Σχολή Θετικών Επιστημών και Τεχνολογίας (ΣΘΕΤ)

Τμήμα Πληροφορικής

Πτυχιακή Εργασία

Ελεγκτής VGA βασισμένος σε μικροελεγκτή περιορισμένων πόρων

Παναγιώτης Αργυρόπουλος

Επιβλέπων καθηγητής: Αθανάσιος Κακαρούντας

Πάτρα, Μάιος 2026

Ελεγκτής VGA βασισμένος σε μικροελεγκτή περιορισμένων πόρων

Παναγιώτης Αργυρόπουλος

Επιτροπή Επίβλεψης Πτυχιακής Εργασίας

Επιβλέπων Καθηγητής:

Αθανάσιος Κακαρούντας

Καθηγητής, Τμ. Πληροφορικής με Εφαρμογές στη Βιοϊατρική,

Πανεπιστήμιο Θεσσαλίας

Συν-Επιβλέπων Καθηγητής:

Ευάγγελος Τοπάλης

Ερευνητής, Τμ. Ηλεκτρολόγων Μηχ. & Τεχνολογίας Η/Υ

Πανεπιστήμιο Πατρών

Συν-Επιβλέπων Καθηγητής:

Βασίλειος Πλαγιανάκος

Καθηγητής, Τμ. Πληροφορικής με Εφαρμογές στη Βιοϊατρική,

Πανεπιστήμιο Θεσσαλίας

Πάτρα, Μαίος 2026

«Αφιέρωση – Ευχαριστίες»

Στη Μαρία…

Περίληψη

Η παρούσα πτυχιακή εργασία πραγματεύεται τη μελέτη, τη σχεδίαση και την ολοκληρωμένη υλοποίηση ενός αυτόνομου ελεγκτή VGA, βασισμένου στον 8-bit μικροελεγκτή PIC18F47K42 της Microchip. Το σύστημα λειτουργεί ως μια ανεξάρτητη κάρτα γραφικών, σχεδιασμένη να αποτελεί εξωτερικό περιφερειακό υποσύστημα για embedded εφαρμογές περιορισμένων πόρων, επιτρέποντας την οπτική απεικόνιση δεδομένων χωρίς να επιβαρύνεται ο κεντρικός επεξεργαστής του συστήματος.

Η εργασία καλύπτει αναλυτικά τις τεχνικές προδιαγραφές του προτύπου VGA, ενώ επικεντρώνεται στον σχεδιασμό και την υλοποίηση όλων των απαραίτητων λειτουργικών μονάδων για την υποστήριξή του. Κεντρικό σημείο της αρχιτεκτονικής υλικού αποτελεί η μονάδα παραγωγής σημάτων συγχρονισμού, η οποία αξιοποιεί εσωτερικούς απαριθμητές (timers) και μονάδες PWM για τη δημιουργία των παλμών συγχρονισμού (HSync, VSync) με υψηλή ακρίβεια. Η παραγωγή του αναλογικού σήματος χρώματος (4-bit RGBI) επιτυγχάνεται μέσω εξωτερικού κυκλώματος που συνδυάζει πολυπλέκτες 74AC257 και R-2R ψηφιοαναλογικούς μετατροπείς (DAC). Το κύκλωμα οδηγείται από μια βοηθητική μονάδα παραγωγής σήματος ορατής περιοχής, η οποία αξιοποιεί λογικές κυψέλες (CLC) και γεννήτριες PWM για να δημιουργήσει παλμό ενεργοποίησης κατάλληλου εύρους, διασφαλίζοντας ότι το αναλογικό σήμα παράγεται μόνο εντός του ενεργού τμήματος σάρωσης της οθόνης.

Στο επίπεδο του λογισμικού υιοθετήθηκε μια πολυεπίπεδη αρχιτεκτονική (layered architecture), η οποία προάγει τη συντηρησιμότητα και την επεκτασιμότητα του συστήματος. Η επικοινωνία με τον ελεγκτή πραγματοποιείται μέσω της διεπαφής UART, από την οποία λαμβάνονται εντολές και αποστέλλονται οι αντίστοιχες απαντήσεις. Η διαχείριση των δεδομένων υλοποιείται ασύγχρονα μέσω μηχανισμών Circular Buffer σε συνεργασία με τη μονάδα DMA, ενώ για τη διαχείριση συμβάντων και τη διερμήνευση εντολών εφαρμόστηκαν τα σχεδιαστικά πρότυπα Reactor και State Pattern αντίστοιχα. Ιδιαίτερη έμφαση δόθηκε στη βελτιστοποίηση της απόδοσης, με την υλοποίηση της μονάδας Pixel Generator σε γλώσσα Assembly, εξασφαλίζοντας ρυθμό παραγωγής ενός pixel ανά κύκλο.

Ο ελεγκτής υποστηρίζει τρεις διακριτές καταστάσεις λειτουργίας (Text, Plot και Graphics Mode), ενσωματώνοντας αλγορίθμους Bresenham για τη σχεδίαση γεωμετρικών σχημάτων και τεχνικές χρωματικής συμπίεσης για την προβολή εικόνων παρά τους περιορισμούς μνήμης του μικροελεγκτή.

Η λειτουργικότητα και η σταθερότητα του συστήματος επιβεβαιώθηκαν μέσω στοχευμένων stress tests, κατά τα οποία ο ελεγκτής λειτούργησε υπό αυξημένο φορτίο και συνεχείς ακολουθίες εντολών, διατηρώντας πλήρως σταθερή εικόνα. Οι μετρήσεις παλμογράφου έδειξαν ότι οι χρονισμοί των παραγόμενων σημάτων παρέμειναν εντός των προδιαγραφών του προτύπου VGA, με αποκλίσεις μικρότερες του ±0.5%, διασφαλίζοντας πλήρη συμβατότητα με σύγχρονες οθόνες VGA.

Λέξεις κλειδιά

VGA, NTSC, PWM, SPI, CLC, DAC, R-2R, DMA, Multiplexing, Reactor Pattern, State Pattern, Circular Buffer, Layered Architecture, Embedded Systems, Event-Driven Design

Abstract

This BSc thesis details the research, design, and comprehensive implementation of an autonomous VGA controller based on the 8-bit PIC18F47K42 microcontroller by Microchip. The system functions as a standalone graphics card, specifically designed to serve as an external peripheral subsystem for resource-constrained embedded applications, enabling visual data representation without burdening the host processor.

The thesis provides a detailed analysis of the technical specifications of the VGA standard and focuses on the design and implementation of all functional modules required for its support. A central component of the hardware architecture is the synchronization signal generator, which employs internal timers and PWM units to produce the HSync and VSync pulses with high precision. The analog color signal (4-bit RGBI) is generated by an external circuit combining 74AC257 multiplexers with R-2R digital-to-analog converters (DAC). This circuit is driven by an auxiliary visible-area signal generator that uses configurable logic cells (CLC) and PWM modules to produce an activation pulse of appropriate width, ensuring that the analog output is enabled only during the active portion of the display scan.

At the software level, a layered architecture was adopted to enhance system maintainability and scalability. Communication with the controller is handled via a UART interface, utilizing an asynchronous data management scheme based on circular buffers in conjunction with the DMA unit. For event handling and command interpretation, the Reactor and State design patterns were applied, respectively. Performance optimization was prioritized through the implementation of the Pixel Generator in Assembly, ensuring a strict throughput of one pixel per cycle.

The controller supports three distinct operating modes—Text, Plot, and Graphics Mode—incorporating Bresenham algorithms for geometric rendering and color compression techniques for image display despite the microcontroller’s memory constraints.

The functionality and stability of the system were verified through targeted stress tests, during which the controller operated under increased load and continuous command sequences while maintaining a fully stable image. Oscilloscope measurements confirmed that the timing of the generated signals remained within the VGA specification, with deviations below ±0.5%, ensuring full compatibility with modern VGA displays.

Keywords

VGA, NTSC, PWM, SPI, CLC, DAC, R-2R, DMA, Multiplexing, Reactor Pattern, State Pattern, Circular Buffer, Layered Architecture, Embedded Systems, Event-Driven Design

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Διαγραμμάτων

Κατάλογος Πινάκων

Συντομογραφίες - Ακρωνύμια

Ονομασία	Περιγραφή
AV	Active Video Area
BP	Back Porch
CLC	Configurable Logic Cell - Λογική Κυψέλη
FP	Front Porch
H_AV	Horizontal Active Video Area
HBP	Horizontal Back Porch
HSync	Horizontal Synchronization Signal
INT	Interrupt
PPS	Peripheral Pin Select
PWM	Pulse Width Modulation – Διαμόρφωση Πλάτους Παλμού
RGB	Red, Green, Blue
SPI	Serial Peripheral Interface
Sync	Synchronization Pulse
UART	Universal Asynchronous Receiver Transmitter
V_AV	Vertical Active Video Area
VBP	Vertical Back Porch
VGA	Video Graphics Array
VSync	Vertical Synchronization Signal
FSM	Finite State Machine (Μηχανή Πεπερασμένων Καταστάσεων)
CFG	Context Free Grammar (Γραμματική Χωρίς Συμφραζόμενα)
BNF	Backus-Naur Form (Μορφή Μπάκους-Νάουρ)
FPU	Floating Point Unit

1. Εισαγωγή

1.1 Ανάλυση του Προβλήματος

Τα ενσωματωμένα συστήματα περιορισμένων πόρων, όπως αυτά που βασίζονται σε 8-bit μικροελεγκτές της Microchip (π.χ. Arduino UNO, Arduino Mega), χρησιμοποιούνται ευρέως σε εφαρμογές όπου απαιτούνται μικρό μέγεθος, χαμηλό κόστος και περιορισμένη κατανάλωση ενέργειας. Αν και οι μικροελεγκτές αυτοί προσφέρουν σημαντικές δυνατότητες ελέγχου, επικοινωνίας και αλληλεπίδρασης με περιφερειακά, στερούνται ενσωματωμένων γραφικών μονάδων και δεν παρέχουν υποστήριξη για παραγωγή video εξόδου ή σύνδεση με οθόνες υψηλής ανάλυσης.

Όταν απαιτείται οπτική απεικόνιση δεδομένων, οι περισσότεροι χρήστες καταφεύγουν στη χρήση απλών LCD οθονών δυο γραμμών ή μικρών OLED γραφικών οθονών, οι οποίες περιορίζονται σε πολύ μικρές διαστάσεις (1"–2") και χαμηλή ανάλυση. Αυτές οι λύσεις, παρότι εύχρηστες, δεν επαρκούν για πιο απαιτητικές απεικονίσεις, όπως διαγράμματα, πολύχρωμα στοιχεία και δυναμικά δεδομένα.

Το πρότυπο VGA (Video Graphics Array), το οποίο υπήρξε για σχεδόν τρεις δεκαετίες το de facto, standard και υποστηρίζεται ακόμη και σήμερα από την πλειοψηφία του σύγχρονων οθονών που διαθέτουν τη σχετική θύρα εισόδου, δίνει τη δυνατότητα απεικόνισης δεδομένων με ανάλυση 640x480 pixels. Η δυσκολία υλοποίησης μιας γραφικής μονάδας που υποστηρίζει το πρότυπο, έγκειται στο ότι προϋποθέτει:

- Την παραγωγή ψηφιακών σημάτων συγχρονισμού πολύ υψηλής ακρίβειας (HSync, VSync),
- Συνεχή ροή pixel δεδομένων σε υψηλές ταχύτητες,
- Αναλογική έξοδο RGB, που απαιτεί επιπλέον κυκλώματα DAC (ψηφιοαναλογικούς μετατροπείς).

Αυτές οι απαιτήσεις θεωρούνται συχνά απαγορευτικές για μικροελεγκτές 8-bit, εξαιτίας των περιορισμών τους σε χρονισμό, ταχύτητα, μνήμη και γενική υπολογιστική ικανότητα.

Για τον λόγο αυτό, πλακέτες επέκτασης που υλοποιούν το πρότυπο VGA, βασίζονται σε 32-bit μικροελεγκτές, ARM Cortex-M, ή FPGAs, γεγονός που εισάγει υλικό πολλαπλάσιας πολυπλοκότητας και κόστους σε σχέση με τον βασικό μικροελεγκτή του συστήματος.

Το γεγονός αυτό δημιουργεί μια σημαντική αντίφαση: ένας απλός 8 bit μικροελεγκτής, όπως αυτός που φιλοξενείται σε μια αναπτυξιακή πλακέτα Arduino, να χρησιμοποιεί ως εξωτερικό περιφερειακό, μια πολλαπλά ισχυρότερη μονάδα. Αυτό συνιστά ένα σχεδιαστικό παράδοξο, καθώς αντιστρέφει τη συνήθη ιεραρχία ισχύος μεταξύ κεντρικής μονάδας και υποστηρικτικών περιφερειακών.

1.2 Σκοπός της Εργασίας

Σκοπός της παρούσας εργασίας είναι η μελέτη, σχεδίαση και υλοποίηση ενός αυτόνομου έγχρωμου VGA ελεγκτή, βασισμένου αποκλειστικά σε μικροελεγκτή περιορισμένων πόρων και συγκεκριμένα σε μικροελεγκτή 8-bit της Microchip.

Αν και η χρήση ενός ισχυρού 32/64-bit μικροελεγκτή ή ενός FPGA θα καθιστούσε την υλοποίηση σημαντικά ευκολότερη, θα αναιρούσε τον ίδιο τον σκοπό της εργασίας, ο οποίος δεν ήταν η δημιουργία ενός πανίσχυρου VGA controller, αλλά η απόδειξη ότι ένας μικροελεγκτής χαμηλών δυνατοτήτων, με έξυπνη αξιοποίηση των περιφερειακών του και καλό προγραμματισμό, μπορεί να υποστηρίξει τις αυστηρές απαιτήσεις του VGA προτύπου και να παράγει εικόνα.

1.3 Βασικές Απαιτήσεις

Ο ελεγκτής VGA που θα υλοποιηθεί θα πρέπει να υποστηρίζει τις παρακάτω λειτουργίες:

1. Διεπαφή Εισόδου: Θα δέχεται δεδομένα από μικροελεγκτές ή αναπτυξιακές πλακέτες μέσω ενός καθορισμένου πρωτοκόλλου επικοινωνίας (π.χ., SPI, I2C, UART)
2. Επεξεργασία & Μετατροπή: Θα επεξεργάζεται αυτά τα δεδομένα για να παράγει τα απαραίτητα σήματα για την οδήγηση μιας οθόνης VGA.

3. Διεπαφή Εξόδου (VGA):
    - Αναλογικά Σήματα Χρωμάτων (RGB): Θα παράγει τα αναλογικά σήματα για το Κόκκινο (Red), Πράσινο (Greek) και Μπλε (Blue) χρώμα. Αυτό απαιτεί τη χρήση Ψηφιοαναλογικού Μετατροπέα (DAC), είτε ενσωματωμένου στον μικροελεγκτή είτε υλοποιημένου με εξωτερικά εξαρτήματα (π.χ., δίκτυο αντιστατών R-2R).
    - Ψηφιακά Σήματα Συγχρονισμού: Θα παράγει τα ψηφιακά σήματα Οριζόντιου Συγχρονισμού (Horizontal Sync - HSync) και Κάθετου Συγχρονισμού (Vertical Sync - VSync) με πολύ ακριβείς χρονισμούς, οι οποίοι είναι θεμελιώδεις για τη σωστή απεικόνιση στην οθόνη.

Η εργασία αποσκοπεί στα εξής:

- Εμβάθυνση στα ενσωματωμένα συστήματα και στη σχεδίαση ψηφιακών διατάξεων.

- Διερεύνηση του τρόπου με τον οποίο με τον οποίο τα ψηφιακά και αναλογικά σήματα παράγονται και διαμορφώνονται σε πραγματικό χρόνο, από ένα ενσωματωμένο σύστημα περιορισμένων πόρων.

2. Θεωρητικό Υπόβαθρο

2.1 Το πρότυπο VGA

Το πρότυπο VGA και ο αντίστοιχος ελεγκτής αναπτύχθηκε από την IBM το 1987 [1] και μέσα στα επόμενα τρία έτη κυριάρχησε στην αγορά των IBM συμβατών υπολογιστών. Υποστήριζε ανάλυση 640x480 με 16 χρώματα και χρησιμοποιούσε διεπαφή εισόδου 15 ακροδεκτών.

Το πρότυπο είχε σε μεγάλο βαθμό βασιστεί στο παλαιότερο και καλά εδραιωμένο πρότυπο αναλογικής τηλεόρασης NTSC [2], διατηρώντας πλήθος τεχνικών χαρακτηριστικών, ώστε να διευκολύνεται η δημιουργία μετατροπέων VGA to TV. Διατήρησε την ίδια κάθετη ανάλυση 525 οριζόντιων γραμμών, εκ των οποίων ορατές είναι μόνο οι 480, ενώ επέλεξε ως συχνότητα οριζόντιας σάρωσης τα 31.469 KHz, ακριβώς διπλάσια δηλαδή από αυτή του NTSC προτύπου.

Η αρχή λειτουργίας του βασίζεται στη σειριακή σάρωση της οθόνης, γραμμή προς γραμμή (από αριστερά προς τα δεξιά, από πάνω προς τα κάτω). Για τον σωστό συγχρονισμό με την οθόνη, παράγονται δυο σήματα συγχρονισμού, των οποίων το ενεργό τμήμα βρίσκεται σε στάθμη λογικού μηδέν:

- Ο παλμός οριζόντιου συγχρονισμού (HSYNC), όπου ενεργοποιείται στην αρχή κάθε γραμμής, και
- Ο παλμός κατακόρυφου συγχρονισμού (VSYNC), όπου ενεργοποιείται στο τέλος κάθε καρέ για να επανεκκινήσει η σάρωση από την κορυφή της οθόνης.

Επιπλέον παράγει τρία αναλογικά σήματα στάθμης 0V έως και 0.7V τα οποία αντιστοιχούν στα χρωματικά κανάλια RGB (Red, Green, Blue).

2.2 Σήματα Συγχρονισμού

Το πρότυπο VGA επιβάλλει αυστηρές προδιαγραφές ως προς τη συχνότητα και τη δομή του παραγόμενου σήματος, το οποίο αποτελείται από δύο βασικές συνιστώσες: τον οριζόντιο και τον κατακόρυφο παλμό συγχρονισμού. Ο σωστός χρονισμός εξασφαλίζει τη σταθερή και αξιόπιστη εμφάνιση της εικόνας στην οθόνη, συγχρονίζοντας την εμφάνιση κάθε γραμμής (scanline) και κάθε καρέ (frame). Όλα τα επιμέρους χρονικά διαστήματα βασίζονται σε μια ενιαία συχνότητα ρολογιού, γνωστή ως Pixel Clock.

Η συχνότητα του Pixel Clock είναι 25,175 MHz. Με βάση αυτή τη συχνότητα, υπολογίζονται τόσο ο οριζόντιος όσο και ο κατακόρυφος ρυθμός ανανέωσης, οι οποίοι είναι κρίσιμοι για την ορθή ανανέωση της εικόνας στην οθόνη.

Συγκεκριμένα, το πρότυπο VGA ορίζει τα εξής:

Οριζόντιος Ρυθμός Ανανέωσης	31.469 KHz
Κατακόρυφος Ρυθμός Ανανέωσης	59,94 Hz
Pixel Clock	25,175 MHz

Πίνακας 1: Συχνότητα Ανανέωσης

2.2.1 Σήμα Οριζόντιου Συγχρονισμού

Κάθε γραμμή (scanline) αποτελείται από τα ακόλουθα τμήματα:

1. **Ενεργό μέρος (Active Area)**: Το ενεργό τμήμα κάθε γραμμής που αντιστοιχεί σε 640 pixels.
2. **Front Porch**: Μικρό χρονικό διάστημα πριν το ενεργό τμήμα ($\overline{\text{ON Time}}$) του παλμού.
3. **Sync Part**: Το διάστημα κατά το οποίο ο παλμός είναι ενεργός, σε στάθμη λογικού μηδέν.
4. **Back Porch**: Διάστημα μετά το ενεργό τμήμα του παλμού.

Τμήμα Scanline	Πλήθος Κύκλων	Διάρκεια (μsec)
Active Area	640	25,42
Front Porch	16	0,64
Sync Part	96	3,81
Back Porch	48	1,91
Scanline	800	31,78

Πίνακας 2: Οριζόντιος Χρονισμός

2.2.2 Σήμα Κατακόρυφου Συγχρονισμού

Η ίδια λογική εφαρμόζεται και για το κατακόρυφο επίπεδο. Κάθε πλήρες καρέ (frame) αποτελείται από τα ακόλουθα τμήματα:

1. **Ενεργό μέρος (Active Area)**: αποτελούμενο από 480 γραμμές.
2. **Front Porch**: Μικρό χρονικό διάστημα πριν το ενεργό τμήμα του παλμού.
3. **Sync Part**: Το διάστημα κατά το οποίο ο παλμός είναι ενεργός, σε στάθμη λογικού μηδέν ($\overline{\text{ON Time}}$).
4. **Back Porch**: Το διάστημα μετά το ενεργό τμήμα του παλμού και αμέσως πριν ξεκινήσει το νέο frame.

Τμήμα Καρέ	Πλήθος Γραμμών	Διάρκεια (msec)
Active Area	480	15,25
Front Porch	10	0,32
Sync Part	2	0,06
Back Porch	33	1,05
Πλήρες Καρέ	525	16,68

Πίνακας 3: Κατακόρυφος Χρονισμός

2.3 Αναλογικά Σήματα RGB

Η μετάδοση της πληροφορίας του χρώματος πραγματοποιείται μέσω αναλογικού σήματος RGB (Red, Green, Blue). Το συνολικό σήμα διαχωρίζεται σε τρία ανεξάρτητα κανάλια, τα οποία μεταφέρουν την πληροφορία για κάθε βασικό χρώμα:

- Κανάλι Κόκκινου (Red)
- Κανάλι Πράσινου (Green)
- Κανάλι Μπλε (Blue)

Η στάθμη του σήματος κάθε καναλιού, κυμαίνεται από 0,00 V έως 0,70 V, όπου:

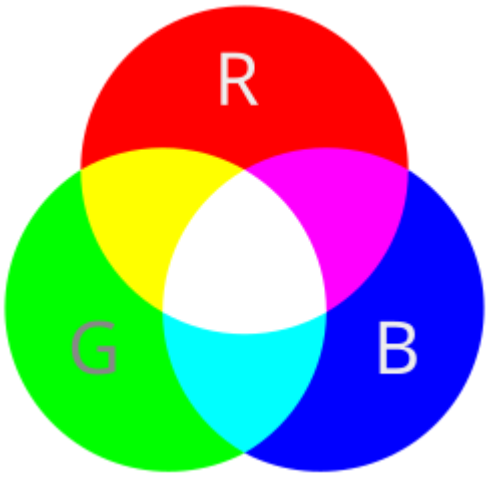
- 0,00 V αντιστοιχεί σε μηδενική ένταση φωτεινότητας (απουσία χρώματος),
- 0,35 V υποδηλώνει μεσαία φωτεινότητα (ημι-φωτεινό σήμα),
- 0,70 V δηλώνει μέγιστη ένταση φωτεινότητας (πλήρης ενεργοποίηση του καναλιού).

Στάθμη (Volt)	Φωτεινότητα
0,00	Ανυπαρξία Χρώματος (Μαύρο)
0,35	Χρώμα Μεσαίας Φωτεινότητας
0,70	Χρώμα Μέγιστης Φωτεινότητας

Πίνακας 4: Στάθμη Σήματος RGB

Το εύρος αυτό παράγεται από ψηφιοαναλογικό μετατροπέα (DAC) που μετατρέπει την ψηφιακή τιμή κάθε χρώματος στο αντίστοιχο αναλογικό επίπεδο τάσεως. Είναι προφανές ότι απαιτούνται τρεις DAC, ένας για κάθε χρωματικό κανάλι.

Η συνδυαστική ενεργοποίηση των καναλιών RGB οδηγεί στην προσθετική σύνθεση χρώματος [3], κάτι που επιτρέπει τη δημιουργία ενός συνολικά μεγάλου χρωματικού φάσματος.



Εικόνα 1: Προσθετική Μίξη Χρωμάτων

Ενδεικτικά, τα παρακάτω επίπεδα στάθμης οδηγούν στη δημιουργία 8 διαφορετικών χρωμάτων:

R	G	B	Χρώμα
0,00	0,00	0,00	
0,00	0,00	0,47	
0,00	0,47	0,00	
0,00	0,47	0,47	
0,47	0,00	0,00	
0,47	0,00	0,47	
0,47	0,47	0,00	
0,47	0,47	0,47	

Πίνακας 5: Ενδεικτικός Χρωματικός Χάρτης

3. Ανάλυση Απαιτήσεων Συστήματος

3.1 Διεπαφή Εξόδου

Το πρότυπο προϋποθέτει τη χρήση ενός ρολογιού 25.175 MHz. Μετά από έρευνα διαπιστώθηκε ότι δεν υπάρχει διαθεσιμότητα στο εμπόριο σε αντίστοιχους παθητικούς κρυστάλλους χρονισμού, ενώ οι ενεργές μονάδες παραγωγής παλμών αυτής της συχνότητας παρουσιάζουν χαμηλή διαθεσιμότητα και υψηλό κόστος. Επιπλέον η μέγιστη συχνότητα εξωτερικού ρολογιού που μπορεί χρησιμοποιηθεί για τον χρονισμό ενός 8 bit μικροελεγκτή της Microchip είναι τα 16MHz, οπότε ούτως ή άλλως είναι αδύνατο να επιλεγούν.

Ωστόσο είναι δυνατόν να επιτευχθεί η ακριβής ονομαστική διάρκεια των σημάτων χρονισμού και με κρυστάλλους διαφορετικής ονομαστικής συχνότητας. Με δεδομένη τη γνωστή μερική προς τα πίσω συμβατότητα με το πρότυπο NTSC, θα επιλεγεί ένας παθητικός κρύσταλλος 14.3182 MHz ο οποίος χρησιμοποιείται στις τηλεοράσεις που υποστηρίζουν NTSC αναλογικό σήμα και είναι ευρέως διαθέσιμος και με εξαιρετικά χαμηλό κόστος.

Κάνοντας αυτή την επιλογή, η μέγιστη συχνότητα χρονισμού αντιστοιχεί στο $\frac{14{,}3182\ MHz}{25{,}175\ MHz} = 56{,}87\%$ του προτύπου. Επομένως οι κύκλοι που θα αντιστοιχούν σε κάθε scanline θα είναι $800 \times 0{,}5687 = 455$. Έτσι εξασφαλίζεται απόλυτη σύγκλιση στις απαιτήσεις χρονισμού του προτύπου καθώς:

$455 \times \frac{1}{14{,}3182\ MHz} = 31{,}78\ \mu sec$, ακριβώς όσο απαιτεί το πρότυπο.

Η επιλογή αυτή, ταυτόχρονα περιορίζει τη μέγιστη οριζόντια ανάλυση η οποία πλέον δεν θα μπορεί να υπερβεί τα $640 \times 0{,}5687 = 364\ pixels$.

Το μέγεθος αυτό, αν και φαίνεται αρκετά μικρότερο από τα 640 pixels που υποστηρίζει το πρότυπο, εντούτοις είναι απολύτως αποδεκτό για απεικόνιση δεδομένων.

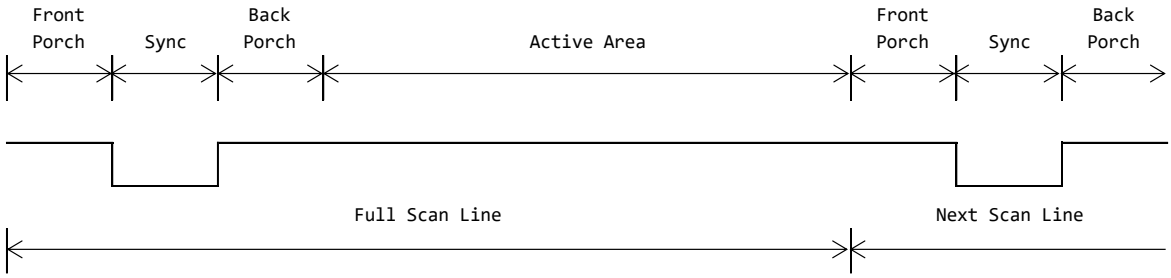
Με δεδομένο ότι για σχεδόν όλη τη δεκαετία του 80, οι 8-bit επιτραπέζιοι υπολογιστές που κυκλοφορούσαν υποστήριζαν ακόμα χαμηλότερες οριζόντιες αναλύσεις που

κυμαίνονταν μεταξύ 160 και 320 pixels, είναι απολύτως ρεαλιστικό μια 8-bit πλατφόρμα χαμηλού κόστους, που βασίζεται ουσιαστικά σε ένα και μόνο ολοκληρωμένο, να μην έχει σημαντικά υψηλότερες προσδοκίες.

3.1.1 Παραγωγή Σήματος Οριζόντιου Συγχρονισμού

Οι απαιτήσεις οριζόντιου συγχρονισμού με βάση το πρότυπο VESA [4], επιβάλουν ακρίβεια $\pm 0.5\%$. Η ακρίβεια αυτή είναι πολύ δύσκολο να επιτευχθεί μόνο με τη χρήση κώδικα. Επομένως πρέπει να αξιοποιηθούν τα περιφερειακά του μικροελεγκτή. Η χρήση ενός απαριθμητή (hardware timer) ο οποίος θα μετρά με ακρίβεια τους κύκλους του ρολογιού που απαιτούνται για τον σχηματισμό μια πλήρους scanline, μπορεί να είναι επαρκής λύση.

Μια πλήρης scan line αποτελείται από τις παρακάτω τέσσερις διακριτές περιοχές, όπως αναφέρθηκε και προηγούμενα.



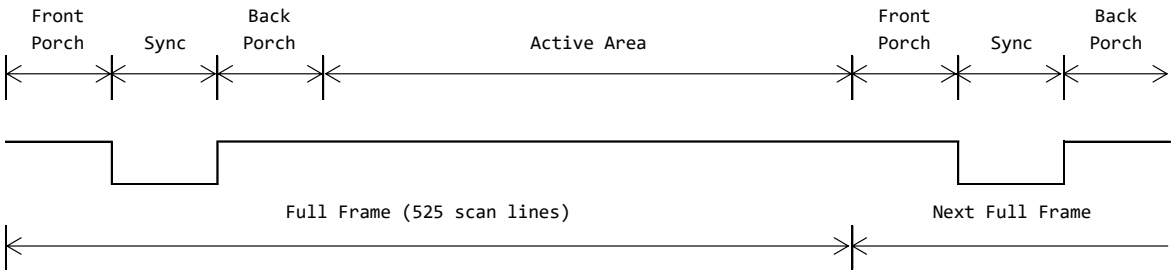
Διάγραμμα 1: Σήμα Οριζόντιου Συγχρονισμού

Εφόσον ο απαριθμητής μετρά 455 παλμούς και κατόπιν επανεκκινεί, είναι εφικτό να επιτευχθεί σταθερός οριζόντιος χρονισμός.

Με την ταυτόχρονη ενεργοποίηση των μονάδων PWM και τη σύνδεσή τους με τον παραπάνω απαριθμητή, είναι εφικτή η ακριβής παραγωγή του παλμού HSync σύμφωνα με τις ακριβείς απαιτήσεις του προτύπου.

3.1.2 Παραγωγή Σήματος Κατακόρυφου Συγχρονισμού

Ο παλμός κατακόρυφου συγχρονισμού έχει διάρκεια δυο οριζόντιων γραμμών και επαναλαμβάνεται κάθε 525 γραμμές.



Διάγραμμα 2: Σήμα Κατακόρυφου Συγχρονισμού

Αν ακολουθηθεί η ίδια μεθοδολογία με προηγουμένως, τότε το γινόμενο παλμών και γραμμών θα έδινε ένα εξαιρετικά μεγάλο πλήθος παλμών ($525 \times 455 = 238.875$).
Με δεδομένο πως $\lceil \log_2 238.875 \rceil = 18\, bits$, σημαίνει πως θα χρειάζονταν ένας απαριθμητής 18 bits για την καταμέτρηση τους και είναι ήδη γνωστό πως οι 8-bit μικροελεγκτές διαθέτουν timers 8 ή 16 bits. Επομένως αυτή η προσέγγιση δεν μπορεί να εφαρμοστεί.

Η λύση είναι να χρησιμοποιηθεί ένας timer ο οποίος θα καταμετρά κάθε παλμό που δημιουργείται από την παλμογεννήτρια που αναπτύχθηκε για την παραγωγή του οριζόντιου χρονισμού.

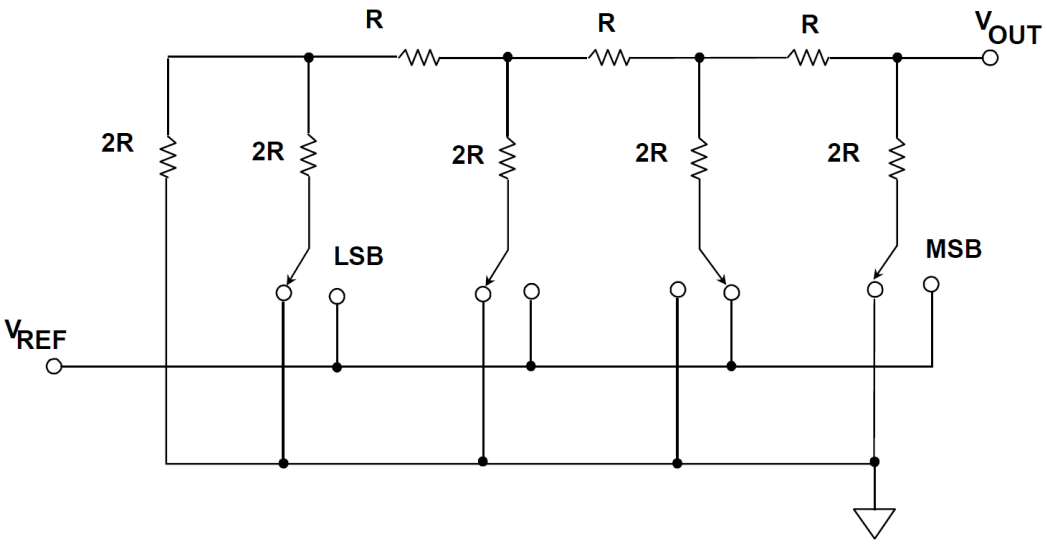
Κατόπιν χρησιμοποιώντας έναν συγκριτή (hardware compare module), θα ελέγχεται αν η τρέχουσα τιμή του μετρητή αντιστοιχεί στη γραμμή του παλμού κάθετου χρονισμού και θα ενεργοποιείται η σχετική έξοδος. Ο λόγος που προτείνεται αυτή η προσέγγιση και όχι ένα περιφερειακό PWM θα γίνει σαφής αργότερα.

3.1.3 Παραγωγή Αναλογικών Σημάτων

Για την παραγωγή 16 διαφορετικών χρωμάτων, είναι απαραίτητη η χρήση ενός 4-bit ψηφιοαναλογικού μετατροπέα (DAC). Εφόσον δε υπάρχουν τρία διαφορετικά χρωματικά κανάλια (R,G,B), είναι απαραίτητη η ύπαρξη τριών διαφορετικών DAC. Τη στιγμή αυτή, από τους συνολικά διακόσιους ενενήντα τέσσερις 8-bit μικροελεγκτές PIC και AVR που διαθέτει η Microchip, μόνο δυο υποστηρίζουν αυτή τη δυνατότητα. Ο περιορισμός αυτός καθιστά απαραίτητο να χρησιμοποιηθεί εξωτερικός DAC ώστε να υπάρχει μεγαλύτερη ευελιξία στην επιλογή μικροελεγκτή.

Ερευνώντας τη διαθεσιμότητα DAC στην αγορά, διαπιστώθηκε ότι η τιμή ενός Parallel In – Analog Out μετατροπέα, ξεκινά από τα 2,03€. Επομένως το συνολικό κόστος για την αγορά τριών DAC, θα ξεπερνά τα 6€. Ωστόσο αυτό υπερβαίνει κατά πολύ το μέσο κόστος αγοράς του ίδιου του μικροελεγκτή. Αυτή τη στιγμή το κόστος αγοράς ενός High End 8-bit μικροελεγκτή κυμαίνεται μεταξύ 1,5€ - 2,5€. Θα ήταν παράδοξο λοιπόν ένα εξωτερικό περιφερειακό να υπερβαίνει τόσο σημαντικά το κόστος της κεντρικής μονάδας.
Επομένως τελικά προκρίνεται η λύση της κατασκευής ενός απλού DAC με δίκτυο αντιστατών R-2R [5]. Το κόστος για την αγορά ενός SMD αντιστάτη, ξεκινά από 0,003€.

Ένα τυπικό διάγραμμα ενός R-2R ψηφιοαναλογικού μετατροπέα 4 bit, παρατίθεται ενδεικτικά παρακάτω:



B. D. Smith, "Coding by Feedback Methods," Proceedings of the I. R. E., Vol. 41, August 1953, pp. 1053-1058

Εικόνα 2: R-2R DAC

Υπολογίζεται πως θα χρειαστούν τέσσερις αντιστάτες ανά χρωματικό κανάλι, οπότε το συνολικό κόστος κτήσης θα είναι πρακτικά ασήμαντο.

3.2 Διεπαφή Εισόδου

3.2.1 Πρωτόκολλο Επικοινωνίας Υλικού

Για την επικοινωνία του μικροελεγκτή με τον έξω κόσμο προτείνεται η υιοθέτηση του ασύγχρονου φυσικού πρωτοκόλλου επικοινωνίας UART. Παρότι η επιλογή κάποιου πρωτοκόλλου σύγχρονης επικοινωνίας όπως SPI ή I2C ίσως φαίνεται πιο κατάλληλη, εντούτοις εισάγει επιπλέον πολυπλοκότητα στη φάση δοκιμών του ελεγκτή. Στην αγορά κυκλοφορούν πάρα πολλοί μετατροπείς USB to RS232 και RS232 to UART. Επομένως όλες οι δοκιμές επικοινωνίας με τον ελεγκτή, μπορούν να γίνουν με τη χρήση του επιτραπέζιου υπολογιστή και οποιουδήποτε λογισμικού υποστηρίζει επικοινωνία τερματικού, όπως το λογισμικό ανοιχτού κώδικα PuTTY ή το Arduino IDE. Έτσι δεν απαιτείται η κατασκευή βοηθητικού εξοπλισμού για να ελεγχθεί η λειτουργικότητα της κατασκευής.

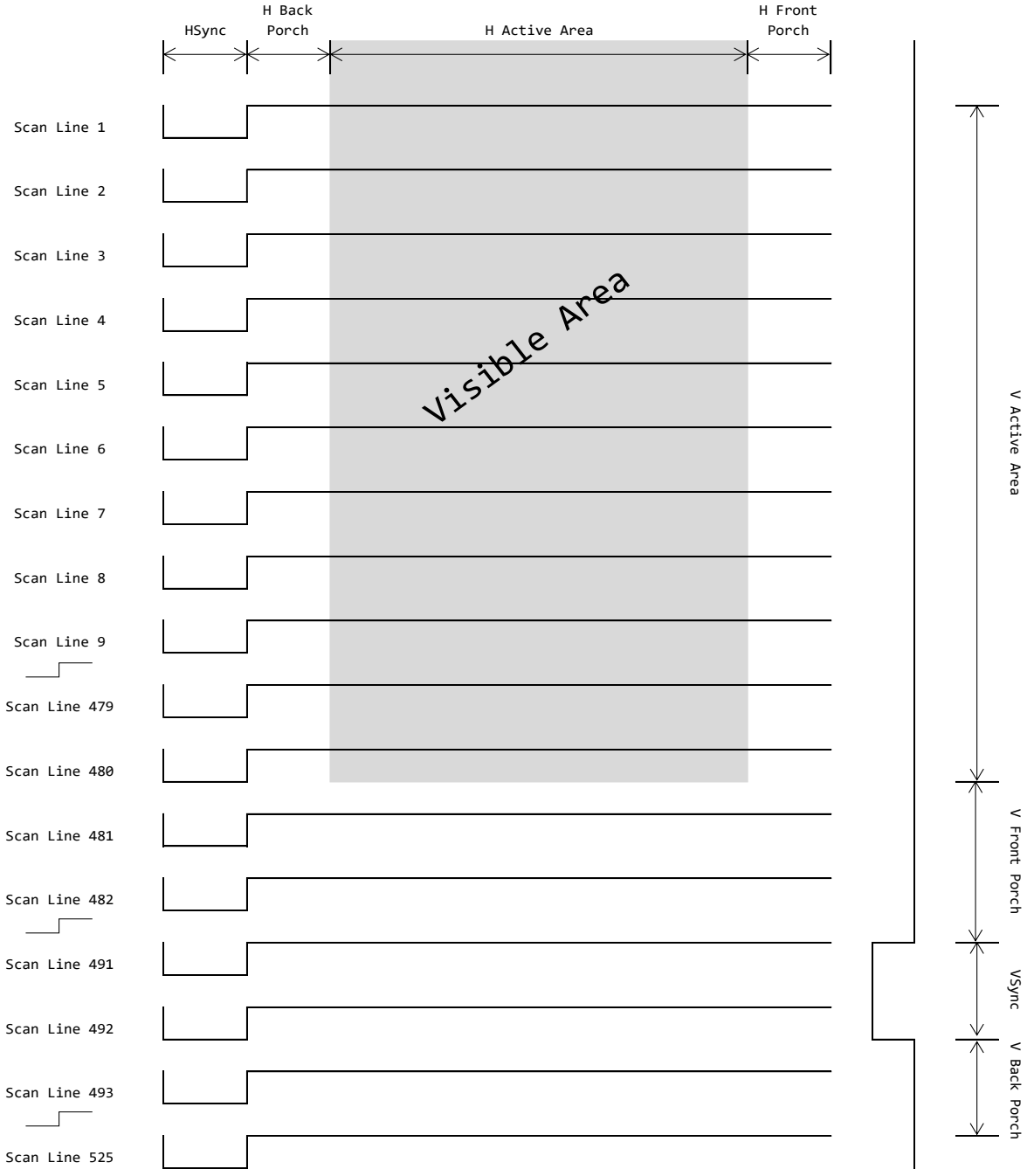
3.2.2 Πρωτόκολλο Επικοινωνίας Λογισμικού

Πάνω από το φυσικό πρωτόκολλο UART θα αναπτυχθεί ένα λογισμικό πρωτόκολλο επικοινωνίας το οποίο θα επιτρέπει στον μικροελεγκτή να λαμβάνει, να αναλύει και να ερμηνεύει εντολές από ένα άλλο σύστημα. Το πρωτόκολλο αυτό θα υλοποιηθεί εξ ολοκλήρου σε επίπεδο εφαρμογής (application layer) και θα σχεδιαστεί ώστε να είναι ελαφρύ και εύκολα επεκτάσιμο.

3.3 Επεξεργασία και Μετατροπή Δεδομένων

Ο μικροελεγκτής θα δέχεται εντολές και δεδομένα που θα πρέπει να μορφοποιηθούν και να παρουσιαστούν στις αντίστοιχες εξόδους RGB τη σωστή χρονική στιγμή. Το πρότυπο VGA ορίζει την ορατή περιοχή ως το χρονικό διάστημα κατά το οποίο οι έξοδοι RGB μπορούν να φέρουν τιμές, ενώ σε κάθε άλλη χρονική στιγμή πρέπει να βρίσκονται στο λογικό μηδέν.

3.3.1 Υπολογισμός Ορατής Περιοχής (Visible Area)



Διάγραμμα 3: Πλήρες Καρέ

Η ορατή περιοχή αποτελείται από 480 ενεργά τμήματα scan line όπως απεικονίζεται παραπάνω και αποτελεί την τομή των περιοχών active area του οριζόντιου και του

κατακόρυφου σήματος χρονισμού. Ο μικροελεγκτής πρέπει να στέλνει τα κατάλληλα αναλογικά σήματα στις εισόδους RGB της οθόνης, μόνο κατά το διάστημα που σημαίνεται με γκρι χρώμα. Επειδή ο χρονισμός είναι κρίσιμος, θα χρησιμοποιηθούν διακοπές (interrupt) όπου θα ενεργοποιούνται κατά την έναρξη κάθε ορατής γραμμής, διακόπτοντας τη φυσιολογική ροή του προγράμματος και θα ενεργοποιούν την αποστολή δεδομένων στην είσοδο της οθόνης.

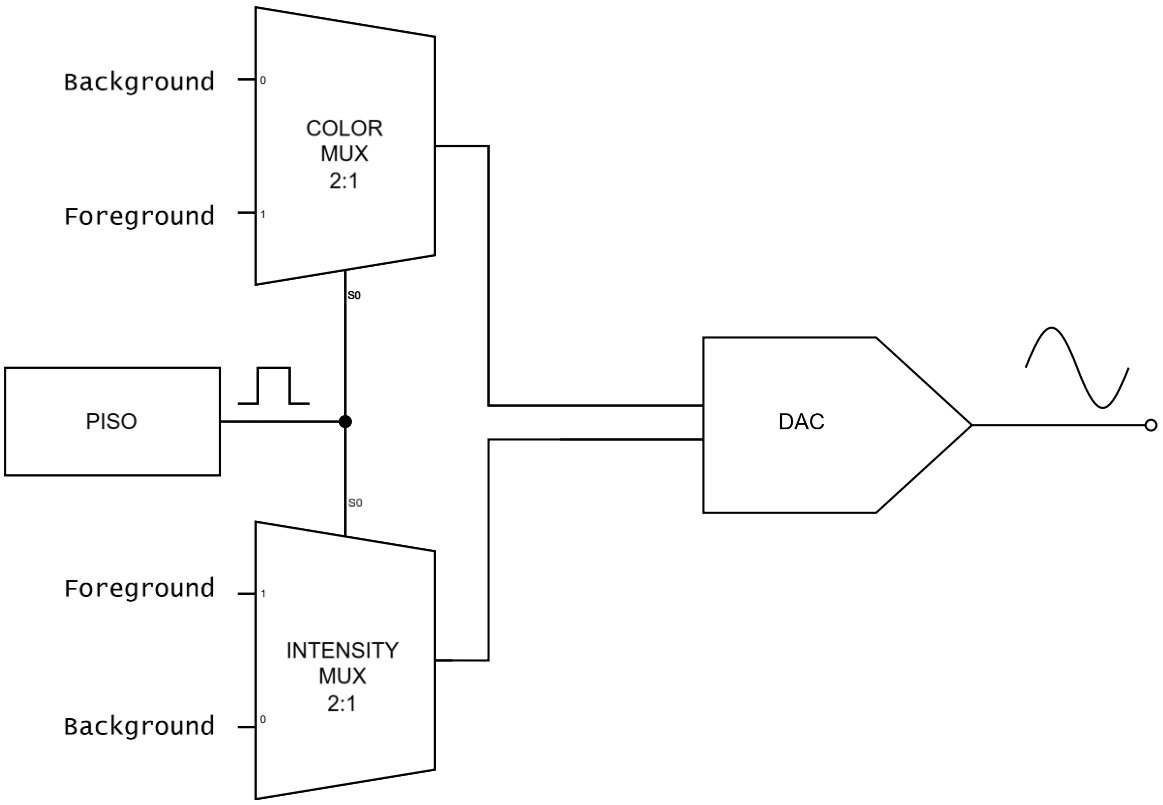
Ο συγκριτής που επελέγη νωρίτερα για την παραγωγή του σήματος κατακόρυφου χρονισμού, έχει την ιδιότητα να μπορεί να ενεργοποιεί ένα interrupt όταν το αποτέλεσμα μιας σύγκρισης επιστρέψει την τιμή TRUE. Επομένως, μπορεί να εξασφαλιστεί ότι θα παράγονται σήματα RGB μόνο κατά το ενεργό τμήμα 480 συναπτών γραμμών, δηλαδή μόνο στην ορατή περιοχή.

3.3.2 Αρχιτεκτονική Γεννήτριας Εικόνας (Pixel Generator)

Κατά το διάστημα που η σάρωση της οθόνης βρίσκεται στην ορατή περιοχή, ο ελεγκτής πρέπει να στέλνει κατάλληλα δεδομένα στον DAC κάθε χρωματικού καναλιού. Έχοντας υιοθετήσει ως συχνότητα χρονισμού τα 14,3182 MHz, ο ελεγκτής είναι υποχρεωμένος να παράγει ένα pixel ανά παλμό του ρολογιού προκειμένου να πετύχει οριζόντια ανάλυση 364 pixels. Αυτή η απαίτηση αποδεικνύεται εξαιρετικά περιοριστική: σε κάθε παλμό, ο ελεγκτής θα πρέπει να διαβάσει την αποθηκευμένη πληροφορία από την εσωτερική μνήμη RAM και να την μετατρέψει σε κατάλληλο σήμα εξόδου 4 bit, το οποίο θα οδηγήσει τον DAC.

Σε πρώτη εκτίμηση, κάτι τέτοιο φαίνεται ανέφικτο χωρίς να υποβιβαστεί επιπλέον η οριζόντια ανάλυση. Επομένως, πρέπει να ακολουθηθεί διαφορετική αρχιτεκτονική για την προετοιμασία και την κωδικοποίηση του σήματος εξόδου, έτσι ώστε να μην επιβαρύνεται η CPU με την παραγωγή κάθε pixel σε πραγματικό χρόνο.

Η λύση που προτείνεται συνοψίζεται στο ακόλουθο block διάγραμμα:



Διάγραμμα 4: MUX-DAC Block Diagram

Διατηρώντας σταθερή την ανάλυση και με τον ταυτόχρονο περιορισμό της χρωματικής πληροφορίας σε δύο μόνο τιμές ανά εικονοστοιχείο, επιτυγχάνεται εντυπωσιακή μείωση τόσο των απαιτήσεων μνήμης όσο και του χρόνου που απαιτείται για την προετοιμασία των δεδομένων εικόνας, ο οποίος μειώνεται δραστικά, σε ελάχιστους μόνο κύκλους ρολογιού.

Κάθε εικονοστοιχείο πλέον χαρακτηρίζεται από το χρώμα προσκηνίου (foreground) και παρασκηνίου (background), επιλέγοντας μέσα από ένα φάσμα 16 διαφορετικών χρωμάτων. Τα pixel κάθε εικονοστοιχείου, μεταφέρονται σειριακά μέσω καταχωρητή PISO (Parallel-In Serial-Out), και ενεργοποιούν την είσοδο επιλογής των πολυπλεκτών. Οι πολυπλέκτες μεταβιβάζουν τη χρωματική πληροφορία στους DAC, η έξοδος των οποίων αποτελεί και ένα χρωματικό κανάλι RGB.

3.3.3 Καταχωρητής Parallel In – Serial Out (PISO)

Σχεδόν όλοι οι 8-bit μικροελεγκτές διαθέτουν τουλάχιστον μία μονάδα SPI, η οποία μπορεί να αξιοποιηθεί ως καταχωρητής Parallel In – Serial Out (PISO). Το συγκεκριμένο περιφερειακό επιτρέπει τη φόρτωσή του με 8-bit δεδομένα σε έναν μόνο κύκλο και τη σειριακή αποστολή τους για τους επόμενους 8 κύκλους, χωρίς να απασχολείται η κεντρική μονάδα του μικροελεγκτή.
Με αυτόν τον τρόπο, δημιουργείται ένα παράθυρο 8 κύκλων, το οποίο παρέχει επαρκή χρόνο για την ανάγνωση των δεδομένων από τη RAM, τη μετατροπή τους σε κατάλληλη μορφή και την προώθησή τους στους DAC των καναλιών χρώματος.

3.3.4 Πολυπλεξία Σήματος Εξόδου

Η έξοδος του περιφερειακού SPI παρέχει διαδοχικά bits, τα οποία αξιοποιούνται για την διέγερση της εισόδου επιλογής ενός συστήματος πολυπλεκτών 2:1. Το κάθε bit καθορίζει αν θα προβληθεί το χρώμα προσκηνίου (foreground) ή το χρώμα παρασκηνίου (background) για το αντίστοιχο pixel.

Οι πολυπλέκτες διακρίνονται σε δυο κατηγορίες:

- Επιλογής Χρώματος: όπου κάθε είσοδος δεδομένων αντιστοιχεί στη διέγερση του αντίστοιχου χρωματικού καναλιού.
- Επιλογής Φωτεινότητας: όπου κάθε είσοδος δεδομένων αντιστοιχεί στο επίπεδο φωτεινότας όλων των καναλιών.

Επομένως, στέλνοντας παράλληλα 8 bit δεδομένων στους πολυπλέκτες και αξιοποιώντας την έξοδο του περιφερειακού SPI ως σήμα επιλογής, μπορεί να μεταδοθεί εντός του χρονικού παραθύρου των 8 κύκλων, πληροφορία κατάλληλη για την οδήγηση του DAC, όπου τα τέσσερα πρώτα bit αντιστοιχούν στο χρώμα προσκηνίου και τα επόμενα στο χρώμα παρασκηνίου και έτσι είναι εφικτό να δημιουργηθούν $2^3 \times 2 = 16$ χρωματικές επιλογές.
Η μεθοδολογία αυτή, γνωστή και ως 4-bit RGBI (Red, Green, Blue, Intensity) [6] χρησιμοποιήθηκε ευρέως από τους πρώτους 8-bit επιτραπέζιους ηλεκτρονικούς υπολογιστές.

Είναι προφανές ότι οι πολυπλέκτες θα αποτελέσουν εξωτερικό κύκλωμα. Ωστόσο το κόστος τους είναι εξαιρετικά χαμηλό. Αυτή τη στιγμή η τιμή ενός Quad 2-input multiplexer είναι μόλις 0,37€.

3.4 Επιλογή Μικροελεγκτή

Για την υλοποίηση της παρούσας κατασκευής επιλέχθηκε ο μικροελεγκτής PIC18F47K42 της Microchip Technology [7], καθώς ανταποκρίνεται στις τεχνικές απαιτήσεις του συστήματος, προσφέροντας επαρκείς δυνατότητες εισόδου/εξόδου και επεξεργαστικής ισχύος. Η επιλογή ενισχύθηκε από το χαμηλό κόστος τόσο του ίδιου του μικροελεγκτή όσο και των συνοδευτικών εργαλείων ανάπτυξης. Ειδικότερα, παρέχεται η δωρεάν σουίτα ανάπτυξης λογισμικού MPLAB X IDE, σε συνδυασμό με τον MPLAB XC8 compiler, ο οποίος υποστηρίζει πλήρως την οικογένεια 8-bit PIC μικροελεγκτών και επιτρέπει την ανάπτυξη κώδικα σε γλώσσα C.

Επιπρόσθετο πλεονέκτημα, αποτελεί η ευρεία διαθεσιμότητα των προγραμματιστών και αποσφαλματωτών PICkit 2 & PICkit 3, οι οποίοι υποστηρίζουν τον συγκεκριμένο μικροελεγκτή, και βασίζονται σε ανοιχτή αρχιτεκτονική υλικού και λογισμικού. Λόγω αυτού, υπάρχει στην αγορά πληθώρα συμβατών εκδόσεων από τρίτους κατασκευαστές, οι οποίες διατίθενται σε σημαντικά χαμηλότερη τιμή από τον επίσημο κατασκευαστή, διατηρώντας ταυτόχρονα τις ίδιες δυνατότητες. Το γεγονός αυτό καθιστά την ανάπτυξη ενσωματωμένων συστημάτων βασισμένα σε μικροελεγκτές PIC, ιδιαίτερα οικονομική.

3.4.1 Τεχνικά Χαρακτηριστικά

Ο PIC18F47K42 είναι ένας μικροελεγκτής 8-bit αρχιτεκτονικής τύπου RISC που ανήκει στην οικογένεια PIC18FxxK42. Διακρίνεται για την υψηλή του ευελιξία, τον μεγάλο αριθμό ενσωματωμένων περιφερειακών και τις λειτουργίες εξοικονόμησης ενέργειας, καθιστώντας τον κατάλληλο για χρήση σε embedded συστήματα με real time απαιτήσεις.

Βασικά Τεχνικά Χαρακτηριστικά:

- Αρχιτεκτονική: 8-bit RISC με pipeline τεχνολογία για βελτιωμένη απόδοση.
- Μικροπυρήνας (Core): PIC18 – Υψηλής απόδοσης με εντολές ενός κύκλου.
- Συχνότητα λειτουργίας: έως 64 MHz, παρέχοντας κύκλο εντολής έως 16 MHz και αποδίδοντας μέχρι 16 εκατομμύρια εντολές ανά δευτερόλεπτο (16 MIPS).
- Μνήμη Flash: 128 KB με 21-bit address bus
- RAM (SRAM): 8 KB με 14-bit address bus
- EEPROM: 1 KB
- Αριθμός Γενικών Εισόδων/Εξόδων (GPIO): έως 36 ψηφιακά pins με δυνατότητα επιλογής περιφερειακού (Peripheral Pin Select - PPS).
- Αναλογικά Περιφερειακά:
    - ADC: 12-bit, έως 35 κανάλια, με δυνατότητα αυτόματης δειγματοληψίας και trigger.
    - DAC: 1 κανάλι 5-bit, για βασική αναλογική έξοδο.
- Ψηφιακές Διασυνδέσεις: Υποστήριξη UART, SPI, I2C, LINbus.
- Διαχείριση ενέργειας: Λειτουργίες IDLE, DOZE, SLEEP για μείωση κατανάλωσης.
- Ενσωματωμένα Περιφερειακά:
    - DMA Controller για απευθείας μεταφορά δεδομένων μεταξύ μονάδων μνήμης.
    - COMPARE/PWM/COMPARATOR Modules.
    - HLVD (High/Low Voltage Detect), Brown-out Reset, Watchdog Timer.
    - Configurable Logic Cells (CLCs), οι οποίες επιτρέπουν την υλοποίηση απλών λογικών κυκλωμάτων χωρίς εξωτερικά στοιχεία, μειώνοντας την πολυπλοκότητα του συστήματος.
- MAP (Memory Access Partition) για προστασία περιοχών μνήμης.
- DIA (Device Information Area) με μοναδικά χαρακτηριστικά συσκευής.
- CRC Engine, Zero-Cross Detect, και δυνατότητα επιβεβαίωσης ακρίβειας μεταφοράς.

3.4.2 Τεκμηρίωση Επιλογής

Η επιλογή του μικροελεγκτή PIC18F47K42 δικαιολογείται τεχνικά από τη δυνατότητα ενσωμάτωσης όλων των κρίσιμων λειτουργικών απαιτήσεων του συστήματος χωρίς την ανάγκη εξωτερικών περιφερειακών. Ενδεικτικά:

- Υποστηρίζει 8/16-bit Timers καθώς και PWM και Compare μονάδες, οι οποίες χρησιμοποιούνται για τη δημιουργία σημάτων χρονισμού HSYNC και VSYNC.
- Περιλαμβάνει μονάδα SPI, η οποία αξιοποιείται ως καταχωρητής Parallel-In Serial-Out (PISO).
- Υποστηρίζει διακοπές υλικού (Hardware Interrupts).
- Διαθέτει μονάδα UART, μέσω της οποίας γίνεται η αλληλεπίδραση με τον χρήστη για την εισαγωγή εντολών και την αποστολή αποτελεσμάτων, αξιοποιώντας απλά πρωτόκολλα μέσω τερματικού (π.χ. PuTTY, Arduino IDE).
- Ενσωματώνει δύο μονάδες DMA, οι οποίες θα αξιοποιηθούν για την αυτόματη μεταφορά δεδομένων μεταξύ περιφερειακών, αποφορτίζοντας επιπλέον την κεντρική μονάδα και δίνοντας περισσότερο χρόνο στη διαδικασία επεξεργασίας και παραγωγής της εικόνας.
- Διαθέτει τέσσερις μονάδες CLC, οι οποίες θα αξιοποιηθούν στην κατασκευή των γεννητριών σήματος, είτε ως λογικές πύλες, είτε ως ακολουθιακά κυκλώματα (Flip Flop), περιορίζοντας ακόμα περισσότερο την ανάγκη για εξωτερικά περιφερειακά. Μέχρι σήμερα, το συγκεκριμένο περιφερειακό παρέχεται αποκλειστικά από τους μικροελεγκτές PIC και δεν απαντάται ούτε σε μικροελεγκτές μεγαλύτερης υπολογιστικής ισχύος.

Τέλος, αξίζει να σημειωθεί ότι η Microchip παρέχει δωρεάν δείγματα για τους περισσότερους μικροελεγκτές, συμπεριλαμβανομένου και του PIC18F47K42, διευκολύνοντας έτσι την εκπαιδευτική και ερευνητική χρήση, χωρίς καμία ουσιαστική οικονομική επιβάρυνση στο αρχικό στάδιο ανάπτυξης.

4. Αρχιτεκτονική Συστήματος

4.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται η αρχιτεκτονική του συστήματος σε δύο διακριτά επίπεδα: το υλικό (hardware) και το λογισμικό (software/firmware), με στόχο την αναλυτική παρουσίαση της δομής και της λειτουργίας κάθε επιμέρους υποσυστήματος.

Ο σχεδιασμός ακολουθεί μία ολοκληρωμένη προσέγγιση, ώστε να διασφαλίζεται η αποδοτική παραγωγή σήματος βίντεο, η αξιόπιστη διαχείριση εισόδου εντολών και η αποτελεσματική αξιοποίηση των διαθέσιμων πόρων του μικροελεγκτή, τόσο σε επίπεδο υπολογιστικής ισχύος όσο και σε επίπεδο περιφερειακών.

Η αρχιτεκτονική υλικού επικεντρώνεται στη διασύνδεση του μικροελεγκτή με τις επιμέρους εσωτερικές και εξωτερικές περιφερειακές μονάδες, οι οποίες αποτελούν κρίσιμα στοιχεία για την λειτουργία του συστήματος. Οι μονάδες αυτές ομαδοποιούνται με βάση τη λειτουργική τους συνεισφορά, ώστε να επιτελούν διακριτούς και εξειδικευμένους ρόλους. Ενδεικτικά, η ομάδα περιφερειακών που σχετίζεται με την παραγωγή σήματος οριζόντιου συγχρονισμού, περιλαμβάνει CLC, Timer και PWM περιφερειακά, τα οποία συνεργάζονται για την υλοποίηση της γεννήτριας του οριζόντιου συγχρονισμού, ως ανεξάρτητη και διακριτή οντότητα.

Η αρχιτεκτονική λογισμικού ακολουθεί πολυεπίπεδη (layered) προσέγγιση, με σκοπό την αύξηση της ευελιξίας, της επαναχρησιμοποίησης και γενικότερα, της τεχνικής βιωσιμότητας του κώδικα. Ο διαχωρισμός του λογισμικού σε διακριτά επίπεδα περιλαμβάνει:

- Low-level drivers, οι οποίοι αλληλεπιδρούν απευθείας με το υλικό και υλοποιούν βασικές ρουτίνες πρόσβασης (π.χ. GPIO, SPI, UART),
- Middleware modules, που προσφέρουν γενική λειτουργικότητα, όπως circular buffers, debouncers κτλ,
- Application/core layer, στο οποίο υλοποιείται η λογική της εφαρμογής και η διαχείριση της αλληλεπίδρασης με τον χρήστη.

Ο εν λόγω διαχωρισμός επιπέδων καθιστά το σύστημα επεκτάσιμο, εύκολα συντηρήσιμο και με υψηλή φορητότητα, ενώ ενισχύει την αξιοπιστία και την αναγνωσιμότητα του πηγαίου κώδικα. Επιπλέον, διευκολύνει την παράλληλη ανάπτυξη και δοκιμή ανά λειτουργικό επίπεδο.

4.2 Αρχιτεκτονική υλικού

Στην ενότητα αυτή, παρουσιάζεται η αρχιτεκτονική υλικού του συστήματος, το οποίο αποτελείται από τρεις ανεξάρτητες λειτουργικές μονάδες, οι οποίες συνεργάζονται για να παραγάγουν όλα τα σήματα που απαιτούνται για την οδήγηση της οθόνης. Συγκεκριμένα, το υλικό περιλαμβάνει τις ακόλουθες οντότητες:

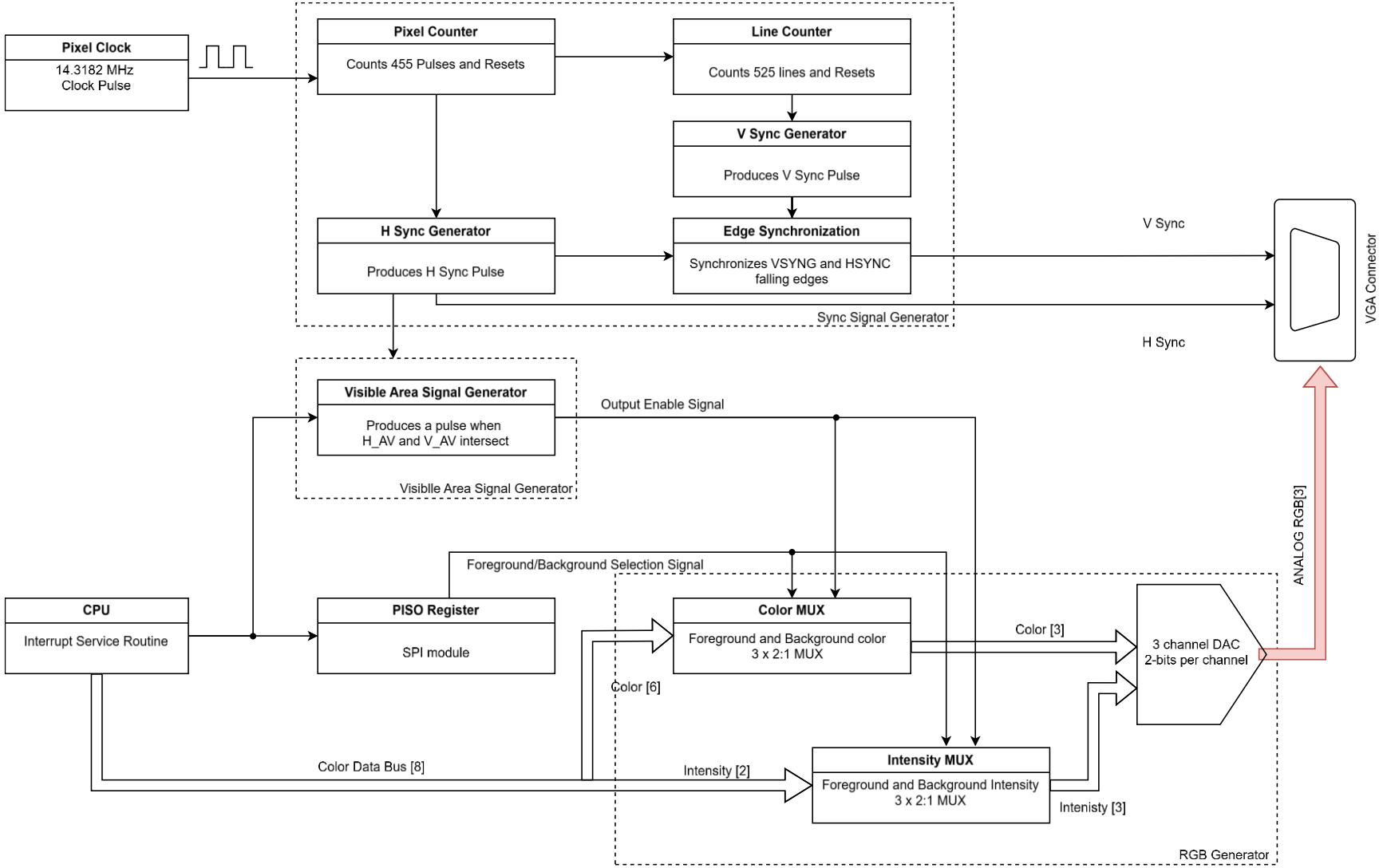
- **Εσωτερική Μονάδα Παραγωγής Σημάτων Συγχρονισμού**
  Υπεύθυνη για την παραγωγή των σημάτων του οριζόντιου και κατακόρυφου συγχρονισμού.

- **Εσωτερική Μονάδα Παραγωγής Σήματος Ορατής Περιοχής**
  Παράγει την τομή της οριζόντιας και της κάθετης ενεργής περιοχής και ενεργοποιεί τους πολυπλέκτες.

- **Εξωτερική Μονάδα Παραγωγής Αναλογικού Σήματος RGB**
  Υλοποιείται εκτός του μικροελεγκτή και περιλαμβάνει τους πολυπλέκτες και τους ψηφιοαναλογικούς μετατροπείς που παράγουν το αναλογικό σήμα RGB

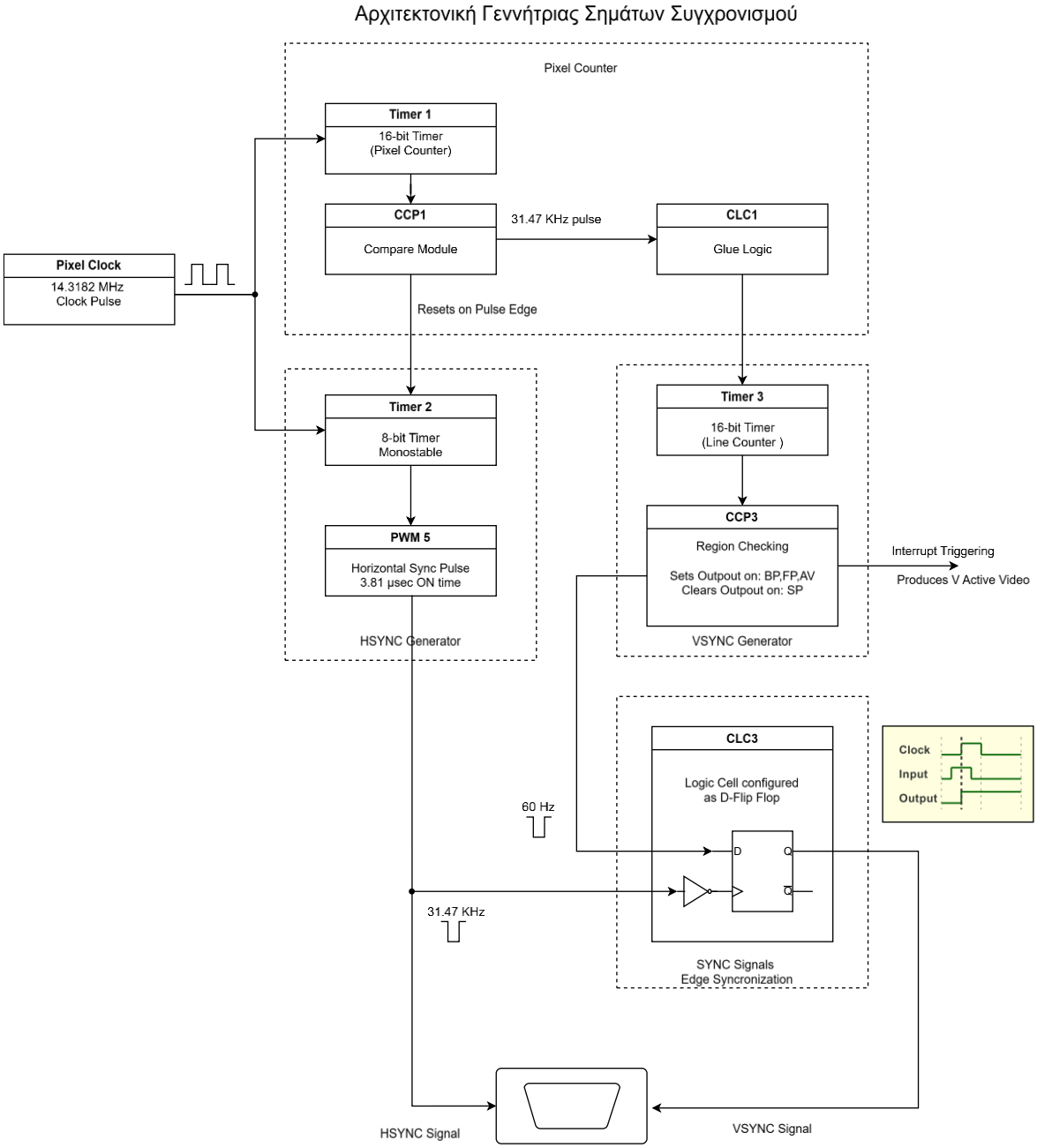
Κατά τον σχεδιασμό, δόθηκε ιδιαίτερη έμφαση στην εκμετάλλευση των ενσωματωμένων περιφερειακών του μικροελεγκτή, προκειμένου να ελαχιστοποιηθεί ο φόρτος της κεντρικής μονάδας επεξεργασίας.

Ακολουθεί η επισκόπηση της συνολικής αρχιτεκτονικής με τη μορφή μπλοκ διαγράμματος, στο οποίο απεικονίζονται οι βασικές μονάδες υλικού και η μεταξύ τους διασύνδεση.



Διάγραμμα 5: Επισκόπηση Αρχιτεκτονικής Υλικού

4.2.1 Εσωτερική Μονάδα Παραγωγής Σημάτων Συγχρονισμού



Διάγραμμα 6: Block Διάγραμμα Μονάδας Παραγωγής Σημάτων Συγχρονισμού

Παρουσιάζεται το block διάγραμμα της Μονάδας Παραγωγής Σημάτων Συγχρονισμού. Αποτελείται από τέσσερις ανεξάρτητες οντότητες, η περιγραφή των οποίων δίνεται παρακάτω:

Μετρητής Γραμμών (Line Counter)

Ο Timer 1 λειτουργεί ως 16-bit απαριθμητής και αυξάνει σε κάθε κύκλο του ρολογιού του συστήματος. Διασυνδέεται με την μονάδα CCP1 (Compare Module) και ενεργοποιείται όταν καταμετρηθούν 455 παλμοί διάρκειας 31,78 μsec, δηλαδή διάρκειας μιας πλήρους scanline. Κατά την ενεργοποίηση παράγει έναν παλμό και επανεκκινεί την καταμέτρηση. Το περιφερειακό CLC1 λειτουργεί ως μονάδα glue logic και απλά εξυπηρετεί στο να καθιστά το σήμα προσβάσιμο σε περιφερειακά που δεν μπορούν να διασυνδεθούν άμεσα με το CCP1.

Γεννήτρια HSYNC

Ο παλμός που παράγεται από τον Μετρητή Γραμμών, ενεργοποιεί τον 8-bit απαριθμητή Timer 2, ο οποίος είναι ρυθμισμένος να λειτουργεί ως μονοσταθής πολυδονητής (monostable multivibrator). Ο Timer 2 διασυνδέεται με την μονάδα PWM 2, η οποία παράγει τον παλμό λογικού μηδέν HSYNC, με διάρκεια 3.91 μs. Με τη μέθοδο αυτή επιτυγχάνεται απόλυτη ακρίβεια στην παραγωγή του απαιτούμενου πλάτους του παλμού. Το σήμα αυτό επαναλαμβάνεται για κάθε γραμμή επιτυγχάνοντας συχνότητα 31.47 KHz.

Γεννήτρια VSYNC

Για τον κατακόρυφο συγχρονισμό, χρησιμοποιείται ο Timer 3, ο οποίος αυξάνεται με κάθε παλμό HSYNC δημιουργώντας έναν επιπλέον μετρητή. Η μονάδα CCP3, ελέγχει αν ο μετρητής βρίσκεται σε κάποια από τις περιοχές VERTICAL BACK PORCH, VERTICAL FRONT PORCH, VERTICAL SYNC PULSE, VERTICAL ACTIVE VIDEO και θέτει ή απενεργοποιεί την έξοδο του περιφερειακού, ενώ ταυτόχρονα ενεργοποιεί μια διακοπή που θα χρησιμοποιηθεί σε δεύτερο χρόνο για την αποστολή δεδομένων RGB αλλά και για τον σχηματισμό της κατακόρυφης ενεργής περιοχής.

Συγχρονισμός Ακμών (Edge Synchronization)

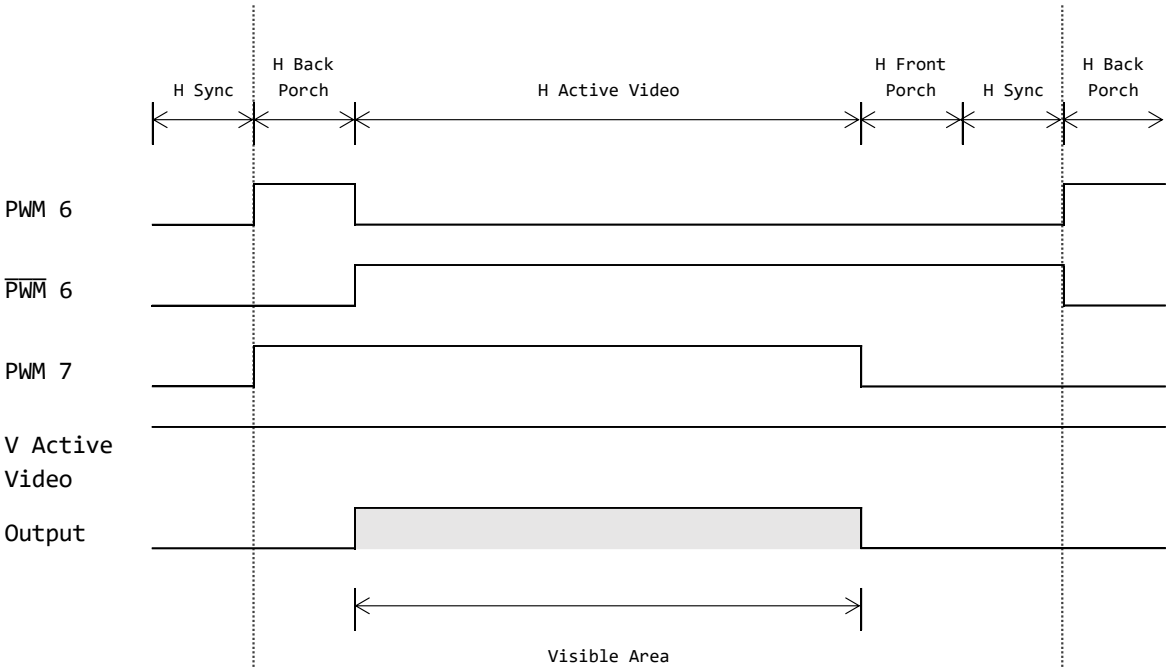
Το πρότυπο επιβάλει οι ακμές των σημάτων VSYNC και HSYNC να είναι συγχρονισμένες. Για να επιτευχθεί αυτό, τα δυο σήματα HSYNC και VSYNC οδηγούνται στο περιφερειακό CLC3, το οποίο έχει διαμορφωθεί ως D Flip-Flop. Το ανεστραμμένο σήμα HSYNC πυροδοτεί την είσοδο CLOCK του D Flip Flop με αποτέλεσμα η μεταβολή της εξόδου VSYNC να συμπίπτει χρονικά την μεταβολή του σήματος HSYNC και έτσι τα δυο σήματα να είναι πλήρως συγχρονισμένα.

Με την παραπάνω αρχιτεκτονική επιτυγχάνεται η ακριβής παραγωγή των σημάτων συγχρονισμού, σε πλήρη συμμόρφωση με τις προδιαγραφές του πρότυπου VGA, χωρίς να απαιτούνται εξωτερικά περιφερειακά και ταυτόχρονα χωρίς να επιβαρύνεται η κεντρική μονάδα.

Η συγκεκριμένη προσέγγιση υπήρξε δύσκολη στη σύλληψη της. Η βασική της καινοτομία συνίσταται στο ότι υλοποιεί λειτουργικότητα που παραδοσιακά βασίζεται σε εξειδικευμένες FPGA [8] ή CPLD μονάδες, αποκλειστικά μέσω των ενσωματωμένων περιφερειακών του μικροελεγκτή. Το γεγονός αυτό, αναδεικνύει τόσο τη δυναμική του προτεινόμενου σχεδιασμού, όσο και την αξιοποίηση στο έπακρο των διαθέσιμων πόρων του μικροελεγκτή.

4.2.2 Εσωτερική Μονάδα Παραγωγής Σήματος Ορατής Περιοχής

Η ορατή περιοχή, όπως αναφέρθηκε και προηγούμενα, είναι η τομή της ενεργής περιοχής των παλμών HSYNC και VSYNC. Ωστόσο, δεν έχει δημιουργηθεί ως τώρα σήμα κατάλληλου πλάτους που να σηματοδοτεί την έναρξη και τη λήξη αυτού του διαστήματος καθώς δεν ήταν απαραίτητο για τον συγχρονισμό της οθόνης. Επομένως, αξιοποιούνται δυο επιπλέον γεννήτριες PWM για να διαμορφώσουν το ακριβές πλάτος του σήματος. Η τομή της θετικής περιόδου των παλμών αυτών και της ενεργής περιοχής του σήματος VSYNC, συνθέτουν το σήμα ενεργοποίησης της Ορατής Περιοχής (Visible Area).

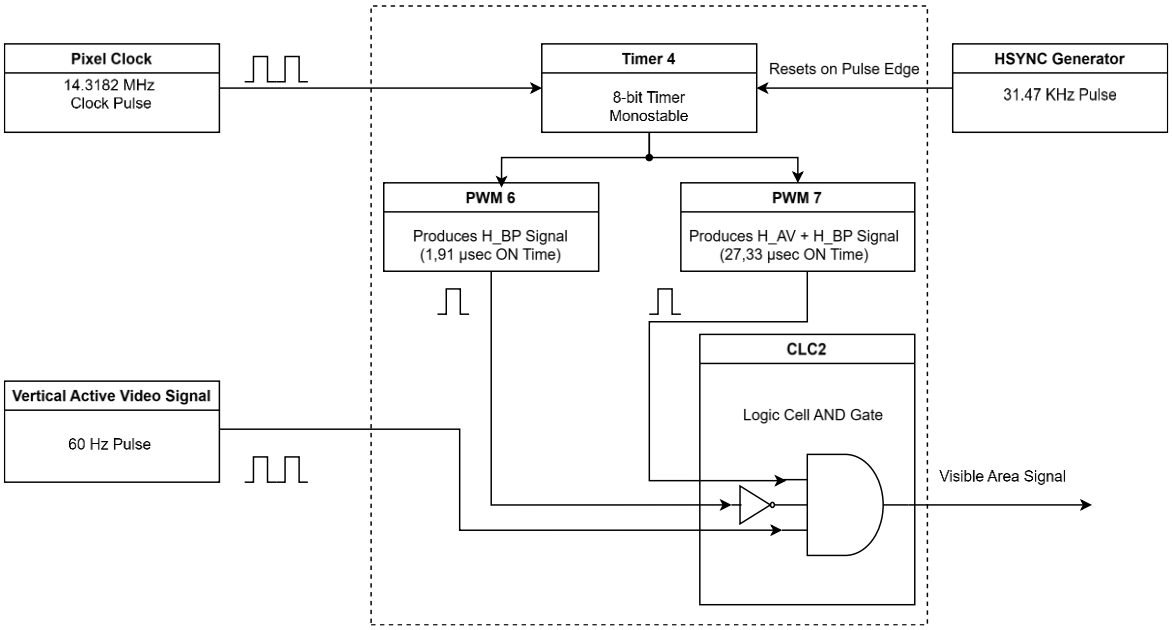


Διάγραμμα 7: Σήμα Ενεργοποίησης Ορατής Περιοχής

Η μονάδα PWM6 παράγει σήμα με διάρκεια ενεργής κατάστασης (on-time) 1.91 μsec, ίσο δηλαδή με τον χρόνο Horizontal Back Porch. Ο μονάδα PWM7 παράγει σήμα με διάρκεια ενεργής κατάστασης (on-time) 27.33 μsec, ίσο δηλαδή με το άθροισμα του χρόνου Horizontal Active Video και Horizontal Back Porch. Η τομή του ανεστραμμένου σήματος PWM 6 και του σήματος PWM 7 ορίζουν το οριζόντιο διάστημα Active Video. Η τομή αυτού του διαστήματος με το αντίστοιχο διάστημα του παλμού κατακόρυφου συγχρονισμού, ορίζει την περιοχή Visible Video.

Η μονάδα παραγωγής του σήματος Visible Video, στήθηκε εξ ολοκλήρου χρησιμοποιώντας τις περιφερειακές μονάδες του μικροελεγκτή και δεν επιβαρύνει την κεντρική μονάδα. Ακολουθεί το block διάγραμμα της μονάδας και η ανάλυση του:

Αρχιτεκτονική Γεννήτριας Σήματος Ορατής Περιοχής



Διάγραμμα 8: Block Διάγραμμα Μονάδας Παραγωγής Σήματος Ορατής Περιοχής

Απαριθμητής Timer 4

Ο απαριθμητής Timer 4 λειτουργεί ως μονοσταθής πολυδονητής. Αυξάνει σε κάθε παλμό του ρολογιού και επανεκκινεί όταν δεχθεί σήμα reset από τη Γεννήτρια Σήματος Οριζόντιου χρονισμού.

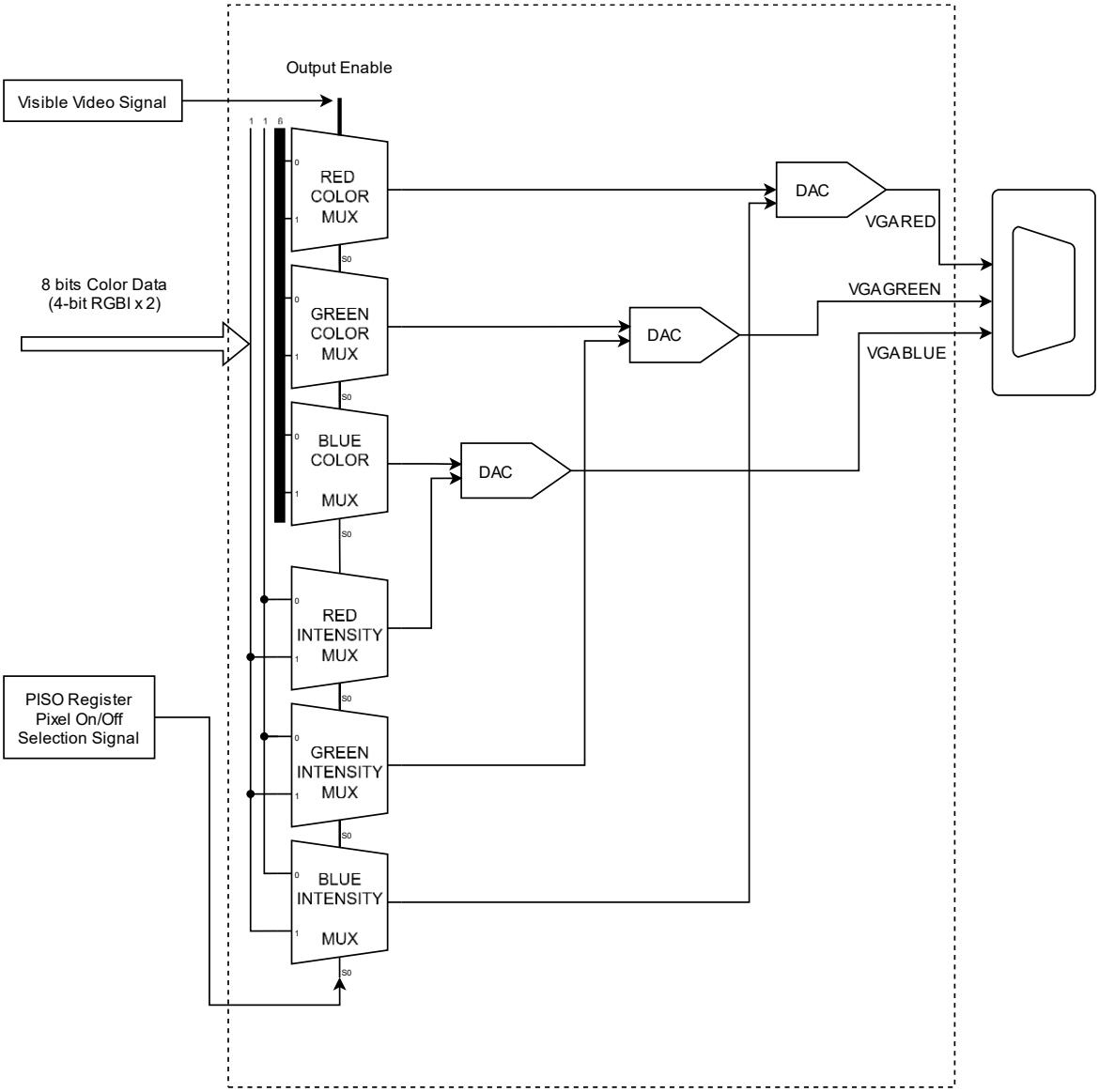
Μονάδες Διαμόρφωσης Πλάτους Παλμού, PWM 6 και PWM 7

Χρησιμοποιώντας τις μονάδες διαμόρφωσης πλάτους PWM 6 και PWM 7 κατά τον τρόπο που περιεγράφηκε νωρίτερα παράγονται δυο σήματα, η τομή των οποίων δημιουργεί την οριζόντια ενεργή περιοχή.

Λογική Κυψέλη CLC2

Η μονάδα CLC2 έχει ρυθμιστεί ως πύλη AND τριών εισόδων. Η έξοδος της δίνει την τομή του σήματος της οριζόντιας ενεργής περιοχής και της κάθετης ενεργής περιοχής, παράγοντας το σήμα της Ορατής Περιοχής (Visible Area).

4.2.3 Εξωτερική Μονάδα Παραγωγής Αναλογικού Σήματος RGB



Διάγραμμα 9: Block Διάγραμμα Μονάδας Παραγωγής Χρώματος

Η αρχιτεκτονική που παρουσιάζεται στο παραπάνω διάγραμμα αποτελεί τον τελικό σταθμό της μετατροπής των δεδομένων εικόνας σε αναλογικό σήμα VGA. Η σχεδίαση υλοποιεί τρία χρωματικά κανάλια RGB με δυνατότητα επιλογής μεταξύ δύο χρωμάτων ανά εικονοστοιχείο. Πρέπει να σημειωθεί ότι το μέγεθος κάθε εικονοστοιχείου δεν είναι σταθερό και μεταβάλλεται ανάλογα με το αν ο ελεγκτής βρίσκεται σε κατάσταση κειμένου (8x12 pixels) ή γραφικών (8x8).

Σήμα Ορατής Περιοχής (Visible Video Signal)

Το σήμα αυτό καθορίζει αν η τρέχουσα θέση σάρωσης της οθόνης αντιστοιχεί στην ορατή περιοχή και οδηγείται στην είσοδο ενεργοποίησης των πολυπλεκτών (OE – Output Enable). Όταν η τιμή του είναι ενεργή (στάθμη λογικού μηδέν), τότε επιτρέπεται η αποστολή δεδομένων προς τους DAC και κατ' επέκταση στην οθόνη.

Δεδομένα Χρώματος

Η πληροφορία χρώματος μεταφέρεται μέσω ενός 8-bit διαύλου. Τα 8 bit χωρίζονται σε δύο μέρη των 4 bits, το καθένα από τα οποία περιγράφει έναν χρωματικό συνδυασμό (RGBI – κόκκινο, πράσινο, μπλε, ένταση φωτεινότητας) κατά τον τρόπο που περιγράφεται στις ενότητες 3.1.3 και 3.3.4. Αυτό επιτρέπει τον καθορισμό δύο χρωμάτων για κάθε εικονοστοιχείο.

Καταχωρητής PISO (Parallel-In Serial-Out)

Η έξοδος του καταχωρητή PISO συνδέεται με την είσοδο επιλογής των πολυπλεκτών και καθορίζει αν η πληροφορία που θα απεικονιστεί, αφορά στο χρώμα προσκηνίου ή στο χρώμα του παρασκηνίου. Υπενθυμίζεται πως αυτή η επιλογή έγινε διότι το συγκεκριμένο περιφερειακό χρειάζεται μόλις έναν κύκλο για να λάβει τιμή και για τους επόμενους 8 πραγματοποιεί τη σειριακή μεταφορά της πληροφορίας, χωρίς να απασχολεί την κεντρική μονάδα.

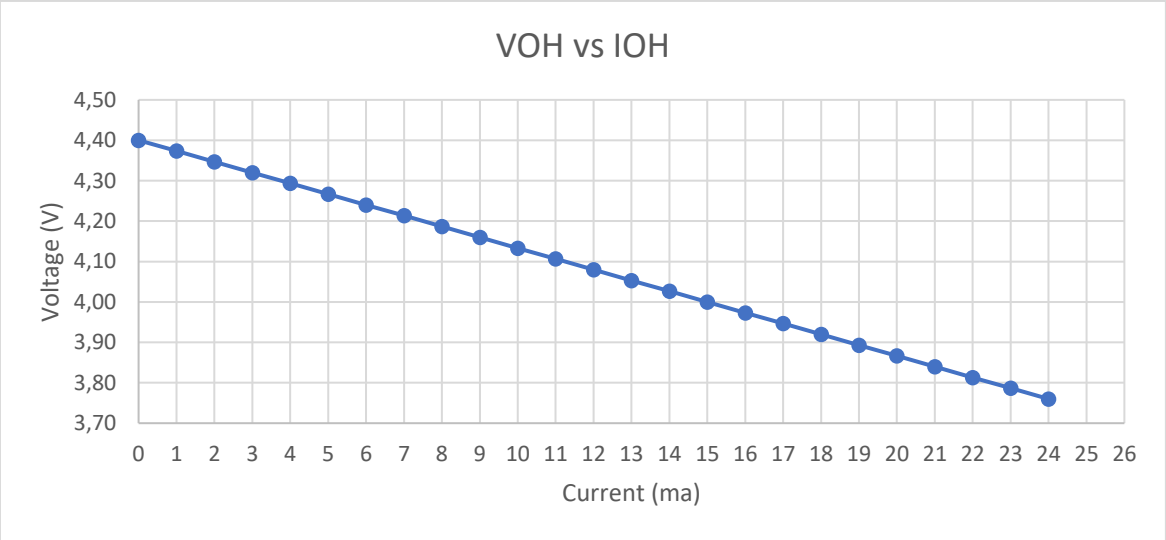
Πολυπλέκτες (MUX 2:1)

Για κάθε χρωματικό κανάλι RGB, χρησιμοποιούνται δύο πολυπλέκτες, επιλογής χρώματος προσκηνίου ή παρασκηνίου και χαμηλής ή υψηλής έντασης φωτεινότητας. Πρέπει να σημειωθεί ότι όλοι οι πολυπλέκτες φωτεινότητας διασυνδέονται, έτσι ώστε η επιλογή χαμηλής ή υψηλής έντασης, να αφορά και στα τρία χρωματικά κανάλια, αλλάζοντας ουσιαστικά τη στάθμη του σήματος που παράγουν οι DAC. Θεωρητικά, επαρκεί ένας πολυπλέκτης φωτεινότητας για να οδηγήσει όλα τα χρωματικά κανάλια. Ωστόσο, μετά από έρευνα στο εμπόριο, διαπιστώθηκε ότι οι πολυπλέκτες 2:1 της σειράς 74HC257 και

74AHC257 (high speed) που αποτελούν τους πιο διαδεδομένους, επιτρέπουν τη διέλευση ρεύματος έντασης 12-35 mA ανά ακροδέκτη (διαφέρει ανά κατασκευαστή και τάση λειτουργίας). Επιπλέον, η πτώση τάσης στις εξόδους διαφοροποιείται σε συνάρτηση με το ρεύμα εξόδου.

Για να εξασφαλιστεί η μέγιστη δυνατή σταθερότητα της τάσης εξόδου που οδηγεί τους DAC αλλά και η ευελιξία επιλογής πολυπλέκτη ανεξαρτήτως κατασκευαστή, επιλέχθηκε η χρήση ενός πολυπλέκτη ανά χρωματικό κανάλι. Με τον τρόπο αυτό αποφεύγονται ανεπιθύμητες μεταβολές στη στάθμη του σήματος και επιτυγχάνεται σταθερότερη χρωματική απεικόνιση.

Στο παρακάτω διάγραμμα δίνεται η θεωρητική τάση στην έξοδο του πολυπλέκτη, σε συνάρτηση με την κατανάλωση ρεύματος (τάση VCC=4.5V, εμπέδηση Z=26.67 Ohm).



Διάγραμμα 10: Πτώση τάσης σε συνάρτηση με το ρεύμα εξόδου (74AC257)

Ψηφιοαναλογικοί Μετατροπείς (DAC)

Οι έξοδοι κάθε ζεύγους πολυπλεκτών, τροφοδοτούν τον 2-bit DAC του αντίστοιχου χρωματικού καναλιού. Η χρωματική πληροφορία, έχοντας ήδη κβαντιστεί σε 4 bit, αντιστοιχίζεται σε 16 διακριτές στάθμες αναλογικού σήματος εύρους 0.0-0,7 V, το οποίο τροφοδοτείται στην οθόνη, παράγοντας 16 διαφορετικά χρώματα.

4.3 Αρχιτεκτονική Λογισμικού

Η αρχιτεκτονική λογισμικού αποτελεί θεμελιώδες στοιχείο του σχεδιασμού ενός ενσωματωμένου συστήματος καθώς καθορίζει τον τρόπο αλληλεπίδρασης μεταξύ των επιμέρους δομικών μονάδων. Η αρχιτεκτονική που ακολουθείται βασίζεται στην πολυεπίπεδη (layered) προσέγγιση, η οποία προσφέρει σαφή διαχωρισμό ευθυνών, εύκολη επεκτασιμότητα και βελτιωμένη δυνατότητα συντήρησης. Με την αξιοποίηση αυτής της σχεδιαστικής φιλοσοφίας, επιτυγχάνεται η τεχνική βιωσιμότητα του λογισμικού.

Η πολυεπίπεδη αρχιτεκτονική διαμορφώνεται από τρία βασικά επίπεδα:

- **Επίπεδο Εφαρμογής (Application/Core Layer):** Αποτελεί το ανώτερο επίπεδο λογισμικού, στο οποίο υλοποιείται η λογική της εφαρμογής και η αλληλεπίδραση με τον χρήστη. Σε αυτό το επίπεδο συγκεντρώνονται και επεξεργάζονται τα δεδομένα που προέρχονται από τα κατώτερα στρώματα.

- **Μεσαίο Επίπεδο (Middleware Modules):** Συνιστά τη λειτουργική γέφυρα ανάμεσα στα χαμηλού επιπέδου υποσυστήματα και το ανώτερο επίπεδο της εφαρμογής. Περιέχει δομικές και λειτουργικές μονάδες όπως κυκλικούς buffers (circular buffers), μονάδα διαχείρισης συμβάντων, μονάδα αποθορυβοποίησης διακοπτών (software debouncer), κτλ.

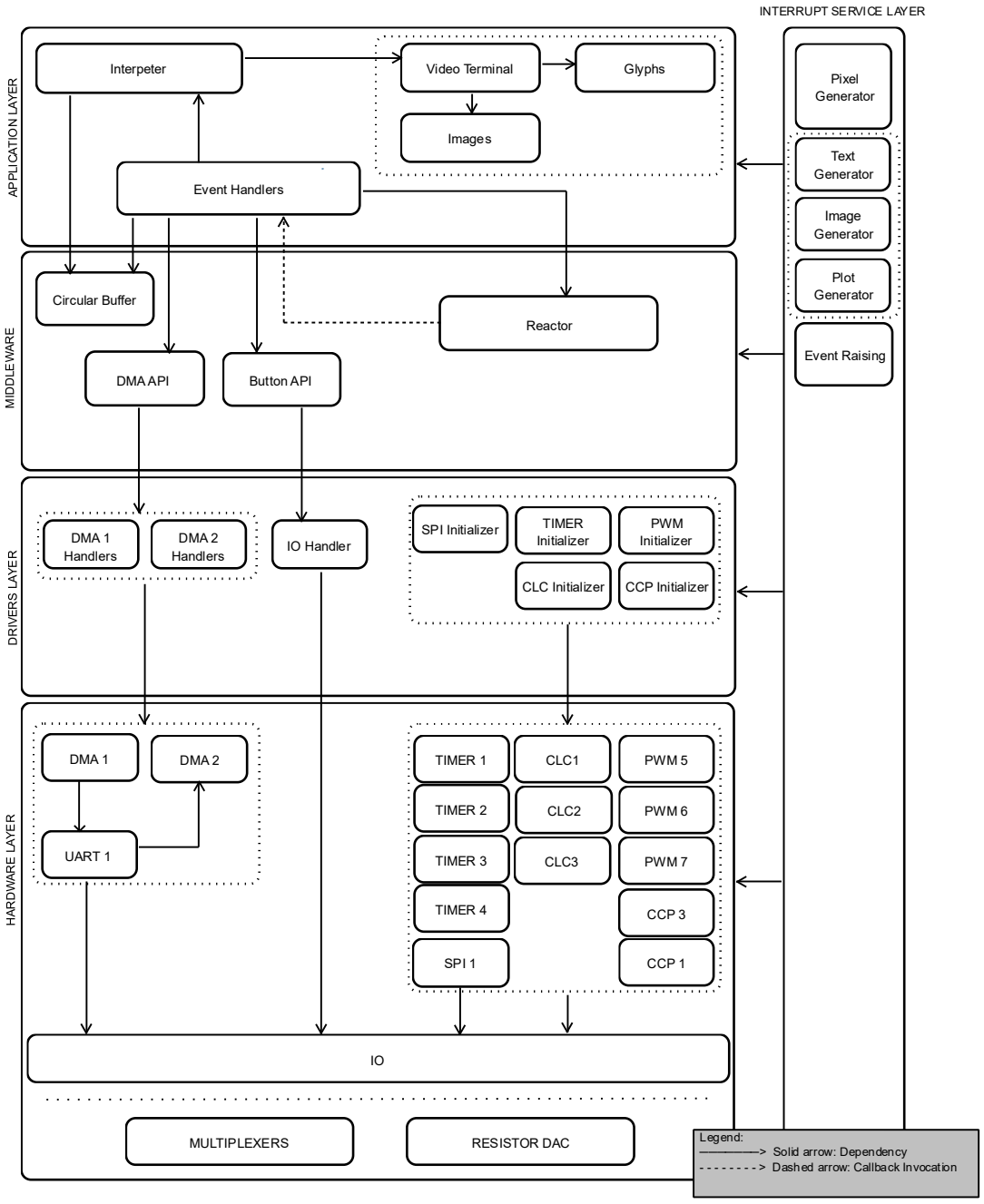
- **Επίπεδο Οδηγών (Low-level Drivers):** Περιλαμβάνει λογισμικές διεπαφές χαμηλού επιπέδου που διαχειρίζονται την απευθείας επικοινωνία με τον μικροελεγκτή και τα περιφερειακά του. Οι οδηγοί αυτοί υλοποιούν βασικές λειτουργίες όπως η πρόσβαση στις σειριακές διεπαφές SPI, UART, I2C, καθώς και στις γενικής χρήσης θύρες εισόδου/εξόδου (GPIO).

Στην παρούσα υλοποίηση έχει προστεθεί ένα ανεξάρτητο επίπεδο που δεν ακολουθεί την παραπάνω διαστρωμάτωση. Πρόκειται για το Επίπεδο Διακοπών (Interrupt Service Layer) το οποίο έχει απευθείας πρόσβαση σε όλα τα άλλα επίπεδα καθώς επίσης και τη

δυνατότητα απευθείας επικοινωνίας με το υλικό. Η ανάλυση της λειτουργίας του θα γίνει παρακάτω.

Η παρουσίαση της εσωτερικής δομής του λογισμικού μέσω block διαγράμματος αποσαφηνίζει τις σχέσεις μεταξύ των επιπέδων και αναδεικνύει τη ροή δεδομένων και εντολών. Επιπλέον, διευκολύνει την τεκμηρίωση, τον έλεγχο και την μελλοντική επέκταση του συστήματος, παρέχοντας μία πλήρη οπτική αναπαράσταση της αρχιτεκτονικής.
Αξίζει να σημειωθεί ότι η επιτυχής εφαρμογή της πολυεπίπεδης προσέγγισης προϋποθέτει σαφή οριοθέτηση των διεπαφών μεταξύ των επιπέδων καθώς και συνεπή τεκμηρίωση του κώδικα. Η ύπαρξη τέτοιων δομικών διαχωρισμών ενισχύει την ανεξαρτησία των επιμέρους modules, επιτρέποντας την παράλληλη ανάπτυξη, τον ευκολότερο εντοπισμό σφαλμάτων και την δυνατότητα επαναχρησιμοποίησης σε νέα έργα.

Η ανάλυση που ακολουθεί εστιάζει στην εσωτερική λειτουργία κάθε επιπέδου, αναλύει τους τρόπους επικοινωνίας μεταξύ τους και παρουσιάζει τους βασικούς σχεδιαστικούς μηχανισμούς που καθιστούν το λογισμικό αποδοτικό, ευέλικτο και μακροχρόνια βιώσιμο.



Διάγραμμα 11: Block Διάγραμμα Αρχιτεκτονικής Λογισμικού

4.3.1 Επίπεδο Εφαρμογής – Application Layer

Το Επίπεδο Εφαρμογής αποτελεί το ανώτερο επίπεδο και περιλαμβάνει τις μονάδες:

- **Interpreter** – Πρόκειται για τον διερμηνευτή των εντολών της κάρτας γραφικών. Επεξεργάζεται τις εισερχόμενες εντολές και τις μεταφράζει σε κατάλληλες λειτουργίες για το υποσύστημα γραφικών.
- **Video Terminal** – Αποτελεί την μονάδα παραγωγής γραφικών. Περιλαμβάνει μια λίστα εντολών η κάθε μια από τις οποίες έχει και συγκεκριμένη λειτουργικότητα σχετική με την παραγωγή γραφικών. Τα γραφικά δεν οδηγούνται απευθείας στην οθόνη αλλά μετά αποθηκεύονται στον Video Buffer. Αποτελείται επιπλέον από τις ακόλουθες υπομονάδες:
    - Glyphs – Μονάδα Γραφικών Συμβόλων. Αντιστοιχίζει κάθε χαρακτήρα στο αντίστοιχο γλυφικό (γραφικό σύμβολο) και υποστηρίζει δύο γραμματοσειρές.
    - Images – Μονάδα Διαχείρισης Εικόνων. Παρέχει δυνατότητες διαχείρισης και προβολής έως τεσσάρων φωτογραφιών διαστάσεων 360 x 480.
- **Event Handlers** – Αποτελεί μια συλλογή από ενέργειες που εκτελούνται όταν προκύψει κάποιο συμβάν. Κάθε event handler εγγράφεται στη λίστα εξυπηρέτησης του Reactor (middleware) και ενεργοποιείται αυτόματα από τον Reactor μόλις προκύψει το αντίστοιχο γεγονός.

4.3.2 Μεσαίο Επίπεδο - Middleware Layer

Το επίπεδο Middleware περιλαμβάνει τις μονάδες:

- **Reactor** – Πρόκειται για τη Μονάδα Διαχείρισης Συμβάντων. Ακουλουθεί το ομώνυμο πρότυπο σχεδίασης «Reactor Pattern» και παρακολουθεί πηγές συμβάντων όπως εισερχόμενα δεδομένα, πάτημα κουμπιών κτλ και ενεργοποιεί τον αντίστοιχο Event Handler που είναι εγγεγραμμένος στη λίστα εξυπηρέτησης.

- **Circular Buffer** – Προσφέρει λειτουργικότητα για τη διαχείριση δομών δεδομένων σταθερού μεγέθους που λειτουργούν σαν κυκλική ουρά (queue). Η λειτουργία του

βασίζεται στην κυκλική διαχείριση ενός πίνακα, όπου τα δεδομένα εισάγονται και ανακτώνται μέσω δύο δεικτών — του δείκτη εγγραφής και του δείκτη ανάγνωσης. Όταν ο δείκτης φτάσει στο τέλος του πίνακα, επιστρέφει στην αρχή, επιτρέποντας συνεχή ροή δεδομένων χωρίς την ανάγκη μετακίνησης ή αναδιάταξης τους.

- **Button API** – Η μονάδα παρέχει διεπαφές για την ανάγνωση της κατάστασης των δυο μηχανικών διακοπτών (tactile switch) που φιλοξενεί η πλακέτα και ταυτόχρονα προσφέρει λειτουργικότητα που αντιμετωπίζει το φαινόμενο της "αναπήδησης" (bouncing) που προκύπτει κατά την πίεση ενός μηχανικού διακόπτη. Όταν ένας χρήστης πατήσει ή ελευθερώσει ένα button, οι επαφές του δεν κλείνουν ή ανοίγουν αμέσως με σταθερό τρόπο· αντίθετα, δημιουργούνται γρήγορες και ανεπιθύμητες εναλλαγές μεταξύ των καταστάσεων ON και OFF για λίγα χιλιοστά του δευτερολέπτου. Αυτό μπορεί να προκαλέσει πολλαπλές λανθασμένες αναγνώσεις από τον μικροελεγκτή οπότε είναι απαραίτητη η αποθορυβοποίηση [9] (debouncing).

- **DMA API** – Η μονάδα παρέχει διεπαφές για την επικοινωνία του ανώτερου επιπέδου με τους οδηγούς DMA του επιπέδου οδηγών. Επίσης προσφέρει βασικές βοηθητικές μεθόδους για την διευκόλυνση της μεταφοράς δεδομένων χρησιμοποιώντας τις DMA μονάδες.

Οι μονάδες του επιπέδου Middleware έχουν σχεδιαστεί με γνώμονα την επαναχρησιμοποίηση, δηλαδή τη δυνατότητα να ενσωματωθούν εύκολα σε διαφορετικά έργα χωρίς να απαιτούνται σημαντικές τροποποιήσεις.

Η επαναχρησιμοποίηση αποτελεί βασική αρχή της σύγχρονης σχεδίασης λογισμικού, καθώς συμβάλλει στην εξοικονόμηση χρόνου ανάπτυξης, στη μείωση των σφαλμάτων και στην αύξηση της αξιοπιστίας του συστήματος.

Για να επιτευχθεί αυτός ο στόχος, κάθε μονάδα Middleware υλοποιείται με τρόπο ανεξάρτητο από το υπόλοιπο σύστημα. Οι λειτουργίες της είναι σαφώς ορισμένες μέσω

γενικών και καλά τεκμηριωμένων διεπαφών (APIs), ενώ η εσωτερική της λογική είναι απομονωμένη από τις λεπτομέρειες του hardware και της εφαρμογής. Έτσι, για παράδειγμα, η μονάδα Circular Buffer μπορεί να χρησιμοποιηθεί τόσο για την προσωρινή αποθήκευση δεδομένων από σειριακή επικοινωνία όσο και για buffering αισθητήρων, χωρίς να απαιτείται αλλαγή στον πυρήνα του κώδικα.

Επιπλέον, η modular σχεδίαση των μονάδων επιτρέπει την ανεξάρτητη συντήρηση και εξέλιξή τους. Αν μια μονάδα, όπως ο Button Debouncer, χρειαστεί βελτίωση ή προσαρμογή για διαφορετικό τύπο κουμπιού ή χρονισμού, μπορεί να τροποποιηθεί χωρίς να επηρεαστούν οι υπόλοιπες μονάδες του συστήματος. Αυτό ενισχύει τη διαλειτουργικότητα και την ευελιξία του λογισμικού, καθιστώντας το κατάλληλο για χρήση σε ευρύ φάσμα embedded εφαρμογών — από απλά bare metal projects έως πιο σύνθετα συστήματα με RTOS.

Τέλος, η επαναχρησιμοποίηση ενισχύεται και από την τυποποίηση του κώδικα, την τεκμηρίωση και την οργάνωση των αρχείων σε ξεχωριστές βιβλιοθήκες ή φακέλους. Με αυτόν τον τρόπο, οι μονάδες Middleware μπορούν να ενσωματωθούν γρήγορα σε νέα έργα, να δοκιμαστούν ανεξάρτητα και να επαναχρησιμοποιηθούν με ελάχιστο κόστος ανάπτυξης.

4.3.3 Επίπεδο Οδηγών – Drivers Layer

Το επίπεδο οδηγών αποτελεί το σημείο σύνδεσης μεταξύ του λογισμικού και του υλικού (hardware). Οι οδηγοί είναι υπεύθυνοι για τη διαχείριση των περιφερειακών μονάδων του μικροελεγκτή, όπως UART, SPI, DMA, κτλ και γενικότερα οποιοδήποτε στοιχείο απαιτεί άμεση επικοινωνία με το φυσικό επίπεδο. Η λειτουργία τους περιλαμβάνει την αρχικοποίηση των συσκευών αλλά και την παροχή διεπαφών προς το υλικό, ώστε τα ανώτερα επίπεδα να μην χρειάζεται να γνωρίζουν λεπτομέρειες για κάθε συσκευή και να αλληλεπιδρούν με αυτά μόνο μέσω των κατάλληλων διεπαφών.
Για την ανάπτυξη του έχει χρησιμοποιηθεί σε μεγάλο βαθμό το εργαλείο MCC – Microchip Code Configurator (αντίστοιχο του STM32cubeMX).

4.3.4 Επίπεδο Διακοπών - Interrupt Service Layer

Όπως φαίνεται και στο block διάγραμμα, το επίπεδο διακοπών δεν εντάσσεται στην κλασσική λογική διαστρωμάτωσης και λειτουργεί παράλληλα με τα υπόλοιπα επίπεδα (οδηγών, middleware, εφαρμογής). Η επιλογή αυτή, έγινε με κύριο στόχο τη συνολική αύξηση της ταχύτητας του συστήματος. Οι μονάδες του επιπέδου μπορούν να έχουν πρόσβαση σε κάθε άλλο επίπεδο συμπεριλαμβανομένου και του φυσικού επιπέδου, χωρίς να μεσολαβούν abstractions. Αυτό ασφαλώς μειώνει τη δυνατότητα επαναχρησιμοποίησης του αλλά αυξάνει σημαντικά τη συνολική ταχύτητα του συστήματος.

Στην παρούσα υλοποίηση, η εκτεταμένη χρήση assembly για την παραγωγή του αναλογικού σήματος παραγωγής εικόνας, λειτουργία η οποία εκτελείται από το Επίπεδο Διακοπών, είχε ήδη ως αποτέλεσμα τη χαμηλή επαναχρησιμοποίηση του συγκεκριμένου τμήματος κώδικα. Συνεπώς, η συγκεκριμένη σχεδιαστική επιλογή δεν επιδείνωσε περαιτέρω αυτήν την παράμετρο αλλά αντιθέτως, ανέδειξε έναν σαφή λειτουργικό διαχωρισμό του επιπέδου διακοπών από τα υπόλοιπα τμήματα του συστήματος. Αυτός ο διαχωρισμός διευκολύνει την προσαρμογή του λογισμικού σε διαφορετικούς μικροελεγκτές, καθώς το επίπεδο διακοπών είναι το μόνο που απαιτεί τροποποίηση για να ανταποκριθεί στις ιδιαιτερότητες του νέου υλικού (hardware). Τα επίπεδα Εφαρμογής και Middleware, παραμένουν πλήρως επαναχρησιμοποιήσιμα, ενώ το επίπεδο των οδηγών, μπορεί να επαναδημιουργηθεί εύκολα με την χρήση του εργαλείου MCC, γεγονός που απλοποιεί τη διαδικασία μεταφοράς και επαναχρησιμοποίησης του λογισμικού σε νέο μικροελεγκτή.

Περιλαμβάνει τις μονάδες:

- VSingal Generator – Παράγει το σήμα κάθετου χρονισμού
- Pixel Generator – αποτελούμενη από τις υπομονάδες Text, Image και Plot Generator, οι οποίες παράγουν το αναλογικό σήμα για την παραγωγή κειμένου, στατικών φωτογραφιών και παραγωγής απλών γραφικών σχεδίων αντίστοιχα.

5. Υλοποίηση Λογισμικού

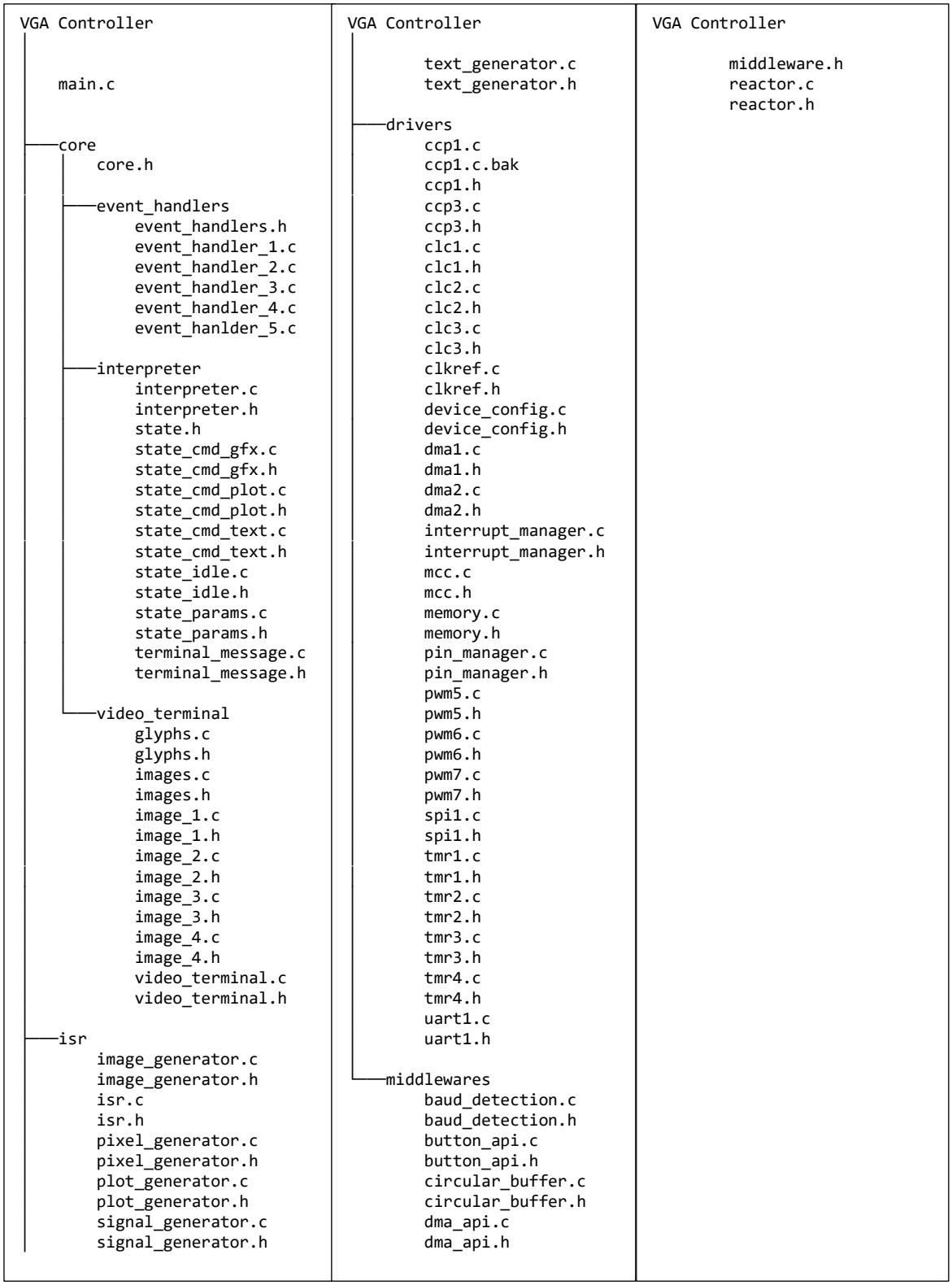
Η συγγραφή του κώδικα έγινε στο MPLAB X IDE και στο NetBeans IDE χρησιμοποιώντας την C99 και τους compilers XC8 και GCC αντίστοιχα.

Η πολυεπίπεδη αρχιτεκτονική που εφαρμόστηκε επέτρεψε την συγγραφή των δυο ανώτερων επιπέδων, σχεδόν αποκλειστικά στο NetBeans IDE χρησιμοποιώντας τον GCC compiler, καθώς δεν απαιτούνταν φυσική σύνδεση με το υλικό. Αυτό συνέβαλε καθοριστικά στην αύξηση της ταχύτητας συγγραφής του κώδικα, κυρίως λόγω της ευκολίας που προσφέρει το NetBeans στον εντοπισμό και τη διόρθωση σφαλμάτων (debugging) αλλά και στην αυξημένη ταχύτητα μεταγλώττισης.

Αφού ολοκληρώθηκαν οι δοκιμές και η λειτουργικότητα των ανώτερων επιπέδων επιβεβαιώθηκε, ο κώδικας ενσωματώθηκε στην κύρια εφαρμογή, η οποία αναπτύχθηκε στο MPLAB X IDE. Εκεί, ολόκληρο το σύστημα μεταγλωττίστηκε εκ νέου με τον XC8 compiler, ώστε να είναι συμβατό με την αρχιτεκτονική του μικροελεγκτή και να μπορεί να εκτελεστεί στο τελικό embedded περιβάλλον. Η μετάβαση από τον GCC στον XC8 δεν απαίτησε παρά ελάχιστες επιπλέον προσαρμογές, καθώς χρησιμοποιήθηκαν κοινές βιβλιοθήκες της C, που υποστηρίζονται και από τους δύο μεταγλωττιστές.

Το MPLAB X IDE χρησιμοποιήθηκε κατά βάση για την ανάπτυξη του χαμηλότερου επιπέδου (firmware) και του επιπέδου διακοπών. Η εκσφαλμάτωση ήταν σημαντικά πιο αργή, καθώς είτε εκτελέστηκε απευθείας στο υλικό, χρησιμοποιώντας εξωτερικό εξοπλισμό (PICkit Debuger), γενονός που προϋπόθετε τη χρονοβόρα διαδικασία μεταφοράς του κώδικα στον μικροελεγκτή, είτε πραγματοποιήθηκε στο ενσωματωμένο περιβάλλον προσομοίωσης που προσφέρει το MPLAB X. Και στις δυο περιπτώσεις η διαδικασία ήταν σημαντικά πιο αργή σε σχέση με το NetBeans.

Ο κώδικας οργανώθηκε σε διακριτούς φακέλους έτσι ώστε να γίνεται σαφής διαχωρισμός ανάμεσα στα επίπεδα αρχιτεκτονικής που περιγράφηκαν νωρίτερα. Ακολουθεί ένα δεντρικό διάγραμμα που αποτυπώνει τη δομή των φακέλων και των αρχείων.



Διάγραμμα 12: Διάγραμμα Δομής Φακέλων & Αρχείων

Θα ακολουθήσει η ανάλυση των βασικών δομικών μονάδων της εφαρμογής, ενώ ο πλήρης κώδικας θα συμπεριληφθεί στο αντίστοιχο παράρτημα.

5.1 Intepreter

Για την επικοινωνία με τον έξω κόσμο η κάρτα γραφικών πρέπει να μπορεί να δέχεται και να ερμηνεύει μια λίστα εντολών, οι οποίες ελέγχουν τη λειτουργία της. Στην πραγματικότητα λειτουργεί ως ένα τερματικό εντολών [10]. Κάθε εντολή που λαμβάνεται μέσω του διαύλου επικοινωνίας UART, επεξεργάζεται κατάλληλα, εξετάζεται ως προς τη συντακτική της ορθότητα και κατόπιν εκτελείται η αντίστοιχη λειτουργία. Επομένως, είναι απαραίτητη η κατασκευή μιας υποτυπώδους γλώσσας επικοινωνίας και ενός διερμηνευτή της γλώσσας αυτής.

Τα τερματικά του παρελθόντος χρησιμοποιούσαν ένα τυποποιημένο σύνολο εντολών ελέγχου· γνωστό, ως ANSI Escape Codes [11]. Το πρωτόκολλο αυτό παραμένει ενεργό έως σήμερα για την παραμετροποίηση και τον έλεγχο της εμφάνισης χαρακτήρων σε εξομοιωτές τερματικών, καθώς και για την αλληλεπίδραση με συστήματα που βασίζονται σε σειριακή επικοινωνία.

Ακολουθώντας αυτή την προσέγγιση, σχεδιάζεται ένα σύνολο εντολών βασισμένο σε γραμματική χωρίς συμφραζόμενα (CFG), με στόχο τη σαφήνεια και την αυστηρή τυποποίηση των συντακτικών κανόνων. Τα τερματικά και μη τερματικά σύμβολα αλλά και οι κανόνες παραγωγής της γλώσσας, ορίζονται χρησιμοποιώντας τεχνική συμβολισμού Backus-Naur Form [12] (BNF). Η εφαρμογή της τεχνικής αυτής εξασφαλίζει την ακριβή και αυστηρή περιγραφή της γλώσσας, επιτρέποντας επίσης την εύκολη μελλοντική της επέκταση.

5.1.1 Γραμματική σε μορφή BNF

$$< esc\ code > ::= < esc > < command > | < character >$$
$$< command > ::= < alpha > < parameters\ seq > | < alpha >$$
$$< parameters\ seq > ::= < parameter > < parameter\ tail >$$
$$< parameters\ tail > ::= < parameter\ delimeter > < parameters\ seq > \ | \ < parameter\ end >$$
$$< parameter > ::= < digit > < parameter > | < digit >$$

$< digit >$::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

$< parameter\ delimeter >$::= " , "

$< parameter\ end >$::= $0x0d$ | " ; "

$< esc >$::= $0x1b$

$< alpha >$::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" |
"o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" |
"A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" |
"O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

$< character >$::=$< digit >$ | $< alpha >$

Από το παραπάνω σύνολο κανόνων σχηματίζονται τρεις ομάδες εντολών που αντιστοιχούν σε κάθε μια από τις καταστάσεις λειτουργίας του ελεγκτή VGA.

5.1.2 Λίστα Εντολών

Εντολές Κατάστασης Κειμένου

$< ESC > E;$	Clear Screen
$< ESC > O;$	Set Cursor ON
$< ESC > F;$	Set Cursor OFF
$< ESC > B;$	Blink ON
$< ESC > B;$	Blink OFF
$< ESC > y < x >,< y >;$	Set Cursor Pos
$< ESC > b < color >;$	Set Background Color
$< ESC > f < color >;$	Set Foreground Color
$< ESC > m < mode >;$	Change Video Mode
$< ESC > d < x_1 >,< y_1 >,< x_2 >,< y_2 >,< 1 >;$	Draw Thin Rectangle
$< ESC > d < x_1 >,< y_1 >,< x_2 >,< y_2 >,< 2 >;$	Draw Thick Rectangle
$< ESC > d < x_1 >,< y_1 >,< x_2 >,< y_2 >,< 10 >;$	Draw Thin Window
$< ESC > d < x_1 >,< y_1 >,< x_2 >,< y_2 >,< 20 >;$	Draw Thick Window

Εντολές Κατάστασης Γραφικών

$< ESC > p < f >,< b >,< x >,< y >;$	Set fore/back ground color at {x,y} image cell
$< ESC > p < f >,< b >;$	Paint Picture
$< ESC > l < picture >;$	Load Picture
$< ESC > m < mode >;$	Change Video Mode

Εντολές Κατάστασης Σχεδίασης

$< ESC > E$	Clear Screen
$< ESC > b \ < color >;$	Set Background Color
$< ESC > f \ < color >;$	Set Foreground Color
$< ESC > p \ < f >, < b >;$	Set both colors
$< ESC > l \ < x_1 >, < y_1 >, < x_2 >, < y_2 >, < color >;$	Draw Line
$< ESC > c \ < x >, < y >, < radius >, < color >;$	Draw Circle
$< ESC > s \ < x >, < y >, < color >;$	Set Pixel
$< ESC > m \ < mode >;$	Change Video Mode

Για την διερμήνευση των εντολών, είναι απαραίτητη η κατασκευή μιας μηχανής πεπερασμένων καταστάσεων (Finite State Machine – FSM) η οποία θα αποτελεί τον πυρήνα του διερμηνευτή εντολών. Τα εισερχόμενα από το δίαυλο UART σύμβολα ελέγχονται ως προς την συμφωνία τους με τη γραμματική της γλώσσας εντολών, καθιστώντας τη μηχανή FSM έναν συντακτικό αναλυτή πραγματικού χρόνου, ο οποίος μεταβαίνει μεταξύ προκαθορισμένων καταστάσεων με βάση το κάθε σύμβολο που λαμβάνεται. Η μέθοδος που επιλέχθηκε για την υλοποίηση της FSM, βασίζεται στο State Pattern [13], ένα σχεδιαστικό πρότυπο που επιτρέπει την αλλαγή συμπεριφοράς ενός αντικειμένου ανάλογα με την τρέχουσα κατάσταση του συστήματος. Η επιλογή του συγκεκριμένου προτύπου έγινε γιατί πλεονεκτεί σημαντικά, σε σχέση με την παραδοσιακή προσέγγιση κατασκευής FSM με πολλαπλά nested conditional statements, κυρίως σε χρονική πολυπλοκότητα [14], ενώ προσφέρει εξαιρετική ευκολία στη συντήρηση αλλά και στη μελλοντική επέκταση, με ουσιαστικό κόστος μόνο τον επιπλέον χρόνο σχεδίασης και ανάπτυξης.

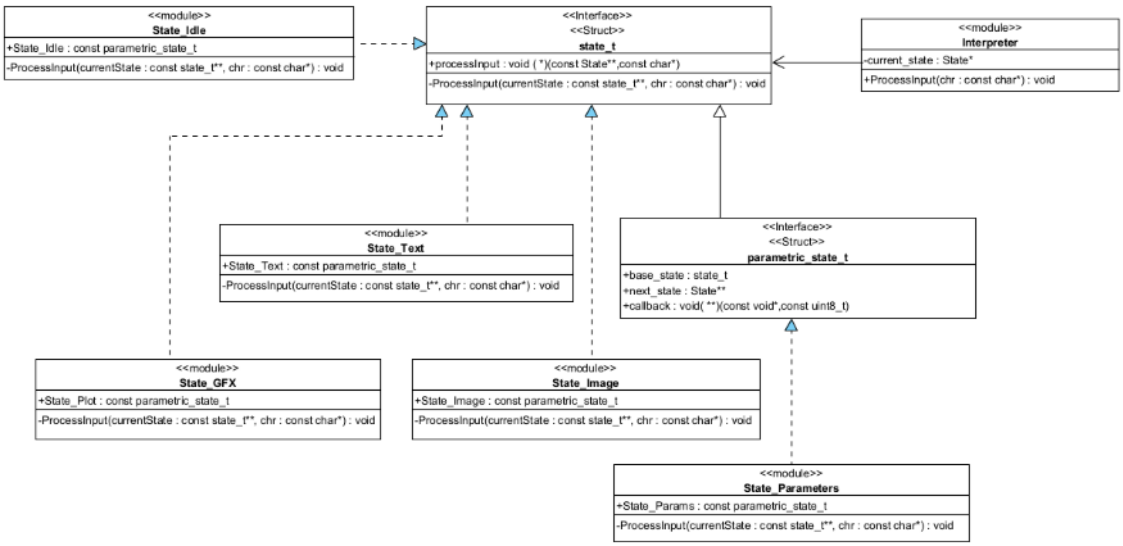
5.1.3 Αρχιτεκτονική

Η υλοποίηση του προτύπου State Pattern συνίσταται σε τρεις βασικές έννοιες:

- Context: Αντιπροσωπεύει τον κορμό της μηχανής FSM. Διατηρεί μια αναφορά (pointer) στα αντικείμενα τύπου State και δίνει τον έλεγχο σε αυτό που αποτελεί την τρέχουσα ενεργή κατάσταση.

- State Interface: Αποτελεί το κοινό Interface το οποίο εφαρμόζει κάθε αντικείμενο κατάστασης. Η επικοινωνία του Context με τα States γίνεται μέσω της διεπαφής αυτής.
- Concrete States: Δομές που υλοποιούν τη λογική κάθε επιμέρους κατάστασης. Υλοποιούν το State Interface και παρέχουν τον μηχανισμό μετάβασης προς κάθε άλλο State

Ακολουθεί το διάγραμμα κλάσεων σε μορφή UML. Αν και η γλώσσα C δεν υποστηρίζει εγγενώς τη δημιουργία κλάσεων, ο μηχανισμός λειτουργίας τους μπορεί να προσομοιωθεί με την χρήση των απλούστερων δομών struct και με την εκτεταμένη χρήση function pointers.



Διάγραμμα 13: State Pattern Class Diagram

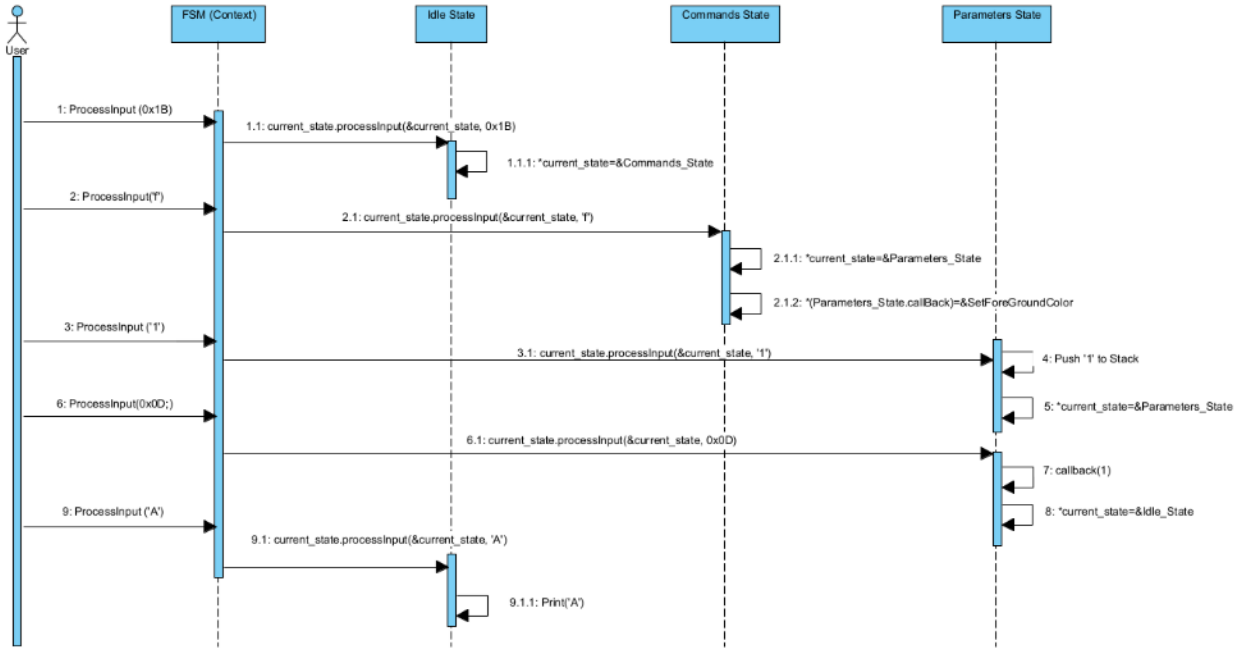
Όλα τα States υλοποιούν το State Interface. Αυτό εξασφαλίζει ότι το Context, μπορεί να καλεί την κοινή μέθοδο ProcessInput, εφόσον γνωρίζει πως κάθε State εφαρμόζει την παραπάνω σύμβαση. Με τον τρόπο αυτό επιτυγχάνεται αποσύζευξη μεταξύ του πυρήνα της FSM και της λογικής των επιμέρους καταστάσεων, καθώς δεν απαιτείται άμεση πρόσβαση στις εσωτερικές μεθόδους κάθε State.

Επιπλέον, η χρήση διπλού δείκτη (pointer to pointer) για τον καθορισμό της τρέχουσας κατάστασης (State) επιτρέπει σε κάθε αντικείμενο State να τροποποιεί την ενεργή

κατάσταση της μηχανής καταστάσεων χωρίς να απαιτείται άμεση πρόσβαση στο αντικείμενο Context. Με αυτόν τον τρόπο, επιτυγχάνεται πλήρης απομόνωση της λογικής μετάβασης αλλά ολοκληρωτική αποσύζευξη μεταξύ των επιμέρους καταστάσεων και του κεντρικού ελεγκτή καθώς δεν υπάρχει καμία αναφορά σε αυτόν από τα αντικείμενα κατάστασης και το αντίστροφο.

Τέλος πρέπει να σημειωθεί ότι το Parameters State Interface επεκτείνει το Generic State Interface και δίνει πρόσθετη λειτουργικότητα στο State Parameters που το υλοποιεί. Η επιλογή αυτής της προσέγγισης θα καταστεί σαφής στο διάγραμμα ακολουθίας, όπου αποτυπώνεται η δυναμική αλληλεπίδραση μεταξύ των State και Context.

Παρατίθεται ενδεικτικά, το διάγραμμα ακολουθίας για την προβολή στην οθόνη του χαρακτήρα «A» σε μπλε χρώμα.



Διάγραμμα 14: FSM Sequence Diagram

5.1.4 Υλοποίηση

Ακολουθούν ενδεικτικά στιγμιότυπα του κώδικα σε γλώσσα C, για την κατασκευή του State Pattern. Ο πλήρης κώδικας θα δοθεί στο αντίστοιχο παράρτημα.

C code - Αρχείο «state.h»

```
#ifndef STATE_H
#define STATE_H

#include <stdint.h>

/**
 * Generic State Interface
 **/
typedef struct State {
    void (*processInput)(const struct State **currentState, const
char* chr);
}state_t;

/**
 * Parametric State interface
*/
typedef struct {
    state_t base_state;
    void (**callback)(const void* args, const uint8_t count);
    const state_t **nextState;
} parametric_state_t;
```

Αρχείο «interpreter.h» (Context)

```
#ifndef INTERPRETER_H
#define INTERPRETER_H
#include "state.h"

void INTERPRETER_Process_Input(const char* chr);
#endif /* INTERPRETER_H */
```

Αρχείο «interpreter.c» (Context)

```
#include "interpreter.h"
#include "state_idle.h"

typedef struct {
    const state_t *currentState;
} context_t;

static context_t Context= {
    .currentState=&StateIdleText
};

void INTERPRETER_Process_Input(const char* chr){
    Context.currentState->processInput(&Context.currentState,chr);
}
```

C code - Αρχείο « state_cmd_text.c »

```
#include "state_params.h"
#include "state_idle.h"
#include "video_terminal.h"
#include "state.h"
#include <ctype.h>
#include "middleware.h"
```

```
#include "terminal_message.h"
#include "glyphs.h"
#include "images.h"
#include "signal_generator.h"

//wrapper functions for the function pointer
static void CmdSetForeGroundColor(const void *var,uint8_t count);
static void CmdSetBackGroundColor(const void *var,uint8_t count);
static void CmdSetCursorXY(const void *var,uint8_t count);
static void CmdDrawBox(const void *var,uint8_t count);
static void CmdDrawWindow(const void *var,uint8_t count);
static void CmdScrollBoxRegionUp(const void *var,uint8_t count);
static void CmdDrawWindowSeparator(const void *var,uint8_t count);
static void CmdFillRegion(const void *var,uint8_t count);
static void CmdChangeMode(const void *var,uint8_t count);
static void CmdBlinkCursor(const void *var,uint8_t count);
static void CmdDrawColor(const void *var,uint8_t count);


static void ProcessInput(const state_t **currentState, const char*
chr){

    switch (*chr){
        //ESC f <c> Foreground color
        case 'f':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdSetForeGroundColor;

            break;
        //ESC b <c> Background color
        case 'b':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdSetBackGroundColor;
            break;
        //ESC b <bool> Set Cursor Blink ON/Off
        case 'x':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdBlinkCursor;
            break;

        //ESC Y <y> <x> Set cursor position
        case 'Y':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdSetCursorXY;
            break;
        //ESC E Clear screen
        case 'E':
            VideoTerminal.clearScreen();
            TERMINAL_MESSAGE_Send(MESSAGE_ACK);
            *currentState=&StateIdleText;

            break;
        //ESC S Scroll Up by one line
        case 'S':
            VideoTerminal.scrollUp();
            TERMINAL_MESSAGE_Send(MESSAGE_ACK);
            *currentState=&StateIdleText;

            break;
        //ESC s Scroll Up a boxed region
        case 's':
```

```
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdScrollBoxRegionUp;
            break;
        //ESC n Send CR LF
        case 'n':
            VideoTerminal.crlf();
            TERMINAL_MESSAGE_Send(MESSAGE_ACK);
            *currentState=&StateIdleText;
            break;
        //ESC d draw box
        case 'd':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdDrawBox;;
            break;
        //ESC w draw window
        case 'w':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdDrawWindow;
            break;
        //ESC W draw window separator
        case 'W':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdDrawWindowSeparator;
            break;
        //ESC e fill region with character
        case 'e':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdFillRegion;
            break;
        case 'm':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdChangeMode;
            break;
        case 'p':
            *currentState=(state_t*)&StateParams;
            *(StateParams.callBack)=&CmdDrawColor;
            break;
        default:
            *currentState=&StateIdleText;
            TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    }
}
const state_t StateCmdText = {
    .processInput=&ProcessInput
};


static void CmdSetBackGroundColor(const void *var,uint8_t count){
    (void)count;
    uint8_t color=(uint8_t)*((const uint16_t*)var);

    if (!VideoTerminal.setBackGroundColor(color)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }
}

static void CmdSetForeGroundColor(const void *var,uint8_t count){
    uint8_t color=(uint8_t)*((const uint16_t*)var);
    (void) count;
```

```
    if (!VideoTerminal.setForeGroundColor(color)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }
}

static void CmdSetCursorXY(const void *var,uint8_t count){
    (void)count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t x = (uint8_t)*ptr++;
    uint8_t y = (uint8_t)*ptr;
    if (!VideoTerminal.locateXY(x, y)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }
}

static void CmdDrawBox(const void *var,uint8_t count){
    (void)count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t x1 = (uint8_t)*ptr++;
    uint8_t y1 = (uint8_t)*ptr++;
    uint8_t x2 = (uint8_t)*ptr++;
    uint8_t y2 = (uint8_t)*ptr++;
    uint8_t style=(uint8_t)*ptr;
    if (!VideoTerminal.drawBox(x1,y1,x2,y2,style)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }

};

static void CmdDrawWindow(const void *var,uint8_t count){
    (void)count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t x1 = (uint8_t)*ptr++;
    uint8_t y1 = (uint8_t)*ptr++;
    uint8_t x2 = (uint8_t)*ptr++;
    uint8_t y2 = (uint8_t)*ptr++;
    uint8_t style = (uint8_t)*ptr;

    if (!VideoTerminal.drawWindow(x1,y1,x2,y2,style)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }

    VideoTerminal.locateXY(x1+2,y1+1);
};

static void CmdDrawWindowSeparator(const void *var,uint8_t count){
    (void)count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t x1 = (uint8_t)*ptr++;
    uint8_t y  = (uint8_t)*ptr++;
    uint8_t x2 = (uint8_t)*ptr++;
    uint8_t style = (uint8_t)*ptr;
```

```
    if (!VideoTerminal.drawSeparator(x1,y,x2,style)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }
}

static void CmdScrollBoxRegionUp(const void *var,uint8_t count){
    (void)count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t x1 = (uint8_t)*ptr++;
    uint8_t y1 = (uint8_t)*ptr++;
    uint8_t x2 = (uint8_t)*ptr++;
    uint8_t y2 = (uint8_t)*ptr++;
    uint8_t direction = (uint8_t)*ptr;

    if (!VideoTerminal.scrollBox(x1,y1,x2,y2,direction)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }

}

static void CmdFillRegion(const void *var,uint8_t count){
    (void)count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t x1 = (uint8_t)*ptr++;
    uint8_t y1 = (uint8_t)*ptr++;
    uint8_t x2 = (uint8_t)*ptr++;
    uint8_t y2 = (uint8_t)*ptr++;
    uint8_t chr = (uint8_t)*ptr;

    if (!VideoTerminal.fillRegion(x1,y1,x2,y2,chr)){
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    } else{
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }
}

static void CmdChangeMode(const void *var,uint8_t count) {
    (void) count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t mode = (uint8_t)*ptr;
     if(mode==1) {
        SIGNAL_GENERATOR_SetMode(1);
        IMAGE_Inititalize();
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
        *(StateParams.nextState)=&StateIdleGfx;


    }else if(mode==2) {
        VideoTerminal.clearBuffer();
        *(StateParams.nextState)=&StateIdlePlot;
        SIGNAL_GENERATOR_SetMode(2);
        TERMINAL_MESSAGE_Send(MESSAGE_ACK);
    }else{
        TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
    }
```

```
}

static void CmdDrawColor(const void *var,uint8_t count){
    (void) count;
    const uint16_t *ptr = (const uint16_t*)var;
    uint8_t bc = (uint8_t)*ptr++;
    uint8_t fc = (uint8_t)*ptr;
    VideoTerminal.drawColor(bc,fc);
}

static void CmdBlinkCursor(const void *var,uint8_t count){
    (void) count;
    const uint16_t *ptr = (const uint16_t*)var;
    bool blink = (bool)*ptr;
    VideoTerminal.setBlink(blink);
}
```

Αρχείο «state_params.c»

```
#include <ctype.h>
#include "state_params.h"
#include "state_idle.h"
#include "state.h"
#include "terminal_message.h"
#include "stddef.h"
#include "string.h"

#define MAX_NO_OF_ARGUMENTS 8
#define PARAMETER_DELIMETER ','
#define PARAMETER_END1 0x0d
#define PARAMETER_END2 ';'
static void (*dynamicCallback)(const void*, uint8_t)=NULL;
static const state_t *next_state=&StateIdleText;
static void ProcessInput(const state_t **currentState, const char*
chr);

const parametric_state_t StateParams={
    .base_state.processInput=&ProcessInput,
    .callBack=&dynamicCallback,
    .nextState=&next_state
};

static void ProcessInput(const state_t **currentState, const char*
chr){
    static uint16_t stack[MAX_NO_OF_ARGUMENTS];
    static uint8_t paramsCounter=0;
    static uint8_t digitsCounter=0;
    static uint16_t num=0;

    //check if incoming character is digit and convert it to number
    if (isdigit(*chr)){
        num=num*10;
        num+=(uint8_t)*chr-'0';
        digitsCounter++;
        return;

    }else if(*chr==PARAMETER_DELIMETER) {
        digitsCounter=0;
```

```
            stack[paramsCounter]=num;
            paramsCounter++;
            num=0;
            return;

        }else if(*chr==PARAMETER_END1 || *chr==PARAMETER_END2){
            digitsCounter=0;
            stack[paramsCounter]=num;
            paramsCounter++;
            num=0;
            stack[paramsCounter]=num;
            if (dynamicCallback) dynamicCallback(stack,paramsCounter);
            dynamicCallback=NULL;
            *currentState=next_state;
            memset(stack,0,sizeof(stack));
            paramsCounter=0;

        }else {
            TERMINAL_MESSAGE_Send(MESSAGE_ERROR);
            *currentState=next_state;
            memset(stack,0,sizeof(stack));

        }
}
```

5.2 Video Terminal

Η μονάδα Video Terminal προσομοιώνει τη λειτουργία ενός video τερματικού παρέχοντας μια πληθώρα λειτουργιών. Αξιοποιεί έναν δισδιάστατο πίνακα (video buffer) συνολικού μεγέθους 5.400 byte, ο οποίος αναπαριστά την επιφάνεια απεικόνισης της οθόνης με τρόπο ανάλογο προς τη φυσική της δομή. Συγκεκριμένα, ο πίνακας οργανώνεται σε γραμμές και στήλες, όπως ακριβώς και η οθόνη, με κάθε στοιχείο να αντιστοιχίζεται στη φυσική θέση απεικόνισης στην οθόνη.

Όλες οι λειτουργίες του Video Terminal μεταβάλλουν μόνο τα περιεχόμενα τον εν λόγω πίνακα και έτσι δεν απαιτείται καμία επικοινωνία με τα κατώτερα επίπεδα και με το υλικό. Η οργάνωση του πίνακα διαφέρει σε σχέση με την επιλεγμένη κατάσταση οθόνης. Έτσι μπορεί να προσπελαύνεται είτε ως δισδιάστη δομή διαφορετικών διαστάσεων, είτε ως ενιαίος γραμμικός πίνακας, ανάλογα με τις απαιτήσεις της κάθε κατάστασης.

Στον πίνακα που ακολουθεί συνοψίζονται οι καταστάσεις που υποστηρίζονται:

Κωδ.	Περιγραφή	Ανάλυση	Πλήθος Χρωμάτων	Βάθος Χρώματος	Μέγεθος Εικονοστοιχείου
0	Text Mode	360x480	16	4-bit	8x12
1	Image Mode	360x480	16	4-bit	8x8
2	Plot Mode	360x120	16	1-bit	1x1

Πίνακας 6: Υποστηριζόμενες Καταστάσεις Οθόνης

5.2.1 Κατάσταση Κειμένου (Text Mode)

Στην κατάσταση λειτουργίας κειμένου, ο πίνακας οργανώνεται σε 60 γραμμές και 45 στήλες εκ των οποίων ορατές είναι μόνο οι πρώτες 40 γραμμές, αντικατοπτρίζοντας πιστά τη φυσική διάταξη της οθόνης. Κάθε στοιχείο του πίνακα αντιστοιχεί σε μία θέση απεικόνισης και αποτελείται από δεδομένα 16-bit. Το πρώτο byte καθορίζει τον προς εμφάνιση χαρακτήρα σε μορφή ASCII, ενώ το επόμενο αφορά στις ιδιότητες χρωματισμού του, κατά τον τρόπο που απεικονίζεται στον παρακάτω πίνακα:

Χαρακτήρας								**Χρώμα**							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Κωδικός ASCII								Χρώμα Παρασκηνίου				Χρώμα Προσκηνίου			

Πίνακας 7: Οργάνωση Video Buffer στην Κατάσταση Κειμένου

Με δεδομένο ότι χρησιμοποιούνται 4 bit για την αναπαράσταση του χρώματος προσκηνίου και παρασκηνίου, είναι εφικτό να αποτυπωθούν 16 χρωματικές επιλογές. Η κωδικοποίηση των χρωμάτων και η ενδεικτική τους απόχρωση, δίνεται στον ακόλουθο πίνακα:

Κωδ. Χρώματος	Περιγραφή	Απόχρωση	Κωδ. Χρώματος	Περιγραφή	Απόχρωση
0	Black		8	Dark Grey	
1	Dark Blue		9	Bright Blue	
2	Dark Green		10	Bright Green	
3	Dark Cyan		11	Bright Cyan	
4	Dark Red		12	Bright Red	
5	Dark Magenta		13	Bright Magenta	
6	Dark Yellow		14	Bright Yellow	
7	Grey		15	White	

Πίνακας 8: Κωδικοποίηση Χρωμάτων

Κάθε χαρακτήρας αντιστοιχεί σε ένα γραφικό σύμβολο (glyph) μεγέθους 12 x 8 pixels. Ακολουθεί η ενδεικτική αποτύπωση του χαρακτήρα «A»:

	1	2	3	4	5	6	7	8	byte
1	0	0	0	0	0	0	0	0	0x00
2	0	0	0	1	1	0	0	0	0x18
3	0	0	1	1	1	1	0	0	0x3C
4	0	1	1	0	0	1	1	0	0x66
5	0	1	0	0	0	0	1	0	0x42
6	0	1	0	0	0	0	1	0	0x42
7	0	1	0	0	0	0	1	0	0x42
8	0	1	1	1	1	1	1	0	0x7E
9	0	1	0	0	0	0	1	0	0x42
10	0	1	0	0	0	0	1	0	0x42
11	0	1	0	0	0	0	1	0	0x42
12	0	0	0	0	0	0	0	0	0x00

Πίνακας 9: Αναπαράσταση Συμβόλου

Είναι προφανές, ότι για να καλυφθεί ολόκληρο το φάσμα του πίνακα ASCII, απαιτούνται 12 x 256 = 3.072 bytes. Η διατήρηση του πίνακα συμβόλων δεν γίνεται στη μνήμη RAM αλλά στη μνήμη FLASH. Η επιλογή αυτή δεν είναι μόνο στρατηγική αλλά είναι και τεχνικά επιβεβλημένη, καθώς η μνήμη RAM του ελεγκτή είναι περιορισμένη, οπότε η χρήση της προορίζεται αυστηρά και μόνο για δυναμικά δεδομένα.

Για την κατασκευή του video buffer, δημιουργήθηκε ο νέος τύπος δεδομένων cell_t και ο πίνακας δηλώθηκε ως δισδιάστατη δομή αυτού του τύπου:

C code
```
typedef struct{
    char character;
    uint8_t color_attribute;
} cell_t;

cell_t video_buffer[ROWS][COLUMNS];
```

Επιπλέον, δημιουργήθηκε ένας μονοδιάστατος πίνακας δεικτών (array of pointers), όπου κάθε στοιχείο δείχνει στην αρχή της αντίστοιχης γραμμής του video buffer. Η επιλογή αυτή έγινε για να μπορούν να αναδιατάσσονται γρήγορα οι σειρές του πίνακα με απλή μετακίνηση των δεικτών του, χωρίς να απαιτείται η φυσική μετακίνηση των στοιχείων του.

Η συγκεκριμένη προσέγγιση προσφέρει τη δυνατότητα ανάπτυξης ταχύτατων συναρτήσεων κύλισης (scrolling). Έστω για παράδειγμα ότι απαιτείται να μετακινηθούν όλα τα περιεχόμενα της οθόνης κατά μια σειρά προς τα πάνω, και να εισαχθεί μια νέα σειρά στο τέλος της οθόνης (scroll up). Η συμβατική προσέγγιση επιβάλει τη φυσική μετακίνηση όλων των στοιχείων του πίνακα και τον καθαρισμό της τελευταίας γραμμής. Απαιτείται λοιπόν να μεταφερθούν 40 x 45 x 16 bit = 3.600 bytes, διαδικασία εξαιρετικά χρονοβόρα για έναν 8-bit μικροελεγκτή. Αντ' αυτού, μετακινούνται κατά μια θέση οι 40 δείκτες, έτσι ώστε ο πρώτος να καταλαμβάνει πλέον την τελευταία θέση και όλοι οι υπόλοιποι μεταφέρονται κατά μια θέση προς τα πάνω. Η διαδικασία αυτή απαιτεί την μεταφορά 40 x 16 bit=80 bytes, δηλαδή ολοκληρώνεται 45 φορές ταχύτερα!

C code
```
cell_t (*videoPtr[ROWS])[COLUMNS];

void VideoTerminal_ScrollUp(void) {
```

```
    cell_t (*tmp_ptr)[COLUMNS]= videoPtr[0];

    // Shift all lines up
    for (uint8_t row = 0; row < ROWS - 1; row++) {
        videoPtr[row] = videoPtr[row + 1];
    }

    // Assign the old top line to the bottom
    videoPtr[ROWS - 1] = tmp_ptr;

    // Clear last line
    cell_t *cell_ptr = (cell_t*)tmp_ptr;
    for (uint8_t col = 0; col < COLUMNS; col++) {
        cell_ptr->character = ' ';
        cell_ptr->color_attribute = cursor.color;
        cell_ptr++;
    }

}
```

Σημαντικό ρόλο στην υλοποίηση διαδραματίζει ο δρομέας (cursor), ο οποίος υλοποιείται ως ξεχωριστή δομή δεδομένων. Η δομή περιλαμβάνει τις συντεταγμένες του στην οθόνη, το χρώμα του, τον χαρακτήρα που αντιστοιχεί στην τρέχουσα θέση του, καθώς και την κατάσταση αναβοσβήματος (blinking) αλλά και το σχήμα του δρομέα.

C code

```
typedef struct {
    uint8_t x;
    uint8_t y;
    union {
        uint8_t color;
        struct {
            uint8_t foreColor : 4;
            uint8_t backColor : 4;
        };
    };
    bool blink;
    uint8_t shape;
    uint8_t active_char;
} cursor_t;
```

Παρέχονται επίσης συναρτήσεις για τη μετακίνηση του δρομέα, την αλλαγή χρώματος, την ενεργοποίηση ή απενεργοποίηση του αναβοσβήματος, κ.α..

Η μονάδα περιλαμβάνει επίσης λειτουργίες για την εκκαθάριση και την κύλιση της οθόνης, την σχεδίαση απλών παραθύρων κ.α. Συγκεκριμένα, η εκκαθάριση μπορεί να εφαρμοστεί είτε σε ολόκληρη την οθόνη είτε σε μία μόνο γραμμή, ενώ η κύλιση υποστηρίζεται τόσο σε μορφή καθολικής κύλισης προς τα πάνω, όσο και σε τοπικό

επίπεδο μέσα σε ένα ορθογώνιο πλαίσιο (scroll box) με δυνατότητα μετατόπισης σε τέσσερις κατευθύνσεις.

Η εμφάνιση κειμένου υποστηρίζεται μέσω της συνάρτησης print, η οποία τοποθετεί χαρακτήρες στην αντίστοιχη θέση του video buffer, ενημερώνει τον δρομέα και εφαρμόζει τις καθορισμένες ρυθμίσεις χρωματισμού. Για την παρουσίαση πιο σύνθετων δομών, υλοποιήθηκαν συναρτήσεις σχεδίασης πλαισίων (boxes) και παραθύρων (windows) με χρήση των εκτεταμένων ASCII χαρακτήρων (extended ASCII). Επιπλέον, δημιουργήθηκαν συναρτήσεις πλήρωσης περιοχών με συγκεκριμένους χαρακτήρες, καθώς και μαζικής αλλαγής χρωμάτων σε όλη την οθόνη.

5.2.2 Κατάσταση Σχεδίασης (Plot Mode)

Στην Κατάσταση Σχεδίασης ο Video Buffer προσπελαύνεται και πάλι ως δισδιάστατη δομή μεγέθους 5.400 bytes, αλλά με διαστάσεις 120 x 45. Κάθε στοιχείο του πίνακα είναι τύπου byte και αντιστοιχεί σε 8 συναπτά pixels. Υποστηρίζεται χρωματικό βάθος 1-bit και η συνολική ανάλυση που επιτυγχάνεται είναι 360 x 120. Ωστόσο είναι πλέον εφικτό να τροποποιηθεί κάθε pixel μεμονωμένα και όχι όπως στην Κατάσταση Κειμένου, όπου οι μεταβολές περιορίζονταν μόνο στην αντικατάσταση τυποποιημένων χαρακτήρων 12 x 8. Υποστηρίζονται και πάλι 16 χρώματα, ωστόσο μόνο δυο χρώματα μπορούν να είναι ορατά ταυτόχρονα στην οθόνη· ένα για το χρώμα παρασκηνίου και ένα για το χρώμα προσκηνίου.

Παρέχεται ασφαλώς συνάρτηση ενεργοποίησης μεμονωμένων Pixel αλλά και συναρτήσεις σχεδίασης γραμμών, κύκλων και ελλείψεων, οι οποίες πραγματοποιούν γεωμετρικούς υπολογισμούς που βασίζονται στους αλγόριθμους του Bresenham [15] [16] και είναι ιδανικοί για συστήματα με περιορισμένους πόρους, όπως μικροελεγκτές χωρίς μονάδα FPU (Floating Point Unit).

Η παραδοσιακή προσέγγιση σχεδίασης γεωμετρικών σχημάτων προϋποθέτει τη χρήση τριγωνομετρικών συναρτήσεων, όπως sin και cos, καθώς και μεταβλητών τύπου double,

ώστε να υποστηρίζεται η απαιτούμενη ακρίβεια δεκαδικών ψηφίων. Κάτι τέτοιο θα επιβάρυνε σημαντικά τον μικροελεγκτή, καθώς όλοι οι υπολογισμοί θα έπρεπε να γίνουν σε επίπεδο λογισμικού. Ακολουθώντας την προσέγγιση του Bresenham, αποφεύχθηκε ολοκληρωτικά η χρήση της μαθηματικής βιβλιοθήκης της C και επιταχύνθηκε σημαντικά η υπολογιστική διαδικασία.

Για να επιτευχθεί ο έλεγχος ενός μεμονωμένου Pixel, απαιτείται η τροποποίηση του αντίστοιχου bit στον video buffer. Αυτό επιτυγχάνεται υπολογίζοντας τη σχετική θέση του στον πίνακα 120 x 45 και πραγματοποιώντας αντίστοιχες λογικές ολισθήσεις της μονάδας, ώστε να επιλεγεί η ακριβής του θέση στο byte. Το αποτέλεσμα είναι η κατασκευή μια bit-μάσκας όπου το λογικό OR με το αντίστοιχο byte ενεργοποιεί το pixel στην οθόνη, ενώ το λογικό AND με την ανεστραμμένη μάσκα το απενεργοποιεί.

Για παράδειγμα, για να ενεργοποιηθεί το pixel στη θέση της οθόνης {y=60, x=84}, γίνονται τα παρακάτω βήματα:

1. $y = 60$
2. $x = \left\lfloor \frac{84}{8} \right\rfloor = 10$
3. $\text{byte} = \text{VideoBuffer}[y][x]$
4. $\text{bit\_mask} = 128 \gg (84 \,\%\, 8) = \text{b10000000} \gg 4 = \text{b00001000} = 8$
5. $\text{byte} \mid = \text{bit\_mask}$
6. $\text{VideoBuffer}[y][x] = \text{byte}$

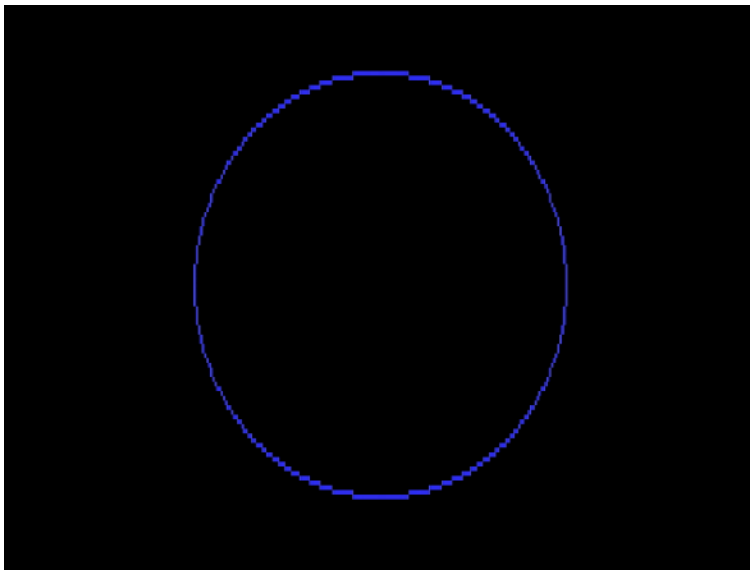
C code

```
void SetPixel(int16_t x, int16_t y, uint8_t color) {
    //Cast to array [120][45]
    uint8_t (*gfx_ptr)[120][45]=(uint8_t(*)[120][45]) &video_buffer;

    int16_t x_offset = x >> 3; // x = x /8

    uint8_t pixel_mask = (uint8_t)(0x80 >> (x & 0x07));

    if (color){
        (*gfx_ptr)[y][x_offset] |= pixel_mask;
    }else {
        (*gfx_ptr)[y][x_offset] &= ~pixel_mask;
    }
}
```

Αντίστοιχο ενδιαφέρον παρουσιάζει και η συνάρτηση σχεδίασης γραμμής. Ο κώδικας που ακολουθεί βασίζεται στον αλγόριθμο του Brasenham και όπως προαναφέρθηκε, δεν χρησιμοποιεί καμία τριγωνομετρική συνάρτηση και όλες οι μεταβλητές που χρησιμοποιούνται είναι τύπου integer, πράγμα που τον καθιστά εξαιρετικά γρήγορο.

C code

```
void DrawLine(int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t
color) {
    if (x1 >= 360 || y1 >= 120 || x2 >= 360 || y2 >= 120) return;

    int16_t dx = x2 - x1 >= 0 ? x2 - x1: x1 - x2; //abs(x1 - x0)

    int16_t dy = y2 - y1 >= 0 ? y2 - y1: y1 - y2; //abs(y1 - y0)

    int8_t sx = (x1 < x2) ? 1 : -1;

    int8_t sy = (y1 < y2) ? 1 : -1;

    int16_t err = dx - dy;


    while (true) {
        SetPixel(x1, y1, color);

        if (x1 == x2 && y1 == y2) break;

        int e2 = 2 * err;
        if (e2 > -dy) { err -= dy; x1 += sx; }
        if (e2 < dx)  { err += dx; y1 += sy; }
    }

}
```



Εικόνα 3: Παράδειγμα Λειτουργίας Plot Mode

5.2.3 Κατάσταση Γραφικών (Graphics Mode)

Η Κατάσταση Γραφικών, επιτρέπει την απεικόνιση εικόνων 480 x 360 pixels και υποστηρίζει επίσης 16 χρώματα. Κάθε εικόνα αυτού του μεγέθους απαιτεί $480 \times 360 \times 4\ bits = 86.400\ bytes$. Η αποθήκευση μιας και μόνο εικόνας στη μνήμη FLASH του μικροελεγκτή, θα καταλάμβανε σχεδόν το 70% της συνολικής του χωρητικότητας. Επιπλέον, όπως εξηγήθηκε στις ενότητες 3.3.3 και 3.3.4, το χρονικό παράθυρο για την αποστολή δεδομένων στη θύρα VGA, διαρκεί μόλις 8 κύκλους εντολής, όσο το διάστημα για να ολοκληρωθεί η αποστολή ενός byte δεδομένων από τον καταχωρητή PISO. Για την αποστολή εικόνας 4-bit, θα χρειάζονταν 4 φορές υψηλότερη ταχύτητα. Επομένως ήταν απαραίτητο να ακολουθηθεί διαφορετική προσέγγιση.

Η λύση που επιλέχθηκε διατηρεί αμετάβλητη την ανάλυση απεικόνισης, ενώ εφαρμόζει συμπίεση στα χρωματικά δεδομένα, έτσι ώστε μειώνονται τόσο οι χωρικές απαιτήσεις όσο και η χρονική πολυπλοκότητα για την προετοιμασία και την αποστολή των δεδομένων στην θύρα VGA.

Χρησιμοποιούνται δυο πίνακες διαστάσεων 480 x 45 bytes και 45 x 60 bytes αντίστοιχα. Ο πρώτος, διατηρεί τα δεδομένα εικόνας βάθους 1-bit και δεσμεύει συνολικά $480\ x\ 360\ x\ 1\ bit = 21.600\ bytes$ μνήμη. Ο δεύτερος, διατηρεί τη συμπιεσμένη χρωματική πληροφορία, η οποία υπόκειται σε λογική διαμέριση 60 γραμμών και 45 στηλών. Κάθε στοιχείο έχει μέγεθος 8 bit, όπου τα πρώτα τέσσερα αφορούν στο χρώμα παρασκηνίου και τα επόμενα στο χρώμα προσκηνίου. Κάθε λογική μονάδα 8 x 8 του πίνακα δεδομένων, συνδέεται με ένα στοιχείο του πίνακα χρωμάτων. Έτσι απαιτούνται $45 \times 60 = 2.700\ bytes$ για την αποθήκευση των χρωμάτων και συνολικά $24{,}3\ Kbytes$ για να αποθηκευτεί μια πλήρης εικόνα. Με την προσέγγιση αυτή, οι συνολικές απαιτήσεις χωρητικότητας μειώνονται σε λιγότερο από το 30% των αρχικών απαιτήσεων και είναι πλέον δυνατό να αποθηκευτούν 4 φωτογραφίες στη μνήμη flash του μικροελεγκτή.

Αυτή η τεχνική συμπίεσης χρησιμοποιήθηκε εκτεταμένα από τους υπολογιστές της δεκαετίας του 80, μεταξύ των οποίων και από τον ZX Spectrum. Συνεπώς, υπήρχαν διαθέσιμες δωρεάν εφαρμογές, που πρόσφεραν τη δυνατότητα μετατροπής αρχείων

εικόνας εφαρμόζοντας τη συγκεκριμένη τεχνική συμπίεσης, οπότε και δεν απαιτήθηκε να κατασκευαστεί συμπληρωτικό λογισμικό επεξεργασίας εικόνας. Οι δοκιμές που πραγματοποιήθηκαν εκ των υστέρων, απέδειξαν πως ήταν χρονικά εφικτό να διπλασιαστεί η ανάλυση της χρωματικής πληροφορίας σε κελιά 4x4 αντί για 8x8· ωστόσο δεν κατέστη εφικτό να εντοπιστεί δωρεάν λογισμικό μετατροπής και επειδή η επένδυση σε χρόνο για την ανάπτυξη σχετικού λογισμικού ήταν σημαντική, διατηρήθηκε η αρχική προσέγγιση.

Για να εξηγηθεί η παραπάνω διαδικασία συμπίεσης δίνεται ενδεικτικά το αποτέλεσμα της μετατροπής μιας εικόνας 16 χρωμάτων με βάθος 4-bit ανά pixel, σε εικόνα 16 χρωμάτων με βάθος 4-bit ανά κελί 8x8.

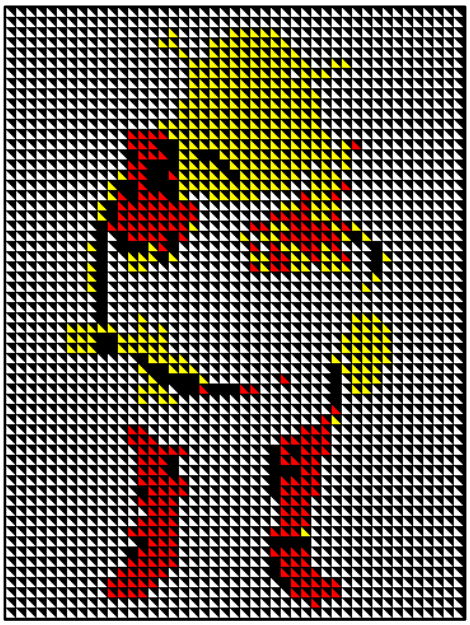
Εικόνα 4: Αρχική εικόνα βάθους 4-bit/Pixel

Εικόνα 5: Συμπιεσμένη Εικόνα βάθους 4-bit/8x8

Για την παραπάνω μετατροπή χρησιμοποιήθηκε το λογισμικό «Image Spectrumizer», το οποίο αποτελεί λογισμικό ανοιχτού κώδικα και διατίθεται ελεύθερα.

Από τη συμπιεσμένη εικόνα εξάγονται τα δεδομένα που περιγράφηκαν νωρίτερα. Σε μια προσπάθεια οπτικοποίησης της επεξεργασμένης πληροφορίας, παρατίθενται οι παρακάτω δυο εικόνες όπου αποτυπώνουν τα περιεχόμενα του πίνακα δεδομένων και της αντίστοιχης χρωματικής πληροφορίας.

Εικόνα 6: Δεδομένα Εικόνας βάθους 1-bit



Εικόνα 7: Χρωματική Πληροφορία

Τα χρωματικά δεδομένα αντιμετωπίζονται ως δυναμική πληροφορία και αποθηκεύονται στον Video Buffer. Έτσι, είναι εφικτό να αλλάζει ο χρωματισμός μιας στατικής εικόνας από τη μονάδα Video Terminal σε πραγματικό χρόνο, όπως ακριβώς γίνεται και στις καταστάσεις Text και Plot. Ο Video Buffer προσπελαύνεται ως δισδιάστατη δομή μεγέθους διαστάσεων 60 x 45 όπου κάθε στοιχείο του πίνακα είναι τύπου byte, αντιστοιχώντας στο χρώμα ενός εικονοστοιχείου 8x8 pixels. Ακολουθεί στιγμιότυπο του κώδικα δυναμικού χρωματισμού μιας εικόνας:

C code

```
//Cast to array [60][45]
uint8_t (*ptr)[60][45] =(uint8_t (*)[60][45]) &videoBuffer;


for (uint8_t x=0; x <45; x++){
    for (uint8_t y=0; y <60; y++){
        (*ptr)[y][x]=(uint8_t) (back_color<<4 |fore_color);
    }
}
```

Δίνονται ενδεικτικά οι παρακάτω χρωματικές αλλαγές, οι οποίες πραγματοποιήθηκαν από τον VGA Controller σε πραγματικό χρόνο. Η εικόνες αποτελούν πραγματικές φωτογραφίες της οθόνης.

Εικόνα 8: Δυναμική αλλαγή χρώματος

5.3 Reactor

Η μονάδα Reactor λειτουργεί ως ένα κεντρικό σύστημα διαχείρισης συμβάντων. Εφαρμόζει μια παραλλαγή του σχεδιαστικού προτύπου «Reactor Pattern» [17], ειδικά διαμορφωμένη για 8-bit μικροελεγκτές και bare-metal εφαρμογές. Το πρότυπο επιτρέπει την παρακολούθηση και την εξυπηρέτηση πολλαπλών συμβάντων (events) μέσα από ένα ενιαίο νήμα εκτέλεσης. Με αυτό τον τρόπο, αποφεύγονται τα προβλήματα συγχρονισμού και η αυξημένη κατανάλωση πόρων που εμφανίζονται συνήθως σε πολυνηματικές υλοποιήσεις.

Το χαρακτηριστικό αυτό καθιστά την προσέγγιση του Reactor ιδιαίτερα κατάλληλη για ενσωματωμένα συστήματα περιορισμένων πόρων. Παράλληλα, η αρχιτεκτονική του Reactor προσφέρει σαφή δομή και επεκτασιμότητα, καθώς η προσθήκη μηχανισμού παρακολούθησης νέων συμβάντων υλοποιείται πολύ εύκολα χωρίς να απαιτούνται δοκιμές αλλαγές στον κώδικα.

5.3.1 Αρχιτεκτονική

Η λειτουργία του Reactor βασίζεται σε τρία κύρια συστατικά:

1. **Event**: Πρόκειται για μια απαρίθμηση (enumeration) που ορίζει τα διαφορετικά είδη συμβάντων τα οποία μπορεί να διαχειριστεί το σύστημα. Κάθε συμβάν αντιστοιχεί σε ένα μοναδικό bit. Επομένως, η αναπαράσταση τους δεν γίνεται με διακριτές αριθμητικές τιμές, αλλά με δυαδικές σημαίες (flags) που μπορούν να συνδυαστούν μεταξύ τους χρησιμοποιώντας δυαδική μάσκα (bit masking). Με τον τρόπο αυτό επιτυγχάνεται ένα είδος πολυπλεξίας, καθώς πολλά συμβάντα μπορούν να απεικονιστούν ταυτόχρονα στην ίδια μεταβλητή, με την ενεργοποίηση των αντίστοιχων bits.

2. **EventHandler:** Αποτελεί μια διεπαφή (interface) που περιέχει:

- Πεδίο event: είναι της μορφής Event και δηλώνει σε ποιο συμβάν αντιστοιχεί ο EventHandler.
- Την αφηρημένη μέθοδο handle, η οποία υλοποιείται από τον αντίστοιχο διαχειριστή συμβάντος και καθορίζει την ενέργεια που πρέπει να εκτελεστεί όταν ενεργοποιηθεί το αντίστοιχο συμβάν.

3. **Reactor:** Αποτελεί το κεντρικό στοιχείου του συστήματος. Διατηρεί έναν πίνακα με όλους τους εγγεγραμμένους διαχειριστές συμβάντος και ελέγχει διαρκώς αν έχει ενεργοποιηθεί κάποιο συμβάν. Στην περίπτωση ενεργού συμβάντος, πραγματοποιεί την αποπλεξία (demultiplexing) του χρησιμοποιώντας και πάλι δυαδική μάσκα, εντοπίζει τους εγγεγραμμένους διαχειριστές που αντιστοιχούν σε αυτό και τους ενεργοποιεί σειριακά (dispatching).

5.3.2 Υλοποίηση

Η αρχιτεκτονική του Reactor είναι σχεδιασμένη με τρόπο που ευνοεί την επεκτασιμότητα. Η προσθήκη νέων συμβάντων υλοποιείται απλά με την επέκταση της απαρίθμησης event_t. Απόσπασμα της υλοποίησης δίνεται παρακάτω:

C code
```c
typedef enum {
    EVENT_BUTTON                = 1 << 0,
    EVENT_DMA1_UPDATE           = 1 << 1,
    EVENT_DMA2_UPDATE           = 1 << 2,
    EVENT_INPUT_BUFFER_UPDATED  = 1 << 3,
    EVENT_UART_OVRN_ERROR       = 1 << 4
} event_t;
```

Για να οριστεί ένα νέο συμβάν π.χ. EVENT_NEW, αρκεί απλά να προστεθεί:
EVENT_NEW = 1 << 6, στην παραπάνω δομή.

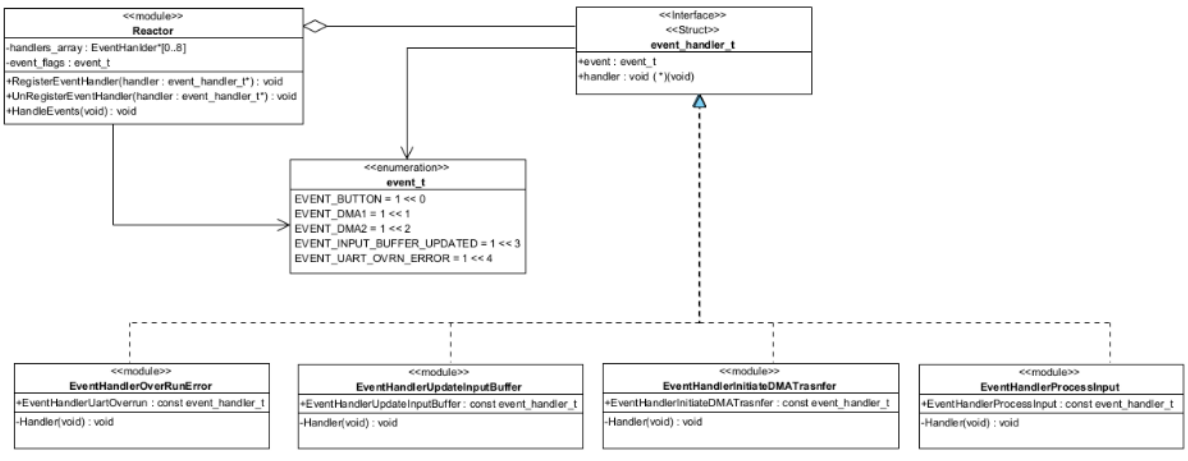
Αντίστοιχα, η προσθήκη νέων διαχειριστών συμβάντων (EventHandlers) είναι εξίσου απλή. Η διεπαφή event_handler_t ορίζει έναν σαφή και ενιαίο τρόπο υλοποίησης:

C code
```c
typedef void (*handler_t)(void);

typedef struct {
    handler_t handler;
    event_t event;
}event_handler_t;
```

Για να προστεθεί ένας νέος διαχειριστής, δημιουργείται μια νέα δομή που υλοποιεί την παραπάνω διεπαφή και παρέχει την κατάλληλη υλοποίηση της μεθόδου handle. Στη συνέχεια, ο νέος handler εγγράφεται στον Reactor μέσω της μεθόδου RegisterEventHandler.

Γίνεται ξεκάθαρο ότι καμία από αυτές τις επεκτάσεις δεν απαιτεί τροποποίηση της βασικής λογικής του Reactor. Ο Reactor παραμένει ο ίδιος και λειτουργεί ως «σκελετός» του συστήματος· αυτό που αλλάζει και επεκτείνεται είναι η λίστα των συμβάντων και οι εξειδικευμένοι EventHandlers. Με αυτό τον τρόπο, η αρχιτεκτονική παραμένει σταθερή, ενώ η λειτουργικότητα μπορεί να αυξάνεται ανάλογα με τις ανάγκες.

Για τον λόγο αυτό, η συγκεκριμένη μονάδα επιλέχθηκε να ενταχθεί στο επίπεδο middleware, ενώ οι υλοποιήσεις των διαχειριστών τοποθετήθηκαν στο επίπεδο εφαρμογής. Έτσι επιτυγχάνεται πλήρης διαχωρισμός ρόλων: το επίπεδο middleware παρέχει τον μηχανισμό διαχείρισης συμβάντων ο οποίος είναι γενικός και επαναχρησιμοποιήσιμος, ενώ το επίπεδο εφαρμογής υλοποιεί τη λογική που είναι εξειδικευμένη για τη συγκεκριμένη υλοποίηση.
Ακολουθεί το διάγραμμα UML που αποτυπώνει την υλοποίση του συγκεκριμένου προτύπου:



Διάγραμμα 15: Reactor Class Diagram

Στιγμιότυπο κώδικα:

C code – «Reactor.c»

```
#include "reactor.h"
#include <stddef.h>
#include <string.h>

__near volatile uint8_t g_event_flags = 0;

static const event_handler_t* handlers_array[HANDLERS_COUNT];

void REACTOR_Init(void){
    //nullify all array entries
    memset(&handlers_array, 0, sizeof(handlers_array));
}


void REACTOR_Register_event_handler(const event_handler_t *handler) {
    for (uint8_t i=0;i<HANDLERS_COUNT;i++){
        if (handlers_array[i]==NULL) {
            handlers_array[i]=handler;
            return;
        }
    }
}

void REACTOR_Unregister_event_handler(const event_handler_t *handler)
{
    for (uint8_t i=0;i<HANDLERS_COUNT;i++){
        if (handlers_array[i]==handler) {
            handlers_array[i]=NULL;
            return;
        }
    }
}

__inline void REACTOR_Raise_Event(event_t event_id){
    g_event_flags |= event_id;

}

void REACTOR_Loop(void) {
    while (1) {
        if (!g_event_flags) continue;

        event_t event_flags_snapshot = g_event_flags;
        g_event_flags &= ~event_flags_snapshot; //clear the flags
        for (uint8_t i = 0; i < HANDLERS_COUNT; ++i) {
            const event_handler_t *handler_ptr = handlers_array[i];
            if (handler_ptr!=NULL && handler_ptr->event &
event_flags_snapshot){

                if (handler_ptr->handle) handler_ptr->handle();

            }
        }
    }
}
```

5.4 Circular Buffer

Ο VGA Controller οφείλει να μπορεί δέχεται συνεχώς εντολές μέσω του διαύλου UART. Ωστόσο, επειδή το μεγαλύτερο μέρος του χρόνου του αφιερώνεται στην παραγωγή των αναλογικών σημάτων VGA, δεν είναι εφικτό να επεξεργάζεται άμεσα κάθε εισερχόμενο δεδομένο. Για τον λόγο αυτό, τα δεδομένα που καταφθάνουν μέσω UART πρέπει να μπορούν να αποθηκεύονται προσωρινά, μέχρι ο μικροελεγκτής να είναι διαθέσιμος για την επεξεργασία τους. Η λύση σε αυτό το πρόβλημα είναι η χρήση ενός buffer δεδομένων.

Ένας buffer δεδομένων αποτελεί μια περιοχή μνήμης που χρησιμοποιείται για την προσωρινή αποθήκευση πληροφοριών, συνήθως κατά τη μεταφορά τους από ένα σημείο σε άλλο, όπως από μια συσκευή εισόδου στη μνήμη RAM.

Η συμβατική μέθοδος υλοποίησης ενός buffer βασίζεται σε έναν γραμμικό πίνακα σταθερού μεγέθους και έναν δείκτη εγγραφής, ο οποίος δείχνει στη θέση που θα αποθηκευτεί το επόμενο στοιχείο. Κάθε εγγραφή στον buffer, αυξάνει τον δείκτη εγγραφής κατά μια θέση. Αντίστοιχα, κάθε ανάγνωση από τον Buffer, αφαιρεί τα αναγνωσμένα δεδομένα, μετακινεί τα εναπομήναντα στοιχεία κατά τον αντίστοιχο αριθμό θέσεων προς την αρχή του πίνακα και ενημερώνει τον δείκτη εγγραφής.

Η προσέγγιση αυτή αν και εξαιρετικά απλή, πάσχει από τα παρακάτω πρόβλήματα:

- Ο χρόνος ανάγνωσης δεν είναι σταθερός καθώς κάθε ανάγνωση απαιτεί και την μετακίνηση διαφορετικού πλήθος δεδομένων εντός του πίνακα.
- Η ανάγκη μετακίνησης των δεδομένων μετά από κάθε ανάγνωση συνεπάγεται σημαντικό φόρτο επεξεργασίας, διαδικασία η οποία δεν προσφέρει καμία ουσιαστική λειτουργική αξία στο σύστημα. Σε έναν γραμμικό buffer μεγέθους 128 byte, αν καταναλωθούν 64 byte και παραμείνουν άλλα 64 μη επεξεργασμένα, αυτά πρέπει να μετακινηθούν στην αρχή ώστε να απελευθερωθεί χώρος. Στην ακραία περίπτωση όπου ο buffer είναι πλήρης, το πλήθος των δεδομένων που πρέπει να μετακινηθούν ισοδυναμεί με το συνολικό του μέγεθος.

- Όταν οι ταχύτητες μεταφοράς δεδομένων είναι υψηλές, τότε αυξάνει ο κίνδυνος το σύστημα να μην μπορεί να αναταποκριθεί έγκαιρα, ο δείκτης εγγραφής να υπερχειλίσει και να χαθούν δεδομένα.

Λύση στα παραπάνω προβλήματα δίνει η εφαρμογή της αρχιτεκτονικής ενός κυκλικού buffer (Circular Buffer).

5.4.1 Αρχιτεκτονική

Η λειτουργία του κυκλικού buffer βασίζεται στη χρήση δύο δεικτών:

- Δείκτης Εγγραφής (Write Index): καθορίζει τη θέση εγγραφής του επόμενου δεδομένου.
- Δείκτης Ανάγνωσης (Read Index): υποδεικνύει τη θέση από την οποία θα αναγνωσθεί το επόμενο δεδομένο.

Η καινοτομία του βασίζεται στον ενσωματωμένο μηχανισμό κυκλικής επαναφοράς (wrap-around) των δεικτών εγγραφής και ανάγνωσης. Όταν οι δείκτες υπερχειλίσουν, τότε μεταφέρονται αυτόματα στην αρχή του πίνακα. Το σύστημα δεν δαπανά χρόνο στην μετακίνηση δεδομένων απλά εξασφαλίζει ότι ο ρυθμός ανάγνωσης θα είναι τέτοιος, ώστε ο δείκτης ανάγνωσης δεν θα προπορευτεί του δείκτη εγγραφής. Καθώς ο χρόνος ανάγνωσης και εγγραφής γίνεται πλέον σταθερός και σημαντικά μικρότερος από την παραδοσιακή προσέγγιση, είναι πλέον εφικτό να γίνονται συχνότερες αναγνώσεις, οπότε και μειώνεται σημαντικά η πιθανότητα απώλειας δεδομένων.

Η ανάγκη υψηλού συγχρονισμού μεταξύ των πηγών δεδομένων και των αντίστοιχων καταναλωτών, πλέον ελαχιστοποείται. Επομένως θα μπορούσαμε να πούμε πως ο συγκεκριμένος μηχανισμός αυξάνει την αποσύζευξη μεταξύ λήψης και κατανάλωσης πληροφορίας και μάλιστα το πετυχαίνει με πολύ μικρότερο υπολογιστικό φόρτο.

5.4.2 Input – Output Circular Buffers

Ο ελεγκτής, δεν δέχεται απλά εντολές αλλά και απαντά σε αυτές, επιβεβαιώνοντας τη λήψη τους και τη συντακτική τους ορθότητα μέσω τυποιημένων μηνύματων ACK, ERR κτλ. Για την υλοποίηση αυτής της αμφίδρομης επικοινωνίας απαιτούνται δύο ξεχωριστοί κυκλικοί buffers: ένας για τα εισερχόμενα και ένας για τα εξερχόμενα δεδομένα.

Αν και η χρήση κυκλικών buffer διευκολύνει σημαντικά τη διαχείριση των δεδομένων, δεν επιλύει πλήρως το πρόβλημα του συγχρονισμού με τον UART δίαυλο, ο οποίος διαθέτει ενσωματωμένο hardware buffer περιορισμένου μεγέθους (μόλις 2 Byte).

Για τα εισερχόμενα δεδομένα, το σύστημα πρέπει να καταναλώνει άμεσα το περιεχόμενο του hardware buffer, πριν εισαχθούν νέα δεδομένα. Αυτό απαιτεί την έγκαιρη μεταφορά τους στον κυκλικό buffer, πριν γίνει νέα εισαγωγή από τον UART και ο hardware buffer υπερχειλίσει.

Για τα εξερχόμενα δεδομένα, η αποστολή προς τον UART πρέπει να γίνεται με τέτοιο ρυθμό, ώστε η μονάδα να έχει προλάβει να ολοκληρώσει τη σειριακή μετάδοση των υφιστάμενων δεδομένων πριν την επόμενη εισαγωγή. Στην περίπτωση που ο ρυθμός μεταφοράς είναι υψηλότερος, ουσιαστικά θα προκληθεί data overwrite και η εξερχόμενη πληροφορία θα παραφθαρεί.

Γίνεται σαφές ότι η μεταφορά δεδομένων από τους κυκλικούς buffer στη μονάδα UART και το αντίστροφο, πρέπει να είναι σε απόλυτο συντονισμό. Ουσιαστικά, πρέπει να ταυτίζεται με τον ρυθμό σειριακής μετάδοσης του διαύλου UART, που στην συγκεκριμένη υλοποίηση, έχει οριστεί στα 57.600 baud.

Αυτό είναι δύσκολο να επιτευχθεί αποκλειστικά μέσω λογισμικού, επομένως θα αξιοποιηθούν οι μονάδες Direct Memory Access (DMA) που διαθέτει ο μικροελεγκτής, σε συνδυασμό με τους κυκλικούς buffer που αναπτύχθηκαν.

5.4.3 Direct Memory Access (DMA)

Η μονάδα DMA επιτρέπει την αυτοματοποιημένη μεταφορά δεδομένων μεταξύ περιοχών μνήμης και στους PIC μικροελεγκτές, διαθέτει δύο καταστάσεις λειτουργίας:

- Stalling Mode: Η μονάδα DMA αποκτά πλήρη έλεγχο του διαύλου δεδομένων, διακόπτοντας προσωρινά την CPU.
- Cycle Stealing Mode: Η μονάδα DMA εκτελεί μεταφορές μόνο όταν η CPU δεν χρησιμοποιεί τον δίαυλο δεδομένων, εκμεταλλευόμενη τις λεγόμενες «φυσαλίδες» (bubbles) – αχρησιμοποίητους κύκλους ρολογιού.

Η επιλογή λειτουργίας γίνεται έμμεσα μέσω του System Arbiter, ορίζοντας προτεραιότητες μεταξύ CPU και DMA. Όταν η CPU έχει υψηλότερη προτεραιότητα, τότε ενεργοποιείται η κατάσταση Cycle Stealing Mode, την οποία και θα εκμεταλευτούμε στη συγκεκριμένη υλοποίηση.

Η αρχή λειτουργίας της είναι σχετικά απλή. Κατά τη διάρκεια κάθε κύκλου εντολής, ανοίγει δυνητικά ένα παράθυρο κατά το οποίο η μονάδα DMA μπορεί να εκτελέσει μεταφορές δεδομένων. Το παράθυρο δημιουργείται από εντολές που δεν που προσπελαύνουν τη μνήμη δεδομένων (SRAM). Αυτές συνήθως είναι ALU πράξεις με τον W καταχωρητή, εντολές NOP ή εντολές που χρειάζονται δυο ή και περισσότερους κύκλους για να ολοκληρωθούν (π.χ. CALL, GOTO, κτλ), οι οποίες ενσωματώνουν την εκτέλεση εντολών NOP στη λειτουργία τους. Σε όλες αυτές τις περιπτώσεις δημιουργείται μια «φυσαλίδα» την οποία εκμεταλεύεται η μονάδα για να πραγματοποιήσει μεταφορά δεδομένων.

Η εκκίνηση της διαδικασίας μεταφοράς γίνεται με αυτοματοποιημένο τρόπο:

- Για τα εισερχόμενα δεδομένα, ενεργοποιείται από το υλικό, μέσω του RX interrupt flag που θέτει ο UART όταν λαμβάνει νέο byte.
- Για τα εξερχόμενα δεδομένα, ενεργοποιείται από τον Reactor, μέσω της συνάρτησης RaiseEvent, που καλείται όταν υπάρχουν δεδομένα προς αποστολή.

5.4.4 Υλοποίηση

Η μονάδα Circular Buffer έχει σχεδιαστεί με γνώμονα την επαναχρησιμοποίηση, επιτρέποντας την αποθήκευση δεδομένων οποιουδήποτε τύπου, κάνοντας χρήση δεικτών τύπου void. Για τον λόγο αυτό επιλέχθηκε η τοποθέτηση της στο επίπεδο Middleware καθώς δεν απαιτείται καμία αλλαγή στον κώδικα στην περίπτωση που οι απαιτήσεις της εφαρμογής αλλάξουν και ενδεχομένως διαφοροποιηθεί ο τύπος δεδομένων που αποθηκεύεται.

Η δομή της ορίζεται από την παρακάτω διεπαφή:

C code
```
typedef struct{
    void *array_ptr;
    size_t array_size;
    uint8_t item_size;
    size_t write_index;
    size_t read_index;
} circular_buffer_t;
```

Οι βασικές λειτουργίες που είναι υπεύθυνες για την ανάγνωση και την εγγραφή δεδομένων στους buffers, δίνονται παρακάτω:

C code
```
circular_buffer_t CircularBuffer_Write(circular_buffer_t *cb_ptr,
const void *item_ptr) {
    buffer_status_t status;
    if (cb_ptr == NULL || item_ptr == NULL) {
        return BUFFER_INVALID;
    }

    // Check if buffer is full *before* attempting write
    if (CircularBuffer_isFull(cb_ptr)) {
        return BUFFER_FULL;
    }

    memcpy((char*) cb_ptr->array_ptr+ cb_ptr->write_index * cb_ptr-
>item_size, item_ptr, cb_ptr->item_size);

    cb_ptr->write_index = (cb_ptr->write_index + 1) % cb_ptr-
>array_size;

    return BUFFER_OK;
}


circular_buffer_t CircularBuffer_Read(circular_buffer_t *cb_ptr, void
*item_ptr) {
```

```
    if (cb_ptr == NULL || item_ptr == NULL) {
        return BUFFER_INVALID;
    }

    // Check if buffer is empty *before* attempting read
    if (CircularBuffer_isEmpty(cb_ptr)){
        return BUFFER_EMPTY;
    }

    memcpy(item_ptr, cb_ptr->array_ptr+cb_ptr->read_index * cb_ptr-
>item_size, cb_ptr->item_size);
    *(uint8_t*)item_ptr= *((uint8_t*)cb_ptr->array_ptr+cb_ptr-
>read_index);

    cb_ptr->read_index = (cb_ptr->read_index + 1) % cb_ptr-
>array_size;

    return BUFFER_OK;
}
```

Η πλήρης υλοποίηση θα δοθεί στο αντίστοιχο παράρτημα.

5.5 Pixel Generator

Η μονάδα Pixel Generator ανήκει στο Επίπεδο Διακοπών και υλοποιήθηκε σχεδόν εξ ολοκλήρου σε Assembly. Είναι υπεύθυνη για την παραγωγή σήματος κατάλληλου για να οδηγηθούν οι DAC.

Αποτελείται από τις μονάδες TEXT Generator, IMAGE Generator και PLOT Generator, κάθε μια από τις οποίες παράγει κατάλληλα διαμορφωμένο σήμα, ανάλογα με την τρέχουσα κατάσταση του Video Terminal.

Η μονάδα ενεργοποιείται από τον Interrupt Manager, ο οποίος εκκινεί σε κάθε παλμό της γεννήτριας VSYNC (4.2.1) και εφόσον η τρέχουσα θέση στην οθόνη αφορά στην ενεργή περιοχή.

5.5.1 Text Generator

Στην Κατάσταση Κειμένου η μονάδα λειτουργεί ώς εξής:
Σαρώνει τον Video Buffer από αριστερά προς τα δεξιά και διαβάζει κάθε στοιχείο του. Αναζητά το σύμβολο που αντιστοιχεί σε κάθε στοιχείο, στον πίνακα συμβόλων (Glyphs). Φορτώνει το σύμβολο στον καταχωρητή PISO (SPI) και εκκινεί την σειριακή αποστολή του προς τον επιλογέα των πολυπλεκτών χρώματος.
Διαβάζει την επόμενη θέση μνήμης που περιέχει την χρωματική πληροφορία και στέλνει το περιεχόμενο της στη θύρα LATD, οι έξοδοι της οποίας είναι συνδεδεμένες με τις εισόδους των πολυπλεκτών.
Επαναλαμβάνει τη διαδικασία για κάθε ορατό χαρακτήρα στην οθόνη.

Η διαδικασία ανάγνωσης, επεξεργασίας και αποστολής των δεδομένων ενός χαρακτήρα, διαρκεί ακριβώς 8 κύκλους, όσο ακριβώς ήταν και το διαθέσιμο παράθυρο (3.3.3). Η επιλογή της συγγραφής της σε Assembly ήταν απόλυτα επιβεβλημένη, καθώς ο αντίστοιχος κώδικας σε C, δαπανούσε σημαντικά περισσότερους κύκλους, οπότε ήταν πρακτικά ανέφικτο να επιτευχθεί η επιδιωκόμενη ανάλυση.

Η παραπάνω διαδικασία αποκρύπτει ένα βασικό πρόβλημα που ανέκυψε κατά την παραγωγή του σήματος. Καθώς οι εντολές εκτελούνται σε σειρά, το σήμα επιλογής έφτανε στους πολυπλέκτες δυο κύκλους ταχύτερα από την πληροφορία του χρώματος, οδηγώντας σε λανθασμένη απεικόνιση (color bleeding).

Εικόνα 9: Φαινόμενο Color Bleeding

Αυτός είναι ένας από τους προφανείς λόγους που τέτοιες εφαρμογές αναπτύσσονται σε FPGA, καθώς επιτρέπεται η ταυτόχρονη παραγωγή σημάτων. Ωστόσο η αντιπετώπιση του αποδείχτηκε απλή.

Πριν την εκκίνηση της διαδικασίας, ο καταχωρητής PISO (SPI) προ-φορτώνεται με δεδομένα κατά τρόπο τέτοιο ώστε να εισάγεται σκόπιμα καθυστέρηση δυο κύκλων στην αμέσως επόμενη φόρτωση. Με αυτό τον τρόπο, τόσο το σήμα επιλογής όσο και η χρωματική πληροφορία, φτάνουν ταυτόχρονα στους πολυπλέκτες και το παραπάνω φαινόμενο εξαλείφεται.

Εικόνα 10: Χωρίς το φαινόμενο Color Bleeding

```
Asm code
MOVF POSTINC1,W,C     ;READ ASCII CHAR - 1 CY
MOVWF TBLPTRL,C       ; 1 CY
TBLRD*                ;READ ASCII GLYPH -  2 CYs
MOVF TABLAT,W,C       ; 1 CY
MOVWF INDF0,C         ;WRITE GLYPH DATA TO SPI -  1 CY
MOVF POSTINC1,W,C     ;READ COLOR -  1 CY
MOVWF LATD,C          ;WRITE COLOR DATA TO MULTIIPLEXERS - 1 CY -  TOTAL: 8 CYs
```

5.5.2 Image Generator

Η μονάδα παραγωγής γραφικών παρουσιάζει παρόμοια συλλογιστική. Ο Video Buffer διατηρεί μόνο τη χρωματική πληροφορία ενώ τα υπόλοιπα δεδομένα φορώνονται απευθείας από τη μνήμη flash του μικροελεγκτή. Η διαδικασία αποδείχτηκε ευκολότερη σε σχέση με αυτή της κατάστασης κειμένου.

Ο μηχανισμός λειτουργίας της έχει ώς εξής:
Σαρώνεται από τη μνήμη flash η επιλεγμένη εικόνα byte προς byte. Κάτε byte φορτώνεται στον καταχωρητή PISO. Διαβάζεται από τον video buffer το χρώμα που αντιστοιχεί στην συγκεκριμένη 8x8 περιοχή, και αποστέλεται στη θύρα LATD.

Η διαδικασία δίνει χρονικό περιθώριο δυο επιπλέον κύκλων, γεγονός που φανερώνει τη δυνατότητα να αυξηθεί η χρωματική ανάλυση σε μελλοντική έκδοση. Στην τρέχουσα υλοποίηση, για λόγους απλότητας, εισήχθησαν στον κώδικα δυο εντολές NOP, ώστε να διατηρηθεί ο ίδιος αριθμός κύκλων ανά παραγόμενο στοιχείο.

```
Asm code
TBLRD*+
MOVF TABLAT, W,C
MOVWF INDF0,C
NOP
NOP
MOVF POSTINC1,W,C
MOVWF LATD,C
```

5.5.3 Plot Generator

Η μονάδα PLOT Generator είναι υπεύθυνη για την παραγωγή σήματος κατάλληλου για να υποστηρίξει την κατάσταση Σχεδίασης. Όλα τα διαθέσιμα δεδομένα βρίσκονται στον Video Buffer καθώς πλέον το παραγόμενο σήμα είναι βάθους 1-bit.

Ο μηχανισμός λειτουργίας της έχει ώς εξής:
Φορτώνει τους πολυπλέκτες με το επιλεγμένο χρώμα προσκηνίου και παρασκηνίου, το οποίο είναι ενιαίο σε όλη την οθόνη.
Σαρώνει τον Video Buffer από αριστερά προς τα δεξιά, byte προς byte. Φορτώνει το περιεχόμενο κάθε θέσης στον καταχωρήτη PISO και ενεργοποιεί τους επιλογείς των πολυπλεκτών.

Όπως και στην προηγούμενη μονάδα, τόσο και εδώ, ο κώδικας έδωσε παράθυρο τεσσάρων επιπλέον κύκλων που για λόγους απλότητας, εισήχθησαν εντολές NOP. Ο περιορισμός στην ανάλυση της Κατάστασης Σχεδίασης, δεν οφείλεται στην ταχύτητα χρονισμού του μικροελεγκτή αλλά στην περιοσριμένη χωρητικότητα της μνήμης. Επομένως, παρότι υπάρχει η δυνατότητα να προβληθούν επιπλέον δεδομένα στο χρονικό παράθυρο των οκτώ κύκλων, δεν υπάρχει επαρκής μνήμη για να αποθηκευθούν.

Ακολουθεί στιγμιότυπο του κώδικα σε Assembly.

Asm code

```
MOVF POSTINC1,W,C
NOP
NOP
NOP
NOP
MOVWF INDF0,C
MOVF INDF2,W,C
MOVWF LATD,C
```

Η εισαγωγή εντολών NOP μπορεί να φαίνεται ως σπατάλη υπολογιστικού χρόνου, ωστόσο αποτελεί ταυτόχρονα μια «φυσαλίδα» την οποία εκμεταλεύεται ο DMA controller για την μεταφορά δεδομένων από και προς τον δίαυλο UART.

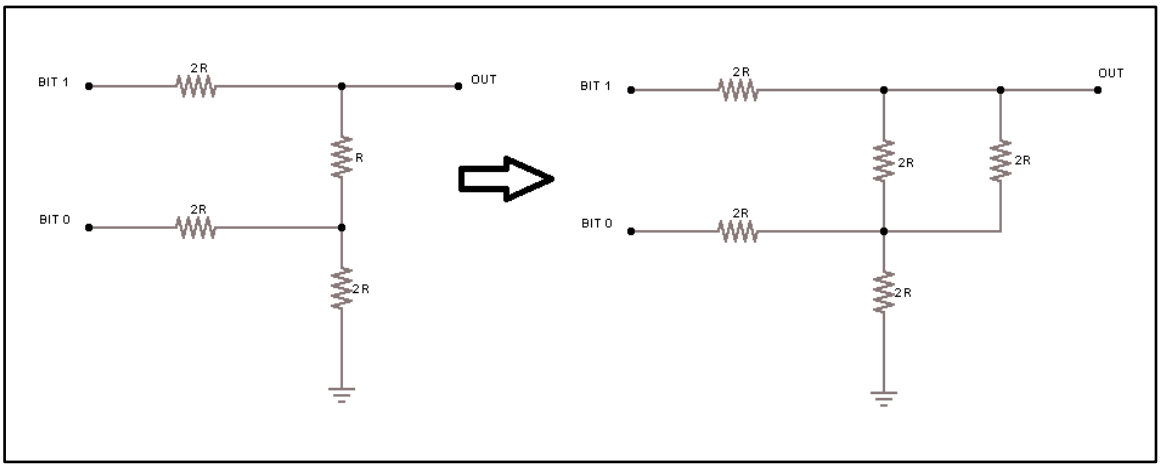
6. Υλοποίηση Υλικού

Το κεφάλαιο αυτό επικεντρώνεται στη διαδικασία σχεδίασης της πλακέτας τυπωμένου κυκλώματος (PCB), η οποία αποτελεί τον πυρήνα της υλοποίησης του υλικού. Η κατασκευή της μονάδας πραγματοποιήθηκε αρχικά σε επιμέρους τμήματα, με χρήση αυτοσχέδιου CNC για τη δημιουργία πρωτοτύπων. Μέσω αυτών των δοκιμαστικών μονάδων επιβεβαιώθηκε η ορθή λειτουργία του κυκλώματος, γεγονός που επέτρεψε την παραγγελία της τελικής πλακέτας σε κατασκευαστή ηλεκτρονικών κυκλωμάτων.

Για τη σχεδίαση χρησιμοποιήθηκε το εξειδικευμένο λογισμικό Eagle CAD, με το οποίο παράχθηκαν τα σχηματικά διαγράμματα και η τελική διάταξη του τυπωμένου κυκλώματος.

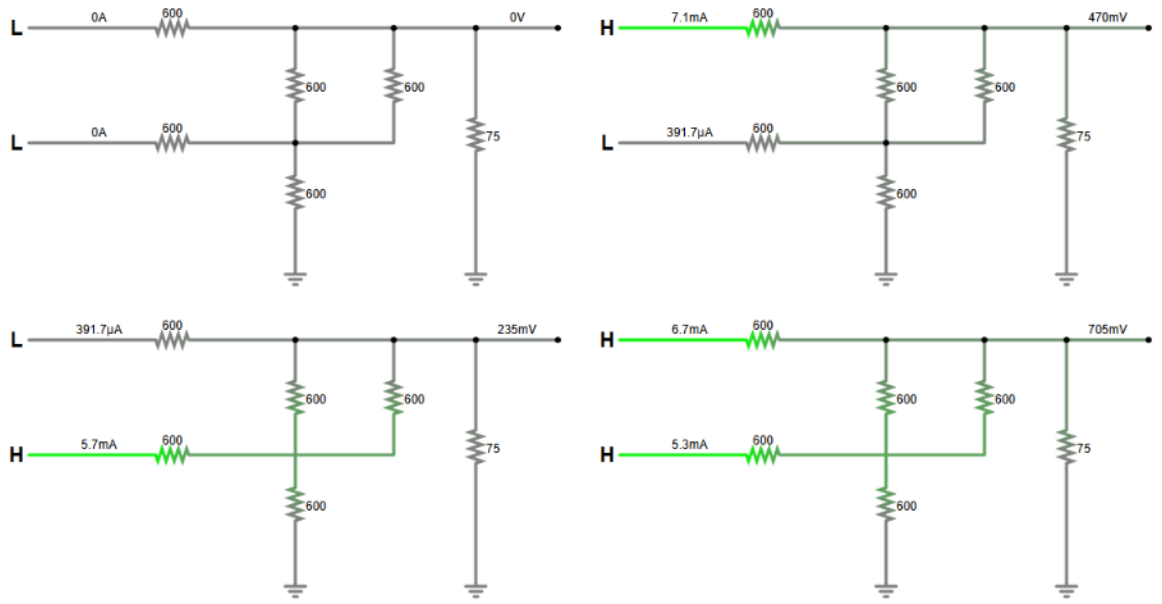
Πριν την εκκίνηση της διαδικασίας σχεδίασης και για να οριστικοποιηθεί η επιλογή των υλικών, μέρος του κυκλώματος σχεδιάστηκε σε πρόγραμμα προσομοίωσης και επιβεβαιώθηκε η ορθή λειτουργία του. Πιο συγκεκριμένα, σχεδιάστηκε ένας 2-bit DAC κατά τον τρόπο που περιγράφηκε στην ενότητα 3.1.3. και η συνδεσμολογία προσομοιώθηκε στο λογισμικό Circuit Simulator ώστε να επιβεβαιώθεί πως το εύρος της τάσης εξόδου βρίσκεται εντός των ορίων που ορίζει το πρότυπο VGA.

Επίσης, για να μειωθεί το πλήθος των διαφορετικών εξαρτημάτων που απαιτούνται για την κατασκευή του κυκλώματος, ο αντιστάτης R αντικαταστάθηκε από δυο παραλληλισμένους αντιστάτες 2R κατά τον τρόπο που φαίνεται παρακάτω, δημιουργώντας ένα ισοδύναμο κύκλωμα:



Εικόνα 11: Κύκλωμα R-2R, 2-Bit DAC

Κατόπιν προσομοιώθηκε το κύκλωμα με εκτιμώμενη τάση λειτουργίας στα 4.7 V. Το εύρος των τιμών της εξόδου του DAC υπολογίστηκε για κάθε πιθανή κατάσταση της εισόδου και επιβεβαιώθηκε πως βρίσκεται μεταξύ 0.0 V και 0.7 V, όσο ακριβώς επιβάλει το πρότυπο.



Εικόνα 12: Προσομοίωση Λειτουργίας του 2-bit DAC

Ο αντιστάτης των 75 Ohm που φαίνεται στην προσομοίωση, δεν αποτελεί μέρος του κυκλώματος αλλά τερματικό αντιστάτη που βρίσκεται εντός της οθόνης. Ωστόσο για την ορθότητα των υπολογισμών, συμπεριλήφθηκε στη διάταξη της προσομοίωσης.

Πέραν του διαγράμματος που ουσιαστικά επιβάλλεται από την αρχιτεκτονική του συστήματος, προστέθηκε και μια απλή διάταξη τροφοδοσίας. Με δεδομένο ότι VGA controller θα αποτελεί περιφερειακό άλλων συστημάτων, δεν ενσωματώθηκε σταθεροποιητής τάσης καθώς είναι αναμενόμενο το κύκλωμα να τροφοδοτείται από τη βασική πλακέτα που θα το οδηγεί (π.χ. Arduino Uno), η οποία ήδη θα έχει σταθεροποιημένη τροφοδοσία. Ωστόσο, για λόγους ασφαλείας, εμποδίζεται η ανάστροφη πόλωση του κυκλώματος αλλά και η τροφοδοσία με τάση υψηλότερη των 5.3V, ώστε να μην καταστραφούν τα ολοκληρωμένα κυκλώματα στην περίπτωση λανθασμένης συνδεσμολογίας.

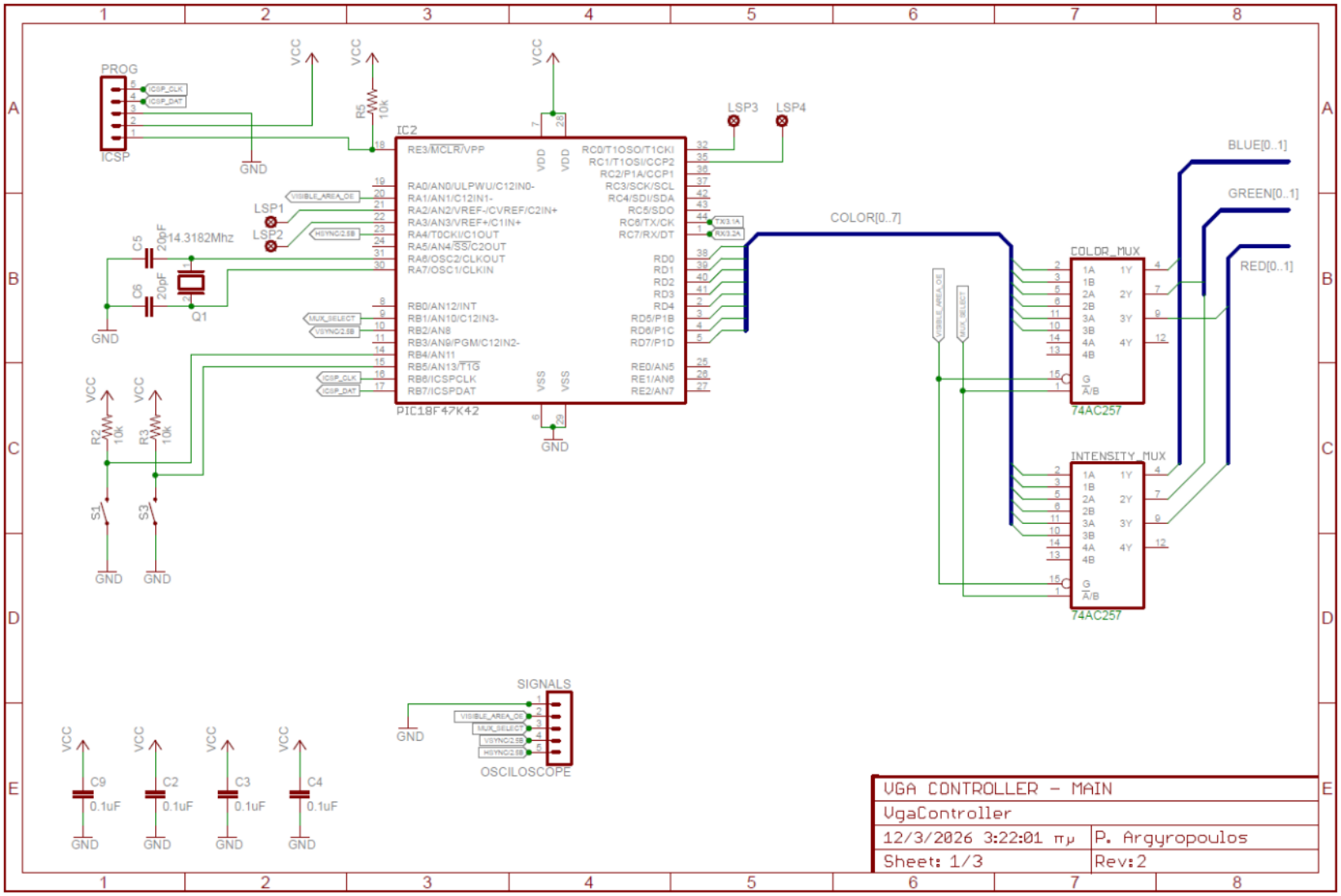
Τέλος, για ευκολία κατά τη φάση ανάπτυξης και αποσφαλμάτωσης του κώδικα, προστέθηκε και μια θύρα mini-USB, επιτρέποντας την άμεση τροφοδοσία της πλακέτας από ηλεκτρονικό υπολογιστή.

6.1 Σχηματικό Διαγράμμα

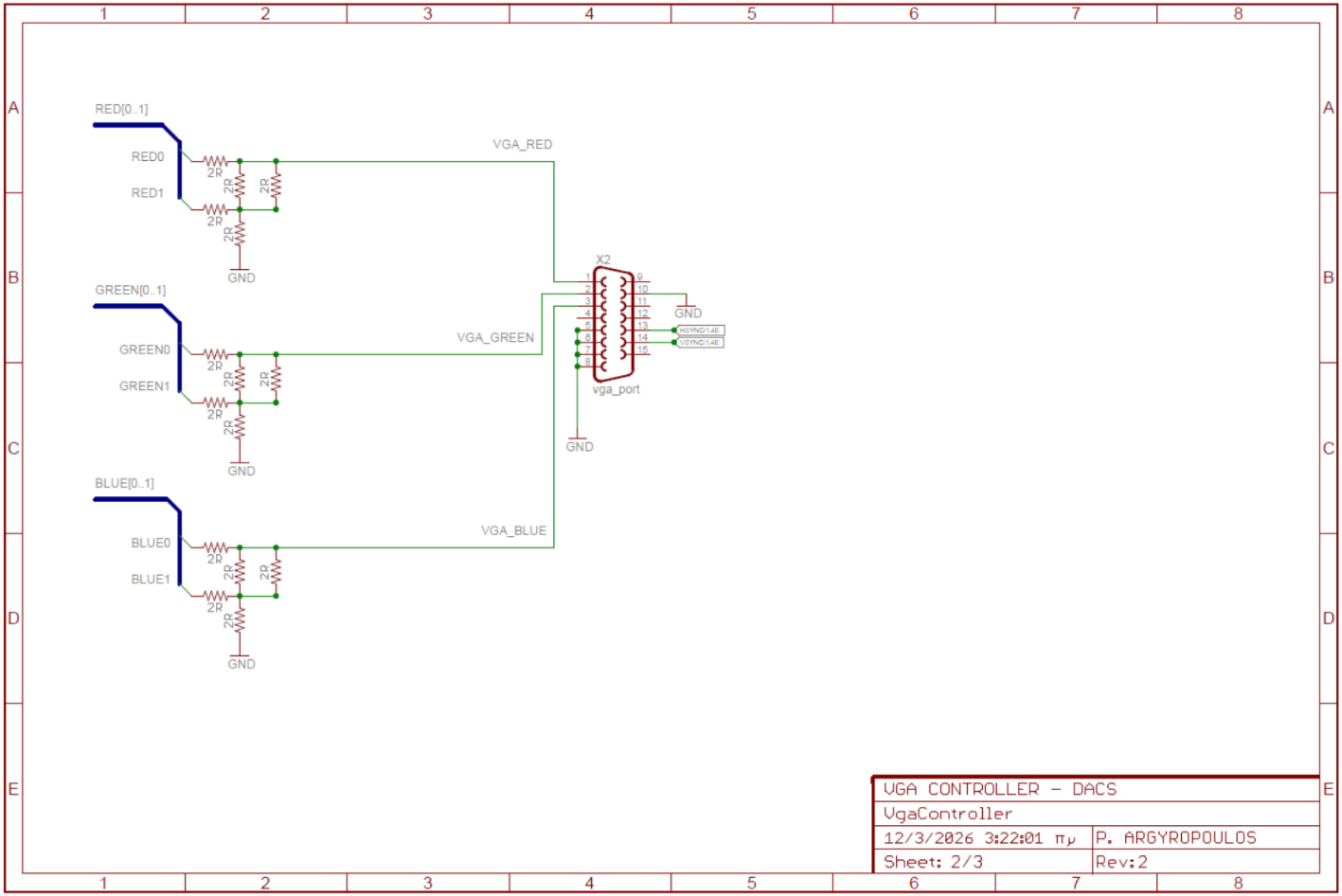
Το 3-σέλιδο σχηματικό διάγραμμα που ακολουθεί, αναπτύχθηκε κατά τρόπο τέτοιο ώστε να ομαδοποιούνται τα επιμέρους κυκλώματα σε σχέση με τον ρόλο τους. Έτσι προέκυψε το βασικό διάγραμμα του μικροελεγκτή και των πολυπλεκτών, το διάγραμμα των ψηφιοαναλογικών μετατροπέων και τέλος το διάγραμμα της τροφοδοσίας.

Οι μπλε διαδρομές στο διάγραμμα, αφορούν στο 8-bit color bus, αλλά και στα τρία 2-bit κανάλια που οδηγούν τους DAC, κατά τον τρόπο που περιγράφηκε στην ενότητα 4.2.

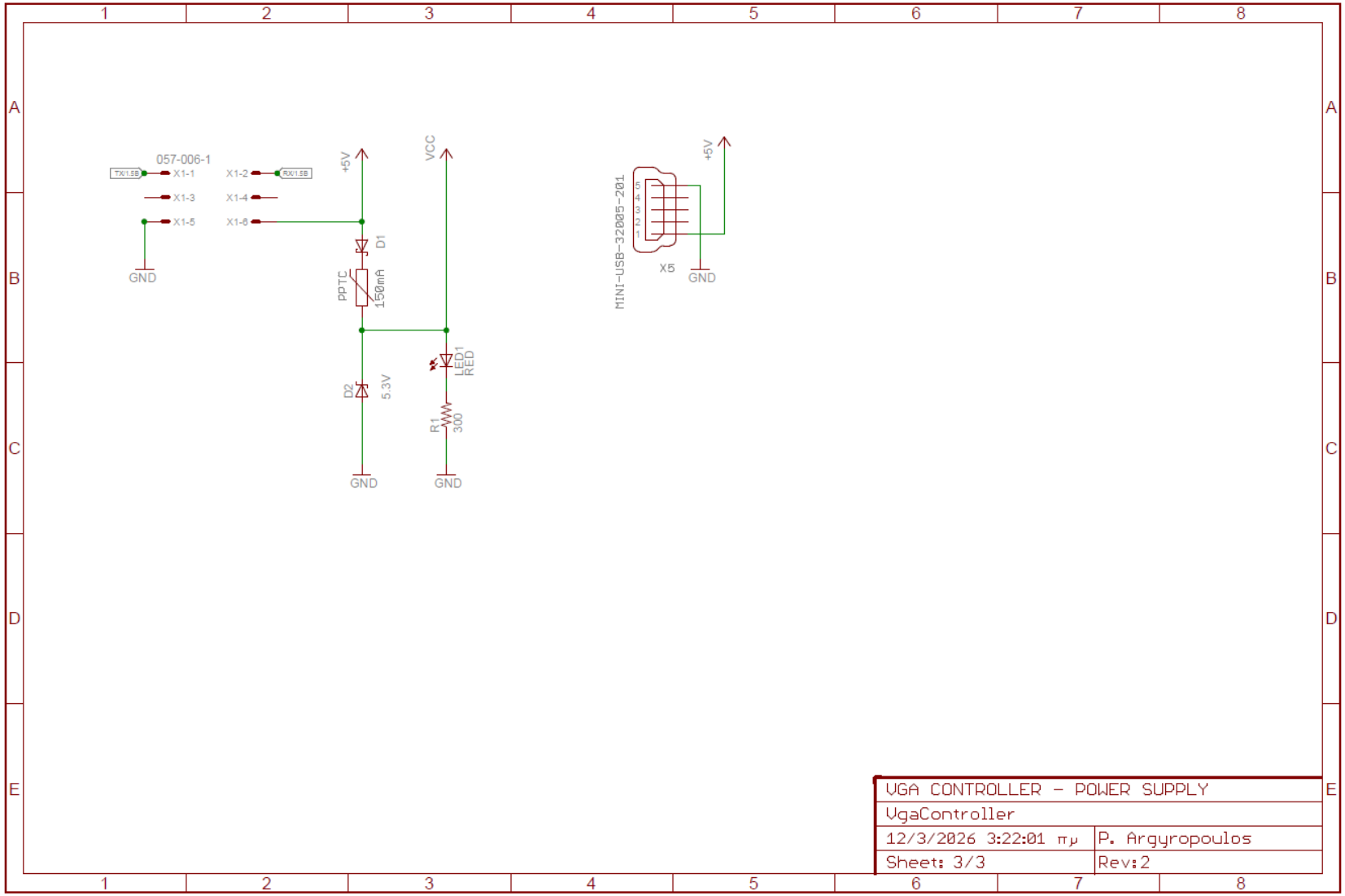
Η διάταξη των πυκνωτών 0.1μF που φαίνεται στη θέση *E1* και *E2* του βασικού διαγράμματος, αφορά σε μίκρο-σταθεροποίηση και αποθορυβοποίηση της τροφοδοσίας (decoupling capacitors) και η χρήση τους συστήνεται από την Microchip στην ενότητα 2.2 του εγχειριδίου του μικροελεγκτή [7], αλλά γενικότερα· η ενσωμάτωση τους θεωρείται καλή πρακτική στη σχεδίαση ψηφιακών κυκλωμάτων.



Εικόνα 13: Σχηματικό Διάγραμμα - Μέρος Α



Εικόνα 14: Σχηματικό Διάγραμμα - Μέρος Β



Εικόνα 15: Σχηματικό Διάγραμμα - Μέρος Γ

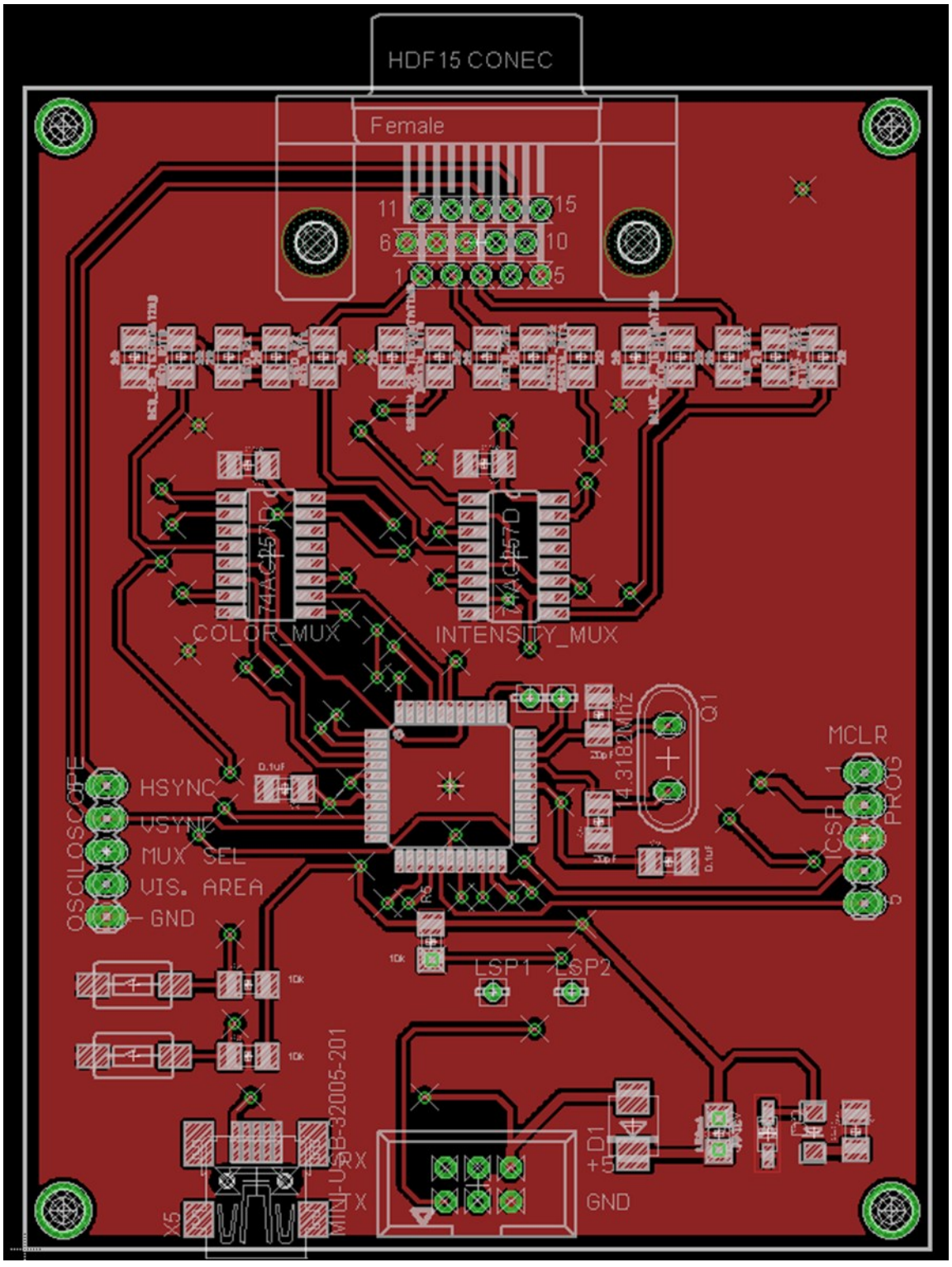
6.2 Διάταξη Πλακέτας

Μετά τη σχεδίαση του σχηματικού διαγράμματος ακολούθησε η σχεδίαση της πλακέτας, διαδικασία κατά την οποία όλα τα εξαρτήματα τοποθετούνται στην πλακέτα και χαράσσονται οι διασυνδέσεις τους.

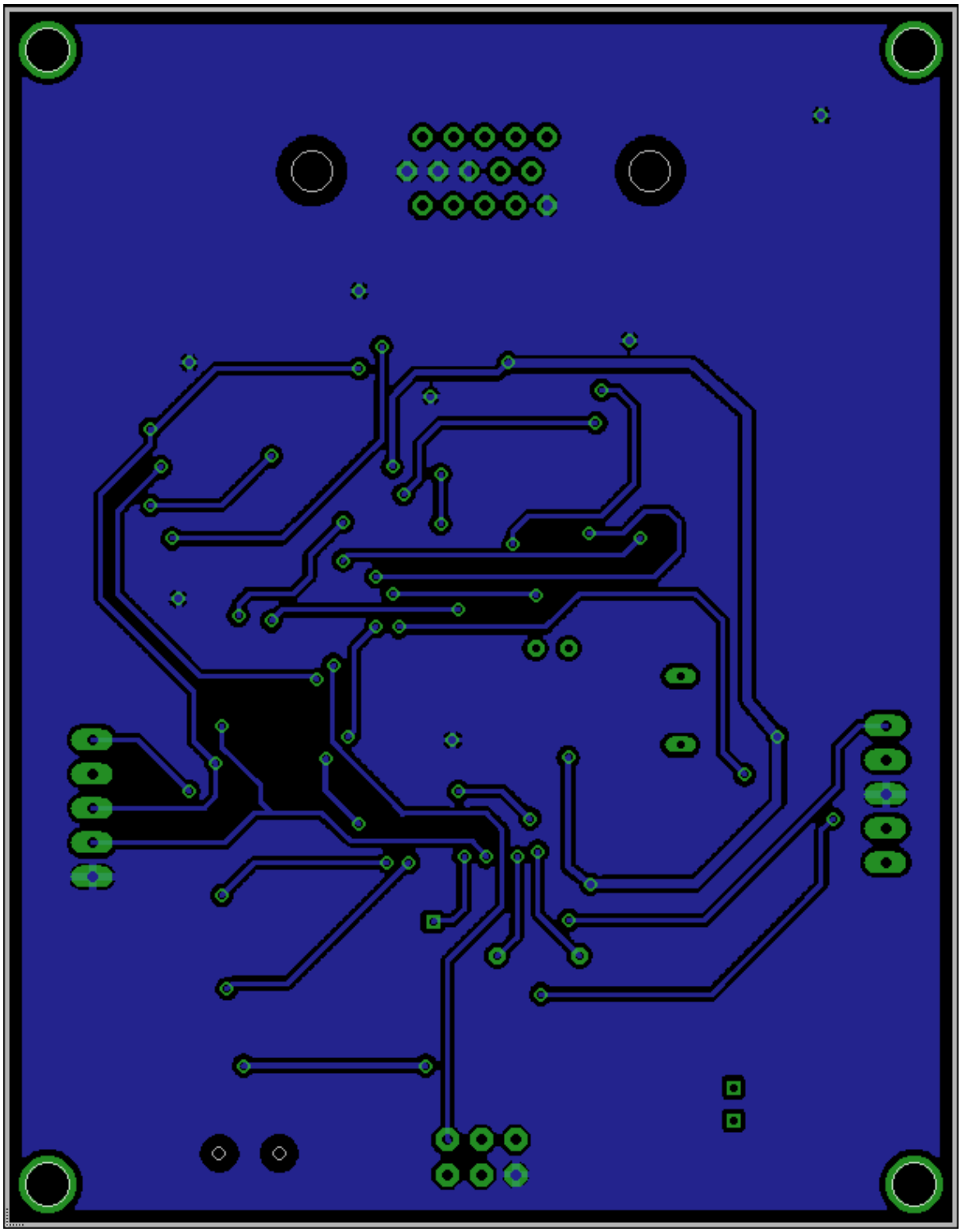
Για να μειωθεί κατά το δυνατόν περισσότερο το τελικό μέγεθος της πλακέτας, όλα τα εξαρτήματα πλην των ακροδεκτών, είναι τύπου SMD. Μοναδική εξαίρεση αποτελεί ο κρύσταλλος 14.3182 MHz, ο οποίος είναι τύπου DIP, καθώς στην μορφή αυτή είναι ευρέως διαθέσιμος στο εμπόριο αλλά επίσης και γιατί δεν παρουσίαζε καμία διαφορά στο μέγεθος σε σχέση με την SMD έκδοση.

Η πλακέτα που δημιουργήθηκε είναι διπλής όψης, με διαστάσεις 70 x 90 mm.

Η πλήρης λίστα των εξαρτημάτων που χρησιμοποιήθηκαν δίνονται στο αντίστοιχο παράρτημα.



Εικόνα 16: Εμπρόσθια Όψη Κυκλώματος



Εικόνα 17: Οπίσθια Όψη Κυκλώματος

7. Έλεγχος Λειτουργίας

Η παρούσα ενότητα αποσκοπεί στην αναλυτική αξιολόγηση της λειτουργικότητας του VGA controller που υλοποιήθηκε στο πλαίσιο της πτυχιακής εργασίας. Ο έλεγχος περιλαμβάνει δύο βασικούς άξονες: αφενός τη μέτρηση και επαλήθευση της χρονικής ακρίβειας των σημάτων συγχρονισμού HSYNC και VSYNC, τα οποία είναι κρίσιμα για τη σωστή απεικόνιση στην οθόνη VGA, και αφετέρου την επιβεβαίωση της ορθής λειτουργίας των τριών υποστηριζόμενων καταστάσεων οθόνης: Text Mode, Image Mode και Plot Mode.

Η αξιολόγηση πραγματοποιήθηκε με τη χρήση παλμογράφου για τη μέτρηση των παραγόμενων σημάτων, καθώς και με οπτική παρατήρηση της εξόδου στην οθόνη VGA για την εκτίμηση της ποιότητας και της σταθερότητας της απεικόνισης. Τα αποτελέσματα των δοκιμών παρουσιάζονται αναλυτικά στις επόμενες ενότητες, συνοδευόμενα από κυματομορφές, φωτογραφίες και συγκριτικούς πίνακες.

Για την ενεργοποίηση και τον έλεγχο των διαφορετικών καταστάσεων λειτουργίας του VGA controller, χρησιμοποιήθηκε σειριακή επικοινωνία μέσω του τερματικού προγράμματος PuTTY. Ο ελεγκτής ήταν συνδεδεμένος με τον υπολογιστή μέσω USB-to-Serial μετατροπέα, και η επικοινωνία πραγματοποιήθηκε με ρυθμό μετάδοσης 57.600 baud.

Οι εντολές αποστέλλονταν χειροκίνητα, με σκοπό την εναλλαγή μεταξύ των υποστηριζόμενων καταστάσεων αλλά και την παραγωγή ενδεικτικών απεικονίσεων. Η ανταπόκριση του συστήματος ήταν άμεση και σταθερή, χωρίς απώλειες ή καθυστερήσεις στην επικοινωνία.

7.1 Σήματα Συγχρονισμού

Ο βασικός έλεγχος λειτουργίας του VGA Controller συνίσταται στη μέτρηση και την επαλήθευση της χρονικής διάρκειας των παραγόμενων σημάτων συγχρονισμού. Για τον σκοπό αυτό, χρησιμοποιήθηκε ο ψηφιακός παλμογράφος RIGOL DS1002Z, με συχνότητα δειγματοληψίας 1GSa/s και μετρήθηκαν με ακρίβεια τα σήματα HSYNC και VSYNC. Τα αποτελέσματα των μετρήσεων καταγράφηκαν και επιβεβαιώθηκε η σύγκλιση με το πρότυπο VGA.

Η απόκλιση του χρονισμού των σημάτων, βρέθηκε σε κάθε περίπτωση μικρότερη από $\pm 0.5\%$ που ορίζει το πρότυπο, γεγονός που επιβεβαιώνει ότι ο ελεγκτής μπορεί να συνεργαστεί με κάθε οθόνη του εμπορίου που διαθέτει διεπαφή VGA.

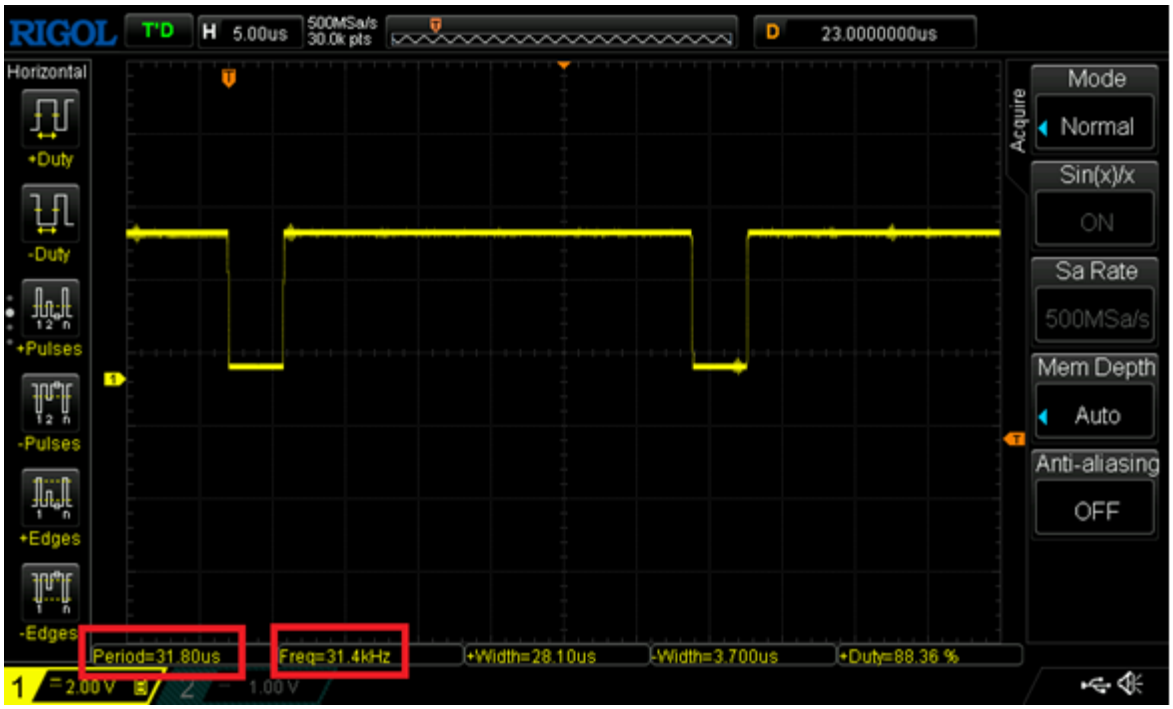
Επιπλέον επιβεβαιώθηκε ότι οι ακμές των δυο σημάτων είναι απόλυτα συγχρονισμένες και η περίοδος τους είναι σταθερή, επομένως εξασφαλίζεται η παραγωγή καθαρής και σταθερής εικόνας, χωρίς φαινόμενα τρεμοπαιξίματος (flickering).

Τα αποτελέσματα συνοψίζονται στον παρακάτω πίνακα:

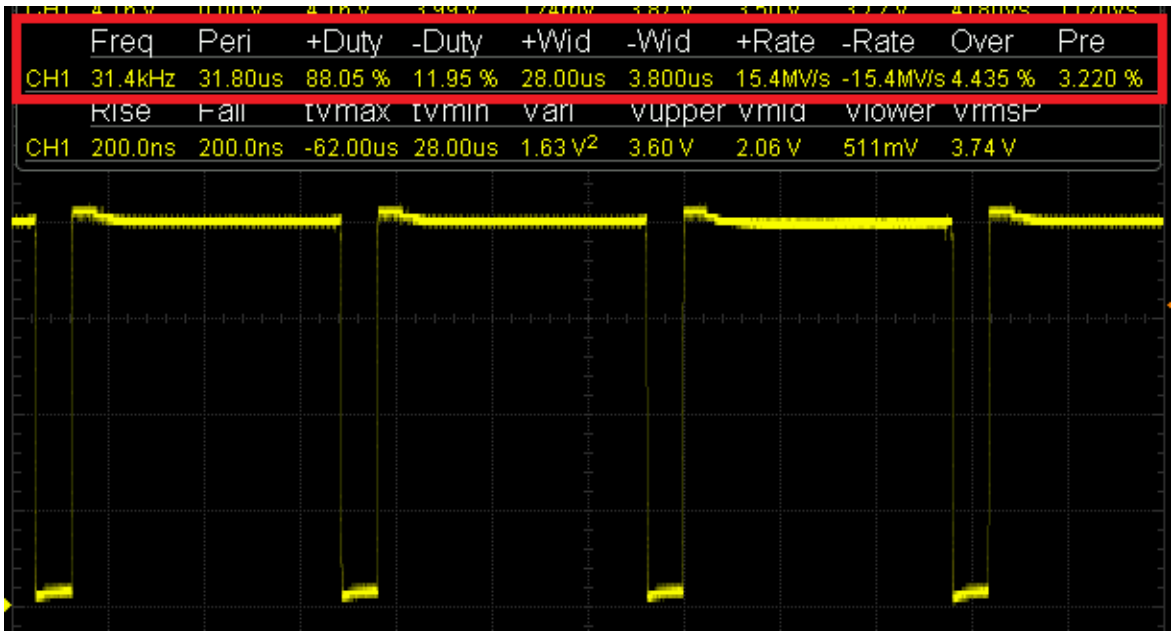
	Τιμή Στόχος (μs)	Μέτρηση (μs)	Απόκλιση %
Παλμός HSYNC	31,78	31,80	0,063%
Θετικό τμήμα	27,97	28,00	0,107%
Αρνητικό τμήμα	3,81	3,80	-0,262%

	Τιμή Στόχος (ms)	Μέτρηση (ms)	Απόκλιση %
Παλμός VSYNC	16,68	16,70	0,120%
Θετικό τμήμα	16,62	16,65	0,202%
Αρνητικό τμήμα	0,06	0,06	0,063%

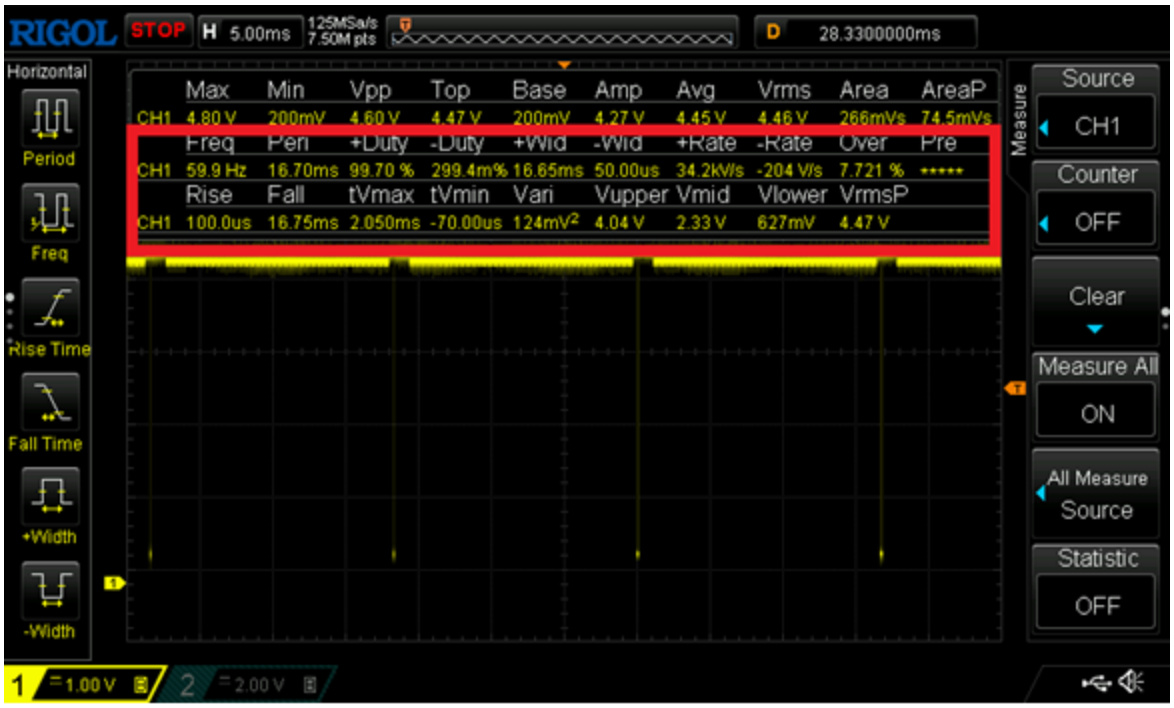
Πίνακας 10: Αποκλίσεις Σημάτων Χρονισμού



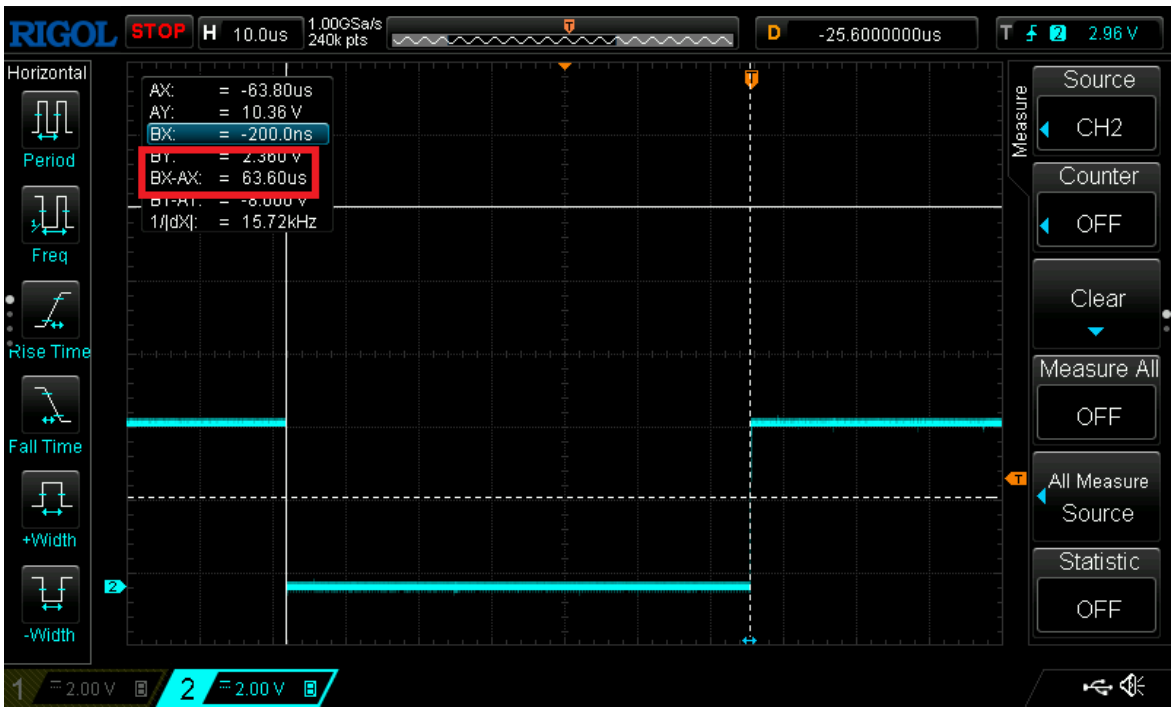
Εικόνα 18: Παλμός HSYNC



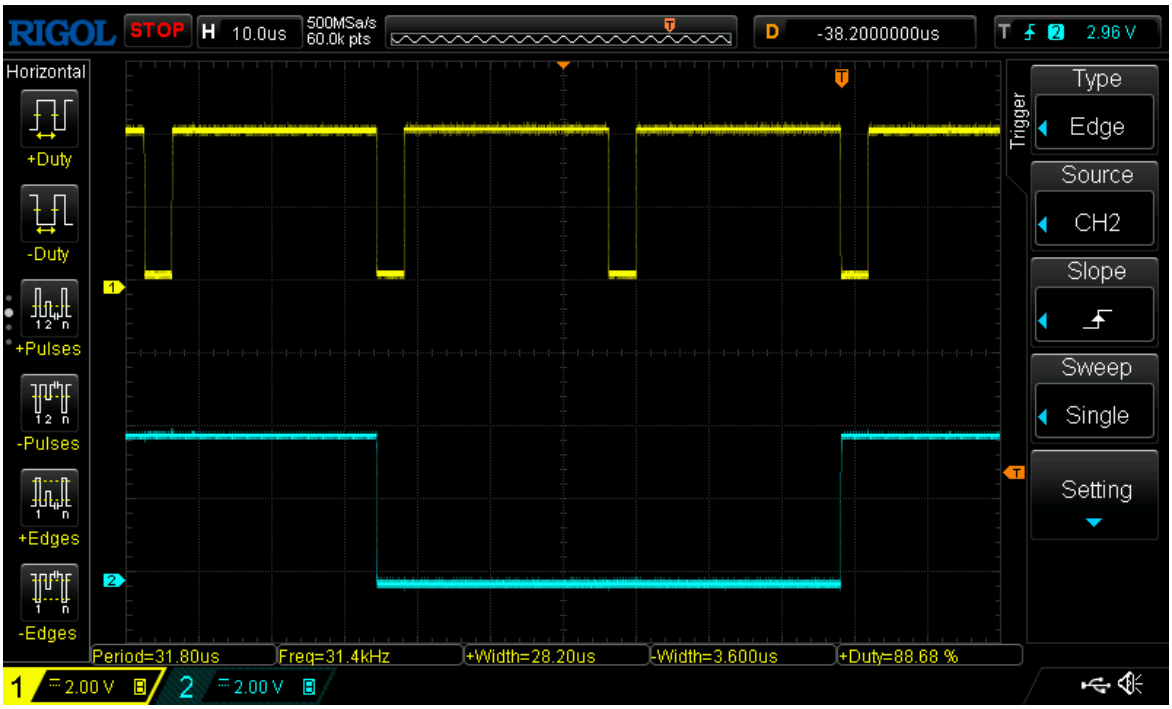
Εικόνα 19: Μικρομετρικές τιμές παλμού HSYNC



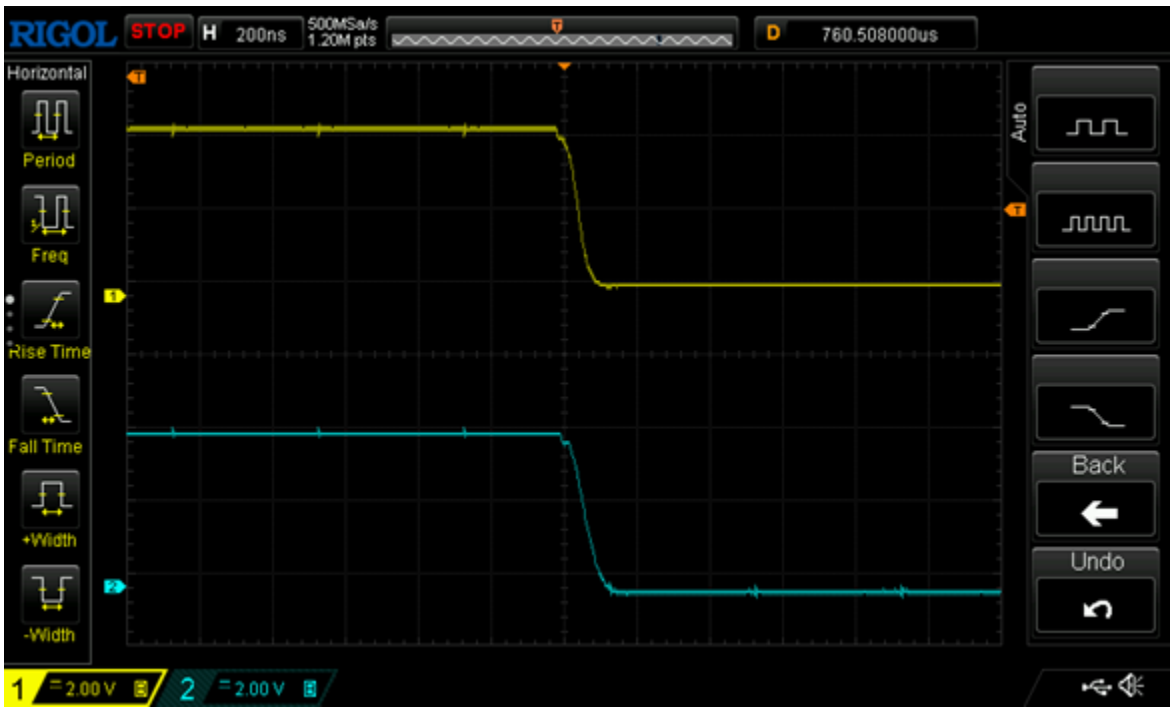
Εικόνα 20: Μικρομετρικές τιμές παλμού VSYNC



Εικόνα 21: Αρνητικό τμήμα κύκλου VSYNC



Εικόνα 22: Παλμοί HSYNC και VSYNC

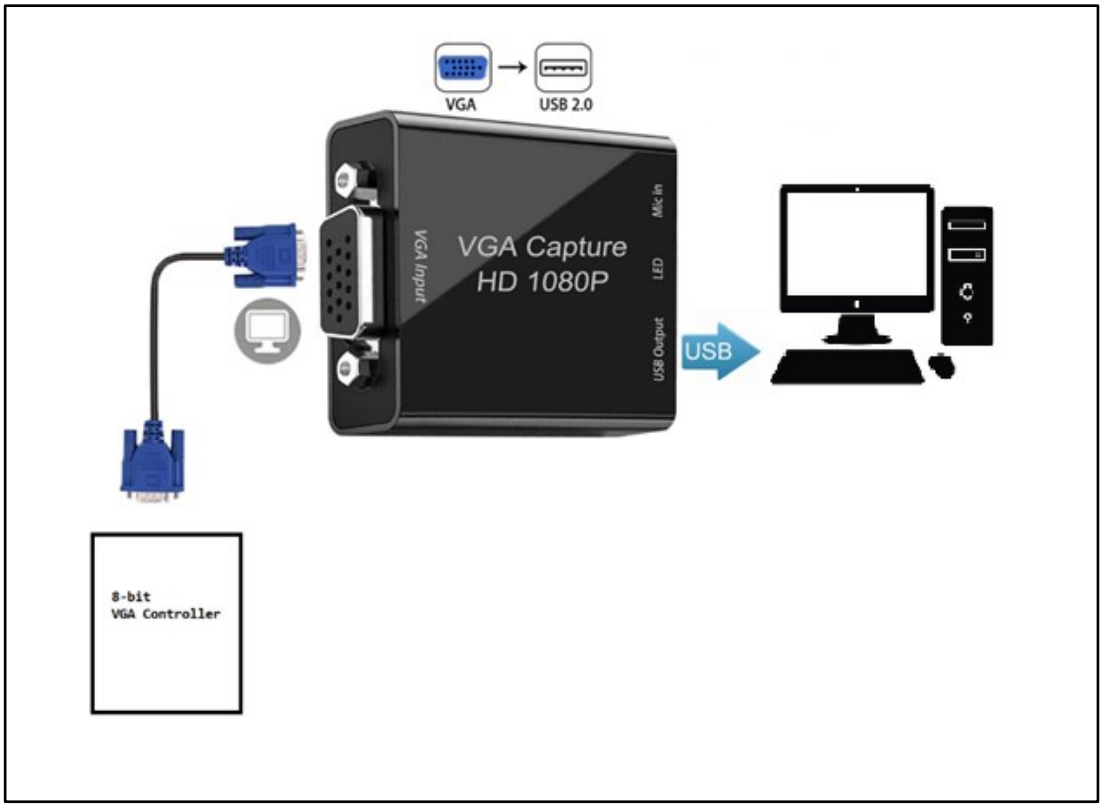


Εικόνα 23: Συγχρονισμός ακμών HSYNC και VSYNC

7.2 Καταστάσεις Οθόνης

Για κάθε κατάσταση δημιουργήθηκε ένα μικρό πρόγραμμα εντολών στην γλώσσα που αναπτύχθηκε ειδικά για τον ελεγκτή. Δίνονται ενδεικτικά τα προγράμματα που δημιουργήθηκαν για κάθε κατάσταση και το αποτέλεσμα που προέκυψε στην οθόνη.

Για την λήψη των στιγμιότυπων της οθόνης, χρησιμοποιήθηκε ένας VGA to USB μετατροπέας, η είσοδος του οποίου συνδέθηκε στη VGA έξοδο του ελεγκτή και η έξοδος του στη θύρα USB του υπολογιστή.

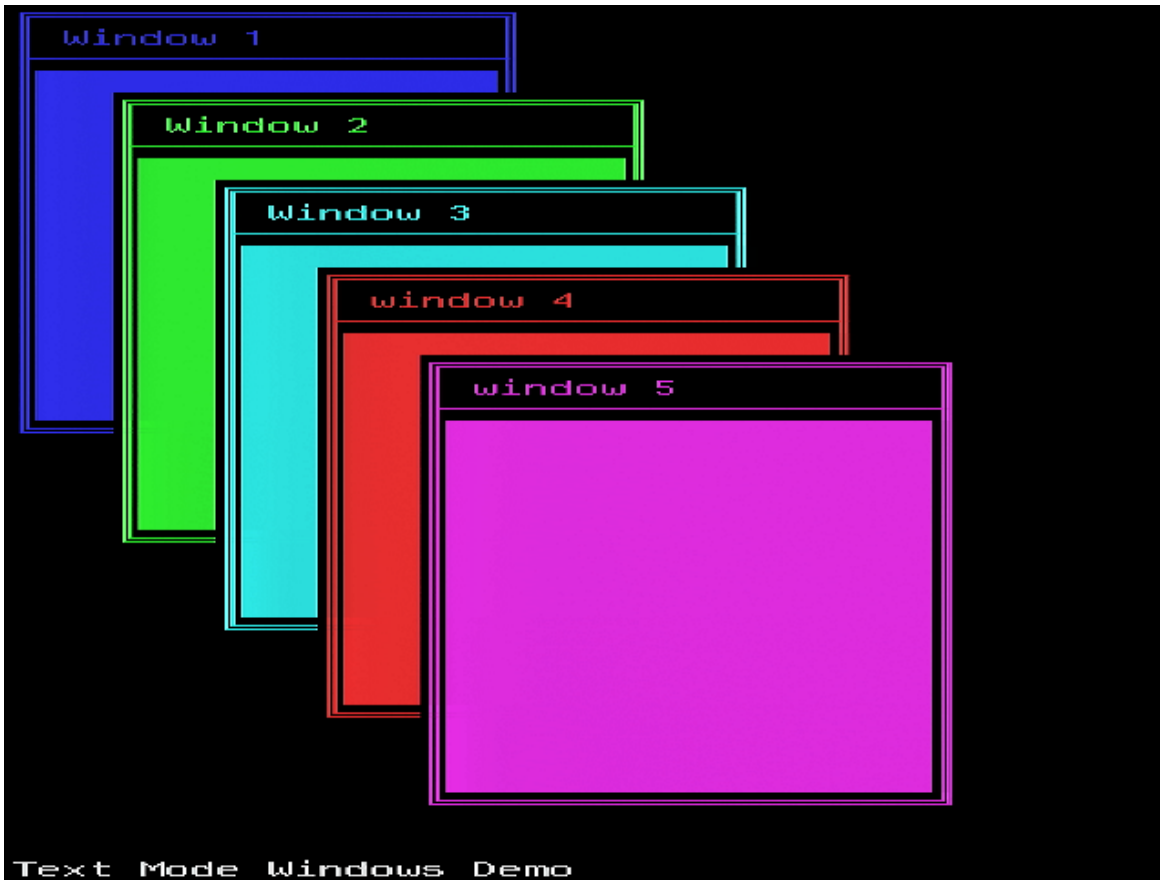


Εικόνα 24: Διασύνδεση VGA controller με τον VGA to USB μετατροπέα

Οι φωτογραφίες που ελήφθησαν παρουσιάζουν μια μικρή χρωματική αλλοίωση, λόγω της μετατροπής που υφίσταται το αναλογικό σήμα, ωστόσο σε κάθε περίπτωση είναι αντιπροσωπευτικές.

7.2.1 Κατάσταση Κειμένου

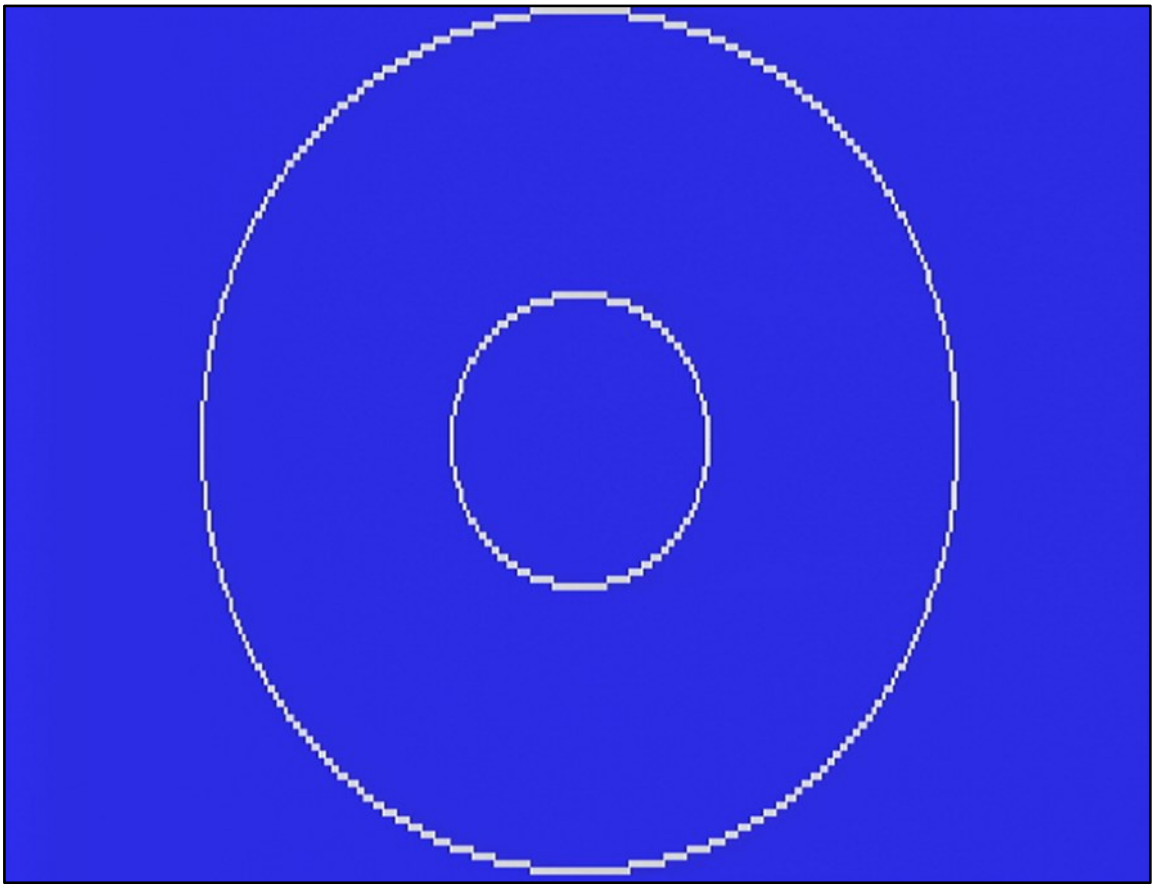
```
VGA Controller Commands
^[m0;              //select text mode
^[x0;              //cursor off
^[b0;              //backcolor=black
^[f15;             //forecolor=bright white
^[E                //clear screen
^[Y0,39;           // locate x=0, y=39
Text Mode Windows Demo // write text

^[f9;              //forecolor=light blue
^[w0,0,19,19,0;    //draw window (0,0,20,20) with thick border
Window 1           //set window title
^[e1,3,18,18,219; //fill box (1,3,19,19) with ascii char 219

^[f10;
^[w4,4,24,24,0;
Window 2
^[e5,7,23,23,219;

^[f11;
^[w8,8,28,28,1;
Window 3
^[e9,11,27,27,219;

^[f12;
^[w12,12,32,32,1;
Window 4
^[e13,15,31,31,219;

^[f13;
^[w16,16,36,36,1;
Window 5
^[e17,19,35,35,219;
```



Εικόνα 25: Text Mode Demo

7.2.2 Κατάσταση Σχεδίασης

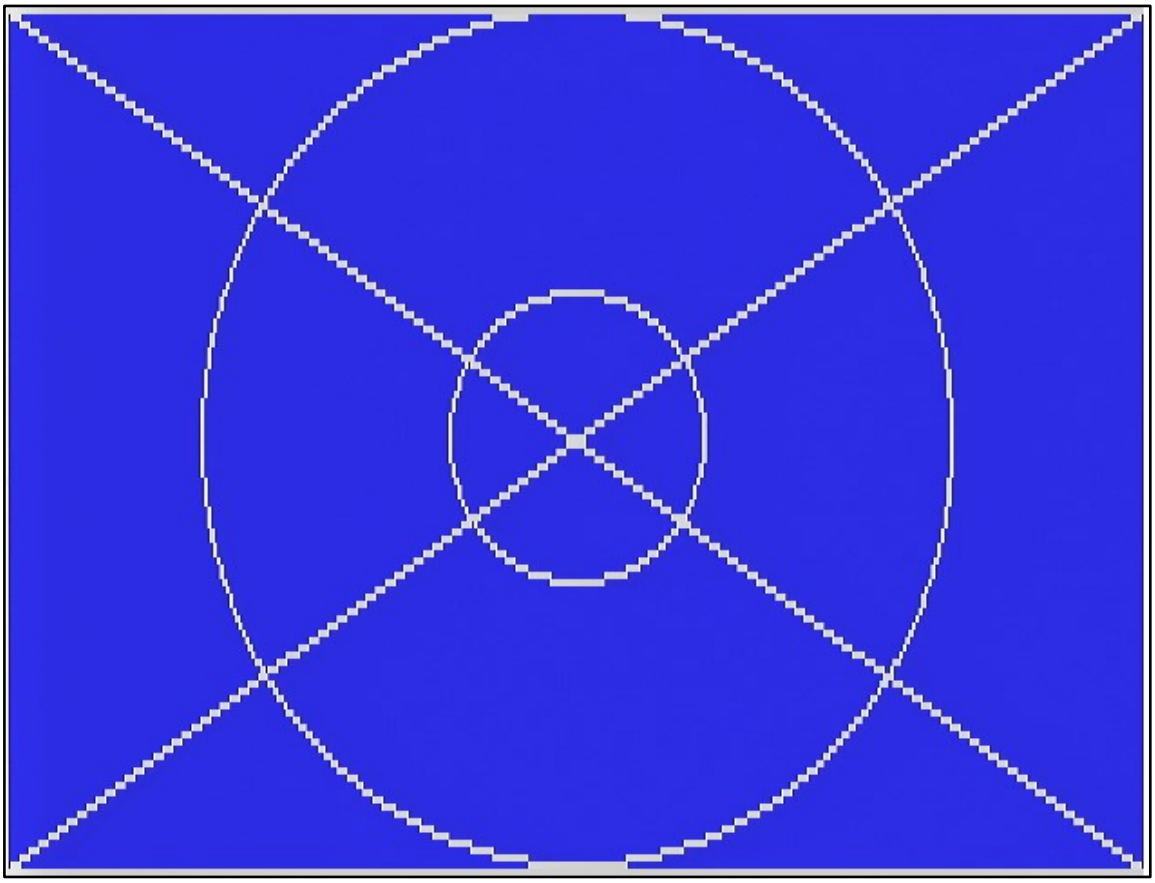
```
VGA Controller Commands
^[m1;              //select plot mode
^[b9;              //backcolor=light blue
^[f15;             //forecolor=bright white

^[c180,59,20,1;    //draw a circle
                   //radius   : 20 px
                   //location : {x=180,y=59}

^[c180,59,59,1;
```



Εικόνα 26: Plot Mode Circles Demo

VGA Controller Commands

```
^[10,0,359,0,1;      // draw line at:
                     // x1=0
                     // y1=0
                     // x2=359
                     // y0=0


^[10,0,0,119,1;

^[10,119,359,119,1;

^[1359,119,359,0,1;

^[10,0,359,119,1;

^[10,119,359,0,1;
```



Εικόνα 27: Plot Mode Circles and Lines Demo

7.2.3 Κατάσταση Γραφικών

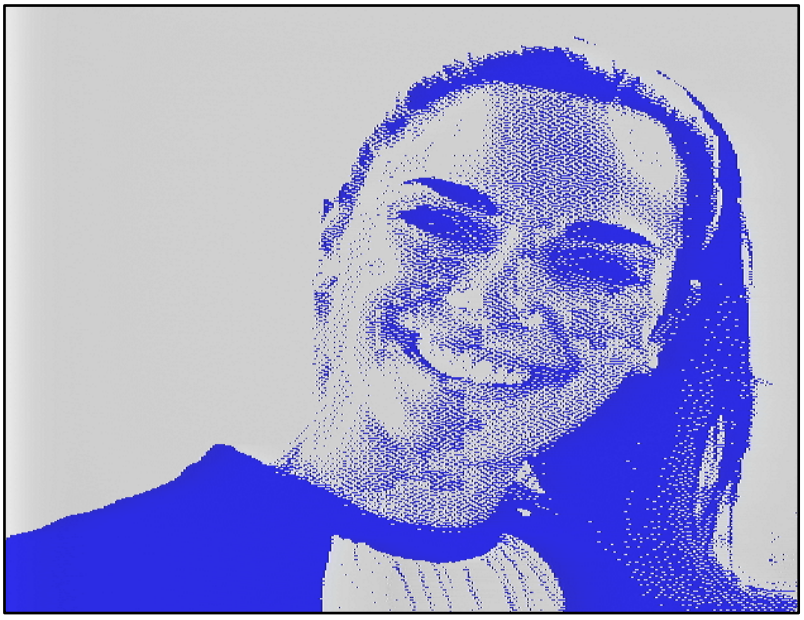
VGA Controller Commands

```
^[m1;              //select gfx mode
^[l0;              //load first stored image
```

Εικόνα 28: Image Mode Demo Picture

VGA Controller Commands

```
^[p9,15;          // change all cell colors
                  // set background color to light blue
                  // and foreground color to bright white
```

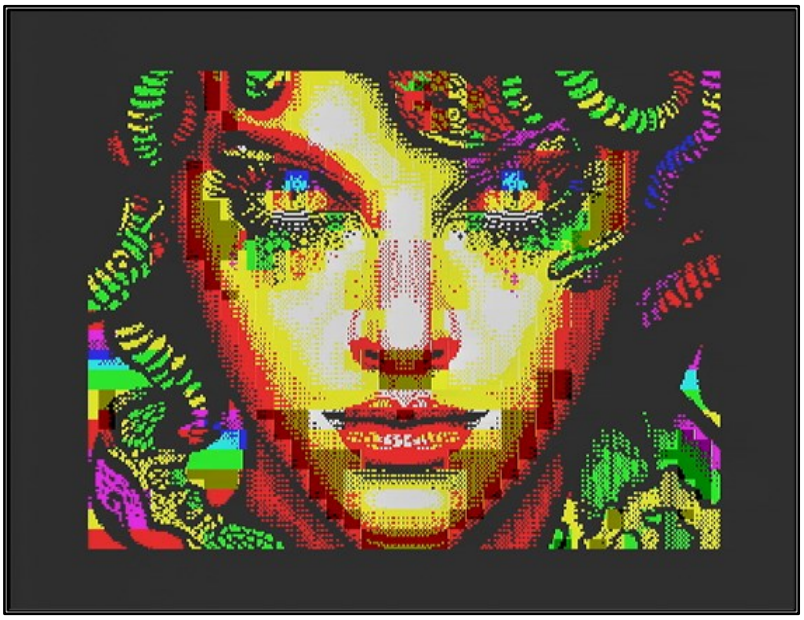


Εικόνα 29: Image Mode Demo - Dynamic Color Change

VGA Controller Commands	
^[11;	//load second stored image

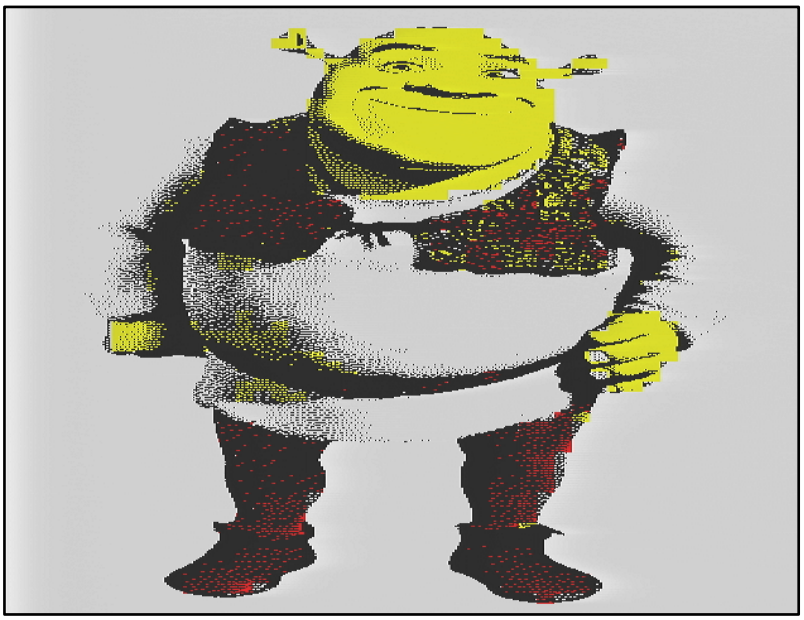
Εικόνα 30: Image Mode Demo Picture 2

VGA Controller Commands	
^[11;	//load third stored image



Εικόνα 31: Image Mode Demo Picture 3

VGA Controller Commands
^[11; //load fourth stored image



Εικόνα 32: Image Mode Demo Picture 4

7.3 Απόκριση Συστήματος

Για να ελεγχθεί η απόκριση του συστήματος υπό συνθήκες φόρτου, έγιναν δοκιμές αποστολής δεδομένων σε διαφορετικές ταχύτητες συγχρονισμού και αξιολογήθηκε η συμπεριφορά του ελεγκτή.

Δημιουργήθηκε μια εφαρμογή με την οποία μετρήθηκε η απόκριση του συστήματος για όγκο δεδομένων 1.800, 18.000 και 36.000 bytes και καταγράφηκε ο χρόνος που χρειάστηκε από την έναρξη της μετάδοσης, μέχρι και την απεικόνιση των δεδομένων στην οθόνη. Ο ελεγκτής συνδέθηκε στη σειριακή θύρα του υπολογιστή και η εφαρμογή έστελνε διαρκώς δεδομένα ενώ ταυτόχρονα έλεγχε για τυχόν εισερχόμενα μηνύματα. Στην περίπτωση που λάμβανε μήνυμα «WT!» από τον ελεγκτή, τότε εισήγαγε τεχνητή καθυστέρηση 100 ms πριν την αμέσως επόμενη αποστολή. Ο πίνακας που ακολουθεί παρουσιάζει τον συνολικό χρόνο σε sec, από την έναρξη της μετάδοσης έως την πλήρη απεικόνιση των δεδομένων στην οθόνη:

		Ταχύτητα Συγχρονισμού (baud)					
		9.600	**14.400**	**19.200**	**28.800**	**38.400**	**57.600**
Δεδομένα (bytes)	**1.800**	1,919	1,283	0,976	0,662	0,598	0,590
	18.000	18,778	12,522	9,406	6,286	6,062	6,080
	36.000	37,510	25,010	18,760	12,525	12,137	12,140

Πίνακας 11: Απόκριση του συστήματος σε sec, ανά ταχύτητα μετάδοσης

Από την ανάλυση των αποτελεσμάτων προκύπτει ότι μέχρι την ταχύτητα των 28.800 baud, η επεξεργασία των δεδομένων πραγματοποιείται σχεδόν σύγχρονα με την αποστολή τους. Αντιθέτως, σε υψηλότερες ταχύτητες παρατηρείται συχνότερη εμφάνιση του μηνύματος «WT!», γεγονός που υποδηλώνει την ανάγκη του ελεγκτή να επιβραδύνει την αποστολή για να προστατεύσει τον buffer από υπερχείλιση και να διατηρήσει την ακεραιότητα των δεδομένων.

Προς επιβεβαίωση των παρατηρήσεων που προέκυψαν από τις αρχικές μετρήσεις χρόνου, οι τιμές του Πίνακας 11: Απόκριση του συστήματος σε sec, ανά ταχύτητα μετάδοσης, μετατράπηκαν στον αντίστοιχο ισοδύναμο ρυθμό μεταφοράς δεδομένων, εκπεφρασμένο σε baud. Η μετατροπή αυτή επιτρέπει την απευθείας σύγκριση της πραγματικής απόδοσης του συστήματος με την ονομαστική ταχύτητα συγχρονισμού της σειριακής επικοινωνίας.

Ο Πίνακας 12: Απόκριση του συστήματος εκπεφρασμένη σε baud, παρουσιάζει τον υπολογισμένο ρυθμό μεταφοράς για κάθε συνδυασμό όγκου δεδομένων και ταχύτητας συγχρονισμού. Παρατηρείται ότι μέχρι και την ταχύτητα των 28.800 baud, ο ελεγκτής επιτυγχάνει σχεδόν πλήρη αξιοποίηση του διαθέσιμου εύρους ζώνης, καθώς ο ισοδύναμος ρυθμός μεταφοράς προσεγγίζει την ονομαστική τιμή. Αυτό υποδηλώνει ότι η διαδικασία παραλαβής, αποκωδικοποίησης και απεικόνισης των δεδομένων πραγματοποιείται σχεδόν σύγχρονα με την αποστολή τους.

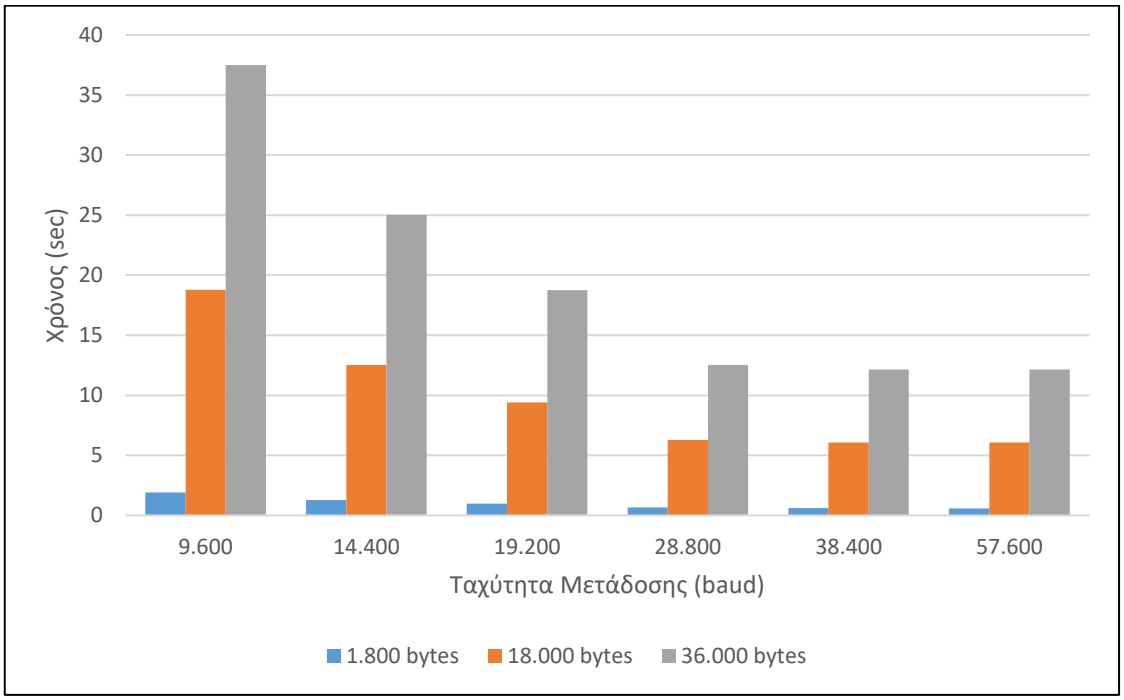
		Ταχύτητα Συγχρονισμού (baud)					
		9.600	**14.400**	**19.200**	**28.800**	**38.400**	**57.600**
Δεδομένα (bytes)	**1.800**	9.380	14.030	18.443	27.190	30.100	30.508
	18.000	9.586	14.375	19.137	28.635	29.693	29.605
	36.000	9.597	14.394	19.190	28.743	29.661	29.654

Πίνακας 12: Απόκριση του συστήματος εκπεφρασμένη σε baud

Αντιθέτως, σε υψηλότερες ταχύτητες (38.400 και 57.600 baud), παρατηρείται κορεσμός της απόδοσης, με τον ισοδύναμο ρυθμό να σταθεροποιείται γύρω από τα 30.000 baud, ανεξαρτήτως του όγκου δεδομένων. Το φαινόμενο αυτό οφείλεται στην ενεργοποίηση του μηχανισμού ελέγχου ροής του ελεγκτή, ο οποίος μέσω του μηνύματος «WT!» αιτείται την προσωρινή διακοπή της αποστολής δεδομένων, ώστε να προστατευθεί ο buffer από υπερχείλιση.

Η συμπεριφορά αυτή επιβεβαιώνει την ύπαρξη ενός δυναμικού ορίου απόδοσης του συστήματος, το οποίο εξαρτάται όχι μόνο από την ταχύτητα συγχρονισμού αλλά και από τις εγγενείς δυνατότητες επεξεργασίας του ελεγκτή. Η γνώση αυτού του ορίου είναι κρίσιμη για τον σχεδιασμό εφαρμογών πραγματικού χρόνου, όπου η σταθερότητα και η αξιοπιστία της επικοινωνίας αποτελούν βασικές απαιτήσεις.

Στο παρακάτω ραβδόγραμμα γίνεται εμφανής η φθίνουσα απόδοση του συστήματος στις υψηλές ταχύτητες, καθώς η βελτίωση που παρατηρείται μεταξύ των 28.8 Kbaud και των 57.6 Kbaud είναι σχεδόν αμελητέα, παρά τον διπλασιασμό της ταχύτητας αποστολής δεδομένων.



Διάγραμμα 16: Χρόνος απόκρισης σε sec

8. Ανασκόπηση Έργου

Το έργο ξεκίνησε με στόχο την υλοποίηση ενός βασικού ελεγκτή VGA, ικανού να δέχεται δεδομένα μέσω καθορισμένου πρωτοκόλλου επικοινωνίας, να τα επεξεργάζεται και να παράγει τα απαραίτητα σήματα για την οδήγηση μιας VGA οθόνης. Οι αρχικές απαιτήσεις περιλάμβαναν:

- Υποστήριξη βασικής διεπαφής εισόδου.
- Παραγωγή αναλογικών RGB σημάτων μέσω DAC.
- Παραγωγής ακριβούς χρονισμού (σήματα HSync και Vsync).

Κατά την εξέλιξη του έργου, οι αρχικές προδιαγραφές όχι μόνο καλύφθηκαν, αλλά ξεπεράστηκαν κατά πολύ, οδηγώντας στην ανάπτυξη ενός πλήρως λειτουργικού συστήματος VGA controller. Ο ελεγκτής που αναπτύχθηκε αποτελεί μια αυτοδύναμη κάρτα γραφικών, με δυνατότητες που τον καθιστούν κατάλληλο για εκπαιδευτική χρήση, ενσωμάτωση σε embedded εφαρμογές, αλλά και προσαρμογή σε πιο σύνθετα γραφικά περιβάλλοντα, όπως απλά παιχνίδια.

Παρά την περιορισμένη υπολογιστική ισχύ του μικροελεγκτή, η έξυπνη διαχείριση των περιφερειακών του και η προσεκτική αρχιτεκτονική του λογισμικού και του υλικού, οδήγησε στην κατασκευή ενός συστήματος πραγματικού χρόνου, το οποίο όχι μόνο επιτυγχάνει να τηρήσει τις αυστηρές απαιτήσεις του προτύπου VGA και να προβάλει στατική εικόνα, αλλά επιπλέον να προβάλει δυναμικά δεδομένα και να υποστηρίζει πολλαπλές καταστάσεις απεικόνισης, κειμένου και γραφικών.

Για την επικοινωνία του ελεγκτή VGA, αναπτύχθηκε μια εξειδικευμένη γλώσσα εντολών βασισμένη σε Escape Sequences, η οποία επιτρέπει τη λήψη εντολών και παραμέτρων με τρόπο δομημένο και επεκτάσιμο. Για τη διερμήνευση της γλώσσας αυτής, υλοποιήθηκε ένας διερμηνευτής εντολών (interpreter), ο οποίος λειτουργεί σε πραγματικό χρόνο, αναλύοντας τα εισερχόμενα δεδομένα και εκτελώντας τις αντίστοιχες λειτουργίες του συστήματος. Κατά την υλοποίηση του διερμηνευτή, εφαρμόστηκε το σχεδιαστικό

πρότυπο State Pattern, το οποίο αποδείχθηκε καθοριστικής σημασίας για την ευελιξία και την επεκτασιμότητα του κώδικα. Η χρήση του προτύπου επέτρεψε την εύκολη ενσωμάτωση νέων εντολών ή και την τροποποίηση των υφιστάμενων, κάτι το οποίο χρειάστηκε αρκετές φορές στη φάση της ανάπτυξης, χωρίς ωστόσο να απαιτούνται δομικές αλλαγές στον πυρήνα του διερμηνευτή, ενισχύοντας έτσι τη συντηρησιμότητα και την επεκτασιμότητα του.

Για την αποτελεσματική διαχείριση της ροής εισερχόμενων δεδομένων, υλοποιήθηκε μονάδα κυκλικής μνήμης (circular buffer), η οποία διασυνδέθηκε με τον DMA controller του μικροελεγκτή. Η συμβατική προσέγγιση απαιτούσε την άμεση επεξεργασία κάθε byte κατά την άφιξή του στον δίαυλο UART, ώστε να ελευθερωθεί ο καταχωρητής UART_RX και να συνεχιστεί η λήψη, αξιοποιώντας κατά κανόνα τεχνικές polling ή hardware interrupts. Αντίθετα, η νέα προσέγγιση επέτρεψε την πλήρη αποσύζευξη μεταξύ λήψης και επεξεργασίας δεδομένων. Ο ελεγκτής επικεντρώνεται στην παραγωγή των VGA σημάτων και επεξεργάζεται τα εισερχόμενα δεδομένα όταν είναι διαθέσιμος, χωρίς να υπάρχει κίνδυνος απώλειας πληροφορίας. Επιπλέον, η ταχύτητα μετάδοσης δεν είναι πλέον αυστηρά προκαθορισμένη και «κλειδωμένη» ώστε να διατηρείται μια εύθραυστη ισορροπία μεταξύ λήψης και επεξεργασίας. Επομένως το σύστημα μπορεί να προσαρμοστεί στις κατά περίπτωση απαιτήσεις, χωρίς να απαιτείται κάποιος επανασχεδιασμός.

Στο πλαίσιο της σχεδίασης του συστήματος, υιοθετήθηκε μια event-driven αρχιτεκτονική βασισμένη στο πρότυπο Reactor. Η προσέγγιση αυτή επέτρεψε στο σύστημα να ανταποκρίνεται σε εξωτερικά γεγονότα — όπως η άφιξη δεδομένων μέσω UART ή η ολοκλήρωση μιας μεταφοράς DMA — χωρίς να απαιτείται συνεχής ενεργή παρακολούθηση από τον επεξεργαστή. Αυτό οδήγησε σε σημαντική μείωση του υπολογιστικού φορτίου και εξασφάλισε ότι κρίσιμες λειτουργίες, όπως η παραγωγή VGA σημάτων, δεν διαταράσσονται.

Η event-driven προσέγγιση προσφέρει υψηλό βαθμό ευελιξίας και επεκτασιμότητας, καθώς το σύστημα μπορεί να προσαρμόζεται δυναμικά σε διαφορετικές συνθήκες

λειτουργίας και να ενσωματώνει νέες πηγές συμβάντων χωρίς να απαιτείται ανασχεδιασμός της βασικής λογικής.

Η υιοθέτηση του προτύπου Reactor δεν αποτελεί απλώς τεχνική επιλογή, αλλά στρατηγική αρχιτεκτονική απόφαση, η οποία επιτρέπει στο σύστημα να λειτουργεί με πραγματικό ασύγχρονο χαρακτήρα, ανταποκρινόμενο αποδοτικά σε κάθε είδους γεγονός. Ο τρόπος με τον οποίο εφαρμόστηκε το πρότυπο Reactor προσομοιάζει τη λειτουργία ενός συνεργατικού συστήματος πραγματικού χρόνου (cooperative RTOS), όπου κάθε μονάδα διαχείρισης συμβάντος εκτελείται όταν είναι απαραίτητο και παραχωρεί τον έλεγχο μόλις ολοκληρώσει την εργασία της, χωρίς την ανάγκη για προεκχωρημένο χρονοπρογραμματισμό.

Σε αντίθεση με τη σύγχρονη τάση για διαρκή αναζήτηση υψηλότερης υπολογιστικής ισχύος με σκοπό την επίλυση σύνθετων προβλημάτων, η εργασία αποδεικνύει ότι ο σωστός σχεδιασμός και ο υψηλής ποιότητας προγραμματισμός, συχνά αποτελούν τους σημαντικότερους παράγοντες επιτυχίας.

8.1 Μελλοντικές επεκτάσεις

Μια σημαντική μελλοντική επέκταση του συστήματος αφορά την υποστήριξη sprites, δηλαδή ανεξάρτητων γραφικών αντικειμένων που μπορούν να κινούνται, να επικαλύπτονται ή να αλληλοεπιδρούν με το υπόλοιπο περιεχόμενο της οθόνης. Η ενσωμάτωση sprites θα επιτρέψει την αξιοποίηση του ελεγκτή για την ανάπτυξη παιχνιδιών σε μικρές αναπτυξιακές πλατφόρμες όπως το Arduino.

Μια δεύτερη κατεύθυνση αφορά στη διασύνδεση του ελεγκτή με εξωτερική μνήμη RAM. Η υφιστάμενη υλοποίηση μπορεί εύκολα να τροποποιηθεί ώστε τα δεδομένα εξόδου να αποστέλλονται στην εξωτερική μνήμη RAM και εκείνη με την σειρά της να συνδέεται με τους DAC. Με αυτή την προσέγγιση είναι δυνατό να αυξηθεί σημαντικά τόσο η οριζόντια ανάλυση όσο το πλήθος των χρωμάτων ανά pixel.

Συμπληρωματικά, μπορεί να προστεθεί ως επιπλέον αποθηκευτικό μέσο μια SD card. Έτσι φωτογραφίες που αυτή τη στιγμή είναι «hard coded» στη μνήμη flash του μικροελεγκτή, μπορούν να αποθηκεύονται εξωτερικά και να εισάγονται εύκολα χωρίς να απαιτείται επαναπρογραμματισμός.

Βιβλιογραφία

[1] D. J. Peddie, Famous Graphics Chips: IBM's VGA, IEEE Computer Society, 2019.

[2] M. P. A. T. ENGINEERS, «Composite Analog Video Signal NTSC for Studio Applications,» *MOTION PICTURE AND TELEVISION ENGINEERS,* 2004.

[3] Noor A. Ibraheem, Mokhtar M. Hasan, Rafiqul Z. Khan, Pramod K. Mishra, «RGB Color Space,» σε *Understanding Color Models: A Review*, 2012.

[4] V. E. S. Association, VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT), Video Electronics Standards Association, 2013.

[5] James Bryant, Walt Kester, «Data Converter Architectures,» σε *Data Conversion Handbook*, Elsevier, Newnes, 2005, pp. 147-171.

[6] IBM, «RGBI Color Space,» σε *IBM Color Graphics Monitor Adapter*, 1981, pp. 6-7.

[7] Microchip, «PIC18F47K42 Data Sheet,» Microchip, [Ηλεκτρονικό]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/PIC18%28L%29F26-27-45-46-47-55-56-57K42-Data-Sheet-40001919G.pdf.

[8] Fangqin Ying, Xiaoqing Feng, «Design and Implementation of VGA Controller Using FPGA,» 2012.

[9] J. G. Ganssle, A Guide to Debouncing, The Ganssle Group, 2008.

[10] wikipedia, «Computer Terminal,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Computer_terminal.

[11] ECMA-48, «Control Functions for Coded Character Sets,» *Standardizing Information and Communication Systems,* 1991.

[12] Backus J, Bauer F, Green J, Katz C, McCarthy J, Naur P, Perlis A, Rutishauser H, Samelson K, Vauquois, «Revised report on the ALGOrithmic Language ALGOL 60,» *The Computer Journal,* τόμ. 5, αρ. 4, pp. 349-367, 1963.

[13] A. Tornhill, «State Pattern,» σε *Patterns in C, Patterns, Idioms and Design Principles*, 2012.

[14] Jonathan Carlgren, Oskarsson Per William, *State Machine Model-To-Code Transformation In C,* Uppsala/Visby: Uppsala Universiter, 2023.

[15] J. E. Bresenham, «Algorithm for computer control of a digital plotter,» *IBM Systems Journal ,* τόμ. 4, αρ. 1, 1965.

[16] J. E. Bresenham, «A Linear Algorithm for Incremental Digital Display of Circular Arcs,» *Comm. ACM,* τόμ. 20, αρ. 2, pp. 100-106, 1977.

[17] D. C. Schmidt, «Reactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events,» Washington University - Department of Computer Science, St. Louis.

Παράρτημα Α – Bill Of Materials

Part	Value
Resistor 2R	600 Ohm
C2	0.1uF
C3	0.1uF
C4	0.1uF
C5	20pF
C6	20pF
C9	0.1uF
COLOR_MUX	74AC257D
D1	Shotcky Diode
D2- 5.3V	Zener Diode 5.3V
MCU	PIC18F47K42
INTENSITY_MUX	74AC257D
LED1	Red SMD Led
OSC. SIGNALS	Connector
PPTC	PPTC-150mA
ICSP	Connector
Q1	Crystal 14.3182Mhz
Resistor R1	300 Ohm
Resistor R2	10k
Resistor R3	10k
Resistor R5	10k
S1	MOMENTARY-SWITCH-SPST-SMD-6.0X3.5MM
S3	MOMENTARY-SWITCH-SPST-SMD-6.0X3.5MM
X1	Connector
X2 - DSUB	VGA Connector - DSUB-15
X5 - USB	MINI-USB-32005-201

Παράρτημα Β – Κώδικας VGA Controller

Ο πλήρης κώδικας βρίσκεται στο αποθετήριο Github στη διεύθυνση https://github.com/pargyropoulos/VgaController και είναι ελεύθερα διαθέσιμος.

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.