή επαληθεύτηκε και με testbench μέσω του ModelSim (Εικόνα 36), αφού ο ήχος δεν μπορεί να καταγραφεί διαφορετικά στην οθόνη.



Εικόνα 36 Προσομοίωση εξόδου ήχου

Συνοψίζοντας, η δοκιμαστική υλοποίηση του υποσυστήματος ήχου, αποτελεί ένα απλό αλλά αποτελεσματικό πλαίσιο για την παραγωγή ψηφιακού ήχου σε FPGA. Η δομή του κώδικα επιτρέπει εύκολη επέκταση, όπως η παραγωγή πολλαπλών συχνοτήτων ή η ενσωμάτωση περισσότερων τόνων, για βελτίωση της ποιότητας του ήχου.

4.1.4 Υποσύστημα ελέγχου παιχνιδιού (Game Logic)

Το υποσύστημα ελέγχου παιχνιδιού (Game Logic) αποτελεί τον πυρήνα της αρχιτεκτονικής του συστήματος, καθώς καθορίζει τη συμπεριφορά και τη ροή του παιχνιδιού, συντονίζοντας την αλληλεπίδραση μεταξύ του παίκτη και του συστήματος. Μέσω ενός συνόλου διαδικασιών και κανόνων, το υποσύστημα αυτό, είναι υπεύθυνο για τη δημιουργία και τη διαχείριση των tetrominoes, τον έλεγχο των κινήσεων και την ανίχνευση συγκρούσεων, την ανίχνευση πλήρων γραμμών και την εκκαθάρισή τους, και την ενημέρωση της κατάστασης του παιχνιδιού. Επιπλέον, παράγει και διαχειρίζεται δεδομένα που απαιτούνται για την απεικόνιση της τρέχουσας κατάστασης του πλέγματος στην οθόνη, καθώς και για την αναπαραγωγή ήχων που συνδέονται με γεγονότα του παιχνιδιού, εξασφαλίζοντας μια ολοκληρωμένη εμπειρία παιχνιδιού.

Για την αποτελεσματική διαχείριση των παραπάνω λειτουργιών, το υποσύστημα οργανώνεται σε διακριτές καταστάσεις όπως η αναπαράσταση του πλέγματος, η γεννήτρια tetromino, η διαχείριση του tetromino σε καθοδική πορεία, η ανίχνευση προσγείωσης, η ανίχνευση και διαγραφή πλήρων γραμμών και ο τερματισμός του παιχνιδιού. Παρακάτω, παρουσιάζεται αναλυτικά η δομή και η λειτουργία των επιμέρους καταστάσεων, επισημαίνοντας τον τρόπο αλληλεπίδρασής τους και τη συμβολή τους στη συνολική ροή του παιχνιδιού.

- ***Αναπαράσταση πλέγματος παιχνιδιού***

Η λειτουργία του υποσυστήματος βασίζεται στην αναπαράσταση του χώρου παιχνιδιού ως μονοδιάστατου πίνακα, το οποίο αποτελείται από ένα σύνολο 200 κελιών. Κάθε κελί μπορεί να βρίσκεται σε κατάσταση κενού ή κατειλημμένου από tetromino. Κάθε tetromino αναπαρίσταται ξεχωριστά, με πληροφορίες που αφορούν τη θέση, τον προσανατολισμό και τον τύπο του, ώστε να διευκολύνεται η κίνηση και η περιστροφή του εντός του πλέγματος. Η αναπαράσταση αυτή επιτρέπει τον αποδοτικό έλεγχο συγκρούσεων και την ανίχνευση πλήρων γραμμών. Όταν όλα τα κελιά μιας γραμμής

γεμίσουν, η γραμμή εκκαθαρίζεται και όλες οι γραμμές που βρίσκονται πάνω από αυτή μετατοπίζονται προς τα κάτω, διατηρώντας τη συνοχή του παιχνιδιού. Η ακριβής παρακολούθηση της κατάστασης του πλέγματος επιτρέπει τη σωστή αλληλεπίδραση μεταξύ των tetrominoes και του περιβάλλοντος παιχνιδιού.

- ***Γεννήτρια Tetromino***

Η γεννήτρια είναι υπεύθυνη για τη δημιουργία νέων tetrominoes που εμφανίζονται στο παιχνίδι. Τα tetrominoes παράγονται με τυχαίο τρόπο και το κάθε ένα εμφανίζεται στην κορυφή του πλέγματος, όταν το προηγούμενο έχει πάρει την οριστική του θέση. Στην υλοποίηση αυτή, εμφανίζεται μόνο το τρέχον tetromino στο πλέγμα και δεν παρέχεται προεπισκόπηση του επόμενου κομματιού στον παίκτη. Ως αποτέλεσμα, η τοποθέτηση των tetrominoes εξαρτάται αποκλειστικά από την άμεση εκτίμηση του παίκτη για το τρέχον tetromino και των διαθέσιμων κενών στο πλέγμα.

- ***Πτώση Tetromino***

Καθώς το tetromino εμφανίζεται στην κορυφή του πλέγματος, κινείται συνεχώς προς τα κάτω έως ότου φτάσει στη βάση του πλέγματος ή ακουμπήσει κάποιο άλλο tetromino. Το σύστημα διατηρεί για κάθε ένα, τις τρέχουσες συντεταγμένες των τεσσάρων τετραγώνων που το αποτελούν, όπως και το σχήμα και τον προσανατολισμό του. Κατά τη διάρκεια της πτώσης, οι συντεταγμένες αυτές ενημερώνονται συνεχώς με βάση τις κινήσεις και τις πιθανές περιστροφές που θα πραγματοποιήσει ο παίκτης, εξασφαλίζοντας ότι το tetromino παραμένει εντός των ορίων του πλέγματος και δεν συγκρούεται με τα ήδη τοποθετημένα κομμάτια. Η συνεχής ενημέρωση της θέσης του επιτρέπει τον στρατηγικό προγραμματισμό των κινήσεων και τη σωστή τοποθέτηση εντός του πλέγματος.

- ***Έλεγχος κινήσεων και συγκρούσεων***

Για κάθε εντολή κίνησης ή περιστροφής που λαμβάνεται από τον χρήστη το υποσύστημα ελέγχου παιχνιδιού, εκτελεί έλεγχο εγκυρότητας πριν ενημερώσει την κατάσταση του παιχνιδιού. Ο έλεγχος αυτός περιλαμβάνει τη σύγκριση της νέας υποψήφιας θέσης του tetromino με τα όρια του πλέγματος και με τα ήδη τοποθετημένα tetrominoes. Αν διαπιστωθεί σύγκρουση, η εντολή απορρίπτεται και η προηγούμενη κατάσταση διατηρείται. Η διαδικασία αυτή εξασφαλίζει ότι το παιχνίδι ακολουθεί πιστά τους κανόνες

του Tetris και αποτρέπει την εμφάνιση μη έγκυρων καταστάσεων. Παράλληλα με τις εντολές του χρήστη, το υποσύστημα ελέγχου παιχνιδιού, λαμβάνει περιοδικά σήματα χρονισμού, τα οποία ενεργοποιούν την αυτόματη πτώση των κομματιών.

- ***Ανίχνευση προσγείωσης και «κλείδωμα» tetromino***

Τα tetrominoes σταματούν να πέφτουν όταν οποιοδήποτε από τα τετράγωνά τους έρθει σε επαφή με κάποιο ήδη τοποθετημένο tetromino ή με τη βάση του πλέγματος. Σε αυτό το σημείο, το πλέγμα ενημερώνεται ώστε να αντικατοπτρίζει την τελική θέση του tetromino που ήταν σε καθοδική πορεία. Το tetromino «κλειδώνει» στο πλέγμα όταν δεν είναι δυνατή περαιτέρω πτώση. Τυχόν πλήρεις γραμμές διαγράφονται και δημιουργείται ένα νέο tetromino.

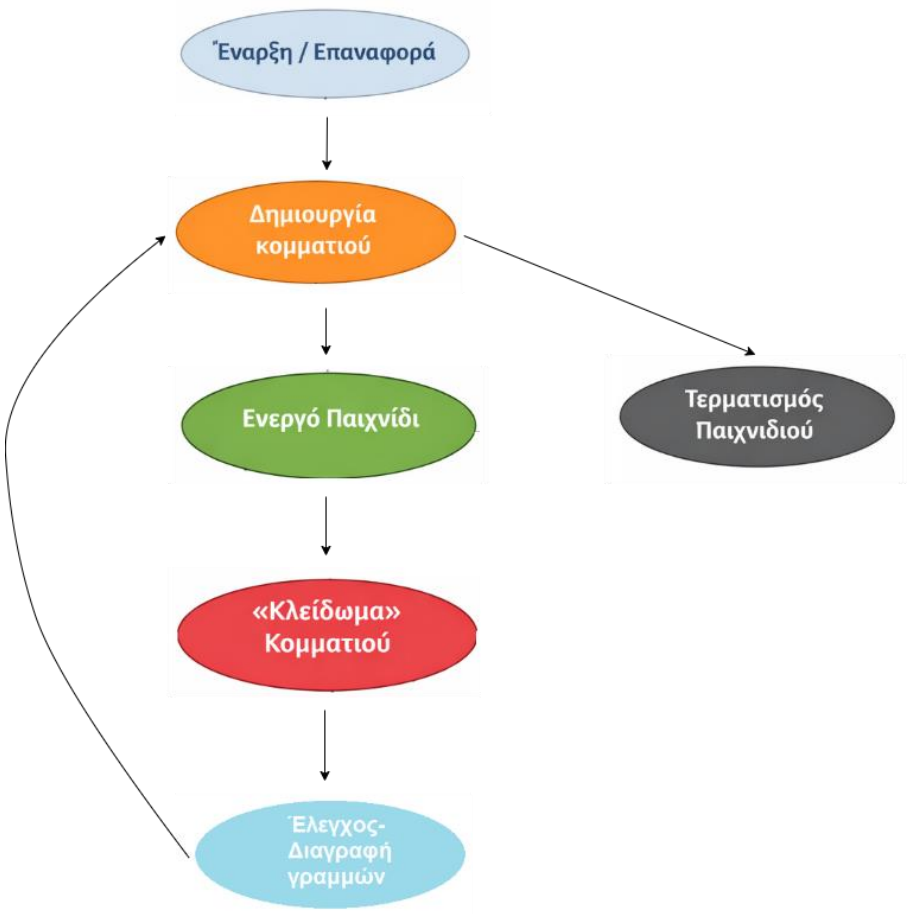
- ***Ανίχνευση και διαγραφή πλήρων γραμμών***

Μετά το «κλείδωμα» ενός tetromino, το υποσύστημα ελέγχου παιχνιδιού ελέγχει κάθε γραμμή του πλέγματος αν είναι πλήρης. Οι πλήρεις γραμμές διαγράφονται και τα tetrominoes που βρίσκονται πάνω από αυτές μετακινούνται προς τα κάτω.

- ***Τερματισμός παιχνιδιού***

Ο τερματισμός του παιχνιδιού πραγματοποιείται όταν το υποσύστημα ελέγχου παιχνιδιού ανιχνεύσει ότι δεν είναι δυνατή η τοποθέτηση νέου tetromino στο πλέγμα. Συγκεκριμένα, κατά τη φάση δημιουργίας ενός νέου tetromino, ελέγχεται αν η αρχική του θέση στην κορυφή του πλέγματος επικαλύπτεται με ήδη κατειλημμένα κελιά. Σε περίπτωση που διαπιστωθεί σύγκρουση, το σύστημα καταλήγει ότι ο διαθέσιμος χώρος έχει εξαντληθεί και το παιχνίδι τερματίζεται. Η επανεκκίνηση του παιχνιδιού μπορεί να επιτευχθεί μόνο μέσω εξωτερικής εντολής, όπως η ενεργοποίηση ενός πλήκτρου επαναφοράς (reset).

Τα παραπάνω μπορούν να θεωρηθούν ως διακριτές καταστάσεις μιας πεπερασμένης μηχανής καταστάσεων (Finite State Machine - FSM), η οποία υλοποιεί τον έλεγχο ροής του παιχνιδιού. Κάθε κατάσταση αντιστοιχεί σε μία συγκεκριμένη φάση λειτουργίας του παιχνιδιού, ενώ οι μεταβάσεις μεταξύ τους καθορίζονται από γεγονότα όπως σήματα χρονισμού, συγκρούσεις και εντολές χρήστη (Εικόνα 37).



Εικόνα 37 Finite State Machine - FSM

Η ***κατάσταση Έναρξης ή Επαναφοράς*** αποτελεί την αρχική κατάσταση της FSM. Ενεργοποιείται κατά την εκκίνηση του συστήματος ή μετά από εξωτερική εντολή επαναφοράς. Κατά τη διάρκειά της, αρχικοποιούνται όλες οι βασικές μεταβλητές του παιχνιδιού, όπως το πλέγμα, οι μετρητές χρονισμού και οι καταχωρητές κατάστασης. Με την ολοκλήρωση της αρχικοποίησης, η FSM μεταβαίνει στην κατάσταση δημιουργίας νέου κομματιού.

Στην ***κατάσταση Δημιουργίας Κομματιού,*** δημιουργείται ένα νέο tetromino, από γεννήτρια με τυχαίο τρόπο, και τοποθετείται στην αρχική του θέση στην κορυφή του πλέγματος. Παράλληλα, πραγματοποιείται έλεγχος σύγκρουσης με ήδη κατειλημμένα κελιά, ώστε να διαπιστωθεί αν υπάρχει διαθέσιμος χώρος για τη συνέχιση του παιχνιδιού. Εφόσον η τοποθέτηση είναι εφικτή, η FSM μεταβαίνει στην κατάσταση ενεργού παιχνιδιού, ενώ σε αντίθετη περίπτωση οδηγείται στην κατάσταση τερματισμού.

Η ***κατάσταση Ενεργού Παιχνιδιού*** αποτελεί την κύρια λειτουργική φάση του συστήματος. Το ενεργό tetromino κινείται καθοδικά με βάση τα σήματα χρονισμού, ενώ παράλληλα γίνεται επεξεργασία των εντολών του χρήστη για οριζόντια μετακίνηση, περιστροφή και επιτάχυνση της πτώσης. Σε κάθε ενέργεια πραγματοποιείται έλεγχος εγκυρότητας, ώστε να διασφαλίζεται η συμμόρφωση με τους κανόνες του παιχνιδιού. Όταν διαπιστωθεί ότι το tetromino δεν μπορεί πλέον να κινηθεί προς τα κάτω, η FSM μεταβαίνει στην κατάσταση κλειδώματος.

Η ***κατάσταση κλειδώματος κομματιού*** ενεργοποιείται όταν το ενεργό tetromino έρθει σε επαφή με τη βάση του πλέγματος ή με κάποιο ήδη τοποθετημένο κομμάτι. Σε αυτή τη φάση, το tetromino ενσωματώνεται οριστικά στο πλέγμα και τα κελιά που το αποτελούν χαρακτηρίζονται ως κατειλημμένα. Με την ολοκλήρωση του κλειδώματος, η FSM μεταβαίνει στην κατάσταση ελέγχου και διαγραφής γραμμών.

Στην ***κατάσταση ελέγχου και διαγραφής γραμμών***, πραγματοποιείται έλεγχος για την ύπαρξη πλήρως συμπληρωμένων γραμμών στο πλέγμα. Οι γραμμές που πληρούν το κριτήριο πληρότητας διαγράφονται και οι γραμμές που βρίσκονται από πάνω, μετατοπίζονται προς τα κάτω. Μετά την ολοκλήρωση της διαδικασίας διαγραφής, το σύστημα επιστρέφει στην κατάσταση δημιουργίας νέου κομματιού.

Η ***κατάσταση τερματισμού*** ενεργοποιείται όταν δεν είναι δυνατή η τοποθέτηση νέου tetromino στο πλέγμα κατά τη φάση δημιουργίας κομματιού. Στην κατάσταση αυτή, η εξέλιξη του παιχνιδιού σταματάει και το σύστημα παραμένει αδρανές μέχρι να δοθεί εντολή επαναφοράς. Η κατάσταση αυτή σηματοδοτεί το τέλος της εκτέλεσης του παιχνιδιού.

Για την ολοκληρωμένη λειτουργία του παιχνιδιού, η FSM πρέπει να λαμβάνει σαφή και αξιόπιστα σήματα εισόδου από τον χρήστη. Τα σήματα αυτά προέρχονται από το υποσύστημα εισόδου, και μετά από διαδικασίες προεπεξεργασίας, όπως εξασφάλιση σταθερότητας και ακρίβειας, τα σήματα εισόδου χαρτογραφούνται σε ενέργειες που καθοδηγούν τη συμπεριφορά του ενεργού tetromino, όπως κίνηση, περιστροφή ή επιτάχυνση πτώσης.

Η χαρτογράφηση αυτή καθορίζει τον τρόπο με τον οποίο κάθε ενέργεια του χρήστη μέσω του χειριστηρίου, αντιστοιχεί σε συγκεκριμένη ενέργεια εντός του παιχνιδιού (Πίνακας 5). Η FSM ελέγχει παράλληλα την εγκυρότητα κάθε ενέργειας, ώστε να διασφαλίζεται ότι οι κινήσεις δεν οδηγούν σε σύγκρουση με τα όρια του πλέγματος ή με ήδη τοποθετημένα

κομμάτια. Με τον τρόπο αυτό, η χαρτογράφηση σημάτων ενσωματώνεται άμεσα στη λογική του παιχνιδιού και αποτελεί κρίσιμο μέρος του υποσυστήματος ελέγχου παιχνιδιού. Η σύνδεση αυτή επιτρέπει την άμεση αντίδραση του παιχνιδιού στις εντολές του χρήστη, εξασφαλίζοντας την ομαλή εξέλιξη του παιχνιδιού σε πραγματικό χρόνο.

Φυσική Είσοδος-Λειτουργία	Ενέργεια παιχνιδιού	Περιγραφή
Button – Έναρξη/ Περιστροφή	Έναρξη παιχνιδιού Περιστροφή tetromino	Γίνεται εκκίνηση του παιχνιδιού. Κατά τη διάρκεια του παιχνιδιού, περιστρέφει το tetromino κατά 90°, εφόσον είναι έγκυρη η κίνηση
Button - Επαναφορά	Επαναφορά παιχνιδιού	Επαναφέρει το παιχνίδι στην αρχική κατάσταση
Joystick - Κάτω	Επιτάχυνση πτώσης	Αυξάνει προσωρινά την ταχύτητα πτώσης του tetromino
Joystick - Αριστερά	Μετακίνηση Αριστερά	Μετακινεί το ενεργό tetromino μία θέση αριστερά, εφόσον δεν υπάρχει σύγκρουση
Joystick - Δεξιά	Μετακίνηση Δεξιά	Μετακινεί το ενεργό tetromino μία θέση δεξιά, εφόσον δεν υπάρχει σύγκρουση

Πίνακας 5 Χαρτογράφηση σημάτων χειριστηρίου

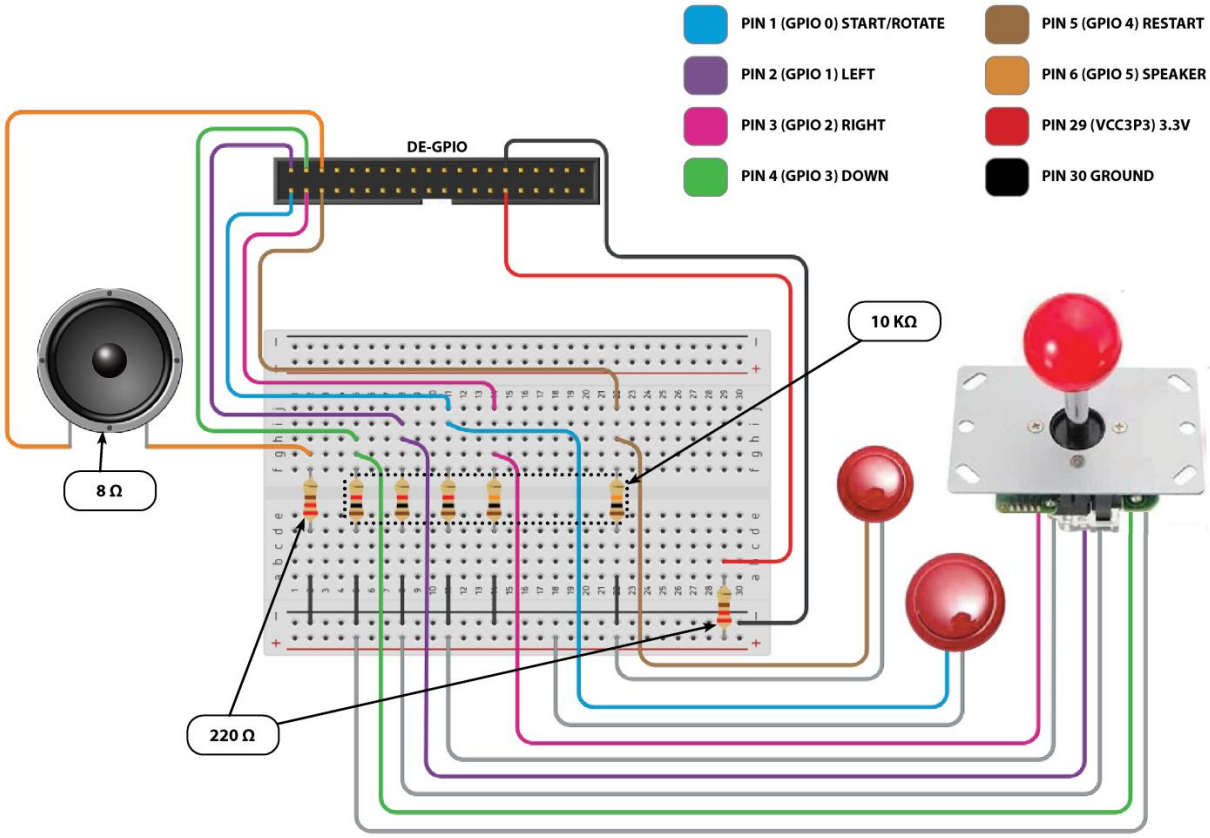
Με την ολοκλήρωση της σχεδίασης των επιμέρους υποσυστημάτων και την επιτυχή δοκιμή τους μέσω των ανεξάρτητων εφαρμογών, το σύστημα είναι έτοιμο για την τελική ενοποίηση. Η ιεραρχική προσέγγιση που ακολουθήθηκε διασφαλίζει ότι η κεντρική μηχανή καταστάσεων (FSM) μπορεί να διαχειριστεί τη ροή του παιχνιδιού βασιζόμενη σε αξιόπιστα σήματα εισόδου και να τροφοδοτήσει με ακρίβεια το υποσύστημα απεικόνισης, επιτυγχάνοντας μια σταθερή υλοποίηση του Tetris σε πραγματικό χρόνο.

4.2 Σχεδίαση συνολικού συστήματος και HDL υλοποίηση

Μετά την ανάλυση της αρχιτεκτονικής του συστήματος, ακολουθεί η φάση της συνολικής σχεδίασης, κατά την οποία προσδιορίζεται ο τρόπος διασύνδεσης και συνεργασίας των επιμέρους υποσυστημάτων. Σε αυτό το στάδιο, η μελέτη μεταβαίνει από την απομονωμένη ανάλυση των υποσυστημάτων, στη σύνθεση μιας ενιαίας αρχιτεκτονικής, η οποία καλείται να διαχειριστεί τη ροή της πληροφορίας σε πραγματικό χρόνο. Η συνολική σχεδίαση εστιάζει στον καθορισμό των πρωτοκόλλων επικοινωνίας μεταξύ των modules, τη διαχείριση των διαφορετικών τομέων ρολογιού και τον συγχρονισμό της λογικής του παιχνιδιού με τον ρυθμό ανανέωσης της οθόνης. Παράλληλα, οι σχεδιαστικές επιλογές αποτυπώνονται σε επίπεδο υλοποίησης μέσω της γλώσσας περιγραφής υλικού (HDL), και συγκεκριμένα σε Verilog, όπου τα επιμέρους υποσυστήματα υλοποιούνται ως διακριτά modules και διασυνδέονται στο ανώτερο επίπεδο σχεδίασης (πλήρης κώδικας στο Παράρτημα Γ). Στόχος είναι η δημιουργία ενός ολοκληρωμένου συστήματος, όπου η αλληλεπίδραση του χρήστη μεταφράζεται άμεσα σε οπτική και ακουστική πληροφορία, διατηρώντας την ακεραιότητα των δεδομένων και την ευστάθεια του χρονισμού. Με τον τρόπο αυτό διασφαλίζεται η συνοχή του συστήματος, η επεκτασιμότητά του και η αξιόπιστη λειτουργία του σε πραγματικό χρόνο.

4.2.1 Συνδεσμολογία

Η συνδεσμολογία του συστήματος αποτελεί το φυσικό επίπεδο υλοποίησης και καθορίζει τον τρόπο με τον οποίο τα εξωτερικά περιφερειακά διασυνδέονται με το FPGA. Στόχος είναι η αξιόπιστη μεταφορά των σημάτων εισόδου από τον χρήστη προς το σύστημα, εξασφαλίζοντας την ηλεκτρική προστασία και τη σταθερότητα της λειτουργίας. Η Εικόνα 38 παρουσιάζει τη φυσική συνδεσμολογία της πλακέτας DE23-Lite με το χειριστήριο τύπου arcade (joystick), δύο κουμπιών (buttons) και ενός ηχείου μέσω ενός breadboard.



Εικόνα 38 Φυσική συνδεσμολογία DE23-Lite με χειριστήριο, κουμπιά εισόδου και ηχείο

Συγκεκριμένα, οι ακροδέκτες GPIO της πλακέτας συνδέονται με ένα χειριστήριο τύπου arcade, δύο κουμπιά και ένα ηχείο. Από το χειριστήριο χρησιμοποιούνται τρεις λειτουργίες, οι οποίες αντιστοιχούν στις κινήσεις αριστερά, δεξιά και κάτω, και συνδέονται στα pins εισόδου 2,3,4 αντίστοιχα. Επιπλέον, το μεγαλύτερο κουμπί, που χρησιμοποιείται για την έναρξη του παιχνιδιού και την περιστροφή των κομματιών, συνδέεται στο pin εισόδου 1, ενώ το μικρότερο, που χρησιμοποιείται για την επανεκκίνηση του παιχνιδιού, αντιστοιχίζεται στο pin εισόδου 5. Κάθε γραμμή εισόδου συνδέεται με μια αντίσταση τιμής 10 kΩ, η οποία λειτουργεί ως pull-down αντίσταση, διασφαλίζοντας ότι το σήμα διατηρεί καθορισμένη λογική στάθμη όταν ο διακόπτης είναι ανοικτός. Όσον αφορά την έξοδο ήχου, το ηχείο αντίστασης 8 Ω συνδέεται με το pin εισόδου 6 παρεμβάλλοντας μια προστατευτική αντίσταση 220Ω. Για την προστασία γενικά του κυκλώματος και τον περιορισμό του ρεύματος, ενσωματώνεται άλλη μια αντίσταση 220Ω. Η τροφοδοσία του κυκλώματος πραγματοποιείται από τις γραμμές

3.3V και γείωσης (GND) της πλακέτας, εξασφαλίζοντας την ομαλή λειτουργία όλων των επιμέρους συνδέσεων.

4.2.2 Υποσύστημα απεικόνισης – HDMI_Controller

Η ανάπτυξη του συστήματος βασίστηκε σε ιεραρχική και αρθρωτή σχεδιαστική προσέγγιση, με διαχωρισμό μεταξύ υποσυστημάτων απεικόνισης, εισόδου, ελέγχου παιχνιδιού και ήχου. Αφετηρία αποτέλεσε η αξιοποίηση των έτοιμων δυνατοτήτων της πλακέτας DE23-Lite, η οποία διαθέτει ενσωματωμένο HDMI transmitter και έτοιμο ελεγκτή διαμόρφωσης μέσω του πρωτοκόλλου I2C (Inter-Integrated Circuit) (πλήρης κώδικας στο Παράρτημα Β). Ο ελεγκτής αυτός λειτουργεί σε χαμηλή συχνότητα χρονισμού 20 KHz, διασφαλίζοντας την μεταφορά 33 πακέτων ρυθμίσεων (LUT Data) προς τους καταχωρητές του πομπού. Με την επιτυχή ολοκλήρωση της παραμετροποίησης, ο ελεγκτής θέτει το σήμα READY σε υψηλή λογική στάθμη, υποδηλώνοντας ότι ο πομπός είναι έτοιμος να δεχθεί δεδομένα βίντεο. Με τον τρόπο αυτό, το υποσύστημα απεικόνισης θεωρείται λειτουργικά ολοκληρωμένο, επιτρέποντας την εστίαση της σχεδίασης στην παραγωγή των δεδομένων εικόνας και όχι στη διαχείριση του πρωτοκόλλου μετάδοσης HDMI.

Για την υποστήριξη της ανάλυσης υψηλής ευκρίνειας, η οποία απαιτεί ρυθμό μετάδοσης δεδομένων υψηλότερο από το βασικό ρολόι του συστήματος (50MHz), κρίνεται απαραίτητη η χρήση ενός κυκλώματος PLL (Phase-Locked Loop). Στην παρούσα υλοποίηση, χρησιμοποιήθηκε η έτοιμη δομή Video PLL της Intel, η οποία αναλαμβάνει τον πολλαπλασιασμό και τη διαίρεση της συχνότητας εισόδου. Συγκεκριμένα, η **μονάδα Vedio PLL** λειτουργεί ως ένας αναλογικός και ψηφιακός πολλαπλασιαστής συχνότητας, ο οποίος λαμβάνει ως σήμα αναφοράς το ρολόι των 50MHz της πλακέτας και παράγει στην έξοδο το pixel clock στα 148,5MHz. Η τιμή αυτή είναι καθορισμένη από το πρότυπο συγχρονισμού για την απεικόνιση 1920x1080 στα 60Hz, καθώς αντιστοιχεί στο γινόμενο των συνολικών pixel ανά γραμμή (2200), των συνολικών γραμμών ανά πλαίσιο (1125) και του ρυθμού ανανέωσης (60 frames/sec). Η χρήση του PLL εξασφαλίζει τη σταθερότητα του σήματος έτσι ώστε η σάρωση της οθόνης να παραμένει συγχρονισμένη με τον ρυθμό ανανέωσης των 60Hz, αποτρέποντας αποτρέποντας φαινόμενα ολίσθησης της εικόνας ή απώλειας σήματος από τον δέκτη HDMI.

4.2.3 Αναπαράσταση πλέγματος – module grid_generator

Ο σχεδιασμός συνεχίζεται με την αναπαράσταση του πλέγματος ως ξεχωριστό module, το οποίο λειτουργεί ως σύνδεσμος μεταξύ της λογικής του παιχνιδιού και της οπτικής απεικόνισης. Η μονάδα αναλαμβάνει την παραγωγή των σημάτων χρονισμού για ανάλυση 1920×1080, σύμφωνα με τις προεπιλεγμένες παραμέτρους χρονισμού της Terasic (Πίνακας 6), και την απόδοση χρώματος σε πραγματικό χρόνο για κάθε παραγόμενο pixel.

Παράμετρος χρονισμού	Οριζόντια	Κάθετα
Ενεργή περιοχή (Active Area)	1920	1080
Front Porch (FP)	88	4
Παλμός Συγχρονισμού (SYNC)	44	5
Back Porch (BP)	148	36
Σύνολο (TOTAL)	2200	1125

Πίνακας 6 Παράμετροι χρονισμού για ανάλυση 1920x1080 60Hz (Προεπιλογή από Terasic)

Το πλέγμα του παιχνιδιού σχεδιάζεται ως μονοδιάστατη δομή συνολικού πλάτους 200 bits (grid_flat[199:0]), που αντιστοιχούν σε διάταξη 10 στηλών και 20 γραμμών, μειώνοντας την πολυπλοκότητα της σύνθεσης. Το μέγεθος κάθε κελιού επιλέχθηκε να αντιστοιχεί σε τετράγωνο μεγέθους 88 pixels, ώστε το συνολικό πλέγμα να είναι κεντραρισμένο στην ενεργή περιοχή της οθόνης για συμμετρική απεικόνιση, έχοντας υπολογίσει και το χώρο που καταλαμβάνει το πλαίσιο γύρω από αυτή.

```
// Grid parameters
localparam GRID_COLS   = 10;
localparam GRID_ROWS   = 20;
localparam SQUARE_SIZE = 88;

// Grid total dimensions
localparam GRID_PIXEL_H = GRID_COLS * SQUARE_SIZE;
localparam GRID_PIXEL_W = GRID_ROWS * SQUARE_SIZE;

// Center grid in screen
localparam GRID_START_X = (H_ACTIVE - GRID_PIXEL_W)/2;
localparam GRID_START_Y = (V_ACTIVE - GRID_PIXEL_H)/2;
```

Γύρω από κάθε τετράγωνο σχεδιάζονται γραμμές διαχωρισμού μεγέθους 2 pixel για καλύτερη οπτική διάκριση.

```
wire vertical_line = (pixel_y_in_tile < 2) || (pixel_y_in_tile >=
SQUARE_SIZE-2);
wire horizontal_line = (pixel_x_in_tile < 2) || (pixel_x_in_tile >=
SQUARE_SIZE-2);
```

Η βασική σχεδιαστική ιδέα στηρίζεται στην απόδοση χρώματος σε πραγματικό χρόνο για κάθε παραγόμενο pixel, αποφεύγοντας τη χρήση ενδιάμεσης μνήμης. Ο εντοπισμός της θέσης στην οθόνη επιτυγχάνεται μέσω δύο μετρητών σάρωσης (h_cnt, v_cnt) που συγχρονίζονται με το pixel clock (148,5MHz), όπου αυξάνονται σε κάθε ακμή του ρολογιού και μηδενίζονται όταν φτάσουν στις τιμές H_TOTAL-1 και V_TOTAL-1 αντίστοιχα, διασφαλίζοντας τον συγχρονισμό της εικόνας.

```
//  Horizontal scan counter
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        h_cnt <= 12'd0;
    end else begin
        if (h_cnt == H_TOTAL-1)
            h_cnt <= 12'd0;
        else
            h_cnt <= h_cnt + 12'd1;
    end
end
//  Vertical scan counter
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        v_cnt <= 12'd0;
    end else if (h_cnt == H_TOTAL-1) begin
        if (v_cnt == V_TOTAL-1)
            v_cnt <= 12'd0;
        else
            v_cnt <= v_cnt + 12'd1;
    end
end
```

Το σύστημα, προσδιορίζει το χρώμα κάθε pixel, με βάση τη θέση του στην οθόνη και τη σχέση αυτής της θέσης με το πλέγμα. Συγκεκριμένα, υπολογίζεται αν το pixel ανήκει στην περιοχή του πλέγματος (inside_grid), αν ναι σε ποιο κελί αντιστοιχεί (μέσω διαίρεσης των συντεταγμένων με το SQUARE_SIZE), αν βρίσκεται σε γραμμή διαχωρισμού και τέλος αν το αντίστοιχο bit στο grid_flat είναι ενεργό ή όχι. Επομένως, για κάθε pixel που σαρώνεται, η μονάδα αναζητά σε πραγματικό χρόνο την κατάσταση του αντίστοιχου κελιού.

```
// Does the rendered pixel belong to the whole grid?
wire inside_grid = (h_cnt >= GRID_START_X) && (h_cnt < GRID_START_X +
GRID_PIXEL_W) && (v_cnt >= GRID_START_Y) && (v_cnt < GRID_START_Y +
GRID_PIXEL_H);

// Square coordinates in the grid
wire [3:0] col = (v_cnt - GRID_START_Y) / SQUARE_SIZE; // vertical index
wire [4:0] row = (h_cnt - GRID_START_X) / SQUARE_SIZE; // horizontal
index

// Does the rendered pixel belong to the square?
wire [6:0] pixel_x_in_tile = h_cnt - (GRID_START_X + row*SQUARE_SIZE);
wire [6:0] pixel_y_in_tile = v_cnt - (GRID_START_Y + col*SQUARE_SIZE);

// Does the rendered pixel belong to a grid line?
wire on_grid_line = inside_grid && (vertical_line || horizontal_line);

// Is this cell set to be on? (white)
wire cell_on = inside_grid && grid_flat[row*GRID_COLS + col_flipped];
```

Η τελική απόδοση χρώματος γίνεται σε πραγματικό χρόνο, σε κάθε κύκλο του pixel clock, με βάση την τρέχουσα κατάσταση του πλέγματος.

```
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        // Clean whole screen on reset
        vpg_r <= 8'h00;vpg_g <= 8'h00;vpg_b <= 8'h00;
    end else if (!inside_grid) begin
        // Gray pixels outside grid
        vpg_r <= 8'h80;vpg_g <= 8'h80;vpg_b <= 8'h80;
    end else if (on_grid_line) begin
        // Black grid lines
        vpg_r <= 8'h00;vpg_g <= 8'h00;vpg_b <= 8'h00;
    end else if (cell_on) begin
        // White square
        vpg_r <= 8'hFF;vpg_g <= 8'hFF;vpg_b <= 8'hFF;
    end else begin
        // Black square
        vpg_r <= 8'h00;vpg_g <= 8'h00;vpg_b <= 8'h00;
    end
end
```

Η σχεδίαση προβλέπει ασπρόμαυρη απεικόνιση και για καλύτερη οπτική ευκρίνεια δημιουργείται πλαίσιο, γκρι χρώματος, γύρω από το συνολικό πλέγμα. Η οθόνη αρχικοποιείται σε μαύρο χρώμα, τα ενεργά κελιά σε άσπρο και οι γραμμές διαχωρισμού σε μαύρο χρώμα.

Δεδομένου ότι η επιλεγμένη οθόνη HDMI λειτουργεί σε οριζόντιο προσανατολισμό, πραγματοποιήθηκε αντιστροφή της σχεδιαστικής λογικής των αξόνων, ώστε να

αξιοποιηθεί το διαθέσιμο εύρος της οθόνης και να βελτιωθεί η εμπειρία του παιχνιδιού του χρήστη.

```
wire [3:0] col_flipped = GRID_COLS - 1 - col;
```

4.2.4 Αναπαράσταση tetrominoes – module tetromino_rom

Στη συνέχεια, για τη διαχείριση και την αναπαράσταση των tetrominoes, επιλέχθηκε η υλοποίηση ενός ξεχωριστού module που λειτουργεί ως στατική δομή δεδομένων (ROM), με στόχο την αποθήκευση όλων των τύπων (I, O, T, S, Z, J, L) και όλων των δυνατών καταστάσεων περιστροφής τους.

Η λογική σχεδίασης αφορά τη δημιουργία ενός τοπικού πλέγματος 4x4, όπου μέσα σε αυτό γίνεται η τοποθέτηση των τεσσάρων τετραγώνων που απαρτίζουν κάθε tetromino, ορίζοντας συγκεκριμένες συντεταγμένες, ώστε να καλύπτει όλους τους τύπους και όλες τις δυνατές περιστροφές (Εικόνα 38). Το συγκεκριμένο πλέγμα επιλέχθηκε ως το ελάχιστο που καλύπτει το μεγαλύτερο tetromino (I) σε οριζόντια και κάθετη διάταξη, ενώ ταυτόχρονα εξασφαλίζει συμμετρία και ευκολία υπολογισμών.

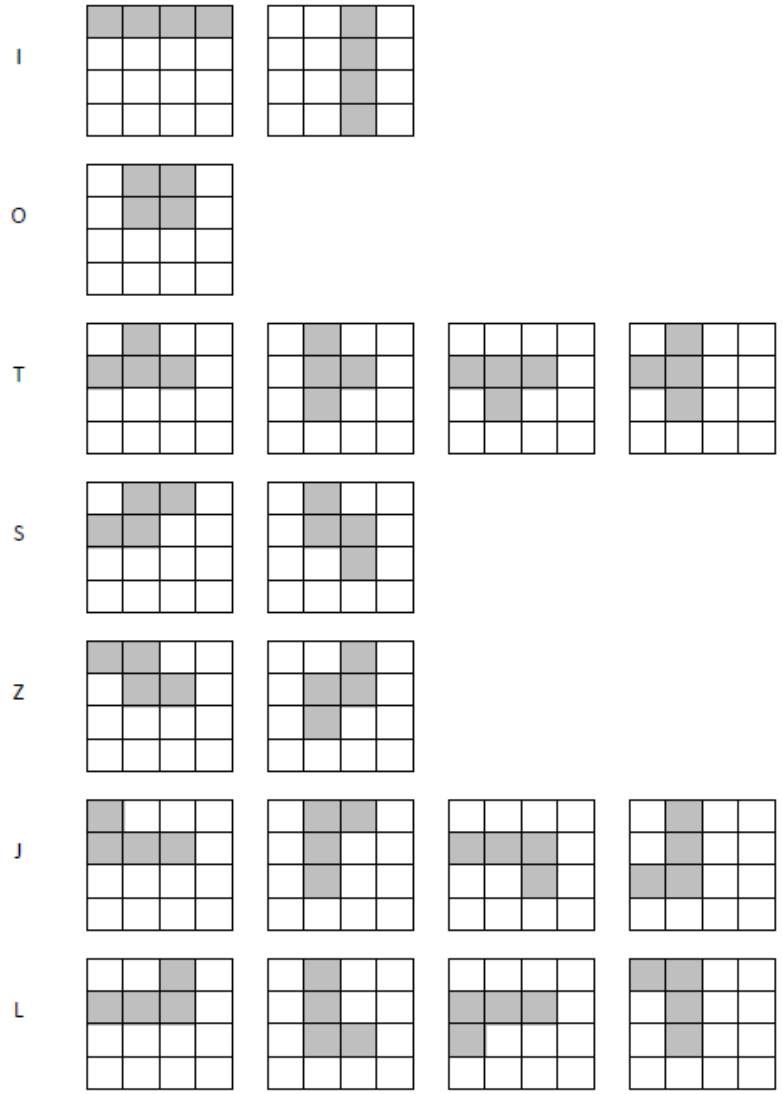
Το module δέχεται ως εισόδους τον τύπο του κομματιού (piece_type), την κατάσταση περιστροφής (rotation_state) και τον δείκτη (block_index) που αντιστοιχεί σε κάθε ένα από τα τέσσερα τετράγωνα του σχήματος. Ως έξοδο, επιστρέφει τις σχετικές μετατοπίσεις dx και dy, οι οποίες υποδεικνύουν τη θέση του κομματιού στο πλέγμα. Η υλοποίηση βασίζεται σε μια ιεραρχική δομή case statements, η οποία επιτρέπει την γρήγορη πρόσβαση στις συντεταγμένες.

```
case (piece_type)
//I piece
   3'b000: begin
        case (rotation_state)
            2'b00, 2'b10: begin // horizontal
                case (block_index)
                    2'b00: begin dx=0; dy=0; end
                    2'b01: begin dx=1; dy=0; end
                    2'b10: begin dx=2; dy=0; end
                    2'b11: begin dx=3; dy=0; end
                endcase
            end
            2'b01, 2'b11: begin // vertical
                case (block_index)
                    2'b00: begin dx=2; dy=0; end
                    2'b01: begin dx=2; dy=1; end
                    2'b10: begin dx=2; dy=2; end
                    2'b11: begin dx=2; dy=3; end
```

```
                endcase
            end
        endcase
    end

//O piece
    3'b001: begin
        case (block_index)
            2'b00: begin dx=1; dy=0; end
            2'b01: begin dx=2; dy=0; end
            2'b10: begin dx=1; dy=1; end
            2'b11: begin dx=2; dy=1; end
        endcase
    end

// λοιπά σχήματα
```

Λόγω της γεωμετρικής συμμετρίας ορισμένων σχημάτων, ομαδοποιούνται οι περιστροφές που παράγουν το ίδιο αποτέλεσμα. Έτσι, για τα tetrominoes I, S, Z οι καταστάσεις 2'b00, 2'b10 αντιμετωπίζονται κοινά όπως και οι 2'b01, 2'b11 αποθηκεύοντας ως δυο διαφορετικές καταστάσεις. Για τα T, J, L αποθηκεύονται τέσσερις καταστάσεις ενώ για το O, το οποίο παρουσιάζει πλήρη συμμετρία, επιστρέφεται η ίδια γεωμετρία ανεξαρτήτως περιστροφής.

Η σημασία του συγκεκριμένου module για τη συνολική λογική του παιχνιδιού είναι καθοριστική, καθώς παρέχει την απαραίτητη πληροφορία για την οπτική απεικόνιση των tetrominoes στην οθόνη, ενώ παράλληλα επιτρέπει στη μονάδα ελέγχου να πραγματοποιεί σημαντικούς υπολογισμούς, όπως τον έλεγχο συγκρούσεων με τα τοιχώματα ή τα ήδη τοποθετημένα κομμάτια. Επομένως, λειτουργεί ως ενδιάμεσο επίπεδο μεταξύ του υποσυστήματος απεικόνισης και του υποσυστήματος ελέγχου παιχνιδιού, συνδέοντας τη στατική περιγραφή των σχημάτων με τη δυναμική εξέλιξη του παιχνιδιού.



Εικόνα 39 Tetromino ROM

4.2.5 Υποσύστημα εισόδου – module input_controller

Μετά τον καθορισμό της οπτικής αναπαράστασης, η σχεδίαση επικεντρώνεται στην αλληλεπίδραση του χρήστη με το σύστημα. Το υποσύστημα εισόδου αποτελεί τη σύνδεση μεταξύ των φυσικών ενεργειών του παίκτη και της ψηφιακής λογικής του παιχνιδιού, μετατρέποντας πέντε βασικές εντολές όπως κίνηση αριστερά, δεξιά, κάτω, έναρξη/περιστροφή (fire) και reset , σε σταθερά σήματα ελέγχου.

Δεδομένου ότι τα σήματα από τα κουμπιά και το joystick είναι ασύγχρονα ως προς το ρολόι του συστήματος (50 MHz), σχεδιάζεται μηχανισμός συγχρονισμού που αφορά τη χρήση δύο διαδοχικών flip-flop για κάθε γραμμή εισόδου ώστε να μειωθεί η πιθανότητα

μετάδοσης μεταστατικών φαινομένων. Κάθε γραμμή εισόδου διέρχεται διαδοχικά από δύο καταχωρητές (sync_ 0, sync_1), συγχρονισμένους με το ρολόι του συστήματος (clk).

```
    reg [4:0] sync_0;
    reg [4:0] sync_1;

    always @(posedge clk) begin
        sync_0 <= gpio_d;
        sync_1 <= sync_0;
    end
```

Η πρώτη καταχώριση (sync_0) δειγματοληπτεί το ασύγχρονο σήμα (gpio_d), ενώ η δεύτερη (sync_1) επαναδειγματοληπτεί την ήδη σταθεροποιημένη τιμή στο επόμενο μέτωπο ρολογιού. Με τον τρόπο αυτό, μειώνεται η πιθανότητα μετάδοσης μεταστατικών φαινομένων στο υπόλοιπο σύστημα.

Επιπλέον, σχεδιάζεται κύκλωμα αποθορυβοποίησης, ώστε να εξαλείφονται τα ανεπιθύμητα πολλαπλά σήματα που προκαλούνται από μηχανικές ταλαντώσεις των διακοπτών. Η λειτουργία του βασίζεται στη χρήση μετρητών χρονικής επιβεβαίωσης. Για κάθε είσοδο υλοποιείται ανεξάρτητος μετρητής που ενεργοποιείται όταν η συγχρονισμένη είσοδος διαφέρει από την τρέχουσα αποθηκευμένη κατάσταση εξόδου.

Η μέγιστη τιμή του μετρητή ορίζεται στα 10ms, και για να εξασφαλιστεί αυτό το χρονικό διάστημα απαιτούνται 500.000 κύκλοι ρολογιού για 50MHz ρολόι συστήματος.

```
localparam DEBOUNCE_MAX = 500_000; // 10ms @ 50MHz
```

Αν η νέα τιμή παραμείνει σταθερή για προκαθορισμένο χρονικό διάστημα (10 ms) τότε η έξοδος ενημερώνεται με τη νέα τιμή. Χαρακτηριστικό απόσπασμα της λογικής αποθορυβοποίησης για την είσοδο «left» παρουσιάζεται παρακάτω, όπου η ίδια δομή επαναλαμβάνεται για τις υπόλοιπες εισόδους:

```
// LEFT

if (sync_1[1] != left_r) begin
    if (cnt_left == DEBOUNCE_MAX) begin
       left_r  <= sync_1[1];
       cnt_left <= 0;
    end
    else
       cnt_left <= cnt_left + 1;
end
else
    cnt_left <= 0;
```

Τελικά, οι σταθεροποιημένες αυτές τιμές αποδίδονται ως έξοδοι του υποσυστήματος εισόδου και αποτελούν τα σήματα ελέγχου που τροφοδοτούν το υποσύστημα λογικής του παιχνιδιού (Game Logic). Με τον τρόπο αυτό, διασφαλίζεται ότι οι κινήσεις του παίκτη μετατρέπονται σε συγχρονισμένες και αξιόπιστες εντολές προς το σύστημα.

4.2.6 Υποσύστημα ήχου – module audio_controller

Παράλληλα με την οπτική αναπαράσταση και τον έλεγχο των κινήσεων, η σχεδίαση συνεχίζεται με τη δημιουργία ενός υποσυστήματος ήχου, με σκοπό τη δημιουργία απλών ηχητικών σημάτων που συνοδεύουν βασικά γεγονότα του παιχνιδιού. Η σχεδίαση βασίζεται στη δημιουργία ψηφιακού τετραγωνικού σήματος συγκεκριμένης συχνότητας και διάρκειας, το οποίο οδηγείται απευθείας σε ένα ηχείο συνδεδεμένο σε GPIO ακροδέκτη της πλακέτας.

Η λογική λειτουργίας του βασίζεται σε τρία διαφορετικά σήματα εισόδου, τα οποία ενεργοποιούνται από το υποσύστημα ελέγχου παιχνιδιού (Game Logic) σε συγκεκριμένα γεγονότα κατά τη διάρκεια του παιχνιδιού, όπως το κλείδωμα του tetromino, την εκκαθάριση γραμμής και τον τερματισμό του παιχνιδιού. Κάθε σήμα ενεργοποιεί διαφορετική ακολουθία παραγωγής ήχου, έτσι ώστε το κάθε γεγονός να ξεχωρίζει ακουστικά και να προσφέρει στον παίκτη άμεση ανατροφοδότηση.

Οι τιμές διάρκειας ορίζονται μέσω των παραμέτρων DUR_SHORT και DUR_LONG, οι οποίες αντιστοιχούν σε 100 ms και 600 ms αντίστοιχα (με βάση τη συχνότητα λειτουργίας του ρολογιού των 50 MHz).

```
//σύντομος ήχος (100ms)- κλείδωμα,εκκαθάριση γραμμής
localparam DUR_SHORT = 5_000_000;
//παρατεταμένος ήχος (600ms)- game over
localparam DUR_LONG  = 30_000_000;
```

Για την παραγωγή του σήματος χρησιμοποιούνται δύο βασικοί μετρητές. Ο πρώτος μετρητής (dur_cnt) καθορίζει τη συνολική διάρκεια του ήχου, ενώ ο δεύτερος μετρητής (pm_cnt) χρησιμοποιείται για τον έλεγχο της συχνότητας του παραγόμενου σήματος. Όσο η κατάσταση αναπαραγωγής (play) είναι ενεργή, ο μετρητής dur_cnt αυξάνεται σε κάθε κύκλο ρολογιού και συγκρίνεται με την τιμή dur_limit_curr, η οποία καθορίζει τη διάρκεια του ήχου. Όταν η συνθήκη dur_cnt < dur_limit_curr σταματά να ισχύει, η αναπαραγωγή τερματίζεται και η έξοδος speaker επανέρχεται στη μηδενική της κατάσταση. Παράλληλα, όσο ο ήχος βρίσκεται σε αναπαραγωγή, η τιμή του μετρητή

pm_cnt συγκρίνεται με τη μεταβλητή freq_limit, η οποία καθορίζει τη συχνότητα του παραγόμενου σήματος. Όταν ο μετρητής φτάσει την τιμή αυτή, μηδενίζεται και η έξοδος speaker αντιστρέφεται, δημιουργώντας έτσι ένα τετραγωνικό κύμα.

```
if (play) begin
   if (dur_cnt < dur_limit_curr) begin
       dur_cnt <= dur_cnt + 1;
       if (pm_cnt >= freq_limit) begin
           pm_cnt <= 0;
           speaker <= ~speaker;
       end else begin
           pm_cnt <= pm_cnt + 1;
       end
   end else begin
       play    <= 0;
       speaker <= 0;
       pm_cnt <= 0;
   end
end
```

Η ενεργοποίηση της αναπαραγωγής ενός ήχου πραγματοποιείται μέσω ανίχνευσης ανερχόμενου μετώπου στα bits του σήματος sound. Για τον σκοπό αυτό αποθηκεύεται η προηγούμενη κατάσταση του σήματος sound στο register sound_d, το οποίο ενημερώνεται σε κάθε κύκλο ρολογιού (sound_d <= sound). Η μετάβαση από 0 σε 1 ανιχνεύεται μέσω της συνθήκης (sound[i] && !sound_d[i]), όπου i αντιστοιχεί στο κάθε ηχητικό γεγονός. Με τον τρόπο, η αναπαραγωγή ενεργοποιείται μόνο τη στιγμή της μετάβασης του σήματος και αποφεύγεται η συνεχής ενεργοποίηση όσο το σήμα παραμένει σε λογικό 1.

Ανάλογα με το bit του sound που ενεργοποιείται, επιλέγονται διαφορετικές τιμές συχνότητας και διάρκειας. Για παράδειγμα, στο κλείδωμα ενός tetromino ενεργοποιείται ένας σύντομος υψηλός τόνος (freq_limit = 200000), στην εκκαθάριση γραμμής παράγεται χαμηλότερος τόνος (freq_limit = 300000) και στην περίπτωση τερματισμού του παιχνιδιού ενεργοποιείται ένας σημαντικά χαμηλότερος και μεγαλύτερης διάρκειας τόνος (freq_limit = 500000), δημιουργώντας ένα ηχητικό αποτέλεσμα που διαφοροποιείται από τα υπόλοιπα γεγονότα.

```
if ((sound[0] && !sound_d[0])) begin
    play          <= 1;
    dur_limit_curr<= DUR_SHORT;
    freq_limit    <= 200_000; // HIGH
    dur_cnt       <= 0;
end
else if ((sound[1] && !sound_d[1])) begin
    play          <= 1;
    dur_limit_curr<= DUR_SHORT;
    freq_limit    <= 300_000; // LOW
    dur_cnt       <= 0;
```

```
end
else if ((sound[2] && !sound_d[2])) begin
    play          <= 1;
    dur_limit_curr<= DUR_LONG;
    freq_limit    <= 500_000; // VERY LOW
    dur_cnt       <= 0;
end
```

4.2.7 Υποσύστημα ελέγχου παιχνιδιού – module game_logic

Η λειτουργία του παιχνιδιού υλοποιείται στο υποσύστημα ελέγχου παιχνιδιού, το οποίο αποτελεί τον κεντρικό ελεγκτή του συστήματος και είναι υπεύθυνο για τη διαχείριση της ροής του παιχνιδιού, την ενημέρωση του πλέγματος, τον έλεγχο συγκρούσεων, τη δημιουργία νέων tetromino και την ενεργοποίηση των αντίστοιχων ηχητικών γεγονότων. Η συμπεριφορά του υποσυστήματος οργανώνεται σύμφωνα με μια Μηχανή Πεπερασμένων Καταστάσεων (FSM), η οποία καθορίζει τα διαδοχικά στάδια λειτουργίας του παιχνιδιού. Κάθε κατάσταση αντιστοιχεί σε ένα συγκεκριμένο στάδιο της εκτέλεσης, όπως η αρχικοποίηση του συστήματος (start/reset), η δημιουργία νέου κομματιού, η κατάσταση ενεργού παιχνιδιού, το κλείδωμα ενός tetromino, ο έλεγχος και η διαγραφή πλήρων γραμμών, καθώς και η κατάσταση τερματισμού του παιχνιδιού. Η μετάβαση μεταξύ των καταστάσεων πραγματοποιείται με βάση γεγονότα που προκύπτουν από την κατάσταση του πλέγματος, τα σήματα εισόδου του χειριστηρίου και την ανίχνευση συγκρούσεων.

- **Κατάσταση Έναρξης/Επαναφοράς**

Η κατάσταση έναρξης ή επαναφοράς αποτελεί το αρχικό στάδιο λειτουργίας του Game Logic και ενεργοποιείται όταν το σύστημα δεχθεί σήμα επαναφοράς (reset_n). Στην κατάσταση αυτή πραγματοποιείται η αρχικοποίηση όλων των βασικών μεταβλητών που χρησιμοποιούνται για την υλοποίηση του παιχνιδιού, ώστε το σύστημα να ξεκινήσει από μια προκαθορισμένη κατάσταση. Συγκεκριμένα, μηδενίζονται τα πλέγματα που αποθηκεύουν την κατάσταση του παιχνιδιού (grid_flat, stack_grid, active_piece), αρχικοποιούνται οι μεταβλητές που καθορίζουν τον τύπο και τη θέση του ενεργού tetromino, ενώ παράλληλα ενεργοποιείται η οθόνη έναρξης (show_start). Επιπλέον, η μεταβλητή που καθορίζει την ταχύτητα πτώσης των tetromino (FALL_FREQ) επαναφέρεται στην αρχική της τιμή. Ταυτόχρονα, καθορίζεται η αρχική θέση του

tetromino μέσω των μεταβλητών base_x και base_y, ώστε να βρίσκεται σχεδόν στο κέντρο της οθόνης.

```
if (!reset_n) begin
    grid_flat      <= 200'b0;
    piece_type     <= 3'd0;
    rotation_state <= 2'd0;
    base_x         <= 3;
    base_y         <= 0;
    stack_grid     <= 200'b0;
    active_piece   <= 200'b0;
    joy_rotate_p   <= 0;
    show_start     <= 1;
    game_over      <= 0;
    sound          <= 3'b000;
    FALL_FREQ      <= INITIAL_FALL_FREQ;
end
```

Μετά την ολοκλήρωση της αρχικοποίησης, το σύστημα παραμένει στην κατάσταση έναρξης μέχρι ο χρήστης να δώσει εντολή εκκίνησης (start) του παιχνιδιού. Κατά τη διάρκεια αυτής της κατάστασης εμφανίζεται στην οθόνη η ένδειξη “PLAY”, όπου μέσω δυαδικών μοτίβων καθορίζουν ποια κελιά θα ενεργοποιηθούν ώστε να σχηματιστεί το κάθε γράμμα στο πλέγμα (grid_flat).

```
if (show_start) begin
//display "PLAY"
   grid_flat[  9:  0] <= 10'b0001111000; // row 0 (P)
   grid_flat[ 19: 10] <= 10'b0001001000; // row 1
   grid_flat[ 29: 20] <= 10'b0001111000; // row 2

// continue with L, A, Y

end
```

Κατά το χρονικό αυτό διάστημα, το σύστημα δεν εκτελεί καμία άλλη λειτουργία παιχνιδιού και παραμένει σε κατάσταση αναμονής. Η μετάβαση στην επόμενη κατάσταση πραγματοποιείται μόνο όταν ο παίκτης πατήσει το κουμπί εκκίνησης. Η εντολή αυτή ανιχνεύεται μέσω του πλήκτρου έναρξης/περιστροφής (joy_rotate), ενώ χρησιμοποιείται και η αποθήκευση της προηγούμενης κατάστασης του σήματος (joy_rotate_p) ώστε να εντοπιστεί η μετάβαση από 0 σε 1. Με αυτόν τον τρόπο, ενεργοποιείται η έναρξη της κανονικής ροής του παιχνιδιού και τη δημιουργία του πρώτου tetromino.

```
joy_rotate_p <= joy_rotate;
if (joy_rotate && !joy_rotate_p)
    show_start <= 0;
```

- **Κατάσταση Δημιουργίας Κομματιού**

Μετά την ενεργοποίηση της εκκίνησης από τον χρήστη, ενεργοποιείται η κατάσταση δημιουργίας κομματιού, στην οποία παράγεται ένα νέο tetromino που θα εισαχθεί στο πλέγμα του παιχνιδιού. Σκοπός είναι η επιλογή του τύπου του κομματιού, η αρχικοποίηση των συντεταγμένων του και η τοποθέτησή του στην αρχική θέση εμφάνισης στο επάνω μέρος του πλέγματος. Η διαδικασία αυτή εκτελείται πριν ξεκινήσει η κανονική εξέλιξη του παιχνιδιού, ώστε το νέο κομμάτι να είναι έτοιμο να κινηθεί προς τα κάτω.

Η επιλογή του τύπου του tetromino γίνεται μέσω μιας γεννήτριας ψευδοτυχαίων αριθμών τύπου Linear Feedback Shift Register (LFSR). Η LFSR είναι ένας 3-bit καταχωρητής ολίσθησης, ο οποίος παράγει μια επαναλαμβανόμενη ψευδοτυχαία ακολουθία, επιτρέποντας στο παιχνίδι να δημιουργεί διαφορετικούς τύπους κομματιών σε κάθε εκτέλεση.

Για τη λειτουργία της LFSR, αρχικά εφαρμόζεται λογικό XOR ανάμεσα στο 3ο bit και το 1ο bit της τιμής (lfsr_feedback = lfsr[2]^ lfsr[0]). Το αρχικό περιεχόμενο του καταχωρητή ολισθαίνει κατά μία θέση προς τα αριστερά και την τελευταία θέση παίρνει το αποτέλεσμα της πράξης XOR ({lfsr[1:0], lfsr_feedback}), παράγοντας, έτσι, μια ακολουθία από 7 διαφορετικές τιμές (από το 1 έως το 7). Η τιμή της LFSR αρχικοποιείται στην τιμή 001 και όχι 000, έτσι ώστε η αρχική πράξη XOR να μην έχει την τιμή 0, με αποτέλεσμα και ο καταχωρητής να έχει μόνιμα την τιμή 0.

```
reg [2:0] lfsr;
wire lfsr_feedback;

assign lfsr_feedback = lfsr[2] ^ lfsr[0]; //bit[2] XOR bit[0]

always @(posedge clk or negedge reset_n) begin
    if (!reset_n)
        lfsr <= 3'b001; //001
    else
        lfsr <= {lfsr[1:0], lfsr_feedback};
end
```

Η τιμή που παράγεται από τη LFSR χρησιμοποιείται ως δείκτης (piece_type) για να επιλεγεί ποιο tetromino θα ανακτηθεί από την tetromino_rom. Ο δείκτης piece_type υπολογίζεται ως lfsr – 1, έτσι ώστε να προκύψει δείκτης από το 0 έως το 6, που αφορούν τους τύπους tetromino που είναι αποθηκευμένοι στην tetromino_rom.

```
tetromino_rom rom0 (piece_type, rotation_state, 2'b00, dx0, dy0);
tetromino_rom rom1 (piece_type, rotation_state, 2'b01, dx1, dy1);
tetromino_rom rom2 (piece_type, rotation_state, 2'b10, dx2, dy2);
tetromino_rom rom3 (piece_type, rotation_state, 2'b11, dx3, dy3);
```

Έτσι, λοιπόν, η κατάσταση αυτή εκτελείται κάθε φορά που ξεκινά το παιχνίδι ή όταν ένα κομμάτι κλειδώνει. Το νέο κομμάτι τοποθετείται στο κέντρο του οριζόντιου άξονα (base_x <= 3) και στην κορυφή του κατακόρυφου (base_y <= 0).

```
base_y <= 0; //top
base_x <= 3; //center
rotation_state <= 0; //no rotated
piece_type <= lfsr - 1;  //random tetromino
```

- **Κατάσταση Ενεργού Παιχνιδιού**

Η κατάσταση ενεργού παιχνιδιού αποτελεί τη βασική φάση λειτουργίας του συστήματος, κατά την οποία το τρέχον tetromino κινείται προς τα κάτω μέσα στο πλέγμα και ο παίκτης μπορεί να το μετακινεί μέσω του χειριστηρίου. Στη φάση αυτή πραγματοποιείται η πτώση του tetromino, η μετακίνησή του δεξιά και αριστερά, η περιστροφή του και ο έλεγχος συγκρούσεων με τα όρια του πλέγματος ή με ήδη τοποθετημένα tetrominoes.

Η **πτώση του tetromino** ελέγχεται από το σήμα fall_tick, το οποίο παράγεται από έναν μετρητή χρονισμού (fall_counter) που λειτουργεί με βάση τη συχνότητα FALL_FREQ. Ο μετρητής fall_counter αυξάνεται σε κάθε κύκλο ρολογιού του συστήματος μέχρι να φτάσει την τιμή της μεταβλητής FALL_FREQ. Όταν ο μετρητής φτάσει αυτή την τιμή, μηδενίζεται και παράγεται ένας παλμός fall_tick. Η ταχύτητα αυτή μπορεί να μεταβάλλεται δυναμικά κατά τη διάρκεια του παιχνιδιού μέσω της μεταβολής της μεταβλητής FALL_FREQ (όπως γίνεται στη διαγραφή γραμμών).

```
if (fall_counter >= FALL_FREQ) begin
            fall_counter <= 0;
            fall_tick <= 1;
```

Κάθε φορά που παράγεται ένα fall_tick, το tetromino μετακινείται μία θέση προς τα κάτω αυξάνοντας την τιμή της μεταβλητής base_y κατά 1, εφόσον δεν υπάρχει σύγκρουση με το κάτω όριο του πλέγματος ή με άλλο tetromino. Η μεταβλητή hit_something ενεργοποιείται όταν κάποιο από τα τετράγωνα του tetromino φτάσει στο κάτω όριο του πλέγματος ή όταν εντοπιστεί σύγκρουση με τα ήδη τοποθετημένα στο stack_grid.

```
if (fall_tick && !hit_something) begin
    base_y <= base_y + 1;
end
```

Κατά τη διάρκεια της κατάστασης ενεργού παιχνιδιού, ο παίκτης μπορεί να **μετακινήσει το tetromino** προς τα αριστερά ή προς τα δεξιά χρησιμοποιώντας το χειριστήριο. Η μετακίνηση πραγματοποιείται, όταν δεν υπάρχει σύγκρουση με τα όρια του πλέγματος ή με τα υπάρχοντα tetrominoes. Ο έλεγχος αυτός πραγματοποιείται μέσω των σημάτων hit_left και hit_right. Έτσι, αν δεν υπάρχει σύγκρουση προς τα αριστερά, η μεταβλητή base_x μειώνεται κατά μία μονάδα και το tetromino μετακινείται μία στήλη προς τα αριστερά. Αντίστοιχα, αν δεν υπάρχει σύγκρουση προς τα δεξιά, η μεταβλητή base_x αυξάνεται κατά μία μονάδα και μετακινείται μία στήλη προς τα δεξιά.

```
if (joy_left && !hit_left) begin
    base_x <= base_x - 1;
end
else if (joy_right && !hit_right) begin
    base_x <= base_x + 1;
end
```

Η υλοποίηση της μετακίνησης δεν πραγματοποιείται άμεσα σε κάθε κύκλο ρολογιού, αλλά ελέγχεται από έναν μηχανισμό χρονισμού που βασίζεται στις παραμέτρους DAS (Delayed Auto Shift) και ARR (Auto Repeat Rate). Το DAS καθορίζει το χρονικό διάστημα που πρέπει να περάσει αφού ο παίκτης κρατήσει πατημένο το κουμπί μετακίνησης, πριν αρχίσει η συνεχόμενη μετακίνηση του tetromino. Το ARR καθορίζει την ταχύτητα με την οποία επαναλαμβάνεται η μετακίνηση του tetromino αφού ολοκληρωθεί η καθυστέρηση DAS.

```
localparam DAS_DELAY = 7_500_000; //150ms
localparam ARR_DELAY = 2_500_000; //50ms
```

Η επιλογή των τιμών έγινε με βάση την ανάλυση των απαιτήσεων του παιχνιδιού, ώστε να διασφαλιστεί ο βέλτιστος συνδυασμός ευαισθησίας και ελέγχου κατά την οριζόντια μετακίνηση των tetromino. Έτσι, δεδομένου ότι το σύστημα λειτουργεί με ρολόι 50 MHz, το παραπάνω πλήθος κύκλων για το DAS αντιστοιχεί περίπου σε 7.500.000/50.000.000 = 150ms ενώ για το ARR σε 2.500.000/50.000.000 = 50ms.

Η υλοποίηση των χρονισμών DAS και ARR πραγματοποιείται με τη χρήση των μετρητών repeat_delay_counter και move_count. Ο μετρητής repeat_delay_counter αυξάνεται σε κάθε κύκλο ρολογιού μέχρι να φτάσει την τιμή repeat_delay. Όταν συμβεί αυτό, επιτρέπεται μια νέα μετακίνηση του tetromino και ο μετρητής μηδενίζεται.

Η πρώτη μετακίνηση πραγματοποιείται άμεσα, καθώς η μεταβλητή repeat_delay αρχικοποιείται στο μηδέν. Μετά την πρώτη μετακίνηση, ενεργοποιείται η καθυστέρηση DAS, ενώ για τις επόμενες μετακινήσεις χρησιμοποιείται η καθυστέρηση ARR.

```
 if (joy_left || joy_right) begin
    if (repeat_delay_counter == repeat_delay) begin
        repeat_delay_counter <= 0;

        if (joy_left && !hit_left) begin
            base_x <= base_x - 1;
        end
        else if (joy_right && !hit_right) begin
            base_x <= base_x + 1;
        end
        move_count <= move_count + 1;
    end
    else begin
        repeat_delay_counter <= repeat_delay_counter + 1;
    end
//after the first immediate tetromino x position move, set the DAS delay
   if (move_count == 0) begin
       repeat_delay <= DAS_DELAY;
   end
   else if (move_count > 1) begin//after the second immediate tetromino
x position move, set the ARR delay
        repeat_delay <= ARR_DELAY;
   end
end else begin //not left or right joystick input, so we reset delay
counters
      repeat_delay <= 0; //no delay for the next input
      move_count <= 0; //reset amount of tetromino moves
      repeat_delay_counter <= 0; //reset delay counter
end
```

Όταν ο παίκτης δεν μετακινεί ούτε αριστερά ούτε δεξιά το χειριστήριο, οι μετρητές repeat_delay, move_count, repeat_delay_counter μηδενίζονται. Έτσι, κάθε νέα εντολή μετακίνησης αρχίζει από την αρχή χωρίς να επηρεάζεται από προηγούμενες κινήσεις, αποφεύγοντας τυχόν ανεπιθύμητες επαναλαμβανόμενες κινήσεις.

Επιπλέον, ο παίκτης, για την **περιστροφή του tetromino**, μπορεί να χρησιμοποιήσει το κουμπί έναρξης/περιστροφής (joy_rotate). Η υλοποίηση της περιστροφής βασίζεται στη μεταβλητή rotation_state, η οποία αναπαριστά την τρέχουσα κατάσταση περιστροφής του tetromino. Κάθε tetromino μπορεί να έχει έως τέσσερις διαφορετικές καταστάσεις περιστροφής (0°, 90°, 180° και 270°). Για τον υπολογισμό της επόμενης κατάστασης περιστροφής χρησιμοποιείται η μεταβλητή rotation_next, η οποία αυξάνει την τιμή του rotation_state κατά μία μονάδα και μηδενίζεται όταν η τιμή φτάσει το 3. Με αυτόν τον τρόπο επιτυγχάνεται κυκλική εναλλαγή μεταξύ των τεσσάρων δυνατών περιστροφών.

```
wire [1:0] rotation_next = (rotation_state == 2'b11) ? 2'b00 :
rotation_state + 1; //wrap around 0-3
```

Στη συνέχεια, η γεωμετρία του tetromino για την επόμενη περιστροφή ανακτάται από το tetromino_rom και διαβάζονται οι νέες μετατοπίσεις (rdx, rdy) για κάθε ένα από τα τέσσερα τετράγωνα του tetromino.

```
wire [3:0] rdx0, rdy0;
wire [3:0] rdx1, rdy1;
wire [3:0] rdx2, rdy2;
wire [3:0] rdx3, rdy3;

tetromino_rom rom_r0 (piece_type, rotation_next, 2'b00, rdx0, rdy0);
tetromino_rom rom_r1 (piece_type, rotation_next, 2'b01, rdx1, rdy1);
tetromino_rom rom_r2 (piece_type, rotation_next, 2'b10, rdx2, rdy2);
tetromino_rom rom_r3 (piece_type, rotation_next, 2'b11, rdx3, rdy3);
```

Οι τιμές αυτές αντιπροσωπεύουν τις σχετικές μετατοπίσεις των τεσσάρων τετραγώνων του tetromino ως προς τη βασική του θέση. Με βάση αυτές τις μετατοπίσεις υπολογίζονται οι νέες θέσεις αυτών στο πλέγμα, όταν γίνει η περιστροφή.

```
wire [8:0] ridx0 = (base_y + rdy0) * GRID_COLS + (rcolx0);
wire [8:0] ridx1 = (base_y + rdy1) * GRID_COLS + (rcolx1);
wire [8:0] ridx2 = (base_y + rdy2) * GRID_COLS + (rcolx2);
wire [8:0] ridx3 = (base_y + rdy3) * GRID_COLS + (rcolx3);
```

Με βάση τις νέες θέσεις, πραγματοποιείται ο έλεγχος σύγκρουσης (rotation_blocked). Αν κάποια από αυτές βρίσκεται εκτός ορίων του πλέγματος ή αν συγκρούεται με ήδη κατειλημμένη θέση του stack_grid, η περιστροφή απορρίπτεται.

Επομένως, αν ο παίκτης πατήσει το κουμπί περιστροφής και δεν υπάρχει εμπόδιο, ενημερώνεται η μεταβλητή rotation_state, ώστε να ενεργοποιηθεί η νέα περιστροφή του tetromino.

```
joy_rotate_p <= joy_rotate; // store previous state, to rotate only once
per click
if (joy_rotate && !rotation_blocked && !joy_rotate_p) begin
    rotation_state <= (rotation_state == 2'b11) ? 2'b00 : rotation_state
+ 1;
end
```

Ο μηχανισμός σύγκρουσης αποτελεί βασικό μέρος της λογικής του παιχνιδιού, καθώς εξασφαλίζει ότι το ενεργό tetromino δεν μπορεί να κινηθεί σε μη έγκυρες θέσεις του grid. Ο έλεγχος πραγματοποιείται πριν από κάθε πιθανή μετακίνηση ή περιστροφή του tetromino.

Αναλυτικότερα, ο έλεγχος της κάθετης σύγκρουσης χρησιμοποιείται για να διαπιστωθεί αν το ενεργό tetromino μπορεί να συνεχίσει να κινείται προς τα κάτω. Για τον σκοπό αυτό, ελέγχεται αν κάποιο από τα τετράγωνα που το αποτελούν πρόκειται να συγκρουστεί είτε με το κάτω όριο του grid είτε με κάποιο ήδη τοποθετημένο τετράγωνο που είναι αποθηκευμένο στο stack_grid.

Οι μεταβλητές dy0–dy3 εκφράζουν τις κατακόρυφες μετατοπίσεις των τετραγώνων του tetromino, όπως αυτές ορίζονται στο tetromino_rom. Η τιμή (base_y + dy) δίνει τη γραμμή στην οποία βρίσκεται κάθε τετράγωνο, όπου base_y είναι η κατακόρυφη θέση του tetromino στο grid. Αν οποιοδήποτε τετράγωνο φτάσει στη γραμμή GRID_ROWS-1, τότε το tetromino έχει φτάσει στο κάτω όριο του grid. Παράλληλα, οι δείκτες idx0_down–idx3_down αντιστοιχούν στις θέσεις θα καταλάμβαναν τα τετράγωνα μετά την μετακίνηση τους μια θέση κάτω. Έτσι, ελέγχεται αν υπάρχει σύγκρουση με ήδη τοποθετημένα τετράγωνα του stack_grid.

Αν κάποια από τις παραπάνω συνθήκες ισχύει, ενεργοποιείται το σήμα hit_something, το tetromino δεν μπορεί να κινηθεί προς τα κάτω και σταθεροποιείται στο grid.

```
// Vertical collision
wire hit_something;
assign hit_something =
    //to bottom
    ((base_y + dy0) == GRID_ROWS-1) ||
    ((base_y + dy1) == GRID_ROWS-1) ||
    ((base_y + dy2) == GRID_ROWS-1) ||
    ((base_y + dy3) == GRID_ROWS-1) ||
    //to another tetromino down
    stack_grid[idx0_down] ||
    stack_grid[idx1_down] ||
    stack_grid[idx2_down] ||
    stack_grid[idx3_down];
```

```
//Compute next down position for the tetromino. Base Y position + tile +
1down * x position
wire [8:0] idx0_down = (base_y + dy0 + 1) * GRID_COLS + (colx0);
wire [8:0] idx1_down = (base_y + dy1 + 1) * GRID_COLS + (colx1);
wire [8:0] idx2_down = (base_y + dy2 + 1) * GRID_COLS + (colx2);
wire [8:0] idx3_down = (base_y + dy3 + 1) * GRID_COLS + (colx3);
```

Ο έλεγχος της οριζόντιας σύγκρουσης χρησιμοποιείται για να διαπιστωθεί αν το ενεργό tetromino μπορεί να μετακινηθεί προς τα αριστερά ή προς τα δεξιά. Αντίστοιχα με την κάθετη σύγκρουση, ελέγχεται αν η μετακίνηση θα προκαλέσει έξοδο από τα όρια του grid ή επικάλυψη με ήδη τοποθετημένα τετράγωνα που είναι αποθηκευμένα στο stack_grid.

Η θέση κάθε τετραγώνου του tetromino ως προς τον οριζόντιο άξονα εκφράζεται μέσω των μεταβλητών colx0–colx3, οι οποίες αντιστοιχούν στη στήλη στην οποία βρίσκεται κάθε τεράγωνο. Αρχικά πραγματοποιείται έλεγχος για σύγκρουση με τα πλευρικά τοιχώματα του grid. Συγκεκριμένα, αν κάποιο τετράγωνο βρίσκεται στη στήλη 0, τότε το tetromino δεν μπορεί να μετακινηθεί περαιτέρω προς τα αριστερά, ενώ αν κάποιο block βρίσκεται στη στήλη GRID_COLS-1, τότε δεν μπορεί να μετακινηθεί προς τα δεξιά.

Παράλληλα, ελέγχεται αν υπάρχει ήδη τοποθετημένο τετράγωνο ακριβώς δίπλα στη θέση προς την οποία επιχειρείται η μετακίνηση. Για τον σκοπό αυτό, γίνεται υπολογισμός της θέσης του κελιού που βρίσκεται μία στήλη αριστερά ή δεξιά από κάθε τετράγωνο.

Αν κάποιο από τα κελιά αυτά είναι ήδη κατειλημμένο στο stack_grid, τότε θεωρείται ότι υπάρχει σύγκρουση και ενεργοποιείται το αντίστοιχο σήμα (hit_left ή hit_right). Στην περίπτωση αυτή, η μετακίνηση προς την αντίστοιχη κατεύθυνση δεν εκτελείται και η θέση του tetromino παραμένει η ίδια.

```
// Horizontal collision
wire hit_left =
    //wall
    ((colx0 == 0) ||
    (colx1 == 0) ||
    (colx2 == 0) ||
    (colx3 == 0)) ||
    //stack
    ((colx0 > 0) && stack_grid[(base_y+dy0)*GRID_COLS + (colx0-1)]) ||
    ((colx1 > 0) && stack_grid[(base_y+dy1)*GRID_COLS + (colx1-1)]) ||
    ((colx2 > 0) && stack_grid[(base_y+dy2)*GRID_COLS + (colx2-1)]) ||
    ((colx3 > 0) && stack_grid[(base_y+dy3)*GRID_COLS + (colx3-1)]);

wire hit_right =
    //wall
    ((colx0 == GRID_COLS-1) ||
    (colx1 == GRID_COLS-1) ||
    (colx2 == GRID_COLS-1) ||
    (colx3 == GRID_COLS-1)) ||
    //stack
    ((colx0 < GRID_COLS-1) && stack_grid[(base_y+dy0)*GRID_COLS +
(colx0+1)]) ||
    ((colx1 < GRID_COLS-1) && stack_grid[(base_y+dy1)*GRID_COLS +
(colx1+1)]) ||
    ((colx2 < GRID_COLS-1) && stack_grid[(base_y+dy2)*GRID_COLS +
(colx2+1)]) ||
    ((colx3 < GRID_COLS-1) && stack_grid[(base_y+dy3)*GRID_COLS +
(colx3+1)]);
```

Ο έλεγχος σύγκρουσης γίνεται και κατά την περιστροφή του tetromino, όπου ελέγχεται αν μπορεί να γίνει χωρίς να συγκρουστεί με τα όρια του grid ή με ήδη τοποθετημένα τετράγωνα του stack_grid. Πραγματοποιείται μέσω του σήματος rotation_blocked και περιλαμβάνει τον έλεγχο εξόδου από τα οριζόντια και κατακόρυφα όρια του grid και τον έλεγχο σύγκρουσης με ήδη τοποθετημένα τετράγωνα στο stack_grid.

```
// Rotation collision (check next rotated position)
wire rotation_blocked;
assign rotation_blocked =
    // horizontal out-of-bounds
    (rcolx0 >= GRID_COLS) ||
       (rcolx1 >= GRID_COLS) ||
       (rcolx2 >= GRID_COLS) ||
       (rcolx3 >= GRID_COLS) ||
    // vertical out-of-bounds
    (base_y + rdy0 >= GRID_ROWS) ||
       (base_y + rdy1 >= GRID_ROWS) ||
       (base_y + rdy2 >= GRID_ROWS) ||
       (base_y + rdy3 >= GRID_ROWS) ||
    // stack
       stack_grid[ridx0] ||
       stack_grid[ridx1] ||
       stack_grid[ridx2] ||
       stack_grid[ridx3];
```

Αρχικά, υπολογίζονται οι νέες συντεταγμένες των τεσσάρων τετραγώνων του tetromino για τον επόμενο προσανατολισμό. Οι οριζόντιες θέσεις των τετραγώνων μετά την περιστροφή εκφράζονται από τις μεταβλητές rcolx0–rcolx3, ενώ οι κατακόρυφες μετατοπίσεις από τις rdy0–rdy3. Οι μεταβλητές αυτές αντιστοιχούν στις πιθανές νέες θέσεις των τετραγώνων, αν η περιστροφή πραγματοποιηθεί. Στη συνέχεια, ελέγχεται αν κάποια από τις νέες οριζόντιες θέσεις των τετραγώνων μετά την περιστροφή (rcolx0–rcolx3) υπερβαίνει το όριο των στηλών (GRID_COLS). Στη συνέχεια ελέγχεται αν η νέα κατακόρυφη θέση των blocks, που υπολογίζεται από τη σχέση (base_y + rdy), υπερβαίνει το κάτω όριο του grid (GRID_ROWS). Τέλος, χρησιμοποιούνται οι δείκτες ridx0–ridx3, οι οποίοι αντιστοιχούν στις θέσεις που θα καταλάμβαναν τα τετράγωνα μετά την περιστροφή και ελέγχεται αν υπάρχει σύγκρουση με ήδη τοποθετημένα τετράγωνα του stack_grid.

Αν ισχύει οποιαδήποτε από τις παραπάνω συνθήκες, ενεργοποιείται το σήμα rotation_blocked και η περιστροφή δεν πραγματοποιείται.

- **Κατάσταση κλειδώματος tetromino**

Όταν το ενεργό tetromino δεν μπορεί να κινηθεί άλλο προς τα κάτω, είτε επειδή έχει φτάσει στο κάτω όριο του πλέγματος είτε επειδή συγκρούεται με ήδη τοποθετημένα κομμάτια, το παιχνίδι μεταβαίνει στην κατάσταση κλειδώματος. Στην κατάσταση αυτή, το κινούμενο tetromino δεν θεωρείται ενεργό και τα τετράγωνά του ενσωματώνονται στο πλέγμα των σταθερών κομματιών (stack_grid). Η ανίχνευση της σύγκρουσης πραγματοποιείται μέσω του σήματος hit_something, το οποίο ενεργοποιείται όταν κάποιο από τα τετράγωνα του tetromino βρίσκεται στο τελευταίο επίπεδο του πλέγματος ή όταν στην επόμενη θέση πτώσης υπάρχει ήδη σταθερό τετράγωνο.

```
wire hit_something;
assign hit_something =
    ((base_y + dy0) == GRID_ROWS-1) || //collision to bottom
    ((base_y + dy1) == GRID_ROWS-1) ||
    ((base_y + dy2) == GRID_ROWS-1) ||
    ((base_y + dy3) == GRID_ROWS-1) ||
    stack_grid[idx0_down] || //collision to another tetromino down
    stack_grid[idx1_down] ||
    stack_grid[idx2_down] ||
    stack_grid[idx3_down];
```

Όταν ξεκινά η πτώση του κομματιού (fall_tick) και ταυτόχρονα ανιχνευθεί σύγκρουση, η λογική του παιχνιδιού εκτελεί τη διαδικασία κλειδώματος. Αρχικά, υπολογίζονται οι θέσεις των τεσσάρων τετραγώνων του tetromino μέσα στο πλέγμα και στη συνέχεια οι θέσεις αυτές αντιγράφονται στο stack_grid. Έτσι, το κινούμενο κομμάτι μετατρέπεται σε σταθερό τμήμα του πλέγματος.

```
if (fall_tick && hit_something) begin
// Compute of moving piece
   idx0 = (base_y + dy0) * GRID_COLS + (colx0);
   idx1 = (base_y + dy1) * GRID_COLS + (colx1);
   idx2 = (base_y + dy2) * GRID_COLS + (colx2);
   idx3 = (base_y + dy3) * GRID_COLS + (colx3);

// Copy into stack grid
   stack_grid[idx0] <= 1'b1;
   stack_grid[idx1] <= 1'b1;
   stack_grid[idx2] <= 1'b1;
   stack_grid[idx3] <= 1'b1;
```

Παράλληλα, ενεργοποιείται ένα σύντομο ηχητικό σήμα που υποδηλώνει την ολοκλήρωση της τοποθέτησης του κομματιού. Στη συνέχεια, αρχικοποιούνται οι παράμετροι για τη δημιουργία του επόμενου κομματιού. Η θέση επαναφέρεται στο επάνω μέρος του πλέγματος, η περιστροφή μηδενίζεται και επιλέγεται ένας νέος τύπος tetromino μέσω της γεννήτριας LFSR.

```
sound <= 3'b010;

base_y <= 0; //top
base_x <= 3; //center
rotation_state <= 0; //no rotated
piece_type <= lfsr - 1;  //random tetromino
```

Στον επόμενο κύκλο ρολογιού πραγματοποιείται έλεγχος για πλήρεις γραμμές μέσω του σήματος any_row_full. Αν κάποια γραμμή είναι πλήρης, ενεργοποιείται η διαδικασία διαγραφής γραμμών, αλλιώς το παιχνίδι συνεχίζει κανονικά με την πτώση του επόμενου tetromino.

- **Κατάσταση ελέγχου και διαγραφής γραμμών**

Μετά το κλείδωμα ενός tetromino στο πλέγμα, το σύστημα ελέγχει αν έχει σχηματιστεί μια ή περισσότερες πλήρεις γραμμές. Ο έλεγχος πραγματοποιείται πάνω στο πλέγμα των σταθερών κομματιών και για κάθε γραμμή του πλέγματος (wire row0_full, wire row1_full…) εφαρμόζεται η λογική πράξη AND σε όλα τα bits της γραμμής. Αν το αποτέλεσμα είναι 1, τότε σημαίνει ότι η συγκεκριμένη γραμμή είναι πλήρης.

```
wire row0_full  = &stack_grid[  9:  0];
wire row1_full  = &stack_grid[ 19: 10];
wire row2_full  = &stack_grid[ 29: 20];
// ομοίως για τις υπόλοιπες γραμμές…
wire row19_full = &stack_grid[199:190];
```

Αν υπάρχει τουλάχιστον μία πλήρης γραμμή στο πλέγμα, όλα τα σήματα των γραμμών συνδυάζονται σε ένα ενιαίο σήμα (wire any_row_full). Το σήμα αυτό χρησιμοποιείται από τη λογική του παιχνιδιού για να καθορίσει αν πρέπει να ενεργοποιηθεί η διαδικασία διαγραφής γραμμών.

```
wire any_row_full =
      row0_full  | row1_full  | row2_full  | row3_full
    | row4_full  | row5_full  | row6_full  | row7_full
    | row8_full  | row9_full  | row10_full | row11_full
    | row12_full | row13_full | row14_full | row15_full
    | row16_full | row17_full | row18_full | row19_full;
```

Όταν το σήμα any_row_full είναι ενεργό, τότε η πλήρης γραμμή αφαιρείται και όλες οι γραμμές που βρίσκονται πάνω από αυτή μετακινούνται μία θέση προς τα κάτω. Η λειτουργία αυτή υλοποιείται με μετατόπιση των αντίστοιχων bits στο stack_grid. Τέλος, η εντολή grid_flat <= stack_grid | active_piece συνδυάζει το πλέγμα των ήδη τοποθετημένων blocks με το τρέχον κινούμενο tetromino, ώστε να παραχθεί η τελική εικόνα που εμφανίζεται στην οθόνη.

```
if (row19_full) begin
    stack_grid[10 +: 190] <= stack_grid[0 +: 190];
end
else if (row18_full) begin
    stack_grid[10 +: 180] <= stack_grid[0 +: 180];
end
// ομοίως για τις υπόλοιπες γραμμές…
else if (row0_full) begin
   stack_grid[0 +: 10] <= 10'b0;
end

grid_flat <= stack_grid | active_piece;
```

Παράλληλα, κάθε φορά που διαγράφεται μία γραμμή, αυξάνεται η ταχύτητα του παιχνιδιού. Αυτό επιτυγχάνεται με τη μείωση της μεταβλητής FALL_FREQ, η οποία καθορίζει τον αριθμό κύκλων ρολογιού που απαιτούνται μέχρι να μετακινηθεί το tetromino μία θέση προς τα κάτω. Συγκεκριμένα, η τιμή της μειώνεται κατά FALL_INCREASE = 1.000.000 κύκλους σε κάθε διαγραφή γραμμής. Δεδομένου ότι το σύστημα λειτουργεί με ρολόι 50 MHz, η μεταβολή αυτή αντιστοιχεί σε μείωση του χρόνου πτώσης κατά περίπου 0.02 sec (Πίνακας 7), αυξάνοντας σταδιακά τη δυσκολία του παιχνιδιού. Κατά τη διάρκεια της διαδικασίας διαγραφής γραμμών ενεργοποιείται επίσης ένα ηχητικό σήμα που ενημερώνει ολοκληρώθηκε η αφαίρεση μιας γραμμής.

```
if (any_row_full) begin
    FALL_FREQ <= FALL_FREQ - FALL_INCREASE;

    sound <= 3'b001;
```

Σύνολο διαγραφόμενων γραμμών	Συχνότητα πτώσης (κύκλοι/sec)	Χρόνος πτώσης (sec)
0	50.000.000	1
1	49.000.000	0.98
5	45.000.000	0.90
10	40.000.000	0.80

Πίνακας 7 Μεταβολή της ταχύτητας πτώσης των tetromino

Να σημειωθεί ότι αν υπάρχουν έως και τέσσερις πλήρεις γραμμές ταυτόχρονα, το σύστημα τις διαγράφει μία-μία ανά κύκλο ρολογιού. Λόγω της συχνότητας των 50MHz, η διαδικασία αυτή ολοκληρώνεται σε nanoseconds, οπότε ο παίκτης αντιλαμβάνεται ότι όλες οι γραμμές σβήστηκαν ταυτόχρονα.

- **Κατάσταση τερματισμού**

Η κατάσταση τερματισμού του παιχνιδιού ενεργοποιείται όταν το πλέγμα γεμίσει έως και τη γραμμή που βρίσκεται στη κορυφή του. Στην περίπτωση αυτή, δεν υπάρχει πλέον διαθέσιμος χώρος για την εμφάνιση νέων tetromino και συνεπώς το παιχνίδι τερματίζεται. Ο έλεγχος αυτός πραγματοποιείται εξετάζοντας αν κάποιο από τα bits της πρώτης γραμμής του stack_grid έχει τιμή 1. Αν πάρει την τιμή 1, αυτό σημαίνει ότι οποιοδήποτε από τα bits της γραμμής είναι ενεργό και άρα κατειλημμένο. Στη συνέχεια, ενεργοποιείται το σήμα game_over, παράγεται ηχητικό σήμα τερματισμού και το σύστημα επιστρέφει στην αρχική οθόνη έναρξης.

```
if (!any_row_full) begin
//game over
    if (|stack_grid[9:0]) begin
        sound <= 3'b100;
        show_start <= 1; // show start screen
        game_over <= 1; // flag game over
    end
```

Όταν ενεργοποιηθεί το σήμα game_over, εμφανίζεται στην οθόνη το μήνυμα GAME OVER, το οποίο σχηματίζεται ενεργοποιώντας συγκεκριμένα bits του πίνακα grid_flat.

```
if (game_over) begin
//display "GAME OVER"
grid_flat[  9:  0] <= 10'b1111001111; // row 0 (G)
grid_flat[ 19: 10] <= 10'b1001000001; // row 1
grid_flat[ 29: 20] <= 10'b1001001101; // row 2
// continue with A, M, E
```

Το παιχνίδι είναι σε αναμονή μέχρι ο παίκτης να πατήσει ξανά το κουμπί START (joy_rotate) για επανεκκίνηση.

4.2.8 Top level– myTetris

Το module myTetris_top αποτελεί το ανώτερο επίπεδο (top-level) της σχεδίασης και συνδέει τη λογική του παιχνιδιού με τους φυσικούς πόρους της πλακέτας. Στο επίπεδο αυτό πραγματοποιείται η διαχείριση των σημάτων εισόδου/εξόδου, η αρχικοποίηση του συστήματος και η διασύνδεση των επιμέρους modules που υλοποιούν τη λειτουργία του παιχνιδιού.

Το module περιλαμβάνει τις βασικές διεπαφές της πλακέτας, όπως το ρολόι συστήματος (CLOCK0_50), τα κουμπιά (KEY), τους διακόπτες (SW), τα LEDs (LEDR), τις segment ενδείξεις (HEX), τη μνήμη SDRAM, τη διεπαφή HDMI και τα GPIO pins που χρησιμοποιούνται για το joystick και τον ήχο.

Αρχικά, απενεργοποιούνται οι πόροι της πλακέτας που δεν χρησιμοποιούνται από την εφαρμογή, ώστε να αποφευχθούν ανεπιθύμητες λειτουργίες, όπως τα LEDs, το 7segment display και η SDRAM.

```
//----LED off device
assign LEDR[8:0] = 9'b111111111;

//----HEX off device
assign HEX0 =7'h7f;
assign HEX1 =7'h7f;
assign HEX2 =7'h7f;
assign HEX3 =7'h7f;
assign HEX4 =7'h7f;
assign HEX5 =7'h7f;

//----SDRAM off
assign DRAM_DQ   =32'hzzzz_zzzzz;
assign DRAM_CS_n =1'b1;
assign DRAM_WE_n =1'b1;
assign DRAM_CAS_n=1'b1;
assign DRAM_RAS_n=1'b1;
assign DRAM_DQM  =4'b1111;
```

Το ρολόι του συστήματος είναι στα 50MHz και χρησιμοποιείται από τα περισσότερα modules του συστήματος, όπως το input_controller, το audio_controller και το game_logic.

```
wire  SYSTEM_50MHZ;
assign SYSTEM_50MHZ = CLOCK0_50;
```

Η αρχικοποίηση του συστήματος πραγματοποιείται μέσω του σήματος reset_n, το οποίο ενεργοποιείται κατά την εκκίνηση του FPGA.

```
wire reset_n; //fpga board startup repair
//--RESET
assign reset_n = ~ninit_done;
//---INIT RESET
wire ninit_done;
      ResetRelease ResetRelease_inst (
            .ninit_done (ninit_done)
);
```

Στη συνέχεια, γίνεται η διασύνδεση των υποσυστημάτων:

HDMI_controller

Τα σήματα V_locked, pll_12M288, vpg_r, vpg_g, vpg_b, HDMI_READY και HDMI_I2C_SCLK χρησιμοποιούνται για τη λειτουργία της διεπαφής βίντεο και την επικοινωνία με την HDMI έξοδο. Τα σήματα vpg_r, vpg_g και vpg_b μεταφέρουν τα δεδομένα RGB της εικόνας που παράγονται από το grid_generator και οδηγούνται στη διεπαφή HDMI για την απεικόνιση του παιχνιδιού στην οθόνη. Παράλληλα, το HDMI_READY δηλώνει ότι η διαδικασία αρχικοποίησης της HDMI συσκευής μέσω του πρωτοκόλλου I2C έχει ολοκληρωθεί επιτυχώς, ενώ το HDMI_I2C_SCLK αποτελεί το ρολόι επικοινωνίας του διαύλου I2C που χρησιμοποιείται για τη ρύθμιση της HDMI διεπαφής.

```
//Terrasic HDMI interface modules
wire  V_locked;
wire  pll_12M288;
wire [7:0] vpg_r;
wire [7:0] vpg_g;
wire [7:0] vpg_b;
wire HDMI_READY ;
wire HDMI_I2C_SCLK ;
```

Η έξοδος εικόνας υλοποιείται μέσω της διεπαφής HDMI. Τα σήματα RGB συνδυάζονται στο bus HDMI_TX_D.

```
assign HDMI_TX_D[23:16] =     vpg_r;
assign HDMI_TX_D[15:8]  =     vpg_g;
assign HDMI_TX_D[7:0]   =     vpg_b;
```

Το module I2C_HDMI_Config χρησιμοποιείται για την αρχικοποίηση και τη ρύθμιση της HDMI διεπαφής μέσω του πρωτοκόλλου επικοινωνίας I2C. Λαμβάνει ως είσοδο το ρολόι συστήματος (SYSTEM_50MHZ) και το σήμα επαναφοράς (reset_n) και επικοινωνεί με τον HDMI transmitter μέσω των γραμμών FPGA_I2C_SCL και FPGA_I2C_SDA. Το σήμα HDMI_TX_INT χρησιμοποιείται για την παρακολούθηση της κατάστασης της HDMI συσκευής, ενώ το σήμα HDMI_READY υποδεικνύει ότι η διαδικασία αρχικοποίησης έχει ολοκληρωθεί επιτυχώς.

```
//--  HDMI I2C    SETTING
I2C_HDMI_Config u_I2C_HDMI_Config (
    .iCLK       (SYSTEM_50MHZ  ),
    .iRST_N     (reset_n),
    .I2C_SCLK   (FPGA_I2C_SCL  ),
```

```
    .I2C_SDAT    (FPGA_I2C_SDA  ),
    .HDMI_TX_INT(HDMI_TX_INT    ),
    .READY       (HDMI_READY    )
);
```

PLL

Το VEDIO_PLL χρησιμοποιείται για την παραγωγή του ρολογιού που απαιτείται για τη λειτουργία της εξόδου βίντεο. Παράγει ένα νέο ρολόι υψηλότερης συχνότητας (vpg_pclk), το οποίο χρησιμοποιείται από το grid_generator και τη διεπαφή HDMI για τον συγχρονισμό της μετάδοσης των δεδομένων βίντεο.

```
VEDIO_PLL u_VEDIO_PLL(
    .refclk        (SYSTEM_50MHZ  ),
    .outclk_0      (vpg_pclk      ),//148.5MHZ
    .locked        (V_locked      ),
    .rst           (~reset_n      )
);
```

Grid_generator

Η απεικόνιση του παιχνιδιού στην οθόνη πραγματοποιείται από το module grid_generator. Μετατρέπει το grid του παιχνιδιού σε σήματα RGB εικόνας και παράγει τα απαραίτητα σήματα συγχρονισμού για την έξοδο HDMI.

```
grid_generator    u_grid_generator (
      .vpg_pclk     (vpg_pclk),//vedio clock input
      .reset_n      (reset_n),
      .grid_flat    (grid_flat_wire),
      .vpg_de       (HDMI_TX_DE ),
      .vpg_hs       (HDMI_TX_HS ),
      .vpg_vs       (HDMI_TX_VS ),
      .vpg_pclk_out(HDMI_TX_CLK),
      .vpg_r        (vpg_r),
      .vpg_g        (vpg_g),
      .vpg_b        (vpg_b)
);
```

Input_controller

Η ανάγνωση των σημάτων από το joystick πραγματοποιείται από το module input_controller, επεξεργάζεται τα σήματα από τα GPIO pins και παράγει τα λογικά σήματα ελέγχου του παιχνιδιού:

```
wire fire, left, right, down, restart; //connect with game logic

input_controller u_input_controller (
    .clk       (SYSTEM_50MHZ),
       //reset on board startup and user input
```

```
    .reset_n   (reset_n & !restart),
    .gpio_d    (GPIO_D),
    .fire      (fire),
    .left      (left),
    .right     (right),
    .down      (down),
    .restart   (restart)
);
```

Audio_controller

Η παραγωγή ήχου υλοποιείται μέσω του module audio_controller. Το σήμα sound παράγεται από το game_logic και καθορίζει το είδος του ηχητικού εφέ που θα αναπαραχθεί. Η έξοδος του ήχου οδηγείται στο pin GPIO_D[5], το οποίο είναι συνδεδεμένο με το ηχείο.

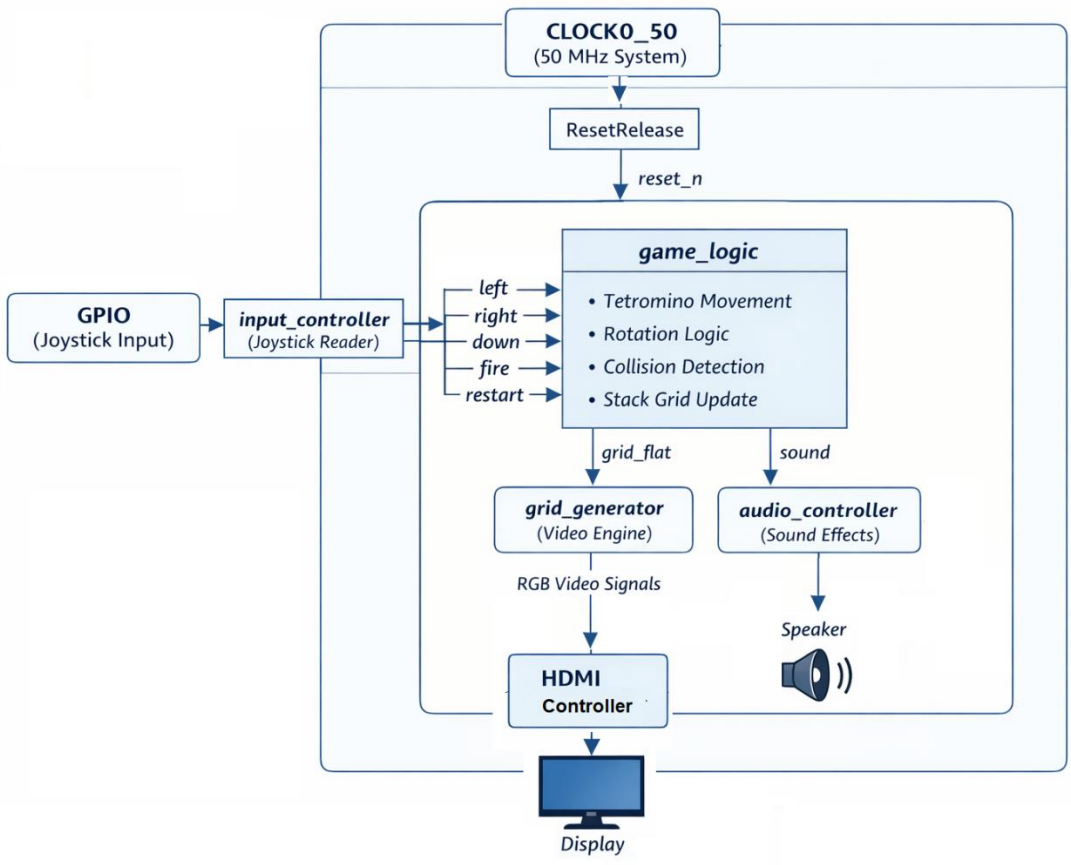
```
wire [2:0] sound; //connect with game logic

audio_controller u_audio_controller (
    .clk       (SYSTEM_50MHZ),
//reset on board startup and user input
    .reset_n   (reset_n & !restart),
    .sound (sound),
    .speaker (GPIO_D[5])
);
```

Game_logic

Η βασική λειτουργία του παιχνιδιού υλοποιείται στο module game_logic. Η κατάσταση του grid εξάγεται μέσω του σήματος grid_flat_wire, το οποίο μεταφέρει τα δεδομένα του παιχνιδιού στο grid_generator.

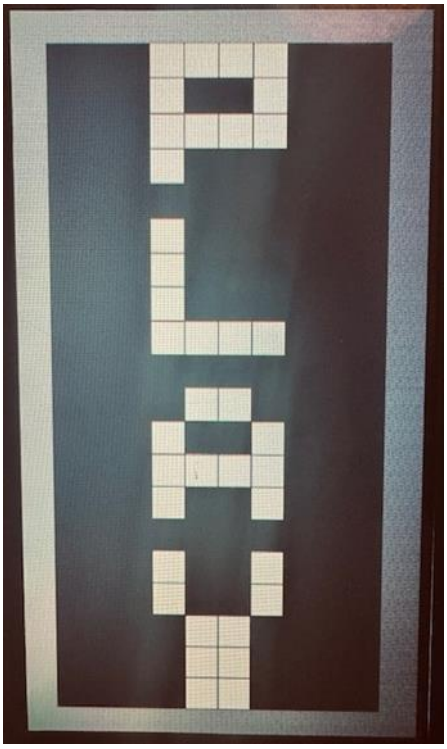
```
wire [199:0] grid_flat_wire;//connect with grid generator

game_logic u_game_logic (
    .clk        (SYSTEM_50MHZ),
       //reset on board startup and user input
    .reset_n    (reset_n & !restart),
    .joy_rotate (fire),
    .joy_left (left),
    .joy_right (right),
    .joy_down (down),
    .sound (sound),
    .grid_flat (grid_flat_wire)
);
```

Συνοπτικά, το top-level module λειτουργεί ως το κεντρικό σημείο διασύνδεσης όλων των υποσυστημάτων της εφαρμογής (Εικόνα 40). Αρχικά, το input_controller λαμβάνει τις εντολές του χρήστη από το joystick και τις μετατρέπει σε σήματα ελέγχου. Τα σήματα αυτά επεξεργάζονται από το game_logic, το οποίο υλοποιεί τη βασική λειτουργία του παιχνιδιού και ενημερώνει την κατάσταση του πλέγματος. Στη συνέχεια, το grid_generator μετατρέπει την κατάσταση του grid σε σήματα εικόνας RGB και τα αντίστοιχα σήματα συγχρονισμού, τα οποία οδηγούνται στη διεπαφή HDMI για την απεικόνιση του παιχνιδιού στην οθόνη. Παράλληλα, το audio_controller λαμβάνει σήματα από τη λογική του παιχνιδιού και παράγει τα αντίστοιχα ηχητικά εφέ. Με τον τρόπο αυτό, επιτυγχάνεται η ολοκληρωμένη λειτουργία του συστήματος, συνδυάζοντας είσοδο χρήστη, λογική παιχνιδιού, απεικόνιση και ήχο.

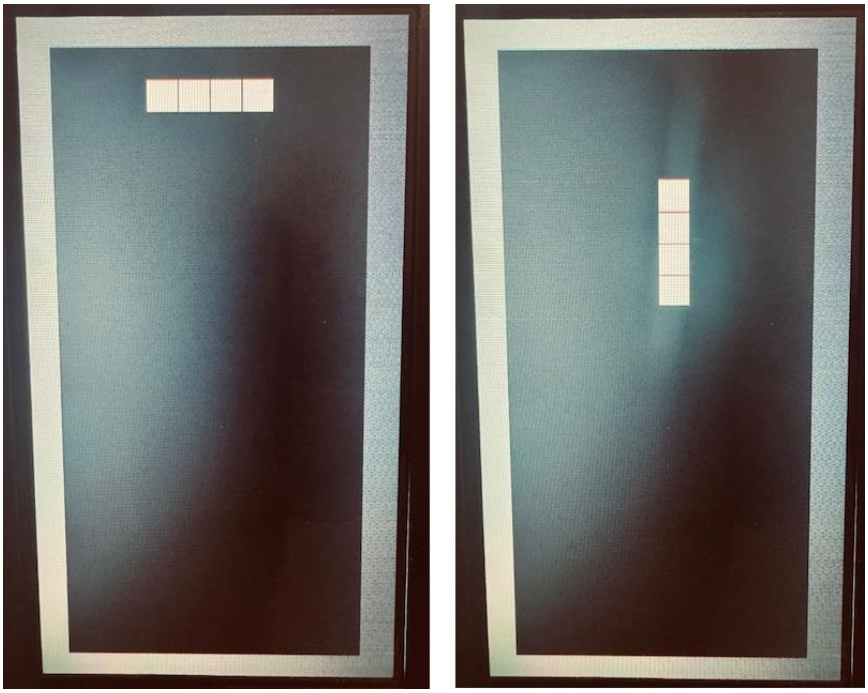


Εικόνα 40 Top level- myTetris

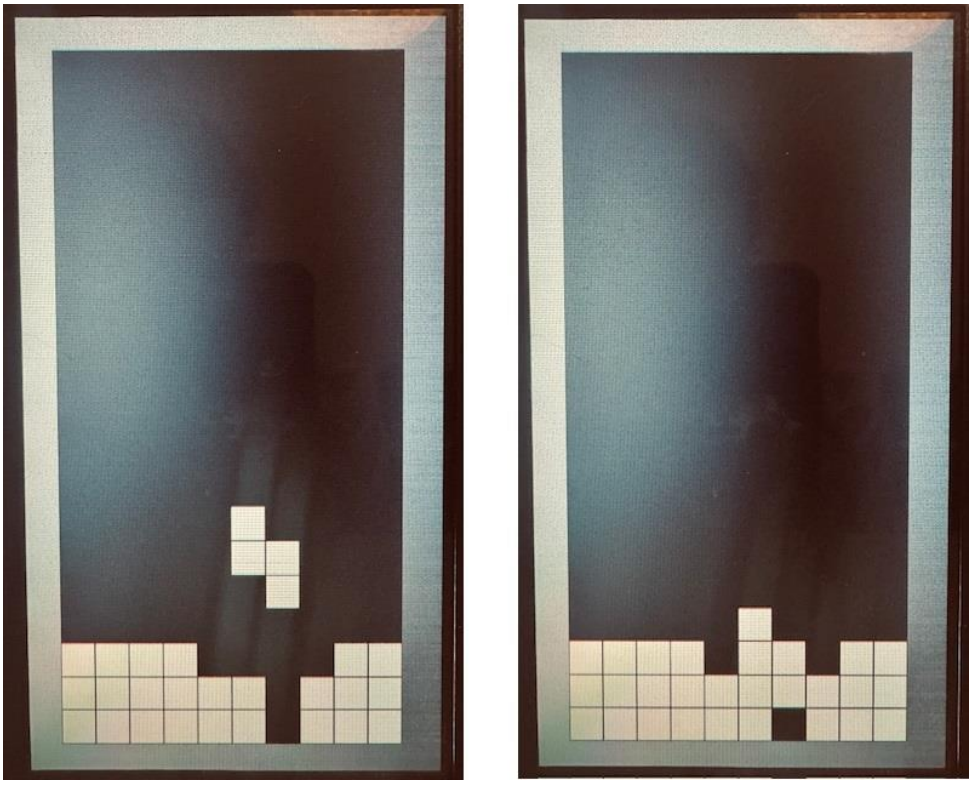
Η υλοποίηση του παιχνιδιού ολοκληρώνεται με επιτυχία και η λειτουργία του συστήματος επαληθεύτηκε στην πλακέτα DE23-Lite. Κατά τη διαδικασία δοκιμών διαπιστώθηκε ότι όλα τα επιμέρους υποσυστήματα συνεργάζονται επιτυχώς, επιτρέποντας την σωστή εκτέλεση του παιχνιδιού και αλληλεπίδραση με τον χρήστη. Στη συνέχεια, παρατίθενται ενδεικτικά στιγμιότυπα από τη λειτουργία του συστήματος, τα οποία απεικονίζουν την εκτέλεση του παιχνιδιού στην οθόνη.



Εικόνα 41 Οθόνη έναρξης του παιχνιδιού

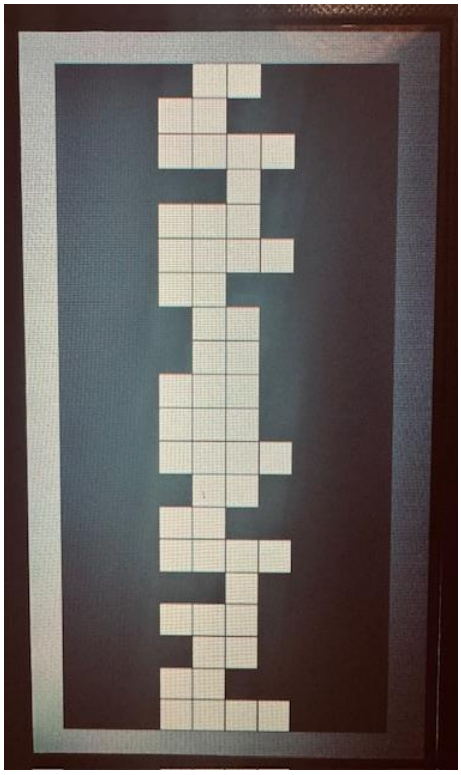


Εικόνα 42 Εμφάνιση tetromino I και περιστροφή του

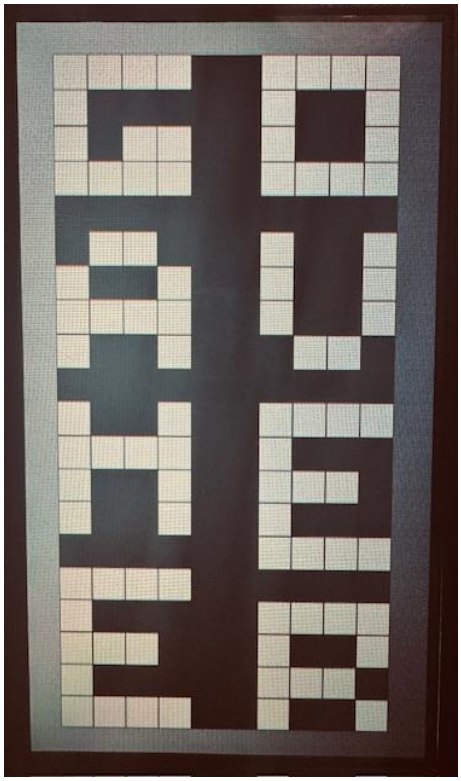


Εικόνα 43 Πτώση tetromino S και κλείδωμα

Εικόνα 44 Διαγραφή πλήρους γραμμής και εμφάνιση νέου tetromino L



Εικόνα 45 Πληρότητα πλέγματος



Εικόνα 46 Οθόνη τερματισμού του παιχνιδιού

Στην Εικόνα 41 φαίνεται η οθόνη έναρξης (PLAY), όπου το παιχνίδι βρίσκεται σε κατάσταση αναμονής, μέχρι ο παίκτης να πατήσει το κουμπί έναρξης/περιστροφής, για να ξεκινήσει το παιχνίδι. Στην Εικόνα 42 απεικονίζεται η εμφάνιση του πρώτου tetromino, το οποίο αρχίζει να κινείται σταδιακά προς τα κάτω στο πλέγμα. Μέσω του ίδιου κουμπιού, ο παίκτης έχει τη δυνατότητα να περιστρέψει το tetromino, επιλέγοντας τον επιθυμητό προσανατολισμό του κατά την πτώση. Στη συνέχεια, στην Εικόνα 43 απεικονίζεται η πτώση και το κλείδωμα ενός tetromino, όταν αυτό φτάσει πάνω σε ήδη τοποθετημένα κομμάτια. Αν συμπληρωθεί μια γραμμή, αυτή διαγράφεται, τα παραπάνω στοιχεία μετακινούνται προς τα κάτω και εμφανίζεται το επόμενο tetromino, κάτι που παρουσιάζεται στην Εικόνα 44. Τέλος, στην Εικόνα 45 απεικονίζεται η κατάσταση πλήρωσης του πλέγματος, κατά την οποία δεν υπάρχει πλέον διαθέσιμος χώρος για την εισαγωγή νέων tetromino. Η συνθήκη αυτή οδηγεί στον τερματισμό του παιχνιδιού και στη μετάβαση στην αντίστοιχη οθόνη λήξης (GAME OVER), ολοκληρώνοντας τον κύκλο λειτουργίας του συστήματος (Εικόνα 46).

4.3 Συμπεράσματα

Στο παρόν κεφάλαιο, παρουσιάστηκε αναλυτικά η σχεδίαση και η ανάπτυξη του συστήματος, ξεκινώντας από τον καθορισμό της αρχιτεκτονικής και καταλήγοντας στην υλοποίηση των επιμέρους υποσυστημάτων σε γλώσσα περιγραφής υλικού. Η ανάλυση περιέλαβε τη λογική οργάνωση του συστήματος αλλά και τη φυσική συνδεσμολογία των περιφερειακών, συμβάλλοντας στην ολοκληρωμένη κατανόηση της λειτουργίας του συστήματος. Η διάσπαση του συστήματος σε διακριτές μονάδες, όπως το υποσύστημα απεικόνισης, το υποσύστημα εισόδου, το υποσύστημα ελέγχου του παιχνιδιού και το υποσύστημα ήχου, επέτρεψε την οργάνωση της σχεδίασης και τη μείωση της πολυπλοκότητας.

Η υλοποίηση σε γλώσσα περιγραφής υλικού επιβεβαίωσε την ορθότητα των σχεδιαστικών επιλογών, καθώς το σύστημα ανταποκρίνεται επιτυχώς στις λειτουργικές απαιτήσεις και παρουσιάζει σταθερή χρονική συμπεριφορά. Ως αποτέλεσμα, επιτυγχάνεται η ομαλή συνεργασία των υποσυστημάτων και η αξιόπιστη απεικόνιση της κατάστασης του παιχνιδιού σε πραγματικό χρόνο. Στο επόμενο κεφάλαιο, ακολουθεί η αξιολόγηση του συστήματος, όπου εξετάζονται η απόδοσή του, η ορθότητα λειτουργίας και οι πιθανές του βελτιώσεις.

5. Αξιολόγηση συστήματος

Η αξιολόγηση αποτελεί σημαντικό στάδιο στην ανάπτυξη κάθε ψηφιακού συστήματος, καθώς επιτρέπει την αντικειμενική επαλήθευση της ορθής λειτουργίας, της απόδοσης και τη σύγκρισή του με τους αρχικούς στόχους που τέθηκαν.

Σκοπός της αξιολόγησης είναι η διερεύνηση της λειτουργικής ορθότητας, της απόδοσης και της αποτελεσματικότητας του υλοποιημένου συστήματος σε πραγματικές συνθήκες λειτουργίας. Πιο συγκεκριμένα, η αξιολόγηση αποσκοπεί αρχικά στην επιβεβαίωση της λειτουργίας του συστήματος, εξετάζοντας κατά πόσο οι μηχανισμοί του παιχνιδιού υλοποιούνται σύμφωνα με τους κανόνες του Tetris και λειτουργούν χωρίς σφάλματα. Παράλληλα, δίνεται ιδιαίτερη έμφαση στην αξιολόγηση της χρονικής απόκρισης και της συμπεριφοράς του συστήματος σε πραγματικό χρόνο. Είναι κρίσιμο να διασφαλιστεί ότι το σύστημα ανταποκρίνεται άμεσα στις εντολές του χρήστη μέσω του arcade χειριστηρίου και ότι η ενημέρωση της εικόνας πραγματοποιείται ομαλά και χωρίς αισθητές καθυστερήσεις, καθώς η απόκριση αυτή επηρεάζει άμεσα την εμπειρία του παίκτη.

Επιπλέον, σημαντικός στόχος της αξιολόγησης είναι η εξέταση της ποιότητας της παραγόμενης εικόνας μέσω της διεπαφής HDMI, συμπεριλαμβανομένης της σταθερότητας του σήματος, της σωστής απεικόνισης των γραφικών στοιχείων του παιχνιδιού και της απουσίας οπτικών σφαλμάτων, όπως τρεμόπαιγμα ή παραμορφώσεις. Ταυτόχρονα, εξετάζεται η αποδοτικότητα στη χρήση των διαθέσιμων πόρων, μέσω της ανάλυσης της κατανάλωσης λογικών στοιχείων, καταχωρητών και μνήμης, προκειμένου να εκτιμηθεί ο βαθμός βελτιστοποίησης της σχεδίασης.

Τέλος, η αξιολόγηση στοχεύει στον εντοπισμό πιθανών περιορισμών και αδυναμιών του συστήματος, καθώς και στην ανάδειξη πιθανών βελτιώσεων. Μέσω αυτής της συστηματικής διαδικασίας δεν επιδιώκεται μόνο η αποτίμηση της παρούσας υλοποίησης, αλλά και η διαμόρφωση κατευθύνσεων για μελλοντική εξέλιξη και επέκταση του συστήματος.

Για την αξιολόγηση χρησιμοποιήθηκαν τόσο ποιοτικές όσο και ποσοτικές μέθοδοι. Στις ποιοτικές μεθόδους περιλαμβάνονται η παρατήρηση της συμπεριφοράς του συστήματος σε διάφορα σενάρια παιχνιδιού, ενώ στις ποσοτικές μεθόδους περιλαμβάνονται μετρήσεις όπως η ανανέωση εικόνας, η καθυστέρηση απόκρισης, η κατανάλωση λογικών πόρων και η ακρίβεια των σημάτων εξόδου.

Μέσα από την παρούσα αξιολόγηση, επιδιώκεται να αποδειχθεί κατά πόσο το υλοποιημένο σύστημα πληροί τους τεχνικούς στόχους που τέθηκαν στην αρχή, να αναδειχθούν τα δυνατά και τα αδύνατα σημεία του και να εντοπιστούν πιθανές βελτιώσεις για μελλοντικές επεκτάσεις. Τα αποτελέσματα που παρουσιάζονται στις επόμενες ενότητες βασίζονται σε επαναλαμβανόμενες δοκιμές και παρέχουν μια ολοκληρωμένη εικόνα του τελικού συστήματος.

5.1 Ανάλυση απόδοσης και συμπεριφοράς συστήματος

Η ανάλυση απόδοσης και συμπεριφοράς συστήματος επικεντρώνεται στα βασικά λειτουργικά και υλικά χαρακτηριστικά της υλοποίησης του Tetris σε FPGA.

Συγκεκριμένα, η αξιολόγηση περιλαμβάνει τον λειτουργικό έλεγχο του συστήματος για την επιβεβαίωση της ορθής λειτουργίας του καθώς και τη μελέτη της χρονικής απόκρισής του, με στόχο την εκτίμηση της ταχύτητας αντίδρασης στις εντολές του χρήστη μέσω του χειριστηρίου. Επιπλέον, εξετάζεται η ποιότητα της παραγόμενης εικόνας μέσω της διεπαφής HDMI, τόσο ως προς τη σταθερότητα του σήματος όσο και ως προς την ορθότητα της απεικόνισης των γραφικών στοιχείων. Ιδιαίτερη έμφαση δίνεται, επίσης, στη χρήση των διαθέσιμων πόρων του FPGA, καθώς αποτελεί κρίσιμο δείκτη αποδοτικότητας της σχεδίασης και καθορίζει τη δυνατότητα επέκτασης του συστήματος.

Συνολικά, η παραπάνω προσέγγιση παρέχει ένα ολοκληρωμένο πλαίσιο αξιολόγησης της απόδοσης και συμπεριφοράς του συστήματος, επιτρέποντας την ανάλυση των επιμέρους τεχνικών χαρακτηριστικών του. Παρακάτω, ακολουθεί αναλυτική παρουσίαση και αξιολόγηση κάθε κριτηρίου ξεχωριστά.

5.1.1 Λειτουργικός έλεγχος συστήματος

Ο λειτουργικός έλεγχος του συστήματος αποσκοπεί στην επιβεβαίωση της ορθής υλοποίησης όλων των βασικών λειτουργιών του παιχνιδιού, σύμφωνα με τους κανόνες του Tetris. Η διαδικασία αυτή επικεντρώνεται στη συμπεριφορά του συστήματος σε επίπεδο λογικής, ανεξάρτητα από την απόδοση και τη χρήση πόρων.

Οι λειτουργικές δοκιμές αποτέλεσαν το βασικό στάδιο επαλήθευσης της σωστής υλοποίησης των κανόνων και της συνολικής συμπεριφοράς του παιχνιδιού. Στόχος τους

ήταν να διασφαλιστεί ότι όλα τα υποσυστήματα λειτουργούν ορθά, τόσο μεμονωμένα όσο και ως ολοκληρωμένο σύστημα, σύμφωνα με τις προδιαγραφές του κλασικού παιχνιδιού.

Αρχικά, πραγματοποιήθηκε αξιολόγηση των επιμέρους υποσυστημάτων, προκειμένου να επιβεβαιωθεί η σωστή λειτουργία κάθε μονάδας ξεχωριστά. Το υποσύστημα εισόδου εξετάστηκε ως προς την αξιόπιστη αναγνώριση των σημάτων του χειριστηρίου, τη σωστή αποθορυβοποίηση και μετατροπή των φυσικών ενεργειών του χρήστη σε λογικές εντολές κίνησης, περιστροφής και επιταχυνόμενης πτώσης. Κατά τη διάρκεια των δοκιμών, ελέγχθηκαν όλες οι εντολές (κίνηση αριστερά, δεξιά, κάτω, έναρξη/περιστροφή) σε διαφορετικές ταχύτητες πίεσης.

Αντίστοιχα, το υποσύστημα ελέγχου παιχνιδιού, υποβλήθηκε σε λεπτομερή έλεγχο, καθώς αποτελεί τον πυρήνα της συνολικής εφαρμογής. Εξετάστηκε η σωστή δημιουργία όλων των τύπων tetromino, η αρχική τοποθέτησή τους στο πλέγμα, η συνεχής πτώση τους με βάση τον προκαθορισμένο χρονισμό, καθώς και η ορθή εφαρμογή κινήσεων και περιστροφών. Ιδιαίτερη έμφαση δόθηκε στον έλεγχο του μηχανισμού ανίχνευσης συγκρούσεων, τόσο με τα όρια του πλέγματος όσο και με ήδη τοποθετημένα tetrominoes. Παράλληλα, αξιολογήθηκε η υλοποίηση των βασικών κανόνων, όπως το κλείδωμα των κομματιών, η δημιουργία νέων tetrominoes και η εκκαθάριση πλήρων γραμμών. Τέλος, εξετάστηκαν ειδικές περιπτώσεις λειτουργίας, όπως η περιστροφή κομματιών κοντά σε εμπόδια ή η ταυτόχρονη εισαγωγή πολλαπλών εντολών (μετακίνηση και περιστροφή), με στόχο την ανίχνευση πιθανών σφαλμάτων.

Το υποσύστημα απεικόνισης αξιολογήθηκε για τη σωστή παραγωγή σημάτων χρονισμού, τη σταθερότητα της εικόνας και την ορθή απεικόνιση όλων των στοιχείων του παιχνιδιού, όπως σωστή απεικόνιση και κεντράρισμα του πλέγματος στην οθόνη, σωστή απόδοση γραμμών διαχωρισμού. Οι δοκιμές επιβεβαίωσαν ότι το πλέγμα, τα tetrominoes και τα υπόλοιπα γραφικά στοιχεία εμφανίζονται σωστά στην οθόνη, χωρίς φαινόμενα αστάθειας ή αλλοιώσεων.

Παράλληλα, για το υποσύστημα ήχου επαληθεύτηκε ότι κάθε γεγονός παράγει τον αντίστοιχο διακριτό ήχο, όπως σύντομος υψηλός τόνος κατά το κλείδωμα tetromino, χαμηλότερος τόνος κατά την εκκαθάριση γραμμής και παρατεταμένος χαμηλός τόνος κατά το Game Over.

Μετά τις μεμονωμένες δοκιμές, πραγματοποιήθηκαν πλήρεις δοκιμές του συστήματος. Οι δοκιμές αυτές προσομοίωναν πλήρη ροή παιχνιδιού από την έναρξη μέχρι τον

τερματισμό, ελέγχοντας την αλληλεπίδραση όλων των υποσυστημάτων σε πραγματικό χρόνο. Αναλυτικότερα, έγιναν δοκιμές σε:

- Βασική ροή παιχνιδιού (εμφάνιση tetromino, κίνηση, περιστροφή, πτώση, κλείδωμα, εκκαθάριση γραμμής, νέο tetromino)
- Ακραίες καταστάσεις (πολύ γρήγορες κινήσεις με το χειριστήριο, γρήγορες περιστροφές, μετακίνηση οριακά πριν κλειδώσει το κομμάτι, πολλαπλές γραμμές που καθαρίζονται ταυτόχρονα)
- Ήχος και οπτική ανατροφοδότηση
- Μακροχρόνια λειτουργία (παιχνίδι διάρκειας άνω των 15 λεπτών)
- Έλεγχο καταστάσεων παιχνιδιού (Start, Game Over)
- Συμπεριφορά κατά την επαναφορά του συστήματος (reset)

Στο αρχικό στάδιο των δοκιμών εντοπίστηκαν συγκεκριμένα λειτουργικά σφάλματα, τα οποία σχετίζονταν κυρίως με τον έλεγχο συγκρούσεων και τη διαχείριση της κατάστασης του πλέγματος. Συγκεκριμένα, παρατηρήθηκε ότι τα tetrominoes μπορούσαν να περιστραφούν εντός ήδη κατειλημμένων περιοχών, γεγονός που υποδείκνυε ανεπαρκή έλεγχο σύγκρουσης. Επιπλέον, σε περιπτώσεις περιστροφής κοντά σε τοίχο, μπορεί να αποτύγχανε αν και υπήρχε επαρκής χώρος. Παράλληλα, διαπιστώθηκε ότι μετά την εκκαθάριση πλήρων γραμμών δεν πραγματοποιούνταν σωστά η καθοδική μετακίνηση των ανώτερων tetrominoes, γεγονός που προκαλούσε ασυνέπεια στη δομή του πλέγματος. Τέλος, εντοπίστηκε σφάλμα στον μηχανισμό τερματισμού του παιχνιδιού, καθώς σε περίπτωση πλήρωσης του πλέγματος δεν οδηγούσε σε τερματισμό, αλλά συνέχιζε να εμφανίζει νεα tetrominoes.

Όλα τα παραπάνω προβλήματα εντοπίστηκαν και διορθώθηκαν μέσω κατάλληλων τροποποιήσεων στη λογική ελέγχου σύγκρουσης, στη διαχείριση του πλέγματος και στον έλεγχο κατάστασης του παιχνιδιού, με αποτέλεσμα την ορθή και σταθερή λειτουργία του συστήματος. Μετά την εφαρμογή των διορθώσεων, έγινε επανάληψη των δοκιμών του συστήματος, με συνεχή χρήση του χειριστηρίου και παρατεταμένη διάρκεια παιχνιδιού. Οι δοκιμές αυτές επιβεβαίωσαν τη σωστή συνεργασία όλων των υποσυστημάτων, τη σταθερότητα της συνολικής λειτουργίας και την απουσία λειτουργικών σφαλμάτων κατά την πλήρη εκτέλεση του παιχνιδιού (Πίνακας 8).

Κατηγορία	Περιγραφή	Αποτέλεσμα
Βασική ροή παιχνιδιού	• Δημιουργία tetromino και εμφάνιση στην κορυφή του πλέγματος • Πτώση tetromino • Κινήσεις • Κλείδωμα tetromino όταν ακουμπήσει στη βάση ή σε άλλα tetrominoes • Διαγραφή πλήρους γραμμής/γραμμών • Εμφάνιση νέου tetromino από την κορυφή του πλέγματος	Ομαλή δημιουργία και πτώση tetromino σε κάθε level Επιτυχές κλείδωμα, σωστή εκκαθάριση και μετατόπιση γραμμών, και εμφάνιση νέου tetromino
Κινήσεις	• Δεξιά • Αριστερά • Γρήγορη πτώση • Περιστροφή	Σωστές μετακινήσεις και περιστροφές των tetrominoes
Συγκρούσεις	• Με τη βάση του πλέγματος • Με τα όρια του πλέγματος • Με ήδη τοποθετημένα tetrominoes	Ύπαρξη σύγκρουσης και διακοπή κίνησης ή περιστροφής
Ακραίες καταστάσεις	• Πολύ γρήγορες κινήσεις με το χειριστήριο, γρήγορες περιστροφές • Μετακίνηση οριακά πριν κλειδώσει το κομμάτι • Πολλαπλές γραμμές που καθαρίζονται ταυτόχρονα	Σωστές μετακινήσεις χωρίς απώλειες και περιστροφές, σωστό κλείδωμα και διαγραφή πολλαπλών γραμμών
Ήχος και οπτική	• Ενεργοποίηση ήχου για	Σωστή ενεργοποίηση

ανατροφοδότηση	κάθε τύπο γεγονότος • Συγχρονισμός ήχου με οπτικά γεγονότα	ήχου και συγχρονισμός με οπτικά γεγονότα
Μακροχρόνια λειτουργία	• Διάρκεια παιχνιδιού >15 λεπτά	Σταθερή λειτουργία χωρίς κολλήματα
Καταστάσεις παιχνιδιού	• Έναρξη παιχνιδιού • Game Over	Σωστή εναλλαγή καταστάσεων
Επαναφορά παιχνιδιού (Reset)	• Πάτημα του κουμπιού reset σε διάφορες φάσεις του παιχνιδιού	Σωστή ανταπόκριση και επαναφορά στην αρχική οθόνη σε οποιαδήποτε φάση

Πίνακας 8 Λειτουργικές δοκιμές και αποτελέσματα

5.1.2 Αξιολόγηση χρονικής απόκρισης

Η αξιολόγηση της χρονικής απόκρισης του συστήματος πραγματοποιείται με στόχο τη σύγκριση της τελικής υλοποίησης με τις αρχικές χρονικές και λειτουργικές προδιαγραφές, όπως αυτές είχαν οριστεί κατά τον σχεδιασμό (Πίνακας 9). Οι παράμετροι αυτοί καθορίζουν τον τρόπο με τον οποίο το σύστημα αποκρίνεται στις εντολές του χρήστη, καθώς και τη συνολική εμπειρία χειρισμού κατά τη διάρκεια του παιχνιδιού.

Η λειτουργία του συστήματος βασίζεται στο κύριο ρολόι των 50MHz, το οποίο χρησιμοποιείται ως σημείο αναφοράς για τον συγχρονισμό των υποσυστημάτων του παιχνιδιού. Μέσω αυτού του σταθερού χρονισμού υλοποιούνται οι απαραίτητοι μετρητές, οι οποίοι καθορίζουν με ακρίβεια τους χρόνους απόκρισης των χειρισμών, την πτώση των tetrominoes, τις καθυστερήσεις μετακίνησης, καθώς και τη συνολική ροή του παιχνιδιού.

Από την ανάλυση προκύπτει ότι οι παράμετροι DAS (Delayed Auto Shift) και ARR (Auto Repeat Rate) υλοποιούνται με μεγάλη ακρίβεια. Συγκεκριμένα, η τιμή DAS αντιστοιχεί σε 150 ms και η τιμή ARR σε 50 ms, καλύπτοντας πλήρως τις αρχικές απαιτήσεις, εξασφαλίζοντας ομαλή οριζόντια μετακίνηση των tetrominoes. Κατά τη διάρκεια της υλοποίησης, πραγματοποιήθηκαν δοκιμές με μικρότερες αλλά και μεγαλύτερες τιμές χρονισμού για τις συγκεκριμένες παραμέτρους, με στόχο την αξιολόγηση της επίδρασής τους στη λειτουργικότητα του συστήματος. Οι μικρότερες

τιμές οδήγησαν σε υπερβολικά γρήγορη οριζόντια μετακίνηση, δυσχεραίνοντας τον ακριβή έλεγχο κινήσεων και προκαλώντας ανεπιθύμητες πολλαπλές μετακινήσεις. Αντίστοιχα, οι μεγαλύτερες τιμές έκαναν την απόκριση πιο αργή, μειώνοντας την αμεσότητα του χειρισμού. Έτσι, οι δοκιμές έδειξαν ότι οι τελικές επιλεγμένες τιμές αποτελούν τη βέλτιστη ισορροπία μεταξύ ταχύτητας και ακρίβειας, εξασφαλίζοντας σταθερή και φυσική εμπειρία χειρισμού.

Στη συνέχεια, η αρχική ταχύτητα πτώσης (Falling Speed) υλοποιήθηκε στο 1 Hz (1000 ms), δηλαδή σε χρονική τιμή μεγαλύτερη από την αρχική προδιαγραφή των 500 ms. Αυτό έχει ως αποτέλεσμα το παιχνίδι να ξεκινά με χαμηλότερη ταχύτητα, έτσι ώστε να έχει πιο ήπιο ρυθμό και να είναι λιγότερο απαιτητικό στην έναρξή του. Πραγματοποιήθηκαν δοκιμές και με την τιμή των 500 ms, σύμφωνα με την αρχική απαίτηση, ωστόσο διαπιστώθηκε ότι ο ρυθμός αυτός ήταν αρκετά γρήγορος για την έναρξη του παιχνιδιού, ιδιαίτερα για αρχάριους χρήστες. Αντίθετα, η επιλογή των 1000 ms αποδείχθηκε περισσότερο προσιτή, παρέχοντας επαρκή χρόνο προσαρμογής. Παράλληλα, η σταδιακή αύξηση της ταχύτητας κατά 20 ms ανά εκκαθάριση γραμμής διατηρεί την απαιτούμενη κλιμάκωση δυσκολίας, εξασφαλίζοντας ότι το παιχνίδι προοδευτικά επιταχύνεται και γίνεται πιο απαιτητικό.

Σχετικά με τη λειτουργία soft drop, εξετάστηκαν διάφορες τιμές ταχύτητας, συμπεριλαμβανομένης της αρχικής προδιαγραφής των 30 ms, όπου η πτώση αποδείχθηκε υπερβολικά γρήγορη για αποτελεσματικό χειρισμό. Κατά την ενεργοποίηση του soft drop, ο μετρητής πτώσης επιλέχτηκε να αυξάνεται κατά 10 ανά κύκλο ρολογιού, με αποτέλεσμα η ταχύτητα πτώσης να γίνεται περίπου δέκα φορές μεγαλύτερη σε σχέση με την κανονική λειτουργία. Με βάση την αρχική ρύθμιση της ταχύτητας πτώσης του παιχνιδιού, αυτό αντιστοιχεί σε περίπου 100 ms ανά βήμα πτώσης, προσφέροντας σημαντική επιτάχυνση της κίνησης χωρίς να χάνεται ο έλεγχος από τον παίκτη. Η τελική αυτή τιμή επιλέχθηκε κατόπιν πρακτικών δοκιμών, καθώς προσφέρει βελτιωμένη εμπειρία χειρισμού, επιτρέποντας ταχεία πτώση και παράλληλα ικανοποιητική ακρίβεια στην τοποθέτηση των tetrominoes, χωρίς να επηρεάζεται αρνητικά η συνολική εμπειρία παιχνιδιού. Αντίθετα, η λειτουργία hard drop δεν υλοποιήθηκε στην παρούσα έκδοση του συστήματος, καθώς κρίθηκε ότι δεν ήταν απαραίτητη για τη βασική λειτουργικότητα του παιχνιδιού. Ωστόσο, προβλέπεται ως πιθανή μελλοντική επέκταση, η οποία θα μπορούσε να ενισχύσει την ταχύτητα και τη στρατηγική του παιχνιδιού.

Αντίθετα, οι υπόλοιπες χρονικές παράμετροι, όπως το ARE (καθυστέρηση εμφάνισης νέου tetromino), το line clear delay, το lock delay και το ground time, δεν υλοποιήθηκαν σύμφωνα με τις αρχικές προδιαγραφές. Στην παρούσα σχεδίαση, η εμφάνιση νέου tetromino πραγματοποιείται στον επόμενο κύκλο πτώσης (fall tick) μετά το κλείδωμα του προηγούμενου, ενώ η εκκαθάριση γραμμών γίνεται χωρίς την προσθήκη ξεχωριστής χρονικής καθυστέρησης. Παράλληλα, δεν ενσωματώθηκε ανεξάρτητος μηχανισμός για το lock delay και το ground time, καθώς το κλείδωμα του tetromino και ο χρόνος παραμονής του στο έδαφος εξαρτώνται από τον επόμενο κύκλο, ο οποίος αντιστοιχεί στο χρόνο του fall tick. Δεδομένου ότι η διάρκεια του fall tick μειώνεται προοδευτικά όσο αυξάνεται η δυσκολία του παιχνιδιού, οι συγκεκριμένοι χρόνοι μεταβάλλονται δυναμικά κατά τη διάρκεια της λειτουργίας, αποκλίνοντας από τις αρχικά επιθυμητές τιμές των 400 ms και 1500 ms αντίστοιχα.

Παρά τις αποκλίσεις από τις αρχικές προδιαγραφές, τα αποτελέσματα των δοκιμών έδειξαν ότι δεν επηρεάζεται η συνολική λειτουργικότητα του παιχνιδιού ούτε η εμπειρία του χρήστη. Η ροή του παιχνιδιού ήταν ομαλή, η λογική διαχείρισης της πτώσης και των συγκρούσεων λειτούργησε με σταθερότητα, ενώ ο χειρισμός από την πλευρά του χρήστη παρέμεινε άμεσος και αποτελεσματικός.

Συνολικά, η σχεδίαση πέτυχε να διατηρήσει την απαιτούμενη λειτουργικότητα και την ευχάριστη εμπειρία χρήσης, ακόμη και χωρίς την πλήρη εφαρμογή όλων των χρονικών χαρακτηριστικών. Οι αποκλίσεις που παρατηρούνται εντοπίζονται κυρίως σε δευτερεύοντες μηχανισμούς βελτιστοποίησης και λεπτομερειών συμπεριφοράς, χωρίς να δημιουργούν προβλήματα στη βασική λειτουργία, στη σταθερότητα ή στην ανταπόκριση του παιχνιδιού. Οι δοκιμές επιβεβαίωσαν ότι το σύστημα παραμένει λειτουργικό και αξιόπιστο, καλύπτοντας αποτελεσματικά τον βασικό στόχο της υλοποίησης ενός πλήρως λειτουργικού Tetris σε FPGA.

Χρονικοί Παράμετροι	Αρχικές Απαιτήσεις (ms)	Υλοποίηση (ms)
DAS	150	150
ARR	50	50
ARE	200	Fall tick
Falling Speed	500	1000
Lock Delay	400	Fall tick
Line Clear Delay	200	Fall tick
Ground Time	1500	Fall tick
Soft Drop	30	100
Hard Drop	30	-

Πίνακας 9 Σύγκριση χρονικών παραμέτρων

5.1.3 Έλεγχος ποιότητας παραγόμενης εικόνας

Η ποιότητα της παραγόμενης εικόνας αξιολογήθηκε με βάση τη σταθερότητα του οπτικού σήματος, την ευκρίνεια της απεικόνισης, τη σωστή γεωμετρία του πλέγματος και τη συνολική οπτική εμπειρία κατά τη λειτουργία του παιχνιδιού. Η υλοποίηση του συστήματος βασίστηκε σε ανάλυση Full HD 1920×1080 μέσω διασύνδεσης HDMI, αξιοποιώντας κατάλληλο PLL για τη δημιουργία pixel clock συχνότητας 148.5 MHz, εξασφαλίζοντας πλήρη συμβατότητα με το πρότυπο χρονισμού 1080p. Η χρήση του PLL επέτρεψε την παραγωγή σταθερού και ακριβούς χρονισμού εικόνας, γεγονός που συνέβαλε στην απουσία οπτικών παραμορφώσεων, τρεμοπαίγματος ή αστάθειας συγχρονισμού.

Η απεικόνιση του πλέγματος πραγματοποιήθηκε με σαφή οριοθέτηση κάθε κελιού μέσω μαύρων γραμμών διαχωρισμού πάχους δύο pixel, προσφέροντας υψηλή ευκρίνεια και διάκριση μεταξύ ενεργών και ανενεργών περιοχών. Τα ενεργά τετράγωνα αποδίδονται με λευκό χρώμα, τα κενά κελιά με μαύρο, ενώ το εξωτερικό φόντο αποδίδεται σε ουδέτερο γκρι χρώμα, επιτυγχάνοντας ικανοποιητική αντίθεση με το πεδίο παιχνιδιού.

Κατά τις δοκιμές σε πραγματική οθόνη HDMI, η εικόνα παρέμεινε σταθερή σε όλη τη διάρκεια λειτουργίας, χωρίς απώλειες συγχρονισμού ή σφάλματα απεικόνισης. Η στοίχιση του πλέγματος στο κέντρο της οθόνης ήταν ακριβής, ενώ η κατακόρυφη διάταξη υλοποιήθηκε σωστά. Παράλληλα, δεν παρατηρήθηκαν φαινόμενα, όπως ασυνεχής οπτική ροή, όπου τα tetrominoes θα εμφανίζονταν με μικρές διακοπές ή άλματα θέσης, ούτε καθυστερήσεις στην ανανέωση της εικόνας, γεγονός που επιβεβαιώνει την συνεργασία μεταξύ της λογικής παιχνιδιού και του υποσυστήματος HDMI controller.

Συνολικά, ο έλεγχος ποιότητας της παραγόμενης εικόνας έδειξε ότι το σύστημα επιτυγχάνει υψηλό επίπεδο οπτικής αξιοπιστίας και σταθερότητας κατά τη λειτουργία του σε πραγματικές συνθήκες. Δεν παρατηρήθηκαν σφάλματα συγχρονισμού, παραμορφώσεις ή αστοχίες στην ανανέωση της εικόνας, ενώ η γεωμετρική απεικόνιση του πλέγματος διατηρείται ακριβής και σωστά κεντραρισμένη στην οθόνη. Επιπλέον, η απουσία φαινομένων ασυνεχούς οπτικής ροής και καθυστερήσεων στην απόδοση της εικόνας επιβεβαιώνει τον ορθό χρονισμό του συστήματος. Επομένως, η σχεδίαση ανταποκρίνεται πλήρως στις απαιτήσεις ενός λειτουργικού συστήματος, παρέχοντας σταθερή και ευανάγνωστη απεικόνιση.

5.1.4 Ανάλυση μετρικών κώδικα

Η αξιολόγηση της υλοποίησης δεν περιορίζεται μόνο στη λειτουργική ορθότητα και την απόδοση του συστήματος, αλλά επεκτείνεται και σε ποσοτικές μετρικές που αποτυπώνουν τη δομή και την πολυπλοκότητα του κώδικα. Οι μετρικές αυτές παρέχουν μια συνολική εικόνα της σχεδιαστικής προσέγγισης, της αρθρωτότητας και της διαχειρισιμότητας του συστήματος.

Βασικές μετρικές που εξετάστηκαν είναι ο αριθμός των modules και το πλήθος των γραμμών κώδικα (Πίνακας 10). Το σύστημα αποτελείται από δέκα διακριτές μονάδες, με συνολικό μέγεθος 1.995 γραμμών κώδικα (συμπεριλαμβανομένων και των σχολίων) σε γλώσσα περιγραφής υλικού.

Project myTetris			
	Modules	**Γραμμές κώδικα** (συμπεριλαμβάνονται τα σχόλια)	**Ποσοστό**
	myTetris_top	206	28.1%
	input_controller	133	8.5%
	grid_generator	169	6.7%
	tetromino_rom	234	11.7%
	audio_controller	77	9.8%
	game_logic	560	7.5%
	I2C_WRITE_WDATA	106	8.3%
	I2C_HDMI_Config	196	5.3%
	I2C_Controller	149	3.9%
	I2C_AV_Config	165	10.3%
Σύνολο	**10**	**1.995**	**100%**

Πίνακας 10 Μετρικές πολυπλοκότητας υλοποίησης

Η κατανομή του κώδικα μεταξύ των modules δεν είναι ομοιόμορφη, καθώς ορισμένα τμήματα του συστήματος εμφανίζουν μεγαλύτερη πολυπλοκότητα από άλλα. Συγκεκριμένα, το module του game_logic, συγκεντρώνει το μεγαλύτερο μέρος του κώδικα, γεγονός που είναι αναμενόμενο, καθώς περιλαμβάνει τον βασικό έλεγχο της ροής του παιχνιδιού και τη διαχείριση των κανόνων λειτουργίας του Tetris. Αντίθετα, modules όπως το audio_controller, εμφανίζουν μικρότερο μέγεθος, καθώς υλοποιούν πιο εξειδικευμένη και περιορισμένη λειτουργικότητα. Επιπλέον, η ύπαρξη πολλαπλών modules που σχετίζονται με το υποσύστημα HDMI και την επικοινωνία μέσω I2C, όπως τα I2C_Controller και I2C_HDMI_Config, αναδεικνύει τη σχεδιαστική πολυπλοκότητα που απαιτείται για τη σωστή διαχείριση της εξόδου εικόνας.

Η ανάλυση των μετρικών αναδεικνύει ότι η υλοποίηση ακολουθεί αρθρωτή αρχιτεκτονική, όπου κάθε module είναι υπεύθυνο για μία συγκεκριμένη λειτουργία. Η προσέγγιση αυτή προσφέρει σημαντικά πλεονεκτήματα, όπως την ευκολότερη κατανόηση του κώδικα, τη διευκόλυνση της ανάπτυξης και αποσφαλμάτωσης του συστήματος, ενώ παράλληλα καθιστά εφικτή τη μελλοντική επέκτασή του.

Συνολικά, οι μετρικές του συστήματος δείχνουν μια οργανωμένη και αρθρωτή υλοποίηση, με διαχωρισμό λειτουργιών και ελεγχόμενη πολυπλοκότητα. Το μέγεθος του κώδικα κρίνεται επαρκές για τις απαιτήσεις της εφαρμογής, ενώ η δομή του επιτρέπει την εύκολη συντήρηση και μελλοντική επέκταση του συστήματος.

5.1.5 Χρήση πόρων FPGA

Η αξιολόγηση της χρήσης πόρων του FPGA αποτελεί κρίσιμο παράγοντα για την εκτίμηση της αποδοτικότητας και της επεκτασιμότητας της υλοποίησης. Η σχεδίαση υλοποιήθηκε στην αναπτυξιακή πλακέτα DE23-Lite, η οποία ενσωματώνει FPGA της οικογένειας Agilex 3 (A3CZ135BB18AE7S). Η ανάλυση της κατανάλωσης πόρων πραγματοποιήθηκε μετά την επιτυχή ολοκλήρωση της σύνθεσης (Synthesis) και της τοποθέτησης-δρομολόγησης (Fitting) με το Quartus Prime Pro Edition (Πίνακας 11).

Fitter Status : Successful - Mon Mar 9 22:03:41 2026
Quartus Prime Version : 25.1.0 Build 129 03/26/2025 Patches 0.36 SC Pro Edition
Revision Name : myTetris
Top-level Entity Name : myTetris_top
Family : Agilex 3
Device : A3CZ135BB18AE7S
Timing Models : Preliminary
Power Models : Preliminary
Device Status : Preliminary
Logic utilization (in ALMs) : 2,622 / 45,800 (6 %)
Total dedicated logic registers : 1222
Total pins : 167 / 208 (80 %)
Total block memory bits : 0 / 7,229,440 (0 %)
Total RAM Blocks : 0 / 353 (0 %)
Total DSP Blocks : 0 / 184 (0 %)
Total HSSI HPS : 0 / 1 (0 %)
Total PLLs : 1 / 4 (25 %)

Πίνακας 11 Fitter report

Η χρήση των λογικών πόρων είναι αρκετά χαμηλή, με το σύστημα να καταναλώνει 6% των συνολικά διαθέσιμων Adaptive Logic Modules (ALMs), γεγονός που υποδηλώνει ιδιαίτερα αποδοτική υλοποίηση. Η χαμηλή αυτή κατανάλωση οφείλεται στη σχετικά απλή λογική του παιχνιδιού, καθώς και στην υιοθέτηση αρθρωτής αρχιτεκτονικής, όπου η λειτουργικότητα διαχωρίζεται σε επιμέρους ανεξάρτητα modules. Ο αριθμός των καταχωρητών που χρησιμοποιούνται είναι επίσης περιορισμένος, γεγονός που δείχνει ότι η σχεδίαση δεν απαιτεί εκτεταμένη αποθήκευση κατάστασης. Η χαμηλή χρήση λογικών

πόρων αποτελεί ένδειξη βελτιστοποίησης και αφήνει περιθώριο για μελλοντικές επεκτάσεις όπως προσθήκη χρωμάτων, εμφάνιση σκορ, προεπισκόπιση επόμενου κομματιού ή ακόμα και multiplayer λειτουργία.

Όσον αφορά τη χρήση μνήμης, παρατηρείται ότι δεν αξιοποιούνται καθόλου block μνήμης (BRAM), που οφείλεται κυρίως στην απλή αναπαράσταση του πλέγματος. Η επιλογή αυτή απλοποιεί τη σχεδίαση και μειώνει την πολυπλοκότητα, αλλά δεν εκμεταλλεύεται τις δυνατότητες της FPGA για αποδοτική αποθήκευση δεδομένων. Σε ενδεχόμενες επεκτάσεις του συστήματος, η αξιοποίηση των block μνήμης θα μπορούσε να συμβάλει στη βελτίωση της διαχείρισης δεδομένων, όπως για παράδειγμα στην αποθήκευση γραφικών ή του τρέχοντος παιχνιδιού.

Παρόμοια εικόνα παρουσιάζεται και στη χρήση των DSP blocks, τα οποία δεν αξιοποιούνται στην παρούσα υλοποίηση. Το γεγονός αυτό είναι αναμενόμενο, δεδομένου ότι το παιχνίδι Tetris δεν απαιτεί πολύπλοκες αριθμητικές πράξεις. Παρόλα αυτά, η διαθεσιμότητα αυτών των πόρων παρέχει σημαντικές δυνατότητες για μελλοντικές επεκτάσεις, όπως προσθήκη σύνθετων γραφικών.

Επιπλέον, δεν χρησιμοποιούνται πόροι υψηλής ταχύτητας σειριακής επικοινωνίας (High-Speed Serial Interface-HSSI) ή ενσωματωμένου επεξεργαστή (Hard Processor System-HPS), καθώς η υλοποίηση βασίζεται αποκλειστικά σε λογική hardware χωρίς την ανάγκη επεξεργασίας από λογισμικό.

Αντίθετα, η χρήση των ακροδεκτών εισόδου/εξόδου (pins) ανέρχεται σε σχετικά υψηλό ποσοστό της τάξης του 80%. Η αυξημένη χρήση οφείλεται κυρίως στη διεπαφή HDMI, η οποία απαιτεί σημαντικό αριθμό σημάτων για τη μετάδοση της εικόνας, καθώς και στη διασύνδεση με το arcade χειριστήριο και τα επιμέρους περιφερειακά.

Τέλος, χρησιμοποιείται μία μονάδα PLL, η οποία αντιστοιχεί στο 25% των διαθέσιμων πόρων. Η χρήση της PLL είναι απαραίτητη για τη δημιουργία του κατάλληλου ρολογιού λειτουργίας (pixel clock 148.5 MHz) της HDMI διεπαφής, που απαιτείται για την υποστήριξη ανάλυσης 1920×1080 60Hz.

Συνολικά, τα αποτελέσματα δείχνουν ότι η υλοποίηση χαρακτηρίζεται από ιδιαίτερα χαμηλή κατανάλωση λογικών και εξειδικευμένων πόρων, γεγονός που υποδηλώνει αποδοτική σχεδίαση. Παράλληλα, η ύπαρξη σημαντικού ποσοστού αχρησιμοποίητων πόρων επιτρέπει την εύκολη επέκταση του συστήματος με επιπλέον λειτουργίες, χωρίς να απαιτείται σημαντική αναδιάρθρωση της αρχιτεκτονικής.

5.2 Αποτελέσματα αξιολόγησης

Η συνολική αξιολόγηση του συστήματος δείχνει ότι η υλοποίηση του παιχνιδιού Tetris σε FPGA ανταποκρίνεται πλήρως στις απαιτήσεις της εφαρμογής, τόσο σε λειτουργικό όσο και σε τεχνικό επίπεδο. Ο λειτουργικός έλεγχος επιβεβαίωσε την ορθή υλοποίηση των βασικών μηχανισμών του παιχνιδιού, συμπεριλαμβανομένης της διαχείρισης των tetrominoes, της ανίχνευσης συγκρούσεων, της εκκαθάρισης γραμμών και της διαχείρισης των καταστάσεων λειτουργίας. Τα προβλήματα που εντοπίστηκαν κατά τη φάση των αρχικών δοκιμών αντιμετωπίστηκαν επιτυχώς, οδηγώντας σε ένα σταθερό και αξιόπιστο σύστημα χωρίς αποκλίσεις από την αναμενόμενη συμπεριφορά.

Σε επίπεδο χρονικής απόκρισης, το σύστημα παρουσιάζει σταθερή και προβλέψιμη συμπεριφορά, με καθυστερήσεις που παραμένουν εντός αποδεκτών ορίων για εφαρμογές πραγματικού χρόνου. Η επιλογή των παραμέτρων χρονισμού, όπως ο χρόνος πτώσης των tetrominoes και οι καθυστερήσεις χειρισμού (DAS, ARR, debounce), συμβάλλει στην επίτευξη ομαλής και άμεσης αλληλεπίδρασης με τον χρήστη.

Η διεπαφή HDMI εξασφαλίζει υψηλής ποιότητας απεικόνιση, με σταθερό χρονισμό και σωστή αναπαράσταση των γραφικών στοιχείων. Η χρήση κατάλληλων παραμέτρων συγχρονισμού οδηγούν σε ομαλή εικόνα χωρίς φαινόμενα αστάθειας.

Η ανάλυση της χρήσης πόρων του FPGA, ανέδειξε μια αποδοτική υλοποίηση, με χαμηλή κατανάλωση λογικών στοιχείων και μη αξιοποίηση εξειδικευμένων πόρων, όπως block μνήμης και DSP μονάδων. Η προσέγγιση αυτή, σε συνδυασμό με την αρθρωτή αρχιτεκτονική του συστήματος, επιτρέπει τη διατήρηση χαμηλής πολυπλοκότητας και παρέχει περιθώρια για μελλοντική επέκταση της λειτουργικότητας.

Συνολικά, το σύστημα χαρακτηρίζεται από υψηλό βαθμό αξιοπιστίας, αποδοτικότητας και επεκτασιμότητας. Η επιτυχής ολοκλήρωση της υλοποίησης και η θετική αποτίμηση σε όλους τους άξονες αξιολόγησης επιβεβαιώνουν την καταλληλότητα του FPGA ως πλατφόρμα για την ανάπτυξη εφαρμογών σε πραγματικό χρόνο. Τα συμπεράσματα που προκύπτουν, καθώς και προτάσεις για μελλοντικές βελτιώσεις, παρουσιάζονται στο επόμενο κεφάλαιο.

6. Συμπεράσματα και μελλοντικές βελτιώσεις

Η ολοκλήρωση της παρούσας εργασίας συνοδεύτηκε από τη συστηματική ανάλυση και αξιολόγηση της υλοποίησης, καλύπτοντας τόσο τη λειτουργική συμπεριφορά όσο και τα τεχνικά χαρακτηριστικά του συστήματος. Η διαδικασία αυτή δεν περιορίστηκε μόνο στην επιβεβαίωση της ορθής λειτουργίας, αλλά επεκτάθηκε στην κατανόηση των σχεδιαστικών επιλογών, των περιορισμών και των δυνατοτήτων που προσφέρει η υλοποίηση σε περιβάλλον FPGA. Μέσα από την ανάπτυξη και τη δοκιμή του συστήματος, αναδείχθηκαν βασικές πτυχές της σχεδίασης ψηφιακών κυκλωμάτων, όπως η σημασία του συγχρονισμού, η διαχείριση καταστάσεων και η ανάγκη για αξιόπιστο έλεγχο εισόδων και εξόδων.

Με βάση τα αποτελέσματα της αξιολόγησης που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, προκύπτουν ορισμένα γενικότερα συμπεράσματα σχετικά με τη σχεδίαση και την υλοποίηση του συστήματος τα οποία αναλύονται παρακάτω.

6.1 Συμπεράσματα

Η ολοκλήρωση του παρόντος έργου συνοδεύτηκε από τη συστηματική ανάλυση, σχεδίαση, ανάπτυξη και αξιολόγηση της υλοποίησης, καλύπτοντας τόσο τη λειτουργική συμπεριφορά όσο και τα τεχνικά χαρακτηριστικά του συστήματος. Η διαδικασία αυτή δεν περιορίστηκε μόνο στην επιβεβαίωση της ορθής λειτουργίας, αλλά επεκτάθηκε στην κατανόηση των σχεδιαστικών επιλογών, των περιορισμών και των δυνατοτήτων που προσφέρει η υλοποίηση. Μέσα από την ανάπτυξη και τη δοκιμή του συστήματος, αναδείχθηκαν βασικές πτυχές της σχεδίασης ψηφιακών κυκλωμάτων, όπως η σημασία του συγχρονισμού, η διαχείριση καταστάσεων και η αξιόπιστη συνεργασία των επιμέρους υποσυστημάτων.

Τα αποτελέσματα της αξιολόγησης έδειξαν ότι η σχεδίαση ανταποκρίνεται στις βασικές απαιτήσεις του έργου, τόσο ως προς την εφαρμογή των μηχανισμών του παιχνιδιού όσο και ως προς την ποιότητα της αλληλεπίδρασης με τον χρήστη. Η άμεση απόκριση στις εντολές εισόδου, η ομαλή απεικόνιση μέσω HDMI και η συνολική σταθερότητα λειτουργίας επιβεβαιώνουν την αποτελεσματικότητα της αρχιτεκτονικής προσέγγισης που ακολουθήθηκε.

Παράλληλα, η ανάλυση χρήσης των πόρων έδειξε ότι η σχεδίαση παρουσιάζει ικανοποιητικό βαθμό αποδοτικότητας, αξιοποιώντας αποτελεσματικά τους διαθέσιμους πόρους, χωρίς επιβάρυνση του συστήματος. Η αποτελεσματική αυτή αξιοποίηση των πόρων ενισχύει τη συνολική αποδοτικότητα της σχεδίασης και παρέχει σημαντικά περιθώρια για μελλοντική επέκταση, βελτίωση και ενσωμάτωση επιπρόσθετων λειτουργιών.

Παρά τα θετικά αποτελέσματα της υλοποίησης, υπήρχαν και ορισμένοι περιορισμοί που επηρεάζουν κυρίως τις δυνατότητες επέκτασης και την εμπειρία χρήσης. Συγκεκριμένα, η απεικόνιση περιορίστηκε σε ασπρόμαυρη μορφή, ενώ δεν υλοποιήθηκαν επιπρόσθετες λειτουργίες όπως η προεπισκόπηση του επόμενου tetromino ή η προβολή του σκορ και του επιπέδου δυσκολίας. Οι περιορισμοί αυτοί δεν μειώνουν τη συνολική αποδοτικότητα του έργου, αλλά αποτελούν σημαντικά σημεία αναφοράς για μελλοντικές βελτιώσεις, συμβάλλοντας στην περαιτέρω αναβάθμιση της λειτουργικότητας, της αποδοτικότητας και της συνολικής εμπειρίας χρήσης.

Συνολικά, η παρούσα εργασία επιβεβαιώνει ότι τα FPGA μπορούν να υποστηρίξουν αποτελεσματικά την ανάπτυξη διαδραστικών ψηφιακών συστημάτων, συνδυάζοντας αξιοπιστία, ευελιξία και αποδοτική αξιοποίηση πόρων. Η υλοποίηση αποδεικνύει τις δυνατότητες της συγκεκριμένης τεχνολογίας όχι μόνο ως μέσο ανάπτυξης της παρούσας εφαρμογής, αλλά και ως ισχυρή βάση για τη μελλοντική δημιουργία πιο σύνθετων, επεκτάσιμων και τεχνολογικά προηγμένων συστημάτων.

6.2 Μελλοντικές βελτιώσεις

Παρ’ ότι η παρούσα υλοποίηση επιτυγχάνει τη βασική λειτουργικότητα του παιχνιδιού και παρουσιάζει σταθερή συμπεριφορά, υπάρχουν αρκετές κατευθύνσεις στις οποίες θα μπορούσε να επεκταθεί ή να βελτιωθεί. Οι πιθανές αυτές βελτιώσεις δεν αφορούν μόνο την προσθήκη νέων χαρακτηριστικών, αλλά και τη βελτιστοποίηση της αρχιτεκτονικής και της αξιοποίησης των διαθέσιμων πόρων του συστήματος.

Μια πρώτη σημαντική κατεύθυνση αφορά την εκμετάλλευση των διαθέσιμων πόρων, όπως τις ενσωματωμένες μονάδες μνήμης όπως τα RAM blocks και DSP blocks, οι οποίες στην παρούσα υλοποίηση δεν αξιοποιούνται. Η χρήση τους θα μπορούσε να επιτρέψει πιο οργανωμένη και αποδοτική αποθήκευση των δεδομένων του παιχνιδιού, όπως την

προσωρινή αποθήκευση εικόνας, γεγονός που θα βελτίωνε σημαντικά την ποιότητα της απεικόνισης και θα διευκόλυνε την υλοποίηση πιο σύνθετων γραφικών χαρακτηριστικών.

Παράλληλα, η αξιοποίηση των εξειδικευμένων αριθμητικών μονάδων (DSP), θα μπορούσε να υποστηρίξει πιο προηγμένες λειτουργίες, όπως παραγωγή ηχητικών εφέ, μουσικής υπόκρουσης ή δημιουργία σύνθετων οπτικών εφέ. Αν και οι λειτουργίες αυτές δεν είναι απαραίτητες για τη βασική εκτέλεση του παιχνιδιού, θα μπορούσαν να προσφέρουν μια πιο ολοκληρωμένη εμπειρία παιχνιδιού.

Επιπλέον, η ενσωμάτωση ενός επεξεργαστή (HPS) θα επέτρεπε την ανάπτυξη πιο σύνθετων λειτουργιών, όπως μενού χρήστη, αποθήκευση ρυθμίσεων, διαχείριση πολλαπλών επιλογών παιχνιδιού ή ακόμη και επικοινωνία με εξωτερικά συστήματα, όπως σύνδεση με υπολογιστικά συστήματα ή δικτυακές υπηρεσίες. Η προσέγγιση αυτή θα αύξανε σημαντικά την πολυπλοκότητα αλλά και τις δυνατότητες του έργου.

Ιδιαίτερο ενδιαφέρον παρουσιάζει, επίσης, η δυνατότητα αξιοποίησης των διαθέσιμων πρωτοκόλλων επικοινωνίας UART και I2C, τα οποία υποστηρίζονται μέσω των αντίστοιχων ακίδων FPGA_UART και FPGA_I2C. Η ανάπτυξη υποσυστήματος UART Transmitter/Receiver θα μπορούσε να επιτρέψει τη διασύνδεση του FPGA με εξωτερικούς μικροελεγκτές ή με ηλεκτρονικούς υπολογιστές, παρέχοντας δυνατότητα ανταλλαγής δεδομένων σε πραγματικό χρόνο και υποστήριξη λειτουργιών όπως online πίνακες κατάταξης καλύτερων επιδόσεων παικτών. Παράλληλα, η αξιοποίηση του διαύλου I2C θα μπορούσε να χρησιμοποιηθεί για επικοινωνία με εξωτερικές μνήμες EEPROM, επιτρέποντας τη μόνιμη αποθήκευση δεδομένων, όπως υψηλές βαθμολογίες, ακόμη και μετά την απενεργοποίηση του συστήματος. Η ενσωμάτωση αυτών των δυνατοτήτων θα αναβάθμιζε σημαντικά την παρούσα υλοποίηση, μετατρέποντάς την από μια αυτόνομη εφαρμογή σε μια πιο ολοκληρωμένη arcade πλατφόρμα με δυνατότητες δικτύωσης, αποθήκευσης και προηγμένης επικοινωνίας.

Σε επίπεδο λειτουργικότητας, θα μπορούσαν να προστεθούν χαρακτηριστικά όπως, προσθήκη χρώματος στα tetrominoes, σύστημα βαθμολογίας, αποθήκευση σκορ υψηλών επιδόσεων, προεπισκόπιση επόμενου κομματιού, επίπεδα δυσκολίας με προσαρμογή της ταχύτητας, προσθήκη hard drop καθώς και οπτικών ή ηχητικών εφέ που θα ενισχύουν την αλληλεπίδραση. Επιπλέον, εφικτή θα ήταν η σύνδεση ενός δεύτερου χειριστηρίου για τη δημιουργία ανταγωνιστικού παιχνιδιού από δυο παίκτες. Τέτοιες προσθήκες θα

μετέτρεπαν την εφαρμογή από μια βασική υλοποίηση Tetris σε μια πιο ολοκληρωμένη εμπειρία.

Τέλος, θα μπορούσε να διερευνηθεί η υποστήριξη υψηλότερων αναλύσεων ή διαφορετικών διεπαφών εξόδου, καθώς και η περαιτέρω βελτιστοποίηση της χρήσης πόρων για την επίτευξη ακόμα πιο αποδοτικής σχεδίασης, ενώ παράλληλα θα μπορούσε να εξεταστεί η προσαρμογή της αρχιτεκτονικής του συστήματος με στόχο την υλοποίηση επιπλέον ρετρό παιχνιδιών, αξιοποιώντας την υπάρχουσα υποδομή ως κοινό πυρήνα για πολλαπλές εφαρμογές πραγματικού χρόνου.

Συνολικά, οι δυνατότητες αυτές αναδεικνύουν ότι το παρόν έργο δεν περιορίζεται μόνο σε μια λειτουργική υλοποίηση παιχνιδιού, αλλά μπορεί να αποτελέσει τόσο βάση για την ανάπτυξη πιο σύνθετων και τεχνολογικά προηγμένων εφαρμογών στο μέλλον όσο και ερευνητική πλατφόρμα για τη μελέτη αρχιτεκτονικών προσεγγίσεων, τεχνικών βελτιστοποίησης και σχεδίασης ψηφιακών συστημάτων πραγματικού χρόνου σε περιβάλλον FPGA.

Βιβλιογραφία

[1] A. Zibell, "What are FPGAs and how do they work," Julius-Maximilians-Universität Würzburg, 2014. Available: https://indico.cern.ch/event/283113/contributions/1632265/attachments/522019/720041/Zibell_How_FPGAs_work.pdf. Accessed: 22-Nov-2025.

[2] "Hardware description language," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/digital-logic/hardware-description-language/. Accessed: 22-Nov-2025.

[3] "A Brief History of Tetris," AtariHQ. [Online]. Available: https://www.atarihq.com/tsr/special/tetrishist.html. Accessed: 22-Nov-2025.

[4] "Tetris Guideline," Tetris Wiki, 2021. [Online]. Available: https://tetris.wiki/Tetris_Guideline. Accessed: 22-Nov-2025.

[5] "Arduino Beginner's Guide," Simplilearn. [Online]. Available: https://www.simplilearn.com/tutorials/programming-tutorial/arduino-beginners-guide. Accessed: 22-Nov-2025.

[6] "Tetris style game on an 8x8 LED," Arduino Project Hub. Available: https://projecthub.arduino.cc/candystan/8x8-tetris-e57938. Accessed: 22-Nov-2025.

[7] "Tetris Clone With OLED SSD1306 (I2C) For Arduino Nano / Uno," Arduino Project Hub. Available: https://projecthub.arduino.cc/BADFEED/tetris-clone-with-oled-ssd1306-i2c-for-arduino-nano-uno-ef8a8a. Accessed: 22-Nov-2025.

[8] "Big and glowy tetris via Arduino," Duino4Projects. Available: https://duino4projects.com/big-and-glowy-tetris-via-arduino. Accessed: 22-Nov-2025.

[9] "LX' Arduino Tetris," Arduino Project Hub. Available: https://projecthub.arduino.cc/RomanSixty/lx-arduino-tetris-708606. Accessed: 22-Nov-2025.

[10] "Colorful Arduino Tetris Game - WS2812B LED Matrix Tutorial," Arduino Project Hub. Available: https://projecthub.arduino.cc/mircemk/colorful-arduino-tetris-game-ws2812b-led-matrix-tutorial-416706. Accessed: 22-Nov-2025.

[11] "Raspberry Pi," Webopedia. [Online]. Available: https://www.webopedia.com/definitions/raspberry-pi. Accessed: 22-Nov-2025.

[12] J. Heller and N. Abramson, "PyTetris project," Cornell ECE Courses. Available: https://courses.ece.cornell.edu/ece5990/ECE5725_Spring2024_Projects/02%20Monday%20May%2013/06%20Tetris/Monday_Group6_JasonHellerNoahAbramson/index.html. Accessed: 22-Nov-2025.

[13] "Pieces of Pi," PiecesOfPi.co.uk. Available: https://piecesofpi.co.uk/tetris-on-leds. Accessed: 22-Nov-2025.

[14] R. Linden and P. Slaats, "LED Tetris project," Harvey Mudd College. Available: https://pages.hmc.edu/harris/class/e155/projects16/Linden_Slaats.pdf. Accessed: 22-Nov-2025.

[15] J. Yang, "Tetris Ever," Snapcraft. Available: https://snapcraft.io/install/tetris-ever/raspbian. Accessed: 22-Nov-2025.

[16] "C++ Tetris Game using raylib," GitHub. Available: https://github.com/educ8s/Cpp-Tetris-Game-with-raylib. Accessed: 22-Nov-2025.

[17] "Tetris AI," GitHub. Available: https://github.com/ChesterHuynh/tetrisAI. Accessed: 22-Nov-2025.

[18] "Gym-simple tetris," GitHub. Available: https://github.com/tristanrussell/gym-simpletetris. Accessed: 22-Nov-2025.

[19] "Python-Tetris," GitHub. Available: https://gist.github.com/WyattJia/5e9e18122a69e964f06979220e513641. Accessed: 22-Nov-2025.

[20] S. Fakunle and A. Alfaraj, "Tetris Video Game via FPGA," ResearchGate, 2022. Available: https://www.researchgate.net/publication/373653441_Tetris_Video_Game_via_FPGA. Accessed: 22-Nov-2025.

[21] S. N. Tural Polat, "A fully embedded tetris game application in VHDL," 2020. Available: https://dergipark.org.tr/en/pub/ngumuh/issue/52170/522790. Accessed: 22-Nov-2025.

[22] O. R. G. Santiago, "BRNO UNIVERSITY OF TECHNOLOGY," 2025. Available: https://dspace.vut.cz/items/4b063454-b741-4da9-8e99-f47e6776d1f4. Accessed: 22-Nov-2025.

[23] A. Jiwarajka, "Parallel Tetris project," Cornell ECE Courses, 2008. Available: https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2008/aj77/aj77/finalreport.html. Accessed: 22-Nov-2025.

[24] Lavingiasa, "FPGA Tetris (Verilog)," GitHub, 2015. Available: https://github.com/lavingiasa/verilogTetris. Accessed: 22-Nov-2025.

[25] J. Schneider, "Field programmable gate arrays (FPGAs) vs. microcontrollers: What's the difference?" IBM. Available: https://www.ibm.com/think/topics/fpga-vs-microcontroller. Accessed: 22-Nov-2025.

[26] R. Rao, "FPGA vs Microcontroller Understanding the Key Differences and Use Cases," Wevolver, 2025. Available: https://www.wevolver.com/article/fpga-vs-microcontroller. Accessed: 22-Nov-2025.

[27] HardwareBee, "FPGA vs. Microcontroller, what to choose?" Available: https://hardwarebee.com/fpga-vs-microcontroller-what-to-choose. Accessed: 22-Nov-2025.

[28] J. Hopkins, "Comparing FPGA vs Microcontroller – Which is Best for Your Needs?" TotalPhase, 2024. Available: https://www.totalphase.com/blog/2021/04/comparing-fpga-vs-microcontroller/. Accessed: 22-Nov-2025.

[29] P. Gupta, "FPGA vs. Microcontroller: Making the Right Choice," FPGA Insights, 2024. Available: https://fpgainsights.com/fpga/fpga-vs-microcontroller/. Accessed: 22-Nov-2025.

[30] "Why are FPGAs faster than CPU?," Reflex ces, 2024. Available: https://www.reflexces.com/newsroom/why-are-fpgas-faster-than-cpu. Accessed: 22-Nov-2025.

[31] "Why use FPGA instead of CPU?," Reflex ces, 2024. Available: https://www.reflexces.com/newsroom/why-use-fpga-instead-of-cpu. Accessed: 22-Nov-2025.

[32] "FPGAs vs processor," 2024. Available: https://logicmadness.com/fpga-vs-processor. Accessed: 22-Nov-2025.

[33] "Where is FPGA stronger than traditional CPU," 2020. Available: https://www.fpgakey.com/technology/details/where-is-fpga-stronger-than-traditional-cpu. Accessed: 22-Nov-2025.

[34] M. Arucu and T. Iliev, "Performance Evaluation of FPGA, GPU, and CPU in FIR Filter Implementation for Semiconductor-Based Systems," MDPI, 2025. Available: https://www.mdpi.com/2079-9268/15/3/40. Accessed: 22-Nov-2025.

[35] "FPGA Pong," 2020. Available: https://projectf.io/posts/fpga-pong/. Accessed: 22-Nov-2025.

[36] "Snake Game on FPGA in Verilog," 2017. Available: https://www.slideshare.net/slideshow/snake-game-on-fpga-in-verilog/81338085. Accessed: 22-Nov-2025.

[37] "A Strategical Approach for Implementing Digital Games on FPGA- Snake game, Spaceman game, Pacman game, Tic-Tac-Toe and Battle tanker games," 2021. Available: https://turcomat.org/index.php/turkbilmat/article/download/5037/4212/9359. Accessed: 22-Nov-2025.

[38] "Space Invaders FPGA Game," 2017. Available: https://hackaday.io/project/19378-space-invaders-fpga-game. Accessed: 22-Nov-2025.

[39] "Two Player Space Invaders Via FPGAs," 2012. Available: https://hackaday.com/2012/01/03/two-player-space-invaders-via-fpgas/. Accessed: 22-Nov-2025.

[40] "VerilogBoy - GameBoy on FPGA," 2019. Available: https://hackaday.io/project/57660-verilogboy-gameboy-on-fpga. Accessed: 22-Nov-2025.

[41] "Game Boy Recreated In Verilog," 2019. Available: https://hackaday.com/2019/03/23/game-boy-recreated-in-verilog/. Accessed: 22-Nov-2025.

[42] C. "Max" Maxfield, The Design Warrior's Guide to FPGAs, 2004. [Online]. Available: https://blog.aku.edu.tr/ismailkoyuncu/files/2017/04/01_ebook.pdf. Accessed: 24-Nov-2025.

[43] "FPGA," Wikipedia. [Online]. Available: https://el.wikipedia.org/wiki/FPGA. Accessed: 24-Nov-2025.

[44] "FPGA: Detailed Guide – Structure, Working Principle, Features," IC Components. [Online]. Available: https://www.ic-components.gr/blog/fpga-detailed-guide-structure,working-principle,features.jsp#10.%20Different%20Applications%20of%20FPGA. Accessed: 24-Nov-2025.

[45] "Automatic FPGA-based Hardware Accelerator Design: A Case Study with Image Processing Applications," EUDL. [Online]. Available: https://eudl.eu/doi/10.4108/eai.12-5-2020.164497. Accessed: 24-Nov-2025.

[46] "Top 10 Emerging Applications of FPGA Technology in 2025," Interwiser Global. [Online]. Available: https://www.interwiser-global.com/news/top-10-emerging-applications-of-fpga-technology-in-2025.html. Accessed: 24-Nov-2025.

[47] "FPGA fundamentals: Architecture, Design, & Applications," DIY Usthad, 2023. [Online]. Available: https://diyusthad.com/2023/08/fpga-fundamentals.html. Accessed: 24-Nov-2025.

[48] "What is a field programmable gate array (FPGA)?," IBM. [Online]. Available: https://www.ibm.com/think/topics/field-programmable-gate-arrays. Accessed: 24-Nov-2025.

[49] A. Zibell, "What are FPGAs and how do they work," Julius-Maximilians-Universität Würzburg, 2014. [Online]. Available: https://indico.cern.ch/event/283113/contributions/1632265/attachments/522019/720041/Zibell_How_FPGAs_work.pdf. Accessed: 24-Nov-2025.

[50] H. Zheng, "FPGA Architecture Overview," University of South Florida. [Online]. Available: https://cse.usf.edu/~haozheng/teach/cda4253/doc/fpga-arch-overview.pdf. Accessed: 27-Nov-2025.

[51] "Ενσωματωμένα Συστήματα – Βασική Αρχιτεκτονική FPGA," Kallipos. [Online]. Available: http://repfiles.kallipos.gr/html_books/1285/bookES.html#%CE%B2%CE%B1%CF%83%CE%B9%CE%BA%CE%AE-

%CE%B1%CF%81%CF%87%CE%B9%CF%84%CE%B5%CE%BA%CF%84%CE%BF%CE%BD%CE%B9%CE%BA%CE%AE-fpga. Accessed: 24-Nov-2025.

[52] “DE23 Lite e-Paper,” Terasic. [Online]. Available: https://mail.terasic.com.tw/epaper/2025/DE23lite.html. Accessed: 27-Nov-2025.

[53] “DE23 Lite FPGA Development Kit,” Terasic. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=44&No=1383&PartNo=2#contents. Accessed: 27-Nov-2025.

[54] “Hello FPGA – Design Flow,” Microchip Technology. [Online]. Available: https://developerhelp.microchip.com/xwiki/bin/view/products/fpga/hello-fpga/design-flow. Accessed: 27-Nov-2025.

[55] “FPGA Design Flow in VHDL Programming Language,” PiEmbSysTech. [Online]. Available: https://piembsystech.com/fpga-design-flow-in-vhdl-programming-language. Accessed: 27-Nov-2025.

[56] “FPGA Design: A Complete Guide,” Vemeko. [Online]. Available: https://www.vemeko.com/blog/fpga-design-complete-guide.html. Accessed: 27-Nov-2025.

[57] “Hardware description language,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Hardware_description_language. Accessed: 27-Nov-2025.

[58] “Evolution of HDLs Part 1: The Birth,” ChipInsights (Substack). [Online]. Available: https://chipinsights.substack.com/p/evolution-of-hdls-part-1-the-birth. Accessed: 27-Nov-2025.

[59] “Hardware Description Languages — VHDL vs Verilog and Their Functional Uses,” Cadence PCB & Design Resources. [Online]. Available: https://resources.pcb.cadence.com/blog/2020-hardware-description-languages-vhdl-vs-verilog-and-their-functional-uses. Accessed: 27-Nov-2025.

[60] “VHDL vs Verilog vs SystemVerilog — What to Learn First,” Medium (Tech Asthetic). [Online]. Available: https://medium.com/@techAsthetic/vhdl-vs-verilog-vs-system-verilog-what-to-learn-first-7ff9e247ffd9. Accessed: 27-Nov-2025.

[61] “High-Level Synthesis (HLS) Overview,” Medium (Sujeeth Selvam). [Online]. Available: https://medium.com/@sujeeth.selvam/high-level-synthesis-hls-overview-dffdce327d7d. Accessed: 27-Nov-2025.

[62] “Introduction to Quartus® Prime Pro Edition,” Intel® FPGA Documentation. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683463/25-3/introduction-to.html. Accessed: 27-Nov-2025.

[63] “ModelSim® and QuestaSim® Guidelines,” Intel® FPGA Documentation. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683080/18-1/and-questasim.html. Accessed: 27-Nov-2025.

[64] “Joystick History,” Scribd. [Online]. Available: https://www.scribd.com/document/811360033/Joystick-History. Accessed: 27-Nov-2025.

[65] “An Ode to Joysticks,” Medium (Design Bootcamp). [Online]. Available: https://medium.com/design-bootcamp/an-ode-to-joysticks-b6c9a556f9c7. Accessed: 27-Nov-2025.

[66] “Atari CX40 joystick,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Atari_CX40_joystick. Accessed: 27-Nov-2025.

[67] “Robotron: 2084,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Robotron:_2084. Accessed: 27-Nov-2025.

[68] “D-pad,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/D-pad. Accessed: 27-Nov-2025.

[69] “Namco Arcade Stick,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Namco_Arcade_Stick. Accessed: 27-Nov-2025.

[70] “X-Arcade,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/X-Arcade. Accessed: 27-Nov-2025.

[71] Κ. Χατζηβασιλείου and Ν. Κανελλόπουλος, “Ο Νίκος Ανερούσης στο Retro Planet,” Retro Planet Magazine (blog), 20 May 2023. [Online]. Available: https://retroplanetmagazine.blogspot.com/2023/05/retro-planet.html. Accessed: 27-Nov-2025.

[72] “Tech Tips: Arcade Controllers,” PrimeTime Amusements. [Online]. Available: https://primetimeamusements.com/tech-tips-controllers/. Accessed: 27-Nov-2025.

[73] “Arcade controller,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Arcade_controller. Accessed: 27-Nov-2025.

[74] “Instructions — FPGA Tetris (Hackaday Project),” Hackaday. [Online]. Available: https://hackaday.io/project/181309/instructions. Accessed: 27-Nov-2025.

[75] “Arcade Controls Wiring Instructions,” The Geek Pub. [Online]. Available: https://www.thegeekpub.com/arcade-controls-wiring-instructions. Accessed: 27-Nov-2025.

[76] “What Are Pull-Up and Pull-Down Resistors?,” Robu.in. [Online]. Available: https://robu.in/what-are-pull-up-and-pull-down-resistors/. Accessed: 27-Nov-2025.

[77] “Analogue to Digital Converter (ADC),” Electronics-Tutorials.ws. [Online]. Available:https://www.electronics-tutorials.ws/combination/analogue-to-digital-converter.html. Accessed: 27-Nov-2025.

[78] “Switch Bounce: How to Deal with It,” All About Circuits. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it. Accessed: 27-Nov-2025.

[79] “What Is Switch Bounce & How to Implement Debounce,” Pico Technology. [Online]. Available: https://www.picotech.com/library/articles/blog/what-is-switch-bounce-how-to-implement-debounce. Accessed: 27-Nov-2025.

[80] “The Engineer’s Guide to Switch Contact Debounce Techniques,” AllPCB / AllelectroHub. [Online]. Available: https://www.allpcb.com/allelectrohub/the-engineers-guide-to-switch-contact-debounce-techniques. Accessed: 27-Nov-2025.

[81] Texas Instruments, “SCEA046B – Switch Bounce and Contact Debounce Techniques,” Texas Instruments Application Report, [Online]. Available: https://www.ti.com/lit/ab/scea046b/scea046b.pdf?ts=1766087748205. Accessed: 27-Nov-2025.

[82] “USB human interface device class,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/USB_human_interface_device_class. Accessed: 27-Nov-2025.

[83] “MAME User Interface: Analog Input Settings,” MAMEDev Documentation. [Online]. Available: https://docs.mamedev.org/usingmame/ui.html#analog-input-settings. Accessed: 27-Nov-2025.

[84] “High-Definition Multimedia Interface,” Wikipedia (Greek). [Online]. Available: https://el.wikipedia.org/wiki/High-Definition_Multimedia_Interface. Accessed: 27-Nov-2025.

[85] “How HDMI Works,” HowStuffWorks. [Online]. Available: https://electronics.howstuffworks.com/hdmi.htm. Accessed: 27-Nov-2025.

[86] “HDMI,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/HDMI. Accessed: 27-Nov-2025.

[87] “HDMI Specifications and Support FAQs,” ASUS. [Online]. Available: https://www.asus.com/gr/support/faq/1004820/. Accessed: 27-Nov-2025.

[88] “What Is the Longest HDMI Cable Length?,” Maplin Expert Advice. [Online]. Available: https://www.maplin.co.uk/blogs/expert-advice/what-is-the-longest-hdmi-cable-length. Accessed: 27-Nov-2025.

[89] “The Importance of HDMI Cable Length: Does It Affect Quality?,” DawnDN. [Online]. Available: https://www.dawndn.com/the-importance-of-hdmi-cable-length-does-it-affect-quality.html. Accessed: 27-Nov-2025.

[90] “Νέα εποχή για το HDMI με την έκδοση 2.2 και υποστήριξη 96 Gbps,” Insomnia.gr. [Online]. Available: https://www.insomnia.gr/articles/various/ces/nea-epohi-gia-to-hdmi-me-tin-ekdosi-22-kai-ypostiriksi-96gbps. Accessed: 27-Nov-2025.

[91] “HDMI Specification 1.3a,” Analog Devices (ez.analog.com). [Online]. Available: https://ez.analog.com/cfs-file/__key/telligent-evolution-components-attachments/00-317-00-00-00-05-21-37/HDMISpecification13a.pdf. Accessed: 28-Nov-2025.

[92] “HDMI,” MADPCB Glossary. [Online]. Available: https://madpcb.com/glossary/hdmi. Accessed: 28-Nov-2025.

[93] “HDMI Protocols: TMDS, CEC, DDC & FRL Explained,” ProDigitalWeb. [Online]. Available: https://www.prodigitalweb.com/hdmi-protocols-tmds-cec-ddc-frl-explained. Accessed: 28-Nov-2025.

[94] “Transition-minimized differential signaling,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Transition-minimized_differential_signaling. Accessed: 28-Nov-2025.

[95] “A Definitive Guide to HDMI® 2.1a,” AVPro Global. [Online]. Available: https://www.avproglobal.com/blogs/news/a-definitive-guide-to-hdmi%C2%AE-2-1a. Accessed: 28-Nov-2025.

[96] “HDMI Protocol,” ProdigyTechno. [Online]. Available: https://www.prodigytechno.com/hdmi-protocol. Accessed: 28-Nov-2025.

[97] “HDMI – Communication channels,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/HDMI#Communication_channels. Accessed: 28-Nov-2025.

[98] “HDMI Explained,” Eaton. [Online]. Available: https://www.eaton.com/us/en-us/products/backup-power-ups-surge-it-power-distribution/backup-power-ups-it-power-distribution-resources/cpdi-vertical-marketing/hdmi-explained.html. Accessed: 28-Nov-2025.

[99] “Terasic FPGA Product Details,” Terasic. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=44&No=1383&PartNo=4#contents. Accessed: 28-Nov-2025.

[100] “The History of Tetris,” AtariHQ. [Online]. Available: http://www.atarihq.com/tsr/special/tetrishist.html. Accessed: 24-Nov-2025.

[101] “Exclusive interview with Alexey Pajitnov – creator of Tetris,” The Shortlisted. [Online]. Available: https://the-shortlisted.co.uk/alexey-pajitnov-tetris-interview/. Accessed: 24-Nov-2025.

[102] V. Gerasimov, “Tetris (history and technical details),” [Online]. Available: http://vadim.oversigma.com/Tetris.htm. Accessed: 24-Nov-2025.

[103] “Playing Tetris reduces unwanted memories,” ScienceDaily, 2009. [Online]. Available: https://www.sciencedaily.com/releases/2009/09/090901082851.htm. Accessed: 24-Nov-2025.

[104] M. A. Jabbar, et al., “A comprehensive review of Tetris-based research,” Entertainment Computing, vol. 38, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389041721000991. Accessed: 24-Nov-2025.

[105] E. Skorka, et al., “Effects of Tetris on the brain: A functional MRI study,” PMC, 2018. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC5836929/. Accessed: 24-Nov-2025.

[106] “Tetris,” Wikipedia. [Online]. Available: https://el.wikipedia.org/wiki/Tetris. Accessed: 24-Nov-2025.

[107] "Tetris The GrandMaster Series," HardDrop. [Online]. Available: https://harddrop.com/wiki/Tetris_The_GrandMaster_Series. Accessed: 22-Nov-2025.

[108] "Tetris (NES)," TetrisWiki. [Online]. Available: https://tetris.wiki/Tetris_%28NES%29. Accessed: 22-Nov-2025.

[109] “Four-tris,” Hard Drop Tetris Wiki. [Online]. Available: https://harddrop.com/wiki/Four-tris. Accessed: 22-Nov-2025.

[110] “Taming Tetris Timing,” TetrisConcept. [Online]. Available: https://tetrisconcept.net/threads/taming-tetris-timing.424/. Accessed: 22-Nov-2025.

[111] “Heboris Unofficial Expansion,” Tetris Fandom Wiki. [Online]. Available: https://tetris.fandom.com/wiki/Heboris_Unofficial_Expansion. Accessed: 22-Nov-2025.

[112] “Tetris Design Guideline,” 2009 Studylib. [Online]. Available: https://studylib.net/doc/27262492/2009-tetris-design-guideline. Accessed: 22-Nov-2025.

[113] “Tetris – Tetromino Game Engine + Terminal Application,” GitHub. [Online]. Available: https://github.com/Strophox/tetrs. Accessed: 22-Nov-2025.

[114] “Tetris The Grand Master 4 – Absolute Eye,” TetrisWiki. [Online]. Available: https://tetris.wiki/Tetris_The_Grand_Master_4_Absolute_Eye. Accessed: 22-Nov-2025

[115] “TGM Guide,” Tetris Wiki. [Online]. Available: https://tetris.wiki/TGM_Guide. Accessed: 24-Nov-2025.

[116] A. Kokkinis and K. Siozos., “Fast resource estimation of FPGA-based MLP Accelerators for TinyML Applications,” Electronics, vol. 14, no. 2, art. 247, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/2/247. Accessed: 26-Nov-2025.

[117] Y. Wang, H. Zhao and J. Zhao, “AFHRE: An Accurate and Fast Hardware Resources Estimation Method for Convolutional Accelerator with Systolic Array Structure on FPGA,” Electronics, vol. 14, no. 1, art. 168, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/1/168. Accessed: 24-Nov-2025.

[118] “Quartus Prime Design Software,” Altera (Intel). [Online]. Available: https://www.altera.com/products/development-tools/quartus. Accessed: 26-Nov-2025.

[119]“ModelSim,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/ModelSim. Accessed: 26-Nov-2025.

[120] “Vivado Design Suite,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Vivado. Accessed: 26-Nov-2025.

[121] “tetris-vhdl,” GitHub repository, [Online]. Available: https://github.com/primiano/tetris-vhdl. Accessed: 24-Nov-2025.

[122] “DE0-Cyclone III FPGA Development Board,” Terasic, [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=183&No=364&PartNo=2#contents. Accessed: 24-Nov-2025.

[123] “Tetris 2-Players Report,” Lund University – EDA385 course, [Online]. Available: https://fileadmin.cs.lth.se/cs/Education/EDA385/HT12/student_doc/final/Tetris2Players/report.pdf. Accessed: 24-Nov-2025.

[124] “Terasic,” Terasic. [Online]. Available: https://www.terasic.com.tw/en/. Accessed: 26-Nov-2025.

[125] “Digilent,” Digilent. [Online]. Available: https://digilent.com/. Accessed: 26-Nov-2025.

[126] “TERASIC DE10-Lite Development and Education Board,” Terasic. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1021. Accessed: 24-Nov-2025.

[127] “TERASIC DE10-Nano Development and Education Board,” Terasic. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1046. Accessed: 24-Nov-2025.

[128] “TERASIC DE23-Lite Development Kit,” Terasic. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=115&No=1383&PartNo=1#contents. Accessed: 24-Nov-2025.

[129] “Arty A7-100T Artix-7 FPGA Development Board,” Digilent. [Online]. Available: https://digilent.com/shop/arty-a7-100t-artix-7-fpga-development-board. Accessed: 24-Nov-2025.

Παράρτημα Α : Demonstration – HDMI_out_RTL of DE-23 Lite

```
module golden_top(

      ///////// CLOCK /////////
      input              CLOCK0_50,
      input              CLOCK1_50,

      ///////// KEY /////////
      input    [ 3: 0]   KEY, //BUTTON is Low-Active

      ///////// SW /////////
      input    [ 9: 0]   SW,

      ///////// LED /////////
      output   [ 9: 0]   LEDR, //LED is Low-Active

      ///////// Seg7 /////////
      output   [ 6: 0]   HEX0,
      output   [ 6: 0]   HEX1,
      output   [ 6: 0]   HEX2,
      output   [ 6: 0]   HEX3,
      output   [ 6: 0]   HEX4,
      output   [ 6: 0]   HEX5,

      ///////// SDRAM /////////
      output             DRAM_CLK,
      output             DRAM_CKE,
      output   [12: 0]   DRAM_ADDR,
      output   [ 1: 0]   DRAM_BA,
      inout    [31: 0]   DRAM_DQ,
      output             DRAM_CS_n,
      output             DRAM_WE_n,
      output             DRAM_CAS_n,
      output             DRAM_RAS_n,
      output   [ 3: 0]   DRAM_DQM,

      ///////// HDMI /////////
      inout              HDMI_LRCLK,
      inout              HDMI_MCLK,
      inout              HDMI_SCLK,
      output             HDMI_TX_CLK,
      output             HDMI_TX_HS,
      output             HDMI_TX_VS,
      output   [23: 0]   HDMI_TX_D,
      output             HDMI_TX_DE,
      input              HDMI_TX_INT,
      inout              HDMI_I2S0,


      ///////// I2C for HDMI and ADC /////////
      inout              FPGA_I2C_SCL,
      inout              FPGA_I2C_SDA,

      ///////// UART /////////
      output             FPGA_UART_TX,
      input              FPGA_UART_RX,
```

```
      ///////// GPIO /////////
      inout    [35: 0]   GPIO_D

);
//----LED off device
assign LEDR[9:7] = 3'h7;

//----HEX off device
assign HEX0 =7'h7f;
assign HEX1 =7'h7f;
assign HEX2 =7'h7f;
assign HEX3 =7'h7f;
assign HEX4 =7'h7f;
assign HEX5 =7'h7f;

//----SDRAM off
assign DRAM_DQ   =32'hzzzz_zzzzz;
assign DRAM_CS_n =1'b1;
assign DRAM_WE_n =1'b1;
assign DRAM_CAS_n=1'b1;
assign DRAM_RAS_n=1'b1;
assign DRAM_DQM  =4'b1111;

//=======================================================
//  wire define
//=======================================================
wire AUD_CTRL_CLK ;
wire AUD_BCLK   ;
wire DAC_DACDAT ;
wire AUD_DACLRCK;
wire reset_n;
wire  V_locked ,A_locked ;
wire  pll_12M288;
wire  SYSTEM_50MHZ;
wire [7:0] vpg_r;
wire [7:0] vpg_g;
wire [7:0] vpg_b;
wire HDMI_READY ;
wire HDMI_I2C_SCLK ;
//=======================================================
//  Structural coding
//=======================================================
//--RESET
assign reset_n = ~ninit_done;
//---INIT RESET
wire ninit_done;
      ResetRelease ResetRelease_inst (
            .ninit_done (ninit_done)
      );
//-- PLL LOCK
assign LEDR[5] = ~V_locked    ;
assign LEDR[6] = ~A_locked    ;

//-- HDMI SET READY
assign LEDR[4] = ~HDMI_READY   ;

//-----heart ----
CLOCKMEM  ckK0( .RESET_n(1), .CLK(HDMI_TX_CLK    ) ,.CLK_FREQ (
148_500_000 )  ,.CK_1HZ  (LEDR[0] ) ) ;
CLOCKMEM  ckK1( .RESET_n(1), .CLK(AUD_DACLRCK    ) ,.CLK_FREQ (
```

```
48_000 )   ,.CK_1HZ  (LEDR[1] ) ) ;
CLOCKMEM  ckK2( .RESET_n(1), .CLK(SYSTEM_50MHZ   ) ,.CLK_FREQ (
50_000_000 )  ,.CK_1HZ  (LEDR[2] ) ) ;
CLOCKMEM  ckK3( .RESET_n(1), .CLK(AUD_CTRL_CLK   ) ,.CLK_FREQ (
12_280_000 )  ,.CK_1HZ  (LEDR[3] ) ) ;

//---AV PLL

VEDIO_PLL u_VEDIO_PLL(
    .refclk       (CLOCK0_50     ),
    .outclk_0     (vpg_pclk      ),//148.5MHZ
    .locked       (V_locked      ),
       .rst          (~reset_n      )
);

AUDIO_PLL u_AUDIO_PLL(
    .refclk       (CLOCK1_50     ),
    .outclk_0     (pll_12M288    ),//12.288136Mhz
    .locked       (A_locked     ),
       .rst          (~reset_n      )
);


assign SYSTEM_50MHZ = CLOCK0_50;//

assign HDMI_TX_D[23:16] =     vpg_r;
assign HDMI_TX_D[15:8]  =     vpg_g;
assign HDMI_TX_D[7:0]   =     vpg_b;
//--HDMI timing generater & //pattern generator
vpg   u_vpg (
      .vpg_pclk    (vpg_pclk   ),//vedio clock input
      .reset_n     (reset_n    ),
      .vpg_de      (HDMI_TX_DE ),
      .vpg_hs      (HDMI_TX_HS ),
      .vpg_vs      (HDMI_TX_VS ),
      .vpg_pclk_out(HDMI_TX_CLK),
      .vpg_r       (vpg_r),
      .vpg_g       (vpg_g),
      .vpg_b       (vpg_b)
      );

//--  HDMI I2C    SETTING
I2C_HDMI_Config u_I2C_HDMI_Config (
                    .iCLK       (SYSTEM_50MHZ  ),
                    .iRST_N     (reset_n & KEY[0]),
                    .I2C_SCLK   (FPGA_I2C_SCL  ),
                    .I2C_SDAT   (FPGA_I2C_SDA  ),
                    .HDMI_TX_INT(HDMI_TX_INT   ),
                    .READY      (HDMI_READY    )
                  );


//---Audio Master L/RCLK , BCK ,DATA  generater

assign AUD_CTRL_CLK =pll_12M288;

AUDIO_DAC   u_AUDIO_DAC (     //    Audio Side
                              .oAUD_BCK       (AUD_BCLK       ),
                              .oAUD_DATA      (DAC_DACDAT     ),
                              .oAUD_LRCK      (AUD_DACLRCK    ),
                              //    Control Signals
```

```
                                .iSrc_Select    (2'b00            ),
                          .iCLK_18_4      (AUD_CTRL_CLK
),//12.288000MHz --12.288136Mhz
                                .iRST_N         (HDMI_READY      )
                                );

// HDMI I2S out
assign HDMI_MCLK  = AUD_CTRL_CLK;
assign HDMI_SCLK  = AUD_BCLK     ;
assign HDMI_LRCLK = AUD_DACLRCK  ;

//--key-in KEY3 ,HDMI out 1k sin-tone
assign HDMI_I2S0 = KEY[3]? 0 : DAC_DACDAT ;

endmodule
```

```
module vga_generator(
  input               clk,
  input               reset_n,
  input       [11:0] h_total,
  input       [11:0] h_sync,
  input       [11:0] h_start,
  input       [11:0] h_end,
  input       [11:0] v_total,
  input       [11:0] v_sync,
  input       [11:0] v_start,
  input       [11:0] v_end,
  input       [11:0] v_active_14,
  input       [11:0] v_active_24,
  input       [11:0] v_active_34,
  output  reg         vga_hs,
  output  reg         vga_vs,
  output  reg           vga_de,
  output  reg [7:0]   vga_r,
  output  reg [7:0]   vga_g,
  output  reg [7:0]   vga_b
);

//=======================================================
//  Signal declarations
//=======================================================
reg                 [11:0]    h_count;
reg                  [7:0]    pixel_x;
reg                 [11:0]    v_count;
reg               h_act;
reg               h_act_d;
reg               v_act;
reg               v_act_d;
reg               pre_vga_de;
wire              h_max, hs_end, hr_start, hr_end;
wire              v_max, vs_end, vr_start, vr_end;
wire              v_act_14, v_act_24, v_act_34;
reg               boarder;
reg        [3:0]  color_mode;

//=======================================================
//  Structural coding
//=======================================================
```

```
assign h_max = h_count == h_total;
assign hs_end = h_count >= h_sync;
assign hr_start = h_count == h_start;
assign hr_end = h_count == h_end;
assign v_max = v_count == v_total;
assign vs_end = v_count >= v_sync;
assign vr_start = v_count == v_start;
assign vr_end = v_count == v_end;
assign v_act_14 = v_count == v_active_14;
assign v_act_24 = v_count == v_active_24;
assign v_act_34 = v_count == v_active_34;

//horizontal control signals
always @ (posedge clk or negedge reset_n)
      if (!reset_n)
      begin
    h_act_d   <=  1'b0;
            h_count           <=    12'b0;
            pixel_x   <=  8'b0;
            vga_hs            <=    1'b1;
            h_act     <=      1'b0;
      end
      else
      begin
    h_act_d   <=  h_act;

    if (h_max)
              h_count   <= 12'b0;
            else
              h_count   <= h_count + 12'b1;

    if (h_act_d)
              pixel_x   <=    pixel_x + 8'b1;
            else
              pixel_x   <=  8'b0;

            if (hs_end && !h_max)
              vga_hs    <=    1'b1;
            else
              vga_hs    <= 1'b0;

            if (hr_start)
              h_act       <=  1'b1;
            else if (hr_end)
              h_act       <=  1'b0;
      end

//vertical control signals
always@(posedge clk or negedge reset_n)
      if(!reset_n)
      begin
            v_act_d       <=  1'b0;
            v_count           <=    12'b0;
            vga_vs            <=    1'b1;
            v_act     <=      1'b0;
            color_mode<=  4'b0;
      end
      else
      begin
            if (h_max)
            begin
```

```
            v_act_d       <=  v_act;

              if (v_max)
                v_count <=    12'b0;
              else
                v_count <=    v_count + 12'b1;

              if (vs_end && !v_max)
                vga_vs  <=    1'b1;
              else
                vga_vs  <=    1'b0;

            if (vr_start)
              v_act <=  1'b1;
              else if (vr_end)
                v_act <=      1'b0;

            if (vr_start)
              color_mode[0] <=      1'b1;
              else if (v_act_14)
                color_mode[0] <=    1'b0;

            if (v_act_14)
              color_mode[1] <=      1'b1;
              else if (v_act_24)
                color_mode[1] <=    1'b0;

            if (v_act_24)
              color_mode[2] <=      1'b1;
              else if (v_act_34)
                color_mode[2] <=    1'b0;

            if (v_act_34)
              color_mode[3] <=      1'b1;
              else if (vr_end)
                color_mode[3] <=    1'b0;
            end
      end

//pattern generator and display enable
always @(posedge clk or negedge reset_n)
begin
      if (!reset_n)
      begin
    vga_de <= 1'b0;
    pre_vga_de <= 1'b0;
    boarder <= 1'b0;
  end
      else
      begin
    vga_de <= pre_vga_de;
    pre_vga_de <= v_act && h_act;

    if ((!h_act_d&&h_act) || hr_end || (!v_act_d&&v_act) || vr_end)
      boarder <= 1'b1;
    else
      boarder <= 1'b0;

            if (boarder)
                  {vga_r, vga_g, vga_b} <= {8'hFF,8'hFF,8'hFF};
            else
```

```
              case (color_mode)
            4'b0001 : {vga_r, vga_g, vga_b} <= {pixel_x,8'h00,8'h00};
            4'b0010 : {vga_r, vga_g, vga_b} <= {8'h00,pixel_x,8'h00};
            4'b0100 : {vga_r, vga_g, vga_b} <= {8'h00,8'h00,pixel_x};
            4'b1000 : {vga_r, vga_g, vga_b} <=
{pixel_x,pixel_x,pixel_x};
            default  : {vga_r, vga_g, vga_b} <= {8'h00,8'h00,8'h00};
      endcase
      end
end

endmodule
```

```
module vpg(
      clk_50,
      reset_n,
      vpg_pclk_out,
      vpg_pclk,
      vpg_de,
      vpg_hs,
      vpg_vs,
      vpg_r,
      vpg_g,
      vpg_b
);

input                   clk_50;
input                   reset_n;
input             vpg_pclk;
output                     vpg_pclk_out;
output                     vpg_de;
output                     vpg_hs;
output                     vpg_vs;
output      [7:0]    vpg_r;
output      [7:0]    vpg_g;
output      [7:0]    vpg_b;

//=========================================================
//  Signal declarations
//=========================================================
//============= assign timing constant
wire  [11:0] h_total, h_sync, h_start, h_end;
wire  [11:0] v_total, v_sync, v_start, v_end;
wire  [11:0] v_active_14, v_active_24, v_active_34;


//=========================================================
//  Sub-module
//=========================================================
//=============== PLL reconfigure
//pll u_pll (
//  .refclk   (clk_50),
//  .rst      (!reset_n),
//  .outclk_0 (vpg_pclk)
//   );

assign vpg_pclk_out = vpg_pclk;//
//=============== pattern generator according to vga timing
vga_generator u_vga_generator (
  .clk(vpg_pclk),
```

```
  .reset_n(reset_n),
  .h_total(h_total),
  .h_sync (h_sync),
  .h_start(h_start),
  .h_end(h_end),
  .v_total(v_total),
  .v_sync(v_sync),
  .v_start(v_start),
  .v_end(v_end),
  .v_active_14(v_active_14),
  .v_active_24(v_active_24),
  .v_active_34(v_active_34),
  .vga_hs(vpg_hs),
  .vga_vs(vpg_vs),
  .vga_de(vpg_de),
  .vga_r(vpg_r),
  .vga_g(vpg_g),
  .vga_b(vpg_b) );

//=======================================================
//  Structural coding
//=======================================================
//============= assign timing constant
//h_total : total - 1
//h_sync : sync - 1
//h_start : sync + back porch - 1 - 2(delay)
//h_end : h_start + avtive

//v_total : total - 1
//v_sync : sync - 1
//v_start : sync + back porch - 1
//v_end : v_start + avtive
//v_active_14 : v_start + 1/4 avtive
//v_active_24 : v_start + 2/4 avtive
//v_active_34 : v_start + 3/4 avtive

//1280x720@60 74.25 MHZ pass
//assign {h_total, h_sync, h_start, h_end} = {12'd1649, 12'd39, 12'd257,
12'd1537};
//assign {v_total, v_sync, v_start, v_end} = {12'd749, 12'd4, 12'd24,
12'd744};
//assign {v_active_14, v_active_24, v_active_34} = {12'd204, 12'd384,
12'd564};

////640x480@60 25.175 MHZ
//assign {h_total, h_sync, h_start, h_end} = {12'd799, 12'd95, 12'd141,
12'd781};
//assign {v_total, v_sync, v_start, v_end} = {12'd524, 12'd1, 12'd34,
12'd514};
//assign {v_active_14, v_active_24, v_active_34} = {12'd154, 12'd274,
12'd394};

//720x480@60 27MHZ (VIC=3, 480P)
//assign {h_total, h_sync, h_start, h_end} = {12'd857, 12'd61, 12'd119,
12'd839};
//assign {v_total, v_sync, v_start, v_end} = {12'd524, 12'd5, 12'd35,
12'd515};
//assign {v_active_14, v_active_24, v_active_34} = {12'd155, 12'd275,
12'd395};

//1024x768@60 65MHZ (XGA)
```

```
//assign {h_total, h_sync, h_start, h_end} = {12'd1343, 12'd135,
12'd293, 12'd1317};
//assign {v_total, v_sync, v_start, v_end} = {12'd805, 12'd5, 12'd34,
12'd802};
//assign {v_active_14, v_active_24, v_active_34} = {12'd226, 12'd418,
12'd610};

//1280x1024@60   108MHZ (SXGA)
//assign {h_total, h_sync, h_start, h_end} = {12'd1687, 12'd111,
12'd357, 12'd1637};
//assign {v_total, v_sync, v_start, v_end} = {12'd1065, 12'd2, 12'd40,
12'd1064};
//assign {v_active_14, v_active_24, v_active_34} = {12'd296, 12'd552,
12'd808};

//1920x1080p60 148.5MHZ
assign {h_total, h_sync, h_start, h_end} = {12'd2199, 12'd43, 12'd189,
12'd2109};
assign {v_total, v_sync, v_start, v_end} = {12'd1124, 12'd4, 12'd40,
12'd1120};
assign {v_active_14, v_active_24, v_active_34} = {12'd310, 12'd580,
12'd850};

//1600x1200p60 162MHZ (VESA)
//assign {h_total, h_sync, h_start, h_end} = {12'd2159, 12'd191,
12'd493, 12'd2093};
//assign {v_total, v_sync, v_start, v_end} = {12'd1249, 12'd2, 12'd48,
12'd1248};
//assign {v_active_14, v_active_24, v_active_34} = {12'd348, 12'd648,
12'd948};

endmodule
```

Παράρτημα Β: HDMI Controller

```
module I2C_WRITE_WDATA  (
   input                       RESET_N ,
      input                    PT_CK,
      input                    GO,
      input       [15:0] REG_DATA,
      input       [7:0]  SLAVE_ADDRESS,
      input              SDAI,
      output reg         SDAO,
      output reg         SCLO,
      output reg         END_OK,

      //--for test
      output reg [7:0]   ST ,
      output reg [7:0]   CNT,
      output reg [7:0]   BYTE,
      output reg         ACK_OK,
   input       [7:0]  BYTE_NUM  // 4 : 4 byte
);

//===reg/wire
reg    [8:0]A ;
reg    [7:0]DELY ;

always @( negedge RESET_N or posedge  PT_CK )begin
if (!RESET_N  ) ST <=0;
else
        case (ST)
          0: begin  //start
                  SDAO   <=1;
               SCLO   <=1;
               ACK_OK <=0;
               CNT    <=0;
               END_OK <=1;
               BYTE   <=0;
               if (GO) ST  <=30 ; // inital

                end
          1: begin  //start
                  ST <=2 ;
                     { SDAO,  SCLO } <= 2'b01;
                        A <= {SLAVE_ADDRESS ,1'b1 };//WRITE COMMAND
                end
          2: begin  //start
                  ST <=3 ;
                     { SDAO,  SCLO } <= 2'b00;
                end

          3: begin
                  ST <=4 ;
                     { SDAO, A } <= { A ,1'b0 };
                end
          4: begin
                  ST <=5 ;
                     SCLO <= 1'b1 ;
                        CNT <= CNT +1 ;
                end
```

```
          5: begin
                      SCLO <= 1'b0 ;
                      if (CNT==9) begin
                              if ( BYTE == BYTE_NUM )  ST <= 6 ;
                                 else  begin
                                          CNT <=0 ;
                                          ST <= 2 ;
                                               if ( BYTE ==0 ) begin BYTE
<=1   ; A <= {REG_DATA[15:8] ,1'b1 }; end
                                          else if ( BYTE ==1 ) begin BYTE
<=2   ; A <= {REG_DATA[7:0] ,1'b1 }; end
                                                    end
                                 if (SDAI ) ACK_OK <=1 ;
                           end
                           else ST <= 2;
                  end

          6: begin          //stop
                   ST <=7 ;
                      { SDAO,  SCLO } <= 2'b00;
         end

          7: begin          //stop
                   ST <=8 ;
                      { SDAO,  SCLO } <= 2'b01;
         end
          8: begin          //stop
                   ST <=9 ;
                      { SDAO,  SCLO } <= 2'b11;

         end
            9:    begin
                   ST     <= 30;
                         SDAO   <=1;
               SCLO   <=1;
               CNT    <=0;
               END_OK <=1;
               BYTE   <=0;
                 end
            //--- END ---
               30: begin
            if (!GO) ST  <=31;
          end
               31: begin  //
                  END_OK<=0;
                        ACK_OK<=0;
                        ST    <=1;
                   end
        endcase
 end

endmodule
```

```
module I2C_HDMI_Config (      //    Host Side
    iCLK,
    iRST_N,
//    I2C Side
    I2C_SCLK,
    I2C_SDAT,
    HDMI_TX_INT,
    READY
 );
//    Host Side
input                   iCLK;
input                   iRST_N;
//    I2C Side
output                  I2C_SCLK;
inout                   I2C_SDAT;
input                   HDMI_TX_INT;
output READY ;

//    Internal Registers/Wires
reg   [15:0]      mI2C_CLK_DIV;
reg   [23:0]      mI2C_DATA;
reg                     mI2C_CTRL_CLK;
reg                     mI2C_GO;
wire                    mI2C_END;
wire                    mI2C_ACK;
reg   [15:0]      LUT_DATA;
reg   [5:0]       LUT_INDEX;
reg   [3:0]       mSetup_ST;
reg READY ;

//    Clock Setting
parameter   CLK_Freq    =     50000000;   //    50    MHz
parameter   I2C_Freq    =     20000;      //    20    KHz
//    LUT Data Number
parameter   LUT_SIZE    =     33;// 31;

////////////////////   I2C Control Clock ////////////////////////
always@(posedge iCLK or negedge iRST_N)
begin
      if(!iRST_N)
      begin
            mI2C_CTRL_CLK     <=    0;
            mI2C_CLK_DIV      <=    0;
      end
      else
      begin
            if( mI2C_CLK_DIV  < (CLK_Freq/I2C_Freq) )
                  mI2C_CLK_DIV      <=    mI2C_CLK_DIV+1;
            else
            begin
                  mI2C_CLK_DIV      <=    0;
                  mI2C_CTRL_CLK     <=    ~mI2C_CTRL_CLK;
            end
      end
end
//////////////////////////////////////////////////////////////////
I2C_Controller    u0    (     .CLOCK(mI2C_CTRL_CLK),  //    Controller
Work Clock
                                  .I2C_SCLK(I2C_SCLK),
      //    I2C CLOCK
                                  .I2C_SDAT(I2C_SDAT),
```

```
      //     I2C DATA
                                    .I2C_DATA(mI2C_DATA),
      //     DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
                                    .GO(mI2C_GO),
            //     GO transfor
                                    .END(mI2C_END),
      //     END transfor
                                    .ACK(mI2C_ACK),
      //     ACK
                                    .RESET(iRST_N)     );
////////////////////////////////////////////////////////////////////
//////////////////////  Config Control     ////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
begin
      if(!iRST_N)
      begin
         READY       <= 0;
            LUT_INDEX   <=     0;
            mSetup_ST   <=     0;
            mI2C_GO           <=     0;
      end
      else
      begin
            if(LUT_INDEX<LUT_SIZE)
            begin
            READY<=0;
                  case(mSetup_ST)
                  0:    begin
                              mI2C_DATA   <=     {8'h72,LUT_DATA};
                              mI2C_GO           <=     1;
                              mSetup_ST   <=     1;
                        end
                  1:    begin
                              if(mI2C_END)
                              begin
                                    if(!mI2C_ACK)
                                    mSetup_ST   <=     2;
                                    else
                                    mSetup_ST   <=     0;

                                    mI2C_GO     <=     0;
                              end
                        end
                  2:    begin
                              LUT_INDEX   <=     LUT_INDEX+1;
                              mSetup_ST   <=     0;
                        end
                  endcase
            end
            else
            begin
              READY<=1; //<---
              if(!HDMI_TX_INT)
              begin
                LUT_INDEX <= 0;
              end
              else
                LUT_INDEX <= LUT_INDEX;
            end
      end
end
```

```
//////////////////////////////////////////////////////////////////
//////////////////////   Config Data LUT     //////////////////////////
always
begin
      case(LUT_INDEX)
      //    Video Config Data
      00    :     LUT_DATA    <=    16'h9803;  //Must be set to 0x03 for
proper operation
      01    :     LUT_DATA    <=    16'h0100;  //Set 'N' value at 6144
      02    :     LUT_DATA    <=    16'h0218;  //Set 'N' value at 6144
      03    :     LUT_DATA    <=    16'h0300;  //Set 'N' value at 6144
      04    :     LUT_DATA    <=    16'h0b2e;  //MCLK Active
      05    :     LUT_DATA    <=    16'h0cbc;  //Serial Audio standard
i2s, R0x0C[1:0] = '00
      06    :     LUT_DATA    <=    16'h1472;  //audio Word Length 16
bit, Set Ch count in the channel status to 8.
      07    :     LUT_DATA    <=    16'h1520;  //Input 444 (RGB or
YCrCb) with Separate Syncs, 48kHz fs
      08    :     LUT_DATA    <=    16'h1630;  //Output format 444, 24-
bit input
      09    :     LUT_DATA    <=    16'h1846;  //Disable CSC
      10    :     LUT_DATA    <=    16'h4080;  //General control packet
enable
      11    :     LUT_DATA    <=    16'h4110;  //Power down control
      12    :     LUT_DATA    <=    16'h49A8;  //Set dither mode - 12-
to-10 bit
      13    :     LUT_DATA    <=    16'h5510;  //Set RGB in AVI
infoframe
      14    :     LUT_DATA    <=    16'h5608;  //Set active format
aspect
      15    :     LUT_DATA    <=    16'h96F6;  //Set interrup
      16    :     LUT_DATA    <=    16'h7307;  //Info frame Ch count to
8
      17    :     LUT_DATA    <=    16'h761f;  //Set speaker allocation
for 8 channels
      18    :     LUT_DATA    <=    16'h9803;  //Must be set to 0x03 for
proper operation
      19    :     LUT_DATA    <=    16'h9902;  //Must be set to Default
Value
      20    :     LUT_DATA    <=    16'h9ae0;  //Must be set to
0b1110000
      21    :     LUT_DATA    <=    16'h9c30;  //PLL filter R1 value
      22    :     LUT_DATA    <=    16'h9d61;  //Set clock divide
      23    :     LUT_DATA    <=    16'ha2a4;  //Must be set to 0xA4 for
proper operation
      24    :     LUT_DATA    <=    16'ha3a4;  //Must be set to 0xA4 for
proper operation
      25    :     LUT_DATA    <=    16'ha504;  //Must be set to Default
Value
      26    :     LUT_DATA    <=    16'hab40;  //Must be set to Default
Value
      27    :     LUT_DATA    <=    16'haf16;  //Select HDMI mode
      28    :     LUT_DATA    <=    16'hba60;  //No clock delay
      29    :     LUT_DATA    <=    16'hd1ff;  //Must be set to Default
Value
      30    :     LUT_DATA    <=    16'hde10;  //Must be set to Default
for proper operation
      31    :     LUT_DATA    <=    16'he460;  //Must be set to Default
Value
      32    :     LUT_DATA    <=    16'hfa7d;  //Nbr of times to look
for good phase
```

```
      default:          LUT_DATA    <=    16'h9803;
      endcase
      //case(LUT_INDEX)
      //
      ////  Video Config Data
      //0   :     LUT_DATA    <=    16'h9803;  //Must be set to 0x03 for
proper operation
      //1   :     LUT_DATA    <=    16'h0100;  //Set 'N' value at 6144
      //2   :     LUT_DATA    <=    16'h0218;  //Set 'N' value at 6144
      //3   :     LUT_DATA    <=    16'h0300;  //Set 'N' value at 6144
      //4   :     LUT_DATA    <=    16'h1470;  // Set Ch count in the
channel status to 8.
      //5   :     LUT_DATA    <=    16'h1520;  //Input 444 (RGB or
YCrCb) with Separate Syncs, 48kHz fs
      //6   :     LUT_DATA    <=    16'h1630;  //Output format 444, 24-
bit input
      //7   :     LUT_DATA    <=    16'h1846;  //Disable CSC
      //8   :     LUT_DATA    <=    16'h4080;  //General control packet
enable
      //9   :     LUT_DATA    <=    16'h4110;  //Power down control
      //10  :     LUT_DATA    <=    16'h49A8;  //Set dither mode - 12-
to-10 bit
      //11  :     LUT_DATA    <=    16'h5510;  //Set RGB in AVI
infoframe
      //12  :     LUT_DATA    <=    16'h5608;  //Set active format
aspect
      //13  :     LUT_DATA    <=    16'h96F6;  //Set interrup
      //14  :     LUT_DATA    <=    16'h7307;  //Info frame Ch count to
8
      //15  :     LUT_DATA    <=    16'h761f;  //Set speaker allocation
for 8 channels
      //16  :     LUT_DATA    <=    16'h9803;  //Must be set to 0x03 for
proper operation
      //17  :     LUT_DATA    <=    16'h9902;  //Must be set to Default
Value
      //18  :     LUT_DATA    <=    16'h9ae0;  //Must be set to
0b1110000
      //19  :     LUT_DATA    <=    16'h9c30;  //PLL filter R1 value
      //20  :     LUT_DATA    <=    16'h9d61;  //Set clock divide
      //21  :     LUT_DATA    <=    16'ha2a4;  //Must be set to 0xA4 for
proper operation
      //22  :     LUT_DATA    <=    16'ha3a4;  //Must be set to 0xA4 for
proper operation
      //23  :     LUT_DATA    <=    16'ha504;  //Must be set to Default
Value
      //24  :     LUT_DATA    <=    16'hab40;  //Must be set to Default
Value
      //25  :     LUT_DATA    <=    16'haf16;  //Select HDMI mode
      //26  :     LUT_DATA    <=    16'hba60;  //No clock delay
      //27  :     LUT_DATA    <=    16'hd1ff;  //Must be set to Default
Value
      //28  :     LUT_DATA    <=    16'hde10;  //Must be set to Default
for proper operation
      //29  :     LUT_DATA    <=    16'he460;  //Must be set to Default
Value
      //30  :     LUT_DATA    <=    16'hfa7d;  //Nbr of times to look
for good phase
  //
      //default:        LUT_DATA    <=    16'h9803;
      //endcase
```

```
end
////////////////////////////////////////////////////////////////////
endmodule
```

```
module I2C_Controller (
      CLOCK,
      I2C_SCLK,//I2C CLOCK
      I2C_SDAT,//I2C DATA
      I2C_DATA,//DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
      GO,      //GO transfor
      END,     //END transfor
      W_R,     //W_R
      ACK,      //ACK
      RESET,
      //TEST
      SD_COUNTER,
      SDO
);
      input  CLOCK;
      input  [23:0]I2C_DATA;
      input  GO;
      input  RESET;
      input  W_R;
      inout  I2C_SDAT;
      output I2C_SCLK;
      output END;
      output ACK;

//TEST
      output [5:0] SD_COUNTER;
      output SDO;


reg SDO;
reg SCLK;
reg END;
reg [23:0]SD;
reg [5:0]SD_COUNTER;

assign I2C_SCLK=(I2C_SCLK_)?1'b1:1'b0 ;
wire   I2C_SCLK_;
assign I2C_SCLK_= SCLK | ( ((SD_COUNTER >= 4) & (SD_COUNTER <=30))?
~CLOCK :0 );
wire I2C_SDAT= SDO?1'bz:0 ;

reg ACK1,ACK2,ACK3;
wire ACK=ACK1 | ACK2 |ACK3;

//--I2C COUNTER
always @(negedge RESET or posedge CLOCK ) begin
if (!RESET) SD_COUNTER=6'b111111;
else begin
if (GO==0)
      SD_COUNTER=0;
      else
      if (SD_COUNTER < 6'b111111) SD_COUNTER=SD_COUNTER+1;
end
end
//----
```

```
always @(negedge RESET or  posedge CLOCK ) begin
if (!RESET) begin SCLK=1;SDO=1; ACK1=0;ACK2=0;ACK3=0; END=1; end
else
case (SD_COUNTER)
      6'd0  : begin ACK1=0 ;ACK2=0 ;ACK3=0 ; END=0; SDO=1; SCLK=1;end
      //start
      6'd1  : begin SD=I2C_DATA;SDO=0;end
      6'd2  : SCLK=0;
      //SLAVE ADDR
      6'd3  : SDO=SD[23];
      6'd4  : SDO=SD[22];
      6'd5  : SDO=SD[21];
      6'd6  : SDO=SD[20];
      6'd7  : SDO=SD[19];
      6'd8  : SDO=SD[18];
      6'd9  : SDO=SD[17];
      6'd10 : SDO=SD[16];
      6'd11 : SDO=1'b1;//ACK

      //SUB ADDR
      6'd12  : begin SDO=SD[15]; ACK1=I2C_SDAT; end
      6'd13  : SDO=SD[14];
      6'd14  : SDO=SD[13];
      6'd15  : SDO=SD[12];
      6'd16  : SDO=SD[11];
      6'd17  : SDO=SD[10];
      6'd18  : SDO=SD[9];
      6'd19  : SDO=SD[8];
      6'd20  : SDO=1'b1;//ACK

      //DATA
      6'd21  : begin SDO=SD[7]; ACK2=I2C_SDAT; end
      6'd22  : SDO=SD[6];
      6'd23  : SDO=SD[5];
      6'd24  : SDO=SD[4];
      6'd25  : SDO=SD[3];
      6'd26  : SDO=SD[2];
      6'd27  : SDO=SD[1];
      6'd28  : SDO=SD[0];
      6'd29  : SDO=1'b1;//ACK


      //stop
    6'd30 : begin SDO=1'b0;   SCLK=1'b0; ACK3=I2C_SDAT; end
    6'd31 : SCLK=1'b1;
    6'd32 : begin SDO=1'b1; END=1; end

endcase
end


endmodule
```

```
module I2C_AV_Config (  //    Host Side
   iCLK,
   iRST_N,
//    I2C Side
   I2C_SCLK,
   I2C_SDAT ,
   READY
```

```
);
//    Host Side
input       iCLK;
input       iRST_N;
//    I2C Side
output            I2C_SCLK;
inout          I2C_SDAT;
//--
output      READY;
//    Internal Registers/Wires
reg            READY;
reg   [15:0]      mI2C_CLK_DIV;
reg   [23:0]      mI2C_DATA;
reg               mI2C_CTRL_CLK;
reg               mI2C_GO;
wire        mI2C_END;
wire        mI2C_ACK;
reg   [15:0]      LUT_DATA;
reg   [5:0] LUT_INDEX;
reg   [3:0] mSetup_ST;

//    Clock Setting
parameter   CLK_Freq    =     50000000;   //    50    MHz
parameter   I2C_Freq    =     20000;      //    20    KHz
//    LUT Data Number
parameter   LUT_SIZE    =     51;
//    Audio Data Index
parameter   Dummy_DATA  =     0;
parameter   SET_LIN_L   =     1;
parameter   SET_LIN_R   =     2;
parameter   SET_HEAD_L  =     3;
parameter   SET_HEAD_R  =     4;
parameter   A_PATH_CTRL =     5;
parameter   D_PATH_CTRL =     6;
parameter   POWER_ON    =     7;
parameter   SET_FORMAT  =     8;
parameter   SAMPLE_CTRL =     9;
parameter   SET_ACTIVE  =     10;
//    Video Data Index
parameter   SET_VIDEO   =     11;

////////////////////   I2C Control Clock ////////////////////////
always@(posedge iCLK or negedge iRST_N)
begin
      if(!iRST_N)
      begin
            mI2C_CTRL_CLK     <=    0;
            mI2C_CLK_DIV      <=    0;
      end
      else
      begin
            if( mI2C_CLK_DIV  < (CLK_Freq/I2C_Freq) )
            mI2C_CLK_DIV      <=    mI2C_CLK_DIV+1;
            else
            begin
                  mI2C_CLK_DIV      <=    0;
                  mI2C_CTRL_CLK     <=    ~mI2C_CTRL_CLK;
            end
      end
end
//////////////////////////////////////////////////////////////////
```

```
I2C_Controller    u0    (     .CLOCK(mI2C_CTRL_CLK),        //
      Controller Work Clock
                                .I2C_SCLK(I2C_SCLK),          //
      I2C CLOCK
                                .I2C_SDAT(I2C_SDAT),          //
      I2C DATA
                                .I2C_DATA(mI2C_DATA),         //
      DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
                                .GO(mI2C_GO),
      //    GO transfor
                                .END(mI2C_END),
      //    END transfor
                                .ACK(mI2C_ACK),
      //    ACK
                                .RESET(iRST_N)    );
////////////////////////////////////////////////////////////////
////////////////////  Config Control    ////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
begin
      if(!iRST_N)
      begin
            READY       <= 0;
            LUT_INDEX   <=    0;
            mSetup_ST   <=    0;
            mI2C_GO     <=    0;
      end
      else
      begin
            if(LUT_INDEX<LUT_SIZE)
            begin
                  READY<=0;
                  case(mSetup_ST)
                  0:    begin
                              if(LUT_INDEX<SET_VIDEO)
                              mI2C_DATA   <=    {8'h34,LUT_DATA};//audio
                              else
                              mI2C_DATA   <=    {8'h40,LUT_DATA};//tv-
decoder
                              mI2C_GO           <=    1;
                              mSetup_ST   <=    1;
                        end
                  1:    begin
                              if(mI2C_END)
                              begin
                                    if(!mI2C_ACK)
                                    mSetup_ST   <=    2;
                                    else
                                    mSetup_ST   <=    0;

                                    mI2C_GO     <=    0;
                              end
                        end
                  2:    begin
                              LUT_INDEX   <=    LUT_INDEX+1;
                              mSetup_ST   <=    0;
                        end
                  endcase
            end
            else        READY<=1;
      end
end
```

```
////////////////////////////////////////////////////////////////////
//////////////////////   Config Data LUT     //////////////////////////
always
begin
      case(LUT_INDEX)
      //    Audio Config Data
      SET_LIN_L   :     LUT_DATA    <=    16'h001A;
      SET_LIN_R   :     LUT_DATA    <=    16'h021A;
      SET_HEAD_L  :     LUT_DATA    <=    16'h047B;
      SET_HEAD_R  :     LUT_DATA    <=    16'h067B;
      A_PATH_CTRL :     LUT_DATA    <=    16'h08F8;
      D_PATH_CTRL :     LUT_DATA    <=    16'h0A06;//48K sample-rate
      POWER_ON       :  LUT_DATA    <=    16'h0C00;
      SET_FORMAT  :     LUT_DATA    <=    16'h0E01;//bit7:0 = BCLK not
inverted (default) ,bit6:0= enable slave mode (default) ,bit
[3:2]:00=16bit ,bit[1:0]:01=00 = right justified
      SAMPLE_CTRL :     LUT_DATA    <=    16'h1000;//0 = support for 256
fS based clock (default)
      SET_ACTIVE  :     LUT_DATA    <=    16'h1200;//default
      //    Video Config Data
      SET_VIDEO+1 :     LUT_DATA       <= 16'h0000;//04
      SET_VIDEO+2 :     LUT_DATA       <= 16'hc301;
    SET_VIDEO+3   :     LUT_DATA       <= 16'hc480;
    SET_VIDEO+4   :     LUT_DATA       <= 16'h0457;
      SET_VIDEO+5 :     LUT_DATA       <= 16'h1741;
      SET_VIDEO+6 :     LUT_DATA       <= 16'h5801;
      SET_VIDEO+7 :     LUT_DATA       <= 16'h3da2;
      SET_VIDEO+8 :     LUT_DATA       <= 16'h37a0;
      SET_VIDEO+9 :     LUT_DATA       <= 16'h3e6a;
      SET_VIDEO+10      :     LUT_DATA       <= 16'h3fa0;
      SET_VIDEO+11      :     LUT_DATA       <= 16'h0e80;
      SET_VIDEO+12      :     LUT_DATA       <= 16'h5581;
      //SET_VIDEO+13    :     LUT_DATA       <= 16'h37A0;   // Polarity
regiser
      SET_VIDEO+13      :     LUT_DATA       <= 16'h3720;   // hs/vs
negitive Polarity , pixel clock normal
      SET_VIDEO+14      :     LUT_DATA       <= 16'h0880;   // Contrast
Register
      SET_VIDEO+15      :     LUT_DATA       <= 16'h0a18;   //
Brightness Register
      SET_VIDEO+16      :     LUT_DATA       <= 16'h2c8e;   // AGC Mode
control
      SET_VIDEO+17      :     LUT_DATA       <= 16'h2df8;   // Chroma
Gain Control 1
      SET_VIDEO+18      :     LUT_DATA       <= 16'h2ece;   // Chroma
Gain Control 2
      SET_VIDEO+19      :     LUT_DATA       <= 16'h2ff4;   // Luma Gain
Control 1
      SET_VIDEO+20      :     LUT_DATA       <= 16'h30b2;   // Luma Gain
Control 2
      SET_VIDEO+21      :     LUT_DATA       <= 16'h0e00;

      default:          LUT_DATA    <=    16'd0 ;
      endcase
end
////////////////////////////////////////////////////////////////////
endmodule
```

Παράρτημα Γ: Πλήρης κώδικας HDL

```
module myTetris_top(

      ///////// CLOCK /////////
      input              CLOCK0_50,

      ///////// KEY /////////
      input    [ 3: 0]   KEY, //BUTTON is Low-Active

      ///////// SW /////////
      input    [ 9: 0]   SW,

      ///////// LED /////////
      output   [ 9: 0]   LEDR, //LED is Low-Active

      ///////// Seg7 /////////
      output   [ 6: 0]   HEX0,
      output   [ 6: 0]   HEX1,
      output   [ 6: 0]   HEX2,
      output   [ 6: 0]   HEX3,
      output   [ 6: 0]   HEX4,
      output   [ 6: 0]   HEX5,

      ///////// SDRAM /////////
      output             DRAM_CLK,
      output             DRAM_CKE,
      output   [12: 0]   DRAM_ADDR,
      output   [ 1: 0]   DRAM_BA,
      inout    [31: 0]   DRAM_DQ,
      output             DRAM_CS_n,
      output             DRAM_WE_n,
      output             DRAM_CAS_n,
      output             DRAM_RAS_n,
      output   [ 3: 0]   DRAM_DQM,

      ///////// HDMI /////////
      inout              HDMI_LRCLK,
      inout              HDMI_MCLK,
      inout              HDMI_SCLK,
      output             HDMI_TX_CLK,
      output             HDMI_TX_HS,
      output             HDMI_TX_VS,
      output   [23: 0]   HDMI_TX_D,
      output             HDMI_TX_DE,
      input              HDMI_TX_INT,
      inout              HDMI_I2S0,

      ///////// I2C for HDMI and ADC /////////
      inout              FPGA_I2C_SCL,
      inout              FPGA_I2C_SDA,

      ///////// UART /////////
      output             FPGA_UART_TX,
      input              FPGA_UART_RX,

      ///////// GPIO used for user input and sound output /////////
      inout    [5: 0]    GPIO_D

);
```

```
//=======================================================
//  Turn OFF / disable unused board resources
//=======================================================

//----LED off device
assign LEDR[8:0] = 9'b111111111;

//----HEX off device
assign HEX0 =7'h7f;
assign HEX1 =7'h7f;
assign HEX2 =7'h7f;
assign HEX3 =7'h7f;
assign HEX4 =7'h7f;
assign HEX5 =7'h7f;

//----SDRAM off
assign DRAM_DQ   =32'hzzzz_zzzzz;
assign DRAM_CS_n =1'b1;
assign DRAM_WE_n =1'b1;
assign DRAM_CAS_n=1'b1;
assign DRAM_RAS_n=1'b1;
assign DRAM_DQM  =4'b1111;

//=======================================================
//  50mhz system clock
//=======================================================
wire  SYSTEM_50MHZ;
assign SYSTEM_50MHZ = CLOCK0_50;

//=======================================================
//  Reset that acts on FPGA board startup
//=======================================================
wire reset_n; //fpga board startup repair
//--RESET
assign reset_n = ~ninit_done;
//---INIT RESET
wire ninit_done;
      ResetRelease ResetRelease_inst (
            .ninit_done (ninit_done)
      );

//=======================================================
//  Sound module
//=======================================================

wire [2:0] sound; //connect with game logic

audio_controller u_audio_controller (
    .clk      (SYSTEM_50MHZ),
//reset on board startup and user input
    .reset_n  (reset_n & !restart),
    .sound (sound),
    .speaker (GPIO_D[5])
);
```

```
//=======================================================
//  Joystick module
//=======================================================

wire fire, left, right, down, restart; //connect with game logic

input_controller u_input_controller (
    .clk      (SYSTEM_50MHZ),
       //reset on board startup and user input
    .reset_n  (reset_n & !restart),
    .gpio_d   (GPIO_D),
    .fire     (fire),
    .left     (left),
    .right    (right),
    .down     (down),
    .restart  (restart)
);

//=======================================================
//  Game logic module
//=======================================================

wire [199:0] grid_flat_wire;//connect with grid generator

game_logic u_game_logic (
    .clk       (SYSTEM_50MHZ),
       //reset on board startup and user input
    .reset_n   (reset_n & !restart),
    .joy_rotate (fire),
    .joy_left (left),
    .joy_right (right),
    .joy_down (down),
    .sound (sound),
    .grid_flat (grid_flat_wire)
);

//=======================================================
//  Grid generator module
//=======================================================

grid_generator    u_grid_generator (
      .vpg_pclk    (vpg_pclk),//vedio clock input
      .reset_n     (reset_n),
      .grid_flat   (grid_flat_wire),
      .vpg_de      (HDMI_TX_DE ),
      .vpg_hs      (HDMI_TX_HS ),
      .vpg_vs      (HDMI_TX_VS ),
      .vpg_pclk_out(HDMI_TX_CLK),
      .vpg_r       (vpg_r),
      .vpg_g       (vpg_g),
      .vpg_b       (vpg_b)
      );

//=======================================================
//  Terrasic HDMI interface modules
//=======================================================

wire  V_locked;
wire  pll_12M288;
wire [7:0] vpg_r;
wire [7:0] vpg_g;
```

```
wire [7:0] vpg_b;
wire HDMI_READY ;
wire HDMI_I2C_SCLK ;

assign HDMI_TX_D[23:16] =     vpg_r;
assign HDMI_TX_D[15:8]  =     vpg_g;
assign HDMI_TX_D[7:0]   =     vpg_b;

//-- HDMI READY LED
assign LEDR[9] = ~HDMI_READY   ;

//---AV PLL
VEDIO_PLL u_VEDIO_PLL(
    .refclk       (SYSTEM_50MHZ  ),
    .outclk_0     (vpg_pclk      ),//148.5MHZ
    .locked       (V_locked      ),
    .rst          (~reset_n      )
);

//--  HDMI I2C    SETTING
I2C_HDMI_Config u_I2C_HDMI_Config (
    .iCLK       (SYSTEM_50MHZ  ),
    .iRST_N     (reset_n),
    .I2C_SCLK   (FPGA_I2C_SCL  ),
    .I2C_SDAT   (FPGA_I2C_SDA  ),
    .HDMI_TX_INT(HDMI_TX_INT   ),
    .READY      (HDMI_READY    )
);

endmodule
```

```
module grid_generator (
    input               vpg_pclk,
    input               reset_n,
    input [199:0]       grid_flat, //Tile grid
    output reg          vpg_de,
    output reg          vpg_hs,
    output reg          vpg_vs,
    output              vpg_pclk_out,
    output reg [7:0]    vpg_r,
    output reg [7:0]    vpg_g,
    output reg [7:0]    vpg_b

);

assign vpg_pclk_out = vpg_pclk;

//=======================================================
//  1080p Timing parameters (Terasic default)
//=======================================================
localparam H_ACTIVE = 1920;
localparam H_FP     = 88;
localparam H_SYNC   = 44;
localparam H_BP     = 148;
localparam H_TOTAL  = H_ACTIVE + H_FP + H_SYNC + H_BP;

localparam V_ACTIVE = 1080;
localparam V_FP     = 4;
localparam V_SYNC   = 5;
localparam V_BP     = 36;
```

```
localparam V_TOTAL  = V_ACTIVE + V_FP + V_SYNC + V_BP;

//=======================================================
//  Horizontal and Vertical pixel counters
//=======================================================
reg [11:0] h_cnt;
reg [11:0] v_cnt;

//=======================================================
//  Grid parameters
//=======================================================
localparam GRID_COLS   = 10;
localparam GRID_ROWS   = 20;
localparam SQUARE_SIZE = 88;

// Grid total dimensions
localparam GRID_PIXEL_H = GRID_COLS * SQUARE_SIZE;
localparam GRID_PIXEL_W = GRID_ROWS * SQUARE_SIZE;

// Center grid in screen
localparam GRID_START_X = (H_ACTIVE - GRID_PIXEL_W)/2;
localparam GRID_START_Y = (V_ACTIVE - GRID_PIXEL_H)/2;

// Does the rendered pixel belong to the whole grid?
wire inside_grid =(h_cnt >= GRID_START_X) && (h_cnt < GRID_START_X +
GRID_PIXEL_W) && (v_cnt >= GRID_START_Y) && (v_cnt < GRID_START_Y +
GRID_PIXEL_H);

// Square coordinates in the grid
wire [3:0] col = (v_cnt - GRID_START_Y) / SQUARE_SIZE; // vertical index
wire [4:0] row = (h_cnt - GRID_START_X) / SQUARE_SIZE; // horizontal
index

// Flip the collumns for portrait (vertical) screen
wire [3:0] col_flipped = GRID_COLS - 1 - col;

// Does the rendered pixel belong to the square?
wire [6:0] pixel_x_in_tile = h_cnt - (GRID_START_X + row*SQUARE_SIZE);
wire [6:0] pixel_y_in_tile = v_cnt - (GRID_START_Y + col*SQUARE_SIZE);

// 2 pixel wide black lines around each square
wire vertical_line = (pixel_y_in_tile < 2) || (pixel_y_in_tile >=
SQUARE_SIZE-2);
wire horizontal_line = (pixel_x_in_tile < 2) || (pixel_x_in_tile >=
SQUARE_SIZE-2);

// Does the rendered pixel belong to a grid line?
wire on_grid_line = inside_grid && (vertical_line || horizontal_line);

// Is this cell set to be on? (white)
wire cell_on = inside_grid && grid_flat[row*GRID_COLS + col_flipped];




//=======================================================
//  Horizontal scan counter
//=======================================================
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        h_cnt <= 12'd0;
```

```
    end else begin
        if (h_cnt == H_TOTAL-1)
            h_cnt <= 12'd0;
        else
            h_cnt <= h_cnt + 12'd1;
    end
end

//=========================================================
//  Vertical scan counter
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        v_cnt <= 12'd0;
    end else if (h_cnt == H_TOTAL-1) begin
        if (v_cnt == V_TOTAL-1)
            v_cnt <= 12'd0;
        else
            v_cnt <= v_cnt + 12'd1;
    end
end
//=========================================================
//  Timing signals
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        vpg_hs <= 1'b1;
        vpg_vs <= 1'b1;
    end else begin
        vpg_hs <= ~((h_cnt >= H_ACTIVE + H_FP) &&
                    (h_cnt <  H_ACTIVE + H_FP + H_SYNC));

        vpg_vs <= ~((v_cnt >= V_ACTIVE + V_FP) &&
                    (v_cnt <  V_ACTIVE + V_FP + V_SYNC));
    end
end

//=========================================================
//  Display activation
//=========================================================
always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n)
        vpg_de <= 1'b0;
    else
        vpg_de <= (h_cnt < H_ACTIVE) && (v_cnt < V_ACTIVE);
end

always @(posedge vpg_pclk or negedge reset_n) begin
    if (!reset_n) begin
        // Clean whole screen on reset
        vpg_r <= 8'h00;
        vpg_g <= 8'h00;
        vpg_b <= 8'h00;

    end else if (!inside_grid) begin
        // Gray pixels outside grid
        vpg_r <= 8'h80;
        vpg_g <= 8'h80;
        vpg_b <= 8'h80;

    end else if (on_grid_line) begin
```

```
        // Black grid lines
        vpg_r <= 8'h00;
        vpg_g <= 8'h00;
        vpg_b <= 8'h00;

    end else if (cell_on) begin
        // White square
        vpg_r <= 8'hFF;
        vpg_g <= 8'hFF;
        vpg_b <= 8'hFF;

    end else begin
        // Black square
        vpg_r <= 8'h00;
        vpg_g <= 8'h00;
        vpg_b <= 8'h00;
    end
end

endmodule
```

```
//0 = I
//1 = O
//2 = T
//3 = S
//4 = Z
//5 = J
//6 = L

module tetromino_rom (
       // The type of the selected piece
    input  [2:0] piece_type,
       // The rotation state of the selected piece
    input  [1:0] rotation_state,
       // Which one of the four squares of the piece we are looking at
    input  [1:0] block_index,

    output reg signed [2:0] dx,
    output reg signed [2:0] dy
);

always @(*) begin
    dx = 0;
    dy = 0;

    case (piece_type)

    // ============================================================
    // 0 = I piece
    // ============================================================
    3'b000: begin
        case (rotation_state)
            2'b0, 2'b10: begin // ----
                case (block_index)
                    2'b00: begin dx=0; dy=0; end
                    2'b01: begin dx=1; dy=0; end
                    2'b10: begin dx=2; dy=0; end
                    2'b11: begin dx=3; dy=0; end
                endcase
            end
```

```
            2'b01, 2'b11: begin // |
                case (block_index)
                    2'b00: begin dx=2; dy=0; end
                    2'b01: begin dx=2; dy=1; end
                    2'b10: begin dx=2; dy=2; end
                    2'b11: begin dx=2; dy=3; end
                endcase
            end
        endcase
    end

    // ============================================================
    // 1 = O piece
    // ============================================================
    3'b001: begin
        case (block_index)
            2'b00: begin dx=1; dy=0; end
            2'b01: begin dx=2; dy=0; end
            2'b10: begin dx=1; dy=1; end
            2'b11: begin dx=2; dy=1; end
        endcase
    end

    // ============================================================
    // 2 = T piece
    // ============================================================
    3'b010: begin
        case (rotation_state)
            2'b00: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=0; dy=1; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=2; dy=1; end
                endcase
            end
            2'b01: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=2; dy=1; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
            2'b10: begin
                case (block_index)
                    2'b00: begin dx=0; dy=1; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=2; dy=1; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
            2'b11: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=0; dy=1; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
        endcase
```

```
    end

    // ============================================================
    // 3 = S piece
    // ============================================================
    3'b011: begin
        case (rotation_state)
            2'b0, 2'b10: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=2; dy=0; end
                    2'b10: begin dx=0; dy=1; end
                    2'b11: begin dx=1; dy=1; end
                endcase
            end
            2'b01, 2'b11: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=2; dy=1; end
                    2'b11: begin dx=2; dy=2; end
                endcase
            end
        endcase
    end

    // ============================================================
    // 4 = Z piece
    // ============================================================
    3'b100: begin
        case (rotation_state)
            2'b0, 2'b10: begin
                case (block_index)
                    2'b00: begin dx=0; dy=0; end
                    2'b01: begin dx=1; dy=0; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=2; dy=1; end
                endcase
            end
            2'b01, 2'b11: begin
                case (block_index)
                    2'b00: begin dx=2; dy=0; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=2; dy=1; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
        endcase
    end

    // ============================================================
    // 5 = J piece
    // ============================================================
    3'b101: begin
        case (rotation_state)
            2'b00: begin
                case (block_index)
                    2'b00: begin dx=0; dy=0; end
                    2'b01: begin dx=0; dy=1; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=2; dy=1; end
```

```
                endcase
            end
            2'b01: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=2; dy=0; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
            2'b10: begin
                case (block_index)
                    2'b00: begin dx=0; dy=1; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=2; dy=1; end
                    2'b11: begin dx=2; dy=2; end
                endcase
            end
            2'b11: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=0; dy=2; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
        endcase
    end

    // ============================================================
    // 6 = L piece
    // ============================================================
    3'b110: begin
        case (rotation_state)
            2'b00: begin
                case (block_index)
                    2'b00: begin dx=2; dy=0; end
                    2'b01: begin dx=0; dy=1; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=2; dy=1; end
                endcase
            end
            2'b01: begin
                case (block_index)
                    2'b00: begin dx=1; dy=0; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=1; dy=2; end
                    2'b11: begin dx=2; dy=2; end
                endcase
            end
            2'b10: begin
                case (block_index)
                    2'b00: begin dx=0; dy=1; end
                    2'b01: begin dx=1; dy=1; end
                    2'b10: begin dx=2; dy=1; end
                    2'b11: begin dx=0; dy=2; end
                endcase
            end
            2'b11: begin
                case (block_index)
                    2'b00: begin dx=0; dy=0; end
```

```
                    2'b01: begin dx=1; dy=0; end
                    2'b10: begin dx=1; dy=1; end
                    2'b11: begin dx=1; dy=2; end
                endcase
            end
        endcase
    end

    endcase
end

endmodule
```

```
module input_controller (
    input         clk,          // 50MHz clock
    input         reset_n,
    input  [4:0] gpio_d,

    output        fire,
    output        left,
    output        right,
    output        down,
    output        restart
);

    // ============================================================
    // Parameters
    // ============================================================
    localparam DEBOUNCE_MAX = 500_000; // 10ms @ 50MHz

    // ============================================================
    // Synchronizers (2-flip-flop to avoid metastability)
    // ============================================================

       //This is a 2-flip-flop synchronizer used to safely bring
asynchronous GPIO inputs into the clk clock domain.
       //The first register (sync_0) captures the raw gpio_d signal,
which may be metastable.
       //The second register (sync_1) captures the already-stabilizing
value one clock later.
       //This greatly reduces the chance of metastability propagating
into the rest of the design.

    reg [4:0] sync_0;
    reg [4:0] sync_1;

    always @(posedge clk) begin
        sync_0 <= gpio_d;
        sync_1 <= sync_0;
    end

    // ============================================================
    // Debounce counters
    // ============================================================
    reg [19:0] cnt_fire, cnt_left, cnt_right, cnt_down, cnt_restart;
    reg fire_r, left_r, right_r, down_r, restart_r;

    // ============================================================
    // Outputs
```

```
    // ============================================================
    assign fire  = fire_r;
    assign left  = left_r;
    assign right = right_r;
    assign down  = down_r;
    assign restart  = restart_r;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            fire_r  <= 0;
            left_r  <= 0;
            right_r <= 0;
            down_r  <= 0;
            restart_r  <= 0;

            cnt_fire  <= 0;
            cnt_left  <= 0;
            cnt_right <= 0;
            cnt_down  <= 0;
            cnt_restart  <= 0;
        end
        else begin

//wait to the dbounce_max value bebore setting output

            // FIRE (GPIO_D[0])
            if (sync_1[0] != fire_r) begin
                if (cnt_fire == DEBOUNCE_MAX) begin
                    fire_r  <= sync_1[0];
                    cnt_fire <= 0;
                end
                else
                    cnt_fire <= cnt_fire + 1;
            end
            else
                cnt_fire <= 0;

            // LEFT (GPIO_D[1])
            if (sync_1[1] != left_r) begin
                if (cnt_left == DEBOUNCE_MAX) begin
                    left_r  <= sync_1[1];
                    cnt_left <= 0;
                end
                else
                    cnt_left <= cnt_left + 1;
            end
            else
                cnt_left <= 0;

            // RIGHT (GPIO_D[2])
            if (sync_1[2] != right_r) begin
                if (cnt_right == DEBOUNCE_MAX) begin
                    right_r  <= sync_1[2];
                    cnt_right <= 0;
                end
                else
                    cnt_right <= cnt_right + 1;
            end
            else
                cnt_right <= 0;
```

```
            // DOWN (GPIO_D[3])
            if (sync_1[3] != down_r) begin
                if (cnt_down == DEBOUNCE_MAX) begin
                    down_r  <= sync_1[3];
                    cnt_down <= 0;
                end
                else
                    cnt_down <= cnt_down + 1;
            end
            else
                cnt_down <= 0;

            // RESTART (GPIO_D[4])
            if (sync_1[4] != restart_r) begin
                if (cnt_restart == DEBOUNCE_MAX) begin
                    restart_r  <= sync_1[4];
                    cnt_restart <= 0;
                end
                else
                    cnt_restart <= cnt_restart + 1;
            end
            else
                cnt_restart <= 0;


        end
    end

endmodule
```

```
module audio_controller(
    input  clk,
    input  reset_n,
    input  [2:0] sound, //3 different sounds
    output reg speaker //GPIO pin 5
);

    // Counters and registers
    reg [16:0] pm_cnt; //counter for output frequency
    reg [27:0] dur_cnt; //counter for output duration
    reg [2:0] sound_d;
    reg play; //currently playing
    reg [16:0] freq_limit; //output frequency

    // Duration selection
    reg [27:0] dur_limit_curr;

    // Localparams for durations
    localparam DUR_SHORT = 5_000_000;  // short sound for tetromino drop
and line clear
    localparam DUR_LONG  = 30_000_000; // long sound for game over

    // Rising edge detection for sounds
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            sound_d       <= 0;
            pm_cnt        <= 0;
            dur_cnt       <= 0;
            speaker       <= 0;
            play          <= 0;
```

```
            freq_limit    <= 0;
            dur_limit_curr<= 0; //applied duration of sound
        end else begin
            // Store previous sound state
            sound_d <= sound;
            // Detect rising edge of sound input
            // we need to set it back to 0 on the game logic
            // module, after setting it to 1 so that the next
            //sound can be produced, since the sound module
            //manages the actual duration of a sound
            if ((sound[0] && !sound_d[0])) begin
                play          <= 1;
                dur_limit_curr<= DUR_SHORT;
                freq_limit    <= 200_000; // HIGH
                dur_cnt       <= 0;
            end else if ((sound[1] && !sound_d[1])) begin
                play          <= 1;
                dur_limit_curr<= DUR_SHORT;
                freq_limit    <= 300_000; // LOW
                dur_cnt       <= 0;
            end else if ((sound[2] && !sound_d[2])) begin
                play          <= 1;
                dur_limit_curr<= DUR_LONG;
                freq_limit    <= 500_000; // VERY LOW
                dur_cnt       <= 0;
            end

            //we first set frequency and duration and then produce the
            //desired output
            if (play) begin
                if (dur_cnt < dur_limit_curr) begin
                    dur_cnt <= dur_cnt + 1;
            //toggle speaker output pin based on freq_limit
                    if (pm_cnt >= freq_limit) begin
                        pm_cnt <= 0;
                        speaker <= ~speaker;
                    end else begin
                        pm_cnt <= pm_cnt + 1;
                    end
                end else begin
                    play    <= 0;
                    speaker <= 0;
                    pm_cnt <= 0;
                end
            end
        end
    end
endmodule
```

```
module game_logic (
    input        clk,
    input        reset_n,
    input        joy_rotate,
    input        joy_left,
    input        joy_right,
    input        joy_down,
    output reg [2:0] sound, //interface with sound module
    output reg [199:0] grid_flat //interface with display grid generator
module
);

//============================================================================
// Locked tiles and moving tetromino
//============================================================================
reg [199:0] stack_grid;      // Locked tetrominos
reg [199:0] active_piece;    // Current moving tetromino

//============================================================================
// Parameters and registers
//============================================================================
localparam GRID_COLS = 10; //grid size
localparam GRID_ROWS = 20;

//fall frequency for when the game starts
localparam INITIAL_FALL_FREQ = 50_000_000; //1hz

//ARR and DAS input repeat delay constants
localparam DAS_DELAY = 7_500_000; //150ms
localparam ARR_DELAY = 2_500_000; //20ms

//counters for input ARR and DAS implementation
reg [27:0] repeat_delay, move_count, repeat_delay_counter;

//fall frequency increase amount per line clear
localparam FALL_INCREASE = 1_000_000;

//current fall frequency
reg [27:0] FALL_FREQ;
initial FALL_FREQ = INITIAL_FALL_FREQ;

//flag to show start screen
reg show_start;
initial show_start = 1;

//flag to show game over screen
reg game_over;
initial game_over = 0;

//============================================================================
// 1 Hz tick timer for falling tetromino
//============================================================================
reg [27:0] fall_counter;
reg        fall_tick;

//increase fall_counter until we reach FALL_FREQ
//then produce one tick for the game logic
always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        fall_counter <= 0;
        fall_tick <= 0;
```

```
    end else begin
        if (fall_counter >= FALL_FREQ) begin
            fall_counter <= 0;
            fall_tick <= 1;
        end else begin

                        if (!joy_down) begin
            fall_counter <= fall_counter + 1;
                        end else
                    //decrease coounter faster when joy_down is pressed

                        if (joy_down) begin
            fall_counter <= fall_counter + 10;
                        end
            fall_tick <= 0;
        end
    end
end

//=============================================================================
// Random tetromino generator (3-bit LFSR)
//=============================================================================
reg [2:0] lfsr;
wire lfsr_feedback;

// 3-bit Linear Feedback Shift Register (LFSR)
// On each clock edge, the register shifts left by one bit.
// The new input bit is generated by XORing bit[2] and bit[0] (taps x^3
+ x + 1).
// Produces a repeating pseudo-random sequence.

assign lfsr_feedback = lfsr[2] ^ lfsr[0]; // taps x^3 + x + 1

always @(posedge clk or negedge reset_n) begin
    if (!reset_n)
        lfsr <= 3'b001;   // must not be 000
    else
        lfsr <= {lfsr[1:0], lfsr_feedback};
end

//=============================================================================
// Tetromino state and position
//=============================================================================
reg [2:0] piece_type;
reg [1:0] rotation_state;
reg signed [5:0] base_x; //x can go negative, y cannot
reg [4:0] base_y;

//precompute x offset from signed into unsigned
wire [5:0] colx0 = $unsigned(base_x + $signed(dx0));
wire [5:0] colx1 = $unsigned(base_x + $signed(dx1));
wire [5:0] colx2 = $unsigned(base_x + $signed(dx2));
wire [5:0] colx3 = $unsigned(base_x + $signed(dx3));

//precompute x offset of rotated tetromino from signed into unsigned
wire [5:0] rcolx0 = $unsigned(base_x + $signed(rdx0));
wire [5:0] rcolx1 = $unsigned(base_x + $signed(rdx1));
wire [5:0] rcolx2 = $unsigned(base_x + $signed(rdx2));
wire [5:0] rcolx3 = $unsigned(base_x + $signed(rdx3));

//=============================================================================
```

```
// Connect to ROM to get the four tetromino square positions
//======================================================================
wire [3:0] dx0, dy0;
wire [3:0] dx1, dy1;
wire [3:0] dx2, dy2;
wire [3:0] dx3, dy3;

tetromino_rom rom0 (piece_type, rotation_state, 2'b00, dx0, dy0);
tetromino_rom rom1 (piece_type, rotation_state, 2'b01, dx1, dy1);
tetromino_rom rom2 (piece_type, rotation_state, 2'b10, dx2, dy2);
tetromino_rom rom3 (piece_type, rotation_state, 2'b11, dx3, dy3);

//======================================================================
// Compute next down position for the tetromino. Base Y position + tile
+ 1down * x position
//======================================================================
wire [8:0] idx0_down = (base_y + dy0 + 1) * GRID_COLS + (colx0);
wire [8:0] idx1_down = (base_y + dy1 + 1) * GRID_COLS + (colx1);
wire [8:0] idx2_down = (base_y + dy2 + 1) * GRID_COLS + (colx2);
wire [8:0] idx3_down = (base_y + dy3 + 1) * GRID_COLS + (colx3);

//======================================================================
// Get rotated position from ROM for collision check
//======================================================================
wire [1:0] rotation_next = (rotation_state == 2'b11) ? 2'b00 :
rotation_state + 1; //wrap around 0-3
wire [3:0] rdx0, rdy0;
wire [3:0] rdx1, rdy1;
wire [3:0] rdx2, rdy2;
wire [3:0] rdx3, rdy3;

tetromino_rom rom_r0 (piece_type, rotation_next, 2'b00, rdx0, rdy0);
tetromino_rom rom_r1 (piece_type, rotation_next, 2'b01, rdx1, rdy1);
tetromino_rom rom_r2 (piece_type, rotation_next, 2'b10, rdx2, rdy2);
tetromino_rom rom_r3 (piece_type, rotation_next, 2'b11, rdx3, rdy3);

//======================================================================
// Compute Next rotated position for the tetromino.
//======================================================================

wire [8:0] ridx0 = (base_y + rdy0) * GRID_COLS + (rcolx0);
wire [8:0] ridx1 = (base_y + rdy1) * GRID_COLS + (rcolx1);
wire [8:0] ridx2 = (base_y + rdy2) * GRID_COLS + (rcolx2);
wire [8:0] ridx3 = (base_y + rdy3) * GRID_COLS + (rcolx3);

//======================================================================
// Collision detection
//======================================================================

// Vertical collision (bottom or stack)
wire hit_something;
assign hit_something =
    ((base_y + dy0) == GRID_ROWS-1) || //collision to bottom
    ((base_y + dy1) == GRID_ROWS-1) ||
    ((base_y + dy2) == GRID_ROWS-1) ||
    ((base_y + dy3) == GRID_ROWS-1) ||
    stack_grid[idx0_down] || //collision to another tetromino down
    stack_grid[idx1_down] ||
    stack_grid[idx2_down] ||
    stack_grid[idx3_down];
```

```
// Horizontal collision with walls or stack
wire hit_left =
       //wall
    ((colx0 == 0) ||
       (colx1 == 0) ||
       (colx2 == 0) ||
       (colx3 == 0)) ||
    //stack
       ((colx0 > 0) && stack_grid[(base_y+dy0)*GRID_COLS + (colx0-1)])
||
    ((colx1 > 0) && stack_grid[(base_y+dy1)*GRID_COLS + (colx1-1)]) ||
    ((colx2 > 0) && stack_grid[(base_y+dy2)*GRID_COLS + (colx2-1)]) ||
    ((colx3 > 0) && stack_grid[(base_y+dy3)*GRID_COLS + (colx3-1)]);

wire hit_right =
       //wall
    ((colx0 == GRID_COLS-1) ||
       (colx1 == GRID_COLS-1) ||
       (colx2 == GRID_COLS-1) ||
       (colx3 == GRID_COLS-1)) ||
    //stack
       ((colx0 < GRID_COLS-1) && stack_grid[(base_y+dy0)*GRID_COLS +
(colx0+1)]) ||
    ((colx1 < GRID_COLS-1) && stack_grid[(base_y+dy1)*GRID_COLS +
(colx1+1)]) ||
    ((colx2 < GRID_COLS-1) && stack_grid[(base_y+dy2)*GRID_COLS +
(colx2+1)]) ||
    ((colx3 < GRID_COLS-1) && stack_grid[(base_y+dy3)*GRID_COLS +
(colx3+1)]);

// Rotation collision (check next rotated position)
wire rotation_blocked;
assign rotation_blocked =
    // horizontal out-of-bounds
    (rcolx0 >= GRID_COLS) ||
       (rcolx1 >= GRID_COLS) ||
       (rcolx2 >= GRID_COLS) ||
       (rcolx3 >= GRID_COLS) ||
    // vertical out-of-bounds
    (base_y + rdy0 >= GRID_ROWS) ||
       (base_y + rdy1 >= GRID_ROWS) ||
       (base_y + rdy2 >= GRID_ROWS) ||
       (base_y + rdy3 >= GRID_ROWS) ||
    // collision with stack
    stack_grid[ridx0] ||
       stack_grid[ridx1] ||
       stack_grid[ridx2] ||
       stack_grid[ridx3];

//=============================================================================
// Wires that tell is a row is full. All bits of each row are connected
to an AND gate
//=============================================================================
wire row0_full  = &stack_grid[  9:  0];
wire row1_full  = &stack_grid[ 19: 10];
wire row2_full  = &stack_grid[ 29: 20];
wire row3_full  = &stack_grid[ 39: 30];
wire row4_full  = &stack_grid[ 49: 40];
wire row5_full  = &stack_grid[ 59: 50];
wire row6_full  = &stack_grid[ 69: 60];
wire row7_full  = &stack_grid[ 79: 70];
```

```
wire row8_full  = &stack_grid[ 89: 80];
wire row9_full  = &stack_grid[ 99: 90];
wire row10_full = &stack_grid[109:100];
wire row11_full = &stack_grid[119:110];
wire row12_full = &stack_grid[129:120];
wire row13_full = &stack_grid[139:130];
wire row14_full = &stack_grid[149:140];
wire row15_full = &stack_grid[159:150];
wire row16_full = &stack_grid[169:160];
wire row17_full = &stack_grid[179:170];
wire row18_full = &stack_grid[189:180];
wire row19_full = &stack_grid[199:190];
//flag to tell if there are any full lines in the grid
//to then crear them
wire any_row_full =
      row0_full  | row1_full  | row2_full  | row3_full
    | row4_full  | row5_full  | row6_full  | row7_full
    | row8_full  | row9_full  | row10_full | row11_full
    | row12_full | row13_full | row14_full | row15_full
    | row16_full | row17_full | row18_full | row19_full;

//============================================================================
// Main logic block
//============================================================================
reg [7:0] idx0, idx1, idx2, idx3; // moving tetromino indices

reg joy_rotate_p;

always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        grid_flat      <= 200'b0;
        piece_type     <= 3'b00;
        rotation_state <= 2'b00;
        base_x         <= 3;
        base_y         <= 0;
        stack_grid     <= 200'b0;
        active_piece   <= 200'b0;
              joy_rotate_p   <= 0;
              show_start        <= 1;
              game_over         <= 0;
              sound <= 3'b000;
              //reset fall frequency
              FALL_FREQ <= INITIAL_FALL_FREQ;
    end else begin

            if (show_start) begin //show the start or game over screen

                  //on joystick button click begin the game
                  joy_rotate_p <= joy_rotate; // store previous state
                  if (joy_rotate && !joy_rotate_p) show_start <= 0;


                  //reset game when showing the start screen
                  grid_flat      <= 200'b0;
                  piece_type     <= 3'b00;
                  rotation_state <= 2'b00;
                  base_x         <= 3;
                  base_y         <= 0;
                  stack_grid     <= 200'b0;
                  active_piece   <= 200'b0;
                  //reset fall frequency
```

```
                    FALL_FREQ <= INITIAL_FALL_FREQ;

                    if (game_over) begin
                          //display "GAME OVER" if game over flag is set
                          grid_flat[  9:  0] <= 10'b1111001111; // row 0
                          grid_flat[ 19: 10] <= 10'b1001000001; // row 1
                          grid_flat[ 29: 20] <= 10'b1001001101; // row 2
                          grid_flat[ 39: 30] <= 10'b1111001111; // row 3
                          grid_flat[ 49: 40] <= 10'b0000000000; // row 4
                          grid_flat[ 59: 50] <= 10'b1001000110; // row 5
                          grid_flat[ 69: 60] <= 10'b1001001001; // row 6
                          grid_flat[ 79: 70] <= 10'b1001001111; // row 7
                          grid_flat[ 89: 80] <= 10'b0110001001; // row 8
                          grid_flat[ 99: 90] <= 10'b0000000000; // row 9
                          grid_flat[109:100] <= 10'b1111001001; // row10
                          grid_flat[119:110] <= 10'b0001001111; // row11
                          grid_flat[129:120] <= 10'b0111001001; // row12
                          grid_flat[139:130] <= 10'b0001001001; // row13
                          grid_flat[149:140] <= 10'b1111000000; // row14
                          grid_flat[159:150] <= 10'b0000001111; // row15
                          grid_flat[169:160] <= 10'b1111000001; // row16
                          grid_flat[179:170] <= 10'b1001000111; // row17
                          grid_flat[189:180] <= 10'b0111000001; // row18
                          grid_flat[199:190] <= 10'b1001001111; // row19

                    end else begin
                       //display "PLAY"
                          grid_flat[  9:  0] <= 10'b0001111000; // row 0
                          grid_flat[ 19: 10] <= 10'b0001001000; // row 1
                          grid_flat[ 29: 20] <= 10'b0001111000; // row 2
                          grid_flat[ 39: 30] <= 10'b0000001000; // row 3
                          grid_flat[ 49: 40] <= 10'b0000000000; // row 4
                          grid_flat[ 59: 50] <= 10'b0000001000; // row 5
                          grid_flat[ 69: 60] <= 10'b0000001000; // row 6
                          grid_flat[ 79: 70] <= 10'b0000001000; // row 7
                          grid_flat[ 89: 80] <= 10'b0001111000; // row 8
                          grid_flat[ 99: 90] <= 10'b0000000000; // row 9
                          grid_flat[109:100] <= 10'b0000110000; // row10
                          grid_flat[119:110] <= 10'b0001001000; // row11
                          grid_flat[129:120] <= 10'b0001111000; // row12
                          grid_flat[139:130] <= 10'b0001001000; // row13
                          grid_flat[149:140] <= 10'b0000000000; // row14
                          grid_flat[159:150] <= 10'b0001001000; // row15
                          grid_flat[169:160] <= 10'b0001001000; // row16
                          grid_flat[179:170] <= 10'b0000110000; // row17
                          grid_flat[189:180] <= 10'b0000110000; // row18
                          grid_flat[199:190] <= 10'b0000110000; // row19

                    end;


              end else //we dont show the start screen, game plays
normally

              if (!any_row_full) begin //there are not any full rows, so
we drop the tetromino one position and accept user input

                    //game over
                    if (|stack_grid[9:0]) begin //if any of the squares on
the first row are stacked, game over
                    sound <= 3'b100; //low frequency long beep for game
```

```
over
                    show_start <= 1; //go back to show the start screen
                    game_over <= 1;
                    end

                    // reset any sound if it is enabled, on tick timer.
audio controler module manages the duration
                    // of each sound, but we need to set its input back to
0 in order to be ready for the next
                    // sound, since audio controller module looks for the
0 -> 1 transition
                    if (fall_tick && (|sound[1:0])) sound <= 3'b000;


                    // Drop tetromino one position by increasing the z
index
                    if (fall_tick && !hit_something) begin
                          base_y <= base_y + 1;

                    end else
                    // else if we hit something copy the tetromino to the
stack grid to lock in in place
                    // locking actually happens on the next tick so no
additional lock delay is needed
                    if (fall_tick && hit_something) begin
                          // Compute indices of moving piece
                          idx0 = (base_y + dy0) * GRID_COLS + (colx0);
                          idx1 = (base_y + dy1) * GRID_COLS + (colx1);
                          idx2 = (base_y + dy2) * GRID_COLS + (colx2);
                          idx3 = (base_y + dy3) * GRID_COLS + (colx3);
                          // Copy indices into stack grid
                          stack_grid[idx0] <= 1'b1;
                          stack_grid[idx1] <= 1'b1;
                          stack_grid[idx2] <= 1'b1;
                          stack_grid[idx3] <= 1'b1;

                          // Make a sound
                          sound <= 3'b010; //high frequency short beep for
tetromino stacking

                          // Spawn new tetromino
                          base_y <= 0; //top
                          base_x <= 3; //center
                          rotation_state <= 0; //no rotated
                          piece_type <= lfsr - 1;  //random tetromino
index
                    end

//=================================================================
// Joystick input, ARR and DAS implementation
//=================================================================

                          //joy rotate button is read on every 50mhz clock
cycle, since we dont need
                          //ARR or DAS as it is rotated only once per user
click
                          joy_rotate_p <= joy_rotate; // store previous
state so to rotate only once per click
                          if (joy_rotate && !rotation_blocked &&
!joy_rotate_p) begin //if we dont hit something rotate
                                //set next rotate state, and wrap around
```

```
0-3
                                rotation_state <= (rotation_state ==
2'b11) ? 2'b00 : rotation_state + 1;
                            end
                            //if joystick is moved left or right we compute
ARR and DAS
                            if (joy_left || joy_right) begin
                                //delay tetromino x position move until
the delay counter reaches the wanted delay
                                //on initial user input repeat_delay is 0
so the first move
                                //is done without delay (only debouce
applies that is managed by the joystick module)
                                //then for the second tetromino x position
move, we set the wanted delay to the DAS value
                                //and for the rest tetromino x position
moves we set the wanted delay to the ARR value
                                if (repeat_delay_counter == repeat_delay)
begin
                                    repeat_delay_counter <= 0;
      //reached the wanted delay, so reset the counter
                                    if (joy_left && !hit_left) begin
//move left if not hit anything
                                        base_x <= base_x - 1;
                                    end else if (joy_right &&
!hit_right) begin //move right if not hit anything
                                        base_x <= base_x + 1;
                                    end
                                    move_count <= move_count + 1;
//count moves of the tetromino to set ARR and DAS accordingly
                                end else begin // if the delay counter has
not reached the wanted delay, increase it
                                    repeat_delay_counter <=
repeat_delay_counter + 1;
                                end
                                //after the first immediate tetromino x
position move, set the DAS delay
                                if (move_count == 0) begin
                                    repeat_delay <= DAS_DELAY;
                                end else
                                //after the second immediate tetromino x
position move, set the ARR delay
                                if (move_count > 1) begin
                                    repeat_delay <= ARR_DELAY;
                                end

                            end else begin //not left or right joystick
input, so we reset delay counters
                                repeat_delay <= 0; //no delay for the next
input
                                move_count <= 0; //reset amount of
tetromino moves
                                repeat_delay_counter <= 0; //reset delay
counter
                            end
```

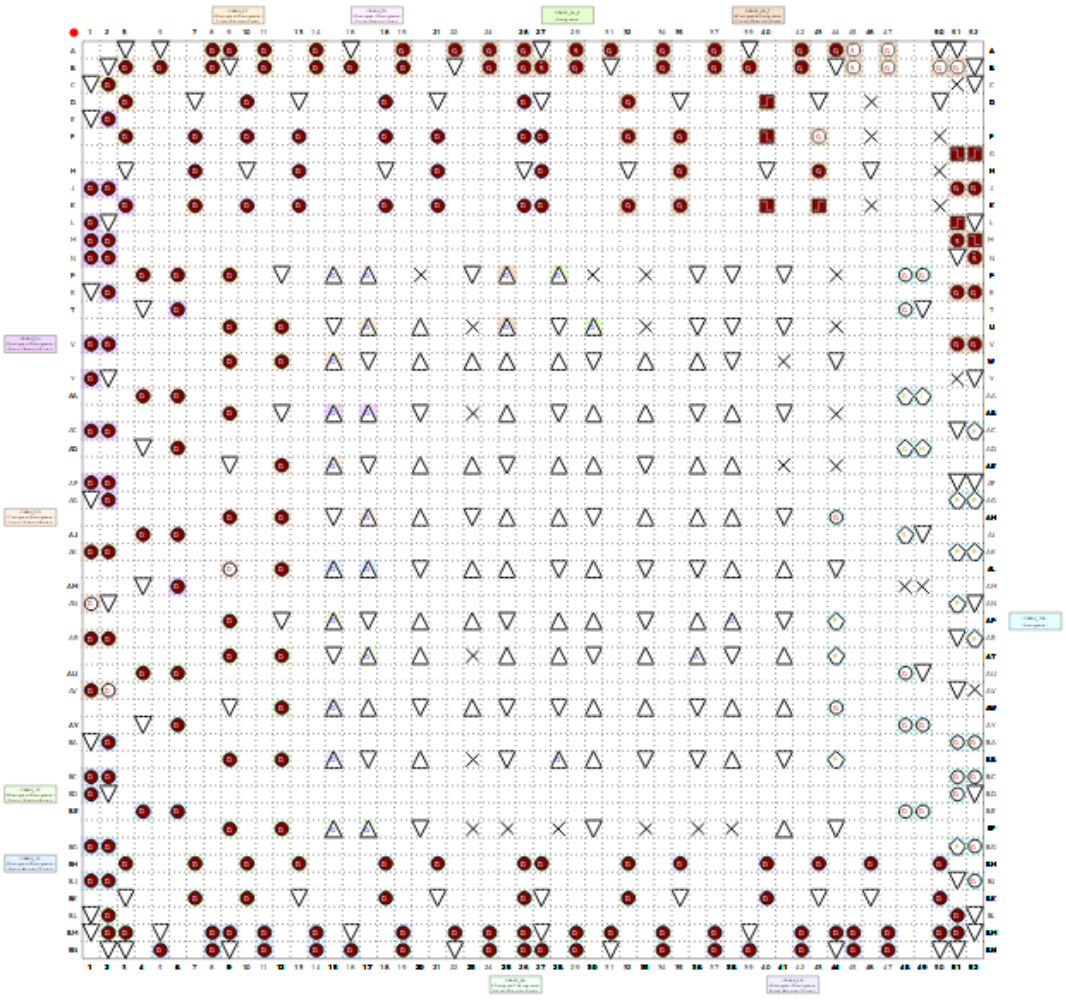
```
//=====================================================================
// Draw moving tetromino
//=====================================================================

                    //clear moving tetromino screen
                    active_piece <= 200'b0;
                    // Compute indices of moving tetromino
                    idx0 = (base_y + dy0) * GRID_COLS + (colx0);
                    idx1 = (base_y + dy1) * GRID_COLS + (colx1);
                    idx2 = (base_y + dy2) * GRID_COLS + (colx2);
                    idx3 = (base_y + dy3) * GRID_COLS + (colx3);
                    //copy indices to active tetromino grid
                    active_piece[idx0] <= 1'b1;
                    active_piece[idx1] <= 1'b1;
                    active_piece[idx2] <= 1'b1;
                    active_piece[idx3] <= 1'b1;

                    // Merged stack grid + moving tetromino grid, is what
the user sees
                    grid_flat <= stack_grid | active_piece;

             end
             //there are full rows, so we ignore user input and dont move
the active tetromino
             //and act only to clear the full rows, one at a time until
no full rows left
             else if (any_row_full) begin
                    //game goes faster on every row clear by increasing
the fall frequency
                    FALL_FREQ <= FALL_FREQ - FALL_INCREASE; // tetrominoes
drop faster
                    //make a sound on line clear
                    sound <= 3'b001; //medium frequency short beep for
line clear

//=====================================================================
// Clear full lines
//=====================================================================


                    //we clear each full line by shifting all the lines
above it one position
                    //down in the stack grid. The top line of the grid it
is cleared
                    //If there are multiple full lines, they are cleared
                    //in sequence at 50mhz so the user sees them clear all
together.
                    //the game speed increases by the FALL_INCREASE amount
for each cleared line

                    // Row 19 (bottom)
                    if (row19_full) begin
                           stack_grid[10 +: 190] <= stack_grid[0 +: 190];
// shift rows 0..18 down, and so on
                    end
                    // Row 18
                    else if (row18_full) begin
                           stack_grid[10 +: 180] <= stack_grid[0 +: 180];
                    end
                    // Row 17
                    else if (row17_full) begin
```

```
                stack_grid[10 +: 170] <= stack_grid[0 +: 170];
            end
            // Row 16
            else if (row16_full) begin
                stack_grid[10 +: 160] <= stack_grid[0 +: 160];
            end
            // Row 15
            else if (row15_full) begin
                stack_grid[10 +: 150] <= stack_grid[0 +: 150];
            end
            // Row 14
            else if (row14_full) begin
                stack_grid[10 +: 140] <= stack_grid[0 +: 140];
            end
            // Row 13
            else if (row13_full) begin
                stack_grid[10 +: 130] <= stack_grid[0 +: 130];
            end
            // Row 12
            else if (row12_full) begin
                stack_grid[10 +: 120] <= stack_grid[0 +: 120];
            end
            // Row 11
            else if (row11_full) begin
                stack_grid[10 +: 110] <= stack_grid[0 +: 110];
            end
            // Row 10
            else if (row10_full) begin
                stack_grid[10 +: 100] <= stack_grid[0 +: 100];
            end
            // Row 9
            else if (row9_full) begin
                stack_grid[10 +: 90] <= stack_grid[0 +: 90];
            end
            // Row 8
            else if (row8_full) begin
                stack_grid[10 +: 80] <= stack_grid[0 +: 80];
            end
            // Row 7
            else if (row7_full) begin
                stack_grid[10 +: 70] <= stack_grid[0 +: 70];
            end
            // Row 6
            else if (row6_full) begin
                stack_grid[10 +: 60] <= stack_grid[0 +: 60];
            end
            // Row 5
            else if (row5_full) begin
                stack_grid[10 +: 50] <= stack_grid[0 +: 50];
            end
            // Row 4
            else if (row4_full) begin
                stack_grid[10 +: 40] <= stack_grid[0 +: 40];
            end
            // Row 3
            else if (row3_full) begin
                stack_grid[10 +: 30] <= stack_grid[0 +: 30];
            end
            // Row 2
            else if (row2_full) begin
                stack_grid[10 +: 20] <= stack_grid[0 +: 20];
```

```
                    end
                    // Row 1
                    else if (row1_full) begin
                           stack_grid[10 +: 10] <= stack_grid[0 +: 10];
                    end
                    // Row 0 (top)
                    else if (row0_full) begin
                           stack_grid[0 +: 10] <= 10'b0;  // nothing
above, just clear
                    end

                    // Merged stack grid + moving tetromino grid, is what
the user sees
                    grid_flat <= stack_grid | active_piece;

             end

    end
end

endmodule
```

Παράρτημα Δ: Pin assignments

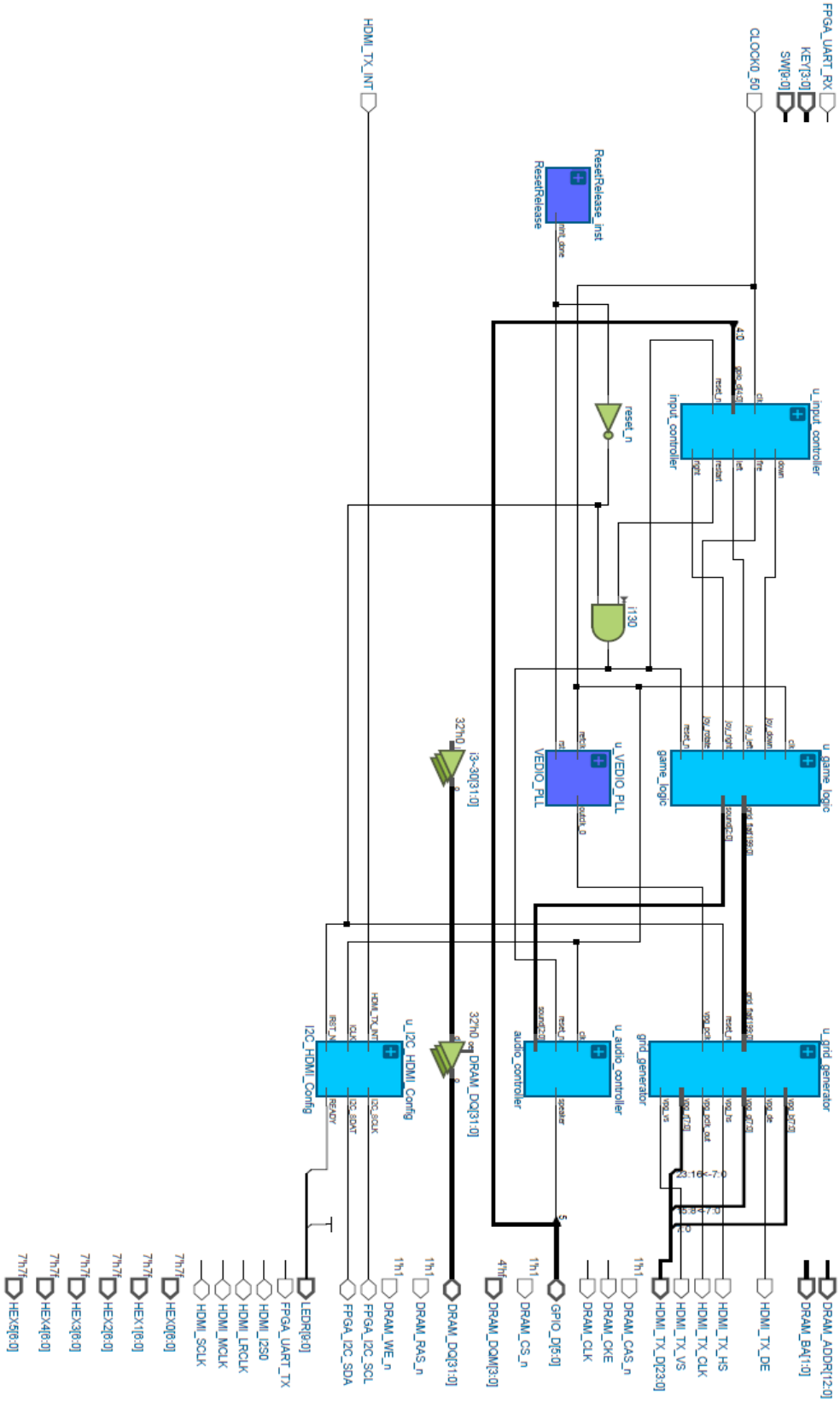
Top View - Flip Chip
Agilex 3 - A3CZ135BB18AE7S



Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard
HEX3[3]	Output	PIN_AY6	6E	PIN_AY6	3.3-V LVCMOS
DRAM_DQ[17]	Bidir	PIN_BC1	6F	PIN_BC1	1.8-V LVCMOS
DRAM_CLK	Output	PIN_BK32	6A	PIN_BK32	1.8-V LVCMOS
HEX3[2]	Output	PIN_BM3	6E	PIN_BM3	3.3-V LVCMOS
DRAM_DQ[16]	Bidir	PIN_BM9	6F	PIN_BM9	1.8-V LVCMOS
DRAM_CKE	Output	PIN_BH40	6B	PIN_BH40	1.8-V LVCMOS
HEX3[1]	Output	PIN_BH13	6E	PIN_BH13	3.3-V LVCMOS
DRAM_DQ[15]	Bidir	PIN_BN45	6B	PIN_BN45	1.8-V LVCMOS
DRAM_CS_n	Output	PIN_BN34	6A	PIN_BN34	1.8-V LVCMOS
HEX3[0]	Output	PIN_BM22	6A	PIN_BM22	1.8-V LVCMOS
DRAM_DQ[14]	Bidir	PIN_BN42	6B	PIN_BN42	1.8-V LVCMOS
DRAM_WE_n	Output	PIN_BM34	6A	PIN_BM34	1.8-V LVCMOS
GPIO_D[5]	Bidir	PIN_K7	6C	PIN_K7	3.3-V LVCMOS
GPIO_D[4]	Bidir	PIN_W9	6C	PIN_W9	3.3-V LVCMOS
GPIO_D[3]	Bidir	PIN_F3	6C	PIN_F3	3.3-V LVCMOS
GPIO_D[2]	Bidir	PIN_D18	6D	PIN_D18	3.3-V LVCMOS
GPIO_D[1]	Bidir	PIN_C2	6C	PIN_C2	3.3-V LVCMOS
GPIO_D[0]	Bidir	PIN_B3	6C	PIN_B3	3.3-V LVCMOS
HEX2[6]	Output	PIN_BN19	6A	PIN_BN19	1.8-V LVCMOS
HEX2[5]	Output	PIN_BN24	6A	PIN_BN24	1.8-V LVCMOS
HEX2[4]	Output	PIN_H35	3A_T	PIN_H35	1.2-V
HEX2[3]	Output	PIN_B29	3A_T	PIN_B29	1.2-V
HEX2[2]	Output	PIN_A31	3A_T	PIN_A31	1.2-V
HEX2[1]	Output	PIN_F35	3A_T	PIN_F35	1.2-V
HEX2[0]	Output	PIN_H43	3A_T	PIN_H43	1.2-V
DRAM_DQ[13]	Bidir	PIN_BM45	6B	PIN_BM45	1.8-V LVCMOS
DRAM_CAS_n	Output	PIN_BK40	6B	PIN_BK40	1.8-V LVCMOS
DRAM_DQ[12]	Bidir	PIN_BM42	6B	PIN_BM42	1.8-V LVCMOS
DRAM_RAS_n	Output	PIN_BM31	6A	PIN_BM31	1.8-V LVCMOS
DRAM_DQ[11]	Bidir	PIN_BM44	6B	PIN_BM44	1.8-V LVCMOS
HDMI_LRCLK	Bidir	PIN_L1	6H	PIN_L1	3.3-V LVCMOS
DRAM_DQ[10]	Bidir	PIN_BN39	6B	PIN_BN39	1.8-V LVCMOS
HDMI_MCLK	Bidir	PIN_T6	6H	PIN_T6	3.3-V LVCMOS
DRAM_DQ[9]	Bidir	PIN_BM37	6B	PIN_BM37	1.8-V LVCMOS
HDMI_SCLK	Bidir	PIN_J2	6H	PIN_J2	3.3-V LVCMOS
DRAM_DQ[8]	Bidir	PIN_BN37	6B	PIN_BN37	1.8-V LVCMOS
HDMI_TX_CLK	Output	PIN_A22	3A_T	PIN_A22	1.2-V
DRAM_DQ[7]	Bidir	PIN_BH46	6B	PIN_BH46	1.8-V LVCMOS
HDMI_TX_HS	Output	PIN_V1	6H	PIN_V1	3.3-V LVCMOS
HDMI_TX_D[23]	Output	PIN_AH12	6G	PIN_AH12	3.3-V LVCMOS
HDMI_TX_D[22]	Output	PIN_AJ6	6G	PIN_AJ6	3.3-V LVCMOS
HDMI_TX_D[21]	Output	PIN_AJ4	6G	PIN_AJ4	3.3-V LVCMOS
HDMI_TX_D[20]	Output	PIN_AH9	6G	PIN_AH9	3.3-V LVCMOS
HDMI_TX_D[19]	Output	PIN_AE12	6G	PIN_AE12	3.3-V LVCMOS
HDMI_TX_D[18]	Output	PIN_AV1	6G	PIN_AV1	3.3-V LVCMOS
HDMI_TX_D[17]	Output	PIN_AR2	6G	PIN_AR2	3.3-V LVCMOS
HDMI_TX_D[16]	Output	PIN_AR1	6G	PIN_AR1	3.3-V LVCMOS
HDMI_TX_D[15]	Output	PIN_AM6	6H	PIN_AM6	3.3-V LVCMOS
HDMI_TX_D[14]	Output	PIN_AK2	6G	PIN_AK2	3.3-V LVCMOS
HDMI_TX_D[13]	Output	PIN_AK1	6G	PIN_AK1	3.3-V LVCMOS
HDMI_TX_D[12]	Output	PIN_AG2	6H	PIN_AG2	3.3-V LVCMOS
HDMI_TX_D[11]	Output	PIN_AF1	6H	PIN_AF1	3.3-V LVCMOS
HDMI_TX_D[10]	Output	PIN_AF2	6H	PIN_AF2	3.3-V LVCMOS
HDMI_TX_D[9]	Output	PIN_AC1	6H	PIN_AC1	3.3-V LVCMOS
HDMI_TX_D[8]	Output	PIN_AB9	6G	PIN_AB9	3.3-V LVCMOS
HDMI_TX_D[7]	Output	PIN_AA4	6G	PIN_AA4	3.3-V LVCMOS
HDMI_TX_D[6]	Output	PIN_AD6	6G	PIN_AD6	3.3-V LVCMOS
HDMI_TX_D[5]	Output	PIN_AA6	6G	PIN_AA6	3.3-V LVCMOS
HDMI_TX_D[4]	Output	PIN_P6	6G	PIN_P6	3.3-V LVCMOS
HDMI_TX_D[3]	Output	PIN_V2	6H	PIN_V2	3.3-V LVCMOS
HDMI_TX_D[2]	Output	PIN_P4	6G	PIN_P4	3.3-V LVCMOS
HDMI_TX_D[1]	Output	PIN_N1	6H	PIN_N1	3.3-V LVCMOS
HDMI_TX_D[0]	Output	PIN_AC2	6H	PIN_AC2	3.3-V LVCMOS
DRAM_DQ[6]	Bidir	PIN_BL51	6B	PIN_BL51	1.8-V LVCMOS
HDMI_TX_VS	Output	PIN_R2	6H	PIN_R2	3.3-V LVCMOS
HEX1[6]	Output	PIN_G52	3A_T	PIN_G52	1.2-V
HEX1[5]	Output	PIN_V52	3A_T	PIN_V52	1.2-V
HEX1[4]	Output	PIN_R52	3A_T	PIN_R52	1.2-V
HEX1[3]	Output	PIN_V51	3A_T	PIN_V51	1.2-V
HEX1[2]	Output	PIN_N52	3A_T	PIN_N52	1.2-V
HEX1[1]	Output	PIN_K40	3A_T	PIN_K40	1.2-V
HEX1[0]	Output	PIN_F40	3A_T	PIN_F40	1.2-V
DRAM_DQ[5]	Bidir	PIN_BH50	6B	PIN_BH50	1.8-V LVCMOS
HDMI_TX_DE	Output	PIN_Y1	6H	PIN_Y1	3.3-V LVCMOS
DRAM_DQ[4]	Bidir	PIN_BM47	6B	PIN_BM47	1.8-V LVCMOS
HDMI_TX_INT	Input	PIN_AL12	6G	PIN_AL12	3.3-V LVCMOS
DRAM_DQ[3]	Bidir	PIN_BN47	6B	PIN_BN47	1.8-V LVCMOS
HDMI_I2S0	Bidir	PIN_M1	6H	PIN_M1	3.3-V LVCMOS
DRAM_DQ[2]	Bidir	PIN_BK50	6B	PIN_BK50	1.8-V LVCMOS
FPGA_I2C_SCL	Bidir	PIN_M2	6H	PIN_M2	3.3-V LVCMOS
DRAM_DQ[1]	Bidir	PIN_BM50	6B	PIN_BM50	1.8-V LVCMOS
FPGA_I2C_SDA	Bidir	PIN_N2	6H	PIN_N2	3.3-V LVCMOS
DRAM_DQ[0]	Bidir	PIN_BM51	6B	PIN_BM51	1.8-V LVCMOS
FPGA_UART_TX	Output	PIN_J1	6H	PIN_J1	3.3-V LVCMOS
DRAM_BA[1]	Output	PIN_BM29	6A	PIN_BM29	1.8-V LVCMOS
DRAM_BA[0]	Output	PIN_BH43	6B	PIN_BH43	1.8-V LVCMOS
FPGA_UART_RX	Input	PIN_BH10	6E	PIN_BH10	3.3-V LVCMOS
HEX0[6]	Output	PIN_R51	3A_T	PIN_R51	1.2-V
HEX0[5]	Output	PIN_M52	3A_T	PIN_M52	1.2-V
HEX0[4]	Output	PIN_M51	3A_T	PIN_M51	1.2-V
HEX0[3]	Output	PIN_J52	3A_T	PIN_J52	1.2-V
HEX0[2]	Output	PIN_G51	3A_T	PIN_G51	1.2-V
HEX0[1]	Output	PIN_J51	3A_T	PIN_J51	1.2-V

Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard
HEX0[0]	Output	PIN_L51	3A_T	PIN_L51	1.2-V
DRAM_ADDR[12]	Output	PIN_BM24	6A	PIN_BM24	1.8-V LVCMOS
DRAM_ADDR[11]	Output	PIN_BN29	6A	PIN_BN29	1.8-V LVCMOS
DRAM_ADDR[10]	Output	PIN_BN27	6A	PIN_BN27	1.8-V LVCMOS
DRAM_ADDR[9]	Output	PIN_BH35	6B	PIN_BH35	1.8-V LVCMOS
DRAM_ADDR[8]	Output	PIN_BK26	6A	PIN_BK26	1.8-V LVCMOS
DRAM_ADDR[7]	Output	PIN_BH32	6A	PIN_BH32	1.8-V LVCMOS
DRAM_ADDR[6]	Output	PIN_BH21	6A	PIN_BH21	1.8-V LVCMOS
DRAM_ADDR[5]	Output	PIN_BK18	6A	PIN_BK18	1.8-V LVCMOS
DRAM_ADDR[4]	Output	PIN_BH26	6A	PIN_BH26	1.8-V LVCMOS
DRAM_ADDR[3]	Output	PIN_BH18	6A	PIN_BH18	1.8-V LVCMOS
DRAM_ADDR[2]	Output	PIN_BM26	6A	PIN_BM26	1.8-V LVCMOS
DRAM_ADDR[1]	Output	PIN_BH27	6A	PIN_BH27	1.8-V LVCMOS
DRAM_ADDR[0]	Output	PIN_BN26	6A	PIN_BN26	1.8-V LVCMOS
LEDR[9]	Output	PIN_D40	3A_T	PIN_D40	1.2-V
LEDR[8]	Output	PIN_B34	3A_T	PIN_B34	1.2-V
LEDR[7]	Output	PIN_A44	3A_T	PIN_A44	1.2-V
LEDR[6]	Output	PIN_A24	3A_T	PIN_A24	1.2-V
LEDR[5]	Output	PIN_B24	3A_T	PIN_B24	1.2-V
LEDR[4]	Output	PIN_A19	3A_T	PIN_A19	1.2-V
LEDR[3]	Output	PIN_F32	3A_T	PIN_F32	1.2-V
LEDR[2]	Output	PIN_K35	3A_T	PIN_K35	1.2-V
LEDR[1]	Output	PIN_B42	3A_T	PIN_B42	1.2-V
LEDR[0]	Output	PIN_B39	3A_T	PIN_B39	1.2-V
HEX5[6]	Output	PIN_K3	6H	PIN_K3	3.3-V LVCMOS
HEX5[5]	Output	PIN_E2	6H	PIN_E2	3.3-V LVCMOS
HEX5[4]	Output	PIN_BB9	6E	PIN_BB9	3.3-V LVCMOS
HEX5[3]	Output	PIN_BB12	6E	PIN_BB12	3.3-V LVCMOS
HEX5[2]	Output	PIN_BF9	6E	PIN_BF9	3.3-V LVCMOS
HEX5[1]	Output	PIN_BF12	6E	PIN_BF12	3.3-V LVCMOS
HEX5[0]	Output	PIN_BH3	6E	PIN_BH3	3.3-V LVCMOS
DRAM_DQM[3]	Output	PIN_BD1	6F	PIN_BD1	1.8-V LVCMOS
DRAM_DQM[2]	Output	PIN_BE6	6F	PIN_BE6	1.8-V LVCMOS
DRAM_DQM[1]	Output	PIN_BA2	6F	PIN_BA2	1.8-V LVCMOS
DRAM_DQM[0]	Output	PIN_BE4	6F	PIN_BE4	1.8-V LVCMOS
SW[9]	Input	PIN_A34	3A_T	PIN_A34	1.2-V
SW[8]	Input	PIN_D32	3A_T	PIN_D32	1.2-V
SW[7]	Input	PIN_A29	3A_T	PIN_A29	1.2-V
SW[6]	Input	PIN_B27	3A_T	PIN_B27	1.2-V
SW[5]	Input	PIN_A26	3A_T	PIN_A26	1.2-V
SW[4]	Input	PIN_K32	3A_T	PIN_K32	1.2-V
SW[3]	Input	PIN_A42	3A_T	PIN_A42	1.2-V
SW[2]	Input	PIN_B26	3A_T	PIN_B26	1.2-V
SW[1]	Input	PIN_B37	3A_T	PIN_B37	1.2-V
SW[0]	Input	PIN_A37	3A_T	PIN_A37	1.2-V
DRAM_DQ[31]	Bidir	PIN_BM19	6F	PIN_BM19	1.8-V LVCMOS
DRAM_DQ[30]	Bidir	PIN_BN14	6F	PIN_BN14	1.8-V LVCMOS
DRAM_DQ[29]	Bidir	PIN_BN16	6F	PIN_BN16	1.8-V LVCMOS
DRAM_DQ[28]	Bidir	PIN_BM11	6F	PIN_BM11	1.8-V LVCMOS
DRAM_DQ[27]	Bidir	PIN_BM14	6F	PIN_BM14	1.8-V LVCMOS
DRAM_DQ[26]	Bidir	PIN_BM8	6F	PIN_BM8	1.8-V LVCMOS
DRAM_DQ[25]	Bidir	PIN_BN11	6F	PIN_BN11	1.8-V LVCMOS
DRAM_DQ[24]	Bidir	PIN_BN8	6F	PIN_BN8	1.8-V LVCMOS
DRAM_DQ[23]	Bidir	PIN_BG1	6F	PIN_BG1	1.8-V LVCMOS
DRAM_DQ[22]	Bidir	PIN_BC2	6F	PIN_BC2	1.8-V LVCMOS
DRAM_DQ[21]	Bidir	PIN_BG2	6F	PIN_BG2	1.8-V LVCMOS
DRAM_DQ[20]	Bidir	PIN_BJ2	6F	PIN_BJ2	1.8-V LVCMOS
DRAM_DQ[19]	Bidir	PIN_BJ1	6F	PIN_BJ1	1.8-V LVCMOS
DRAM_DQ[18]	Bidir	PIN_BN5	6F	PIN_BN5	1.8-V LVCMOS
HEX4[6]	Output	PIN_BH7	6E	PIN_BH7	3.3-V LVCMOS
HEX4[5]	Output	PIN_BL2	6E	PIN_BL2	3.3-V LVCMOS
HEX4[4]	Output	PIN_BM2	6E	PIN_BM2	3.3-V LVCMOS
HEX4[3]	Output	PIN_BK7	6E	PIN_BK7	3.3-V LVCMOS
HEX4[2]	Output	PIN_AP9	6E	PIN_AP9	3.3-V LVCMOS
HEX4[1]	Output	PIN_AT12	6E	PIN_AT12	3.3-V LVCMOS
HEX4[0]	Output	PIN_AT9	6E	PIN_AT9	3.3-V LVCMOS
KEY[3]	Input	PIN_W12	6C	PIN_W12	3.3-V LVCMOS
KEY[2]	Input	PIN_BK10	6E	PIN_BK10	3.3-V LVCMOS
KEY[1]	Input	PIN_K27	6D	PIN_K27	3.3-V LVCMOS
KEY[0]	Input	PIN_K26	6D	PIN_K26	3.3-V LVCMOS
HEX3[6]	Output	PIN_AW12	6E	PIN_AW12	3.3-V LVCMOS
HEX3[5]	Output	PIN_AU4	6E	PIN_AU4	3.3-V LVCMOS
HEX3[4]	Output	PIN_AU6	6E	PIN_AU6	3.3-V LVCMOS
CLOCK0_50	Input	PIN_K43	3A_T	PIN_K43	1.2-V
CLOCK1_50	Unknown	PIN_A8	6C		3.3-V LVCMOS
GPIO_D[6]	Unknown	PIN_F7	6D		3.3-V LVCMOS
GPIO_D[7]	Unknown	PIN_H7	6D		3.3-V LVCMOS
GPIO_D[8]	Unknown	PIN_B8	6C		3.3-V LVCMOS
GPIO_D[9]	Unknown	PIN_B5	6C		3.3-V LVCMOS
GPIO_D[10]	Unknown	PIN_K10	6D		3.3-V LVCMOS
GPIO_D[11]	Unknown	PIN_F10	6D		3.3-V LVCMOS
GPIO_D[12]	Unknown	PIN_F13	6D		3.3-V LVCMOS
GPIO_D[13]	Unknown	PIN_D10	6D		3.3-V LVCMOS
GPIO_D[14]	Unknown	PIN_K13	6D		3.3-V LVCMOS
GPIO_D[15]	Unknown	PIN_H13	6D		3.3-V LVCMOS
GPIO_D[16]	Unknown	PIN_U9	6C		3.3-V LVCMOS
GPIO_D[17]	Unknown	PIN_D3	6C		3.3-V LVCMOS
GPIO_D[18]	Unknown	PIN_B11	6C		3.3-V LVCMOS
GPIO_D[19]	Unknown	PIN_A9	6C		3.3-V LVCMOS
GPIO_D[20]	Unknown	PIN_B14	6C		3.3-V LVCMOS

Παράρτημα Ε: RTL Analyzer



Παράρτημα ΣΤ: Εγχειρίδιο χρήσης παιχνιδιού

ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΠΑΙΧΝΙΔΙΟΥ TETRIS

1. Εισαγωγή

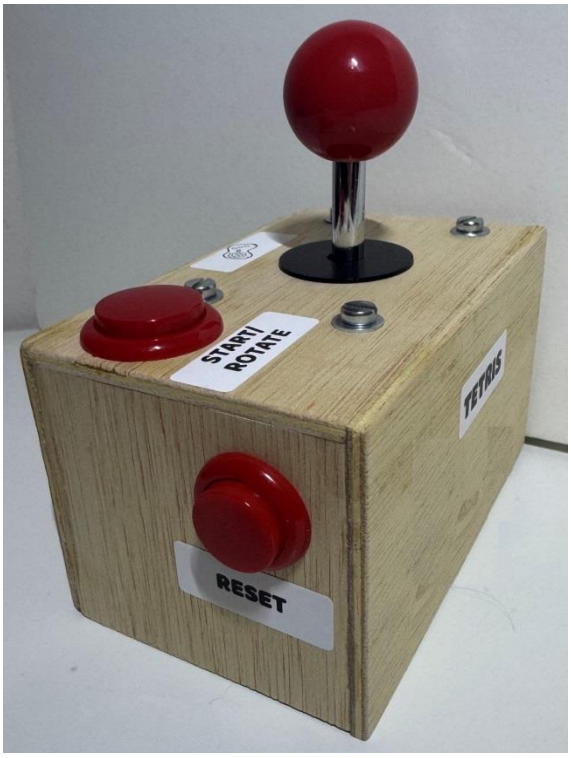
Το παρόν εγχειρίδιο περιγράφει τη χρήση του συστήματος Tetris, το οποίο έχει υλοποιηθεί σε πλατφόρμα FPGA και υποστηρίζει είσοδο μέσω arcade χειριστηρίου και έξοδο εικόνας μέσω διεπαφής HDMI. Στόχος του εγχειριδίου είναι να παρέχει στον χρήστη τις απαραίτητες πληροφορίες για τη σωστή εκκίνηση, τον χειρισμό και την κατανόηση της λειτουργίας του παιχνιδιού.

2. Σκοπός του παιχνιδιού

Σκοπός του παιχνιδιού είναι η σωστή τοποθέτηση των tetrominoes στο πλέγμα, με στόχο τη δημιουργία πλήρων οριζόντιων γραμμών. Όταν μια γραμμή συμπληρωθεί πλήρως, διαγράφεται και τα ανώτερα κομμάτια μετακινούνται προς τα κάτω, αυξάνοντας τη διάρκεια του παιχνιδιού. Το παιχνίδι τερματίζεται όταν το πλέγμα γεμίσει και δεν υπάρχει διαθέσιμος χώρος για την εμφάνιση νέου tetromino.

3. Χειρισμός παιχνιδιού

Η αλληλεπίδραση του χρήστη με το σύστημα πραγματοποιείται μέσω arcade χειριστηρίου. Η διάταξη των πλήκτρων του χειριστηρίου παρουσιάζεται στην παρακάτω Εικόνα:



Οι βασικές λειτουργίες χειρισμού είναι οι εξής:

- Έναρξη του παιχνιδιού πατώντας το κουμπί START/ROTATE.
- Επαναφορά του παιχνιδιού πατώντας το κουμπί RESET.
- Μετακίνηση αριστερά και δεξιά για αλλαγή θέσης του tetromino στο πλέγμα μετακινώντας το μοχλό δεξιά/αριστερά.
- Περιστροφή του tetromnino πατώντας το κουμπί START/ROTATE.
- Επιτάχυνση πτώσης μετακινώντας το μοχλό πρός τα κάτω

Η απόκριση του συστήματος στις εντολές είναι άμεση, με ενσωματωμένους μηχανισμούς αποθορυβοποίησης ώστε να αποφεύγονται λανθασμένες ή πολλαπλές ενεργοποιήσεις.

4. Κανόνες παιχνιδιού

Κατά τη διάρκεια του παιχνιδιού ισχύουν οι εξής βασικοί κανόνες:

- Κάθε tetromino κινείται προς τα κάτω με σταθερό ρυθμό πτώσης
- Η κίνηση σταματά όταν υπάρξει επαφή με άλλο κομμάτι ή με το κάτω όριο του πλέγματος
- Μετά την ακινητοποίηση, το τετρασχήμα ενσωματώνεται στο ήδη διαμορφωμένο πλέγμα
- Όταν συμπληρωθεί πλήρης οριζόντια γραμμή, αυτή διαγράφεται και τα ανώτερα κομμάτια μετακινούνται προς τα κάτω

- Το παιχνίδι ολοκληρώνεται όταν δεν υπάρχει διαθέσιμος χώρος για νέο tetromino στην κορυφή του πλέγματος

5. Σημειώσεις λειτουργίας

- Η απόκριση του συστήματος είναι σχεδιασμένη για πραγματικό χρόνο
- Η ταχύτητα πτώσης των tetrominos αυξάνεται σταδιακά με την πρόοδο του παιχνιδιού
- Η σωστή λειτουργία εξαρτάται από τη σταθερή παροχή ρολογιού και την ορθή σύνδεση του χειριστηρίου και της εξόδου εικόνας

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.