TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Siim Salonen, 221912IASM

FROM 8-BIT TO HDMI: ENHANCING A Z80-BASED COMPUTER USING FPGA TECHNOLOGY

Master's Thesis

Supervisor: Tara Ghasempouri, PhD

Co-Supervisor: Veiko Rütter

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Siim Salonen, 221912IASM

8-BITIST HDMI-NI: Z80-PÕHISE ARVUTI TÄIUSTAMINE FPGA TEHNOLOOGIA ABIL

Magistritöö

Juhendaja: Tara Ghasempouri, PhD

Kaasjuhendaja: Veiko Rütter

Tallinn 2025

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Siim Salonen

16.03.2025

Abstract

This thesis focuses on restoring and enhancing a vintage Z80-based computer, specifically a Soviet clone of the Sinclair ZX Spectrum 48K (Leningrad model). The goal is to develop an FPGA-based add-on that integrates modern features while maintaining compatibility with the original system.

The main enhancement is the addition of HDMI output, as contemporary displays lack the analog inputs required by the original hardware. The FPGA captures video memory data directly from the Z80's memory bus and generates an HDMI signal with optional pixel-scaling algorithms, such as hq2x and Super2xSaI, to improve the visual output. Another key feature is the emulation of the Yamaha AY-3-8912 sound chip, enabling improved audio output via HDMI.

Additionally, the project explores memory expansion beyond the ZX Spectrum 128K model, implementing a memory paging system to support configurations up to 512KB. This is achieved by integrating SRAM controlled via the FPGA, ensuring compatibility with existing software. Further improvements include an SD card interface for loading software snapshots and potential LAN connectivity for debugging and remote file transfers.

The thesis covers the entire development process, including the restoration of the original hardware, schematic and PCB design of the FPGA-based expansion board, and software development for the FPGA. The project demonstrates how retrocomputing can be enhanced using modern digital design techniques, balancing preservation with innovation.

The results of this work will contribute to both hardware preservation efforts and the broader retrocomputing community, showcasing how FPGA technology can extend the lifespan and usability of vintage computing platforms.

Annotatsioon

Antud magistritöö keskendub Z80-põhise arvuti taastamisele ja täiustamisele, eelkõige Sinclair ZX Spectrum 48K klooni ajakohastamisele. Projekti eesmärk on arendada FPGA-põhine laiendusmoodul, mis lisab tänapäevaseid funktsioone, säilitades samal ajal ühilduvuse algse süsteemiga.

Peamine täiustus on HDMI-väljundi lisamine, kuna tänapäevased ekraanid ei toeta enam analoogühendusi. FPGA loeb videomälu andmeid otse Z80 andmesiinilt ja genereerib HDMI-signaali, kasutades kvaliteedi tõstmiseks skaleerimisalgoritme, nagu hq2x ja Super2xSaI, et parandada pildi kvaliteeti. Teine oluline täiendus on Yamaha AY-3-8912 helikiibi emuleerimine, võimaldades paremat heli edastamist HDMI kaudu.

Lisaks uuritakse mälu laiendamise võimalusi algsest 48kB'st 512kB'ni. Selle saavutamiseks kasutatakse FPGA kaudu juhitavat SRAM-i, tagades ühilduvuse olemasoleva tarkvaraga. Edasised parandused hõlmavad SD-kaardi tuge tarkvara laadimiseks ning võimalikku LAN-ühendust.

Töös käsitletakse kogu arendusprotsessi, sealhulgas algse riistvara taastamist, FPGA-laiendusplaadi skeemi ja PCB disaini ning FPGA tarkvaraarendust. Projekt demonstreerib, kuidas retroarvuteid saab täiustada tänapäevaste digitaaltehnoloogiatega, ühendades säilitamise ja innovatsiooni.

Selle töö tulemused aitavad kaasa nii riistvara säilitamisele kui ka laiemale retroarvutite kogukonnale, näidates, kuidas FPGA-tehnoloogia saab pikendada vanade arvutite eluiga ja kasutatavust.

List of abbreviations and terms

DFF	D-type Flip Flop
DRAM	Dynamic Random-Access Memory
DVI	Digital Visual Interface
EBR	Embedded Block RAM
EDID	Extended Display Identification Data
EoL	End-of-Life
FFC	Flexible Flat Cable
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
IC	Integrated Circuit
IO	Input-Output
I2S	Inter-Integrated Circuit Sound
LC	Logic Cell
LDO	Low Dropout
LSB	Least Significant Bit
LUT	Look-up Table
LVCMOS	Low-Voltage Complementary Metal Oxide Semiconductor
LVDS	Low-Voltage Differential Signaling
MMC	MultiMediaCard
MSB	Most Significant Bit
NMI	Non-Maskable Interrupt
PC	Program Counter
PCB	Printer Circuit Board

PCM	Pulse Code Modulation
PLB	Programmable Logic Blocks
PLL	Phase Locked Loops
PSG	Programmable Sound Generator
RGB	R = red, G = green, B = blue
SD	Secure Digital
SOT	Small Outline Transistor
SP	Stack Pointer
SPI	Serial Peripheral Interface
S/PDIF	Sony/Philips Digital Interface
TERC4	TMDS Error Reduction Coding – 4 bits
TMDS	Transition-Minimized Differential Signaling
TTL	Transistor–transistor Logic
ULA	Uncommitted Logic Array
YCbCr	Y = luminance, Cb = Chroma component blue, Cr = Chroma component red

Contents

List of Figures

List of Tables

1 Introduction

Retrocomputing has gained significant popularity in recent years, with enthusiasts restoring and upgrading vintage computers to preserve digital history and explore the evolution of computing technology. Among these, the Sinclair ZX Spectrum, as shown in Figure 1 [1], holds a special place as one of the most iconic 8-bit computers of the 1980s. Various official and unofficial clones of this machine were developed worldwide, including in the Soviet Union, where the Leningrad model emerged as a most well-known version.

Figure 1. Sinclair 48K ZX Spectrum

During the late 1980s, Western personal computers were largely unavailable in the Soviet Union and other Eastern Bloc due to import restrictions and high costs. As a result, enthusiasts and engineers reverse-engineered the ZX Spectrum, leading to numerous unofficial clones. More than 120 distinct variants have been documented globally, with over 50 originating in the former Soviet Union alone [2] [3]. These machines were built using readily available Soviet or Eastern Bloc components, including Z80 microprocessor clones like the T34VM1 or the East German U880, which sometimes led to subtle timing differences compared to the original Sinclair hardware.

One of the most successful was the Leningrad series, named after the city where it was developed (now St. Petersburg), evolving through multiple revisions (Leningrad‑1, Leningrad‑2, etc.). Unlike the original ZX Spectrum, which relied on a custom ULA (Uncommitted Logic Array) chip, the Leningrad clone replaced this with discrete logic chips, making it easier to build with locally available components. The design was widely distributed in magazines and technical manuals, leading to home-assembled versions across the Soviet Union. Many Soviet clones introduced enhancements over the original Spectrum, including discrete‑logic video generators, TR‑DOS floppy‑disk interfaces, and expanded RAM.

This project stems from a personal discovery: an author's first computer, Leningrad-2 ZX Spectrum clone with 5¼-inch floppy drive running TR-DOS, found in the attic in a non-working condition. The goal is not only to restore it to full functionality but also to enhance its capabilities using modern FPGA technology. The limitations of original hardware, such as analog video output, inaccurate video timing signals, low RAM capacity and lack of modern storage solutions, make it challenging to use with contemporary displays and peripherals. This motivates the development of an FPGA-based expansion board, which will bring new features while maintaining the authenticity of the system.

1.1 Objectives

The primary objective of this thesis is to design and implement an FPGA‑based expansion module that plugs into the Spectrum’s original floppy‑drive connector - requiring minimal, if any, modifications to the host computer - while preserving full compatibility with its unaltered hardware. This enhances the original Leningrad ZX Spectrum clone by adding:

- HDMI output with optional pixel-scaling algorithms to ensure compatibility with modern displays.
- Expanded memory, supporting configurations beyond the original 128K model, up to 512KB.
- Yamaha AY-3-8912 sound chip emulation, enabling improved audio output via HDMI.

- SD card support for loading software snapshots.
- LAN capabilities for debugging and potential file transfer.

By achieving these enhancements, the project aims to demonstrate how FPGA technology can modernize legacy computing systems while preserving their original architecture.

1.2 Research Challenges

The project presents several technical challenges, including:

- Repairing the Leningrad ZX Spectrum clone to working state.
- Generating an HDMI signal from a system that originally used composite video, while maintaining low-latency output.
- Memory paging and expansion, ensuring compatibility with existing ZX Spectrum software.
- Efficient FPGA resource management, as the selected Lattice iCE40 FPGA has limited hardware capacity.
- Developing a stable interface between the FPGA and the Z80-based system without interfering with normal operation.

1.3 Overview of Existing Video‑Output Add‑Ons

A number of hardware projects have been developed to provide modern video outputs for the Sinclair ZX Spectrum. These solutions vary significantly in their technological approach, output capabilities, and features. Most published add‑on boards for the Sinclair ZX Spectrum target the original 48‑pin edge connector. In contrast, this work employs a custom MPN‑44 connector. Table 1 summarizes the most notable solutions and highlights their key limitations relative to a pure FPGA‑based HDMI add‑on. Price information in this table is omitted: some projects are purely open‑source hobby designs, while others are produced in such small volumes that their retail cost often bears little relation to the actual bill of materials.

Table 1. Comparison of ZX Spectrum Modern Video Output Solutions

Solution	Form Factor	Output Interf.	Hardware	Scaling	Sound over HDMI	Memory Expansion	SD Card	Video Generation
ZX-VGA-Joy [4]	Add-on module	VGA	CPLD? + joystick interface	-	No	No	No	Bus Snooping / Shadow FB
ZX-VGA [5]	Add-on module	VGA	Xilinx XC95144XL	-	No	No	No	Bus Snooping / Shadow FB
ZX-PiE [6]	Add-on module	HDMI	Raspberry Pi Zero, Altera MAX II CPLD	x2	No	No	Yes	Bus Snooping / Shadow FB
ZX-HD [7]	Add-on module	HDMI	Raspberry Pi Zero, Xilinx XC9572XL	x2	No	No	Yes	Bus Snooping / Shadow FB
ZX-Uno [8]	Full-board replacement	VGA	Xilinx Spartan XC6SLX9-2TQG144C	-	No	512 kB	Yes	Full Reimplementation
MiSTer Spectrum Core [9]	Full-board replacement	HDMI / VGA	Altera Cyclone V SoC	x2, hq2x, etc.	Yes	configurable	Yes	Full Reimplementation
ZX Spectrum Next [10]	Full-board replacement	HDMI / VGA	Xilinx Artix-7 XC7A15T	x2	Yes	1 MB	Yes	Full Reimplementation
Current work	Add-on module	HDMI	Lattice iCE40 HX4K-TQ144	x2, hq2x	Yes	512 kB	Yes	Bus Snooping / Shadow FB

2 Sinclair ZX Spectrum Architecture

The Sinclair ZX Spectrum's architecture represents a unique balance of cost-effective design and technical innovation, with key engineering decisions that influenced both its capabilities and limitations. Understanding these elements is crucial when examining compatibility challenges in clones like the Leningrad, which replaced the Spectrum's custom ULA chip with discrete logic components.

2.1 Core Architecture of the Sinclair ZX Spectrum

The original ZX Spectrum 48K motherboard is shown in Figure 2 [11]. It features:

- CPU: Zilog Z80A at 3.5 MHz
- RAM: 48 KB, split between lower and upper sections
- ROM: 16 KB, containing Sinclair BASIC and core routines
- ULA: A custom IC (designed by Richard Altwasser) integrated critical functions into a single chip:
    - Memory management: The ULA shared RAM access between the CPU (Z80) and video display, dynamically halting the CPU during screen refresh cycles to avoid contention [12]. This created "contended memory" timing quirks that software had to accommodate.
    - Video Signal Generation: Producing composite video (and RF-modulated output) with a 256×192 pixel, organized into 32×24 character blocks. Each 8×8 block used 1 byte for foreground/background colors (3 bits each), brightness (1 bit), and flash (1 bit). This caused "color clash," where pixels in a block couldn’t have independent hues [13].
    - I/O Handling: Managing keyboard scanning and tape operations
    - Sound: Driving a simple 1-bit beeper

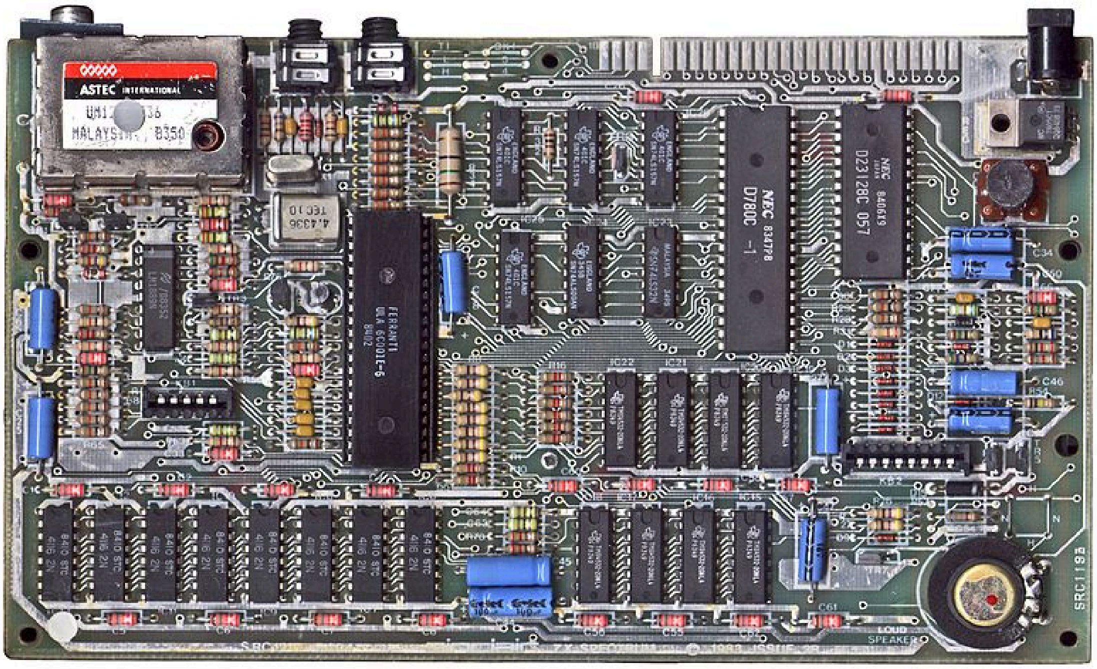


Figure 2. Sinclair ZX Spectrum 48K motherboard, issue 3B

2.2 Leningrad ZX Spectrum Clone Architecture

The Leningrad clone follows the overall design of the ZX Spectrum 48K but replaces the ULA with a combination of standard TTL logic chips. Key differences include:

- Video Generation: Instead of a ULA, the Leningrad clone utilizes a combination of 74HC298N and 74HC257N input multiplexers, 74HC86N NOR gates and 74HC74N D-type flip-flops to generate the video timing and composite signal [14].
- Memory Contention and Timing: In the original Spectrum, the ULA enforces precise memory contention, pausing the CPU during video fetches. In the Leningrad clone, discrete logic circuits based on 74HC138 (3-to-8 line decoders) and 74HC161 (synchronous binary counters) manage timing.
- In the original Spectrum, the ULA enforces precise memory contention, pausing the CPU during video fetches. In the Leningrad clone, discrete logic circuits based on 74HC138N and 74HC4520N synchronous binary counters, 74HC74N D-type flip-flops and 74HC series logic gates manage timing. These

components, while functionally similar, introduce slight variations in timing, leading to potential issues with software that relies on the ULA’s exact behavior.

- I/O and Keyboard Scanning: The original ULA also handled keyboard matrix scanning. In the Leningrad clone, this is implemented using 74HC257N input multiplexers, which can result in minor differences in key recognition and debounce behavior.

The circuit boards feature an empty breadboard area, which is seen in upper-right of Figure 3, that can be used for future enhancements or modifications like adding extra memory, Yamaha AY-3-8912 sound chip or custom I/O interfaces. This area allows hobbyists or engineers to add custom features without major redesigns.

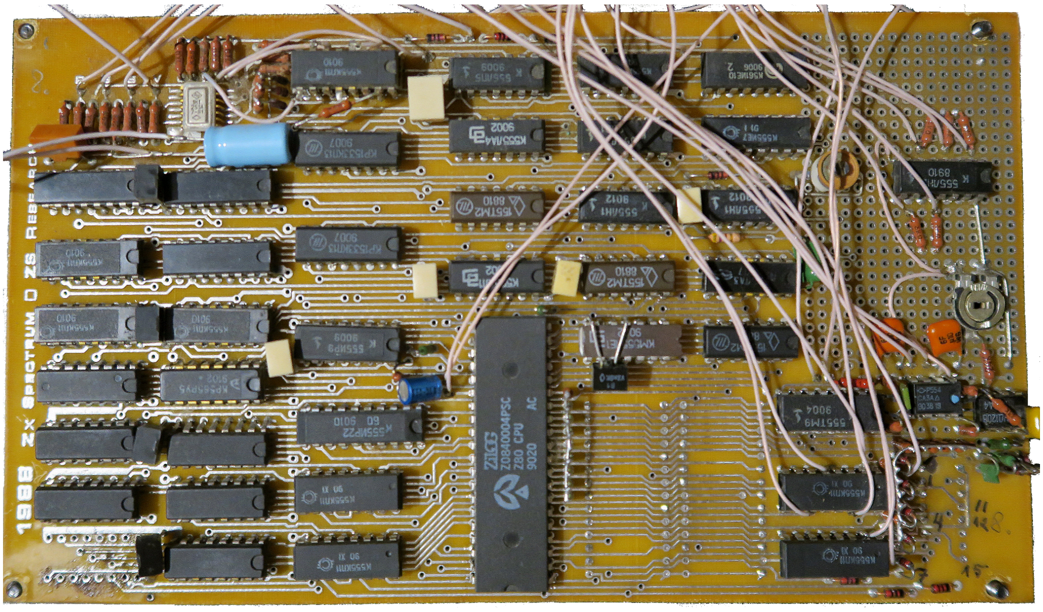


Figure 3. Leningrad 48K motherboard

2.3 TR-DOS and the 5.25-Inch Floppy Interface

A specific ZX Spectrum was later modified to allow the attachment of a 5.25-inch floppy drive. For this, an MPH44-1 socket was added to the side of the case and bodge-wired to the Z80 CPU's pins and ROM was moved to the floppy drive unit. Additionally RDROM and IOREQ lines were cut and both ends wired to the side connector.

The floppy drive unit utilizes the КР1818ВГ93, a soviet clone of the Western Digital WD1793 floppy disk controller, and operates under TR-DOS (Technology Research Disk Operating System). TR-DOS was originally developed by Technology Research Ltd. in the UK to work in conjunction with the Beta Disk Interface. This interface became widely adopted throughout Eastern Europe and the Soviet Union, particularly among various Spectrum clones, where it served as the de facto standard for disk-based software.

TR-DOS allows structured file access on floppy disks using both machine-code routines and extended BASIC commands. It supports common operations such as loading and saving programs, managing disk files, and launching software directly from disk. Compatibility with TR-DOS ensures access to a vast library of historical ZX Spectrum software distributed on floppy disks.

To enter TR-DOS, a special command is used: RANDOMIZE USR 15616. This attempts to execute machine code from address 0x3D00, which in the built-in ROM normally contains font data. However, when a read occurs in the range 0x3D00–0x3DFF and both the /RD and /M1 signals are low (which happens during instruction fetch cycle), a special signal called ROMCS is generated. This signal disables the built-in ROM and enables the external TR-DOS bootstrap ROM in its place. While ROMCS is active, the following I/O ports can be used to communicate with the floppy drive: #1F, #3F, #5F, #7F, and #FF [15].

Understanding the architecture and behavior of the Beta Disk interface and TR-DOS is critical for ensuring compatibility with existing disk-based software and hardware configurations. Since these systems rely on specific memory addresses and I/O ports, the new expansion module must avoid conflicting with TR-DOS-reserved ports and must be capable of launching and coexisting with TR-DOS. Additionally, the ROM must properly handle the ROMCS mechanism to correctly trigger TR-DOS booting when requested. By replicating the original boot and access behavior, the FPGA-based system can ensure seamless coexistence with TR-DOS, allowing users to load and run disk-based software exactly as on the original hardware.

2.4 Comparison: Original ZX Spectrum vs. Leningrad Clone

The ULA in the original ZX Spectrum tightly controlled video timing to generate composite video (later modulated into RF for TV use). On CRT monitors and televisions—which were the norm in the 1980s—minor timing variations were largely imperceptible, as these displays were designed to work with the analog nature of the composite signal. The Spectrum's video was typically in YUV format (via a UHF modulator), which could suffer from color bleeding and lower sharpness.

The Leningrad clone, using discrete logic chips, reproduced video timing through circuits that sometimes resulted in slight imperfections. While these differences had little impact on CRT displays due to their forgiving analog nature, they can cause noticeable problems when interfacing with modern digital hardware that demands precise timing. However, the Leningrad clone offers a key advantage: it produces an RGB signal directly. This results in better picture quality, with improved sharpness and color fidelity compared to the Spectrum's YUV output. Table 2 lists key differences of ULA vs discrete TTL logic.

Table 2. Discrete Logic vs. ULA

Feature	**ZX Spectrum (ULA)**	**Leningrad (TTL)**
Memory Contention	CPU halted only during ULA screen fetches [12]	Delays added to all M1 cycles, altering instruction timing [14]
Floating Bus Behavior	Present (used for raster effects in games) [16]	Absent, breaking titles like Arkanoid [17]
Interrupt Timing	Frame interrupt 64 lines before visible screen	Interrupt at first line after visible screen [17]
Overall Compatibility	Optimized for exact hardware behavior	High compatibility with occasional edge-case differences

Impact on software compatibility:

- Timing-sensitive code: Multicolor demos and cycle-exact routines fail due to altered memory access patterns. 8x1 graphics engines like Bifrost, Nirvana and ZXodus fail to display multicolor [18].
- Raster effects: Games relying on the floating bus (e.g., Cobra, Arkanoid 1) freeze or glitch [17].
- Sprite rendering: Programs using "beam racing" techniques exhibit flickering due to shifted interrupt timing.

Discrete logic chips are generally easier to diagnose, replace, and repair compared to a custom ULA chip. This modularity simplifies maintenance and upgrades, making the system more accessible for restoration projects and experimental modifications.

3 Hardware Design

This chapter describes the hardware design of the enhanced ZX Spectrum system, detailing the core components, their interconnections, and the rationale behind the selection of parts.

The hardware design integrates legacy ZX Spectrum signals with modern digital processing. The system is built around a Lattice HX4K FPGA - selected for its compatibility with open-source tools (Yosys, nextpnr-ice40 [19]) and its TQFP package, which simplifies assembly and debugging. Critical functions include HDMI signal generation (handled via PTN3366), power supply regulation to generate 3.3V and 1.2V rails, level shifting of 5V signals, memory expansion using high-speed SRAM, and an SD-card interface for loading spectrum image files.

The final device connects directly to the computer via the original floppy drive slot, effectively replacing the internal floppy controller. On the opposite side, a female connector allows the original floppy drive to be reconnected, forming a bridge between the ZX Spectrum and its floppy drive. Since the male connector is PCB-mounted and the female connector is edge-mounted, a second small PCB is used to raise the female connector to the correct height. The two PCBs are linked via a flat flexible cable (FFC). The complete assembly is shown in Figure 4.

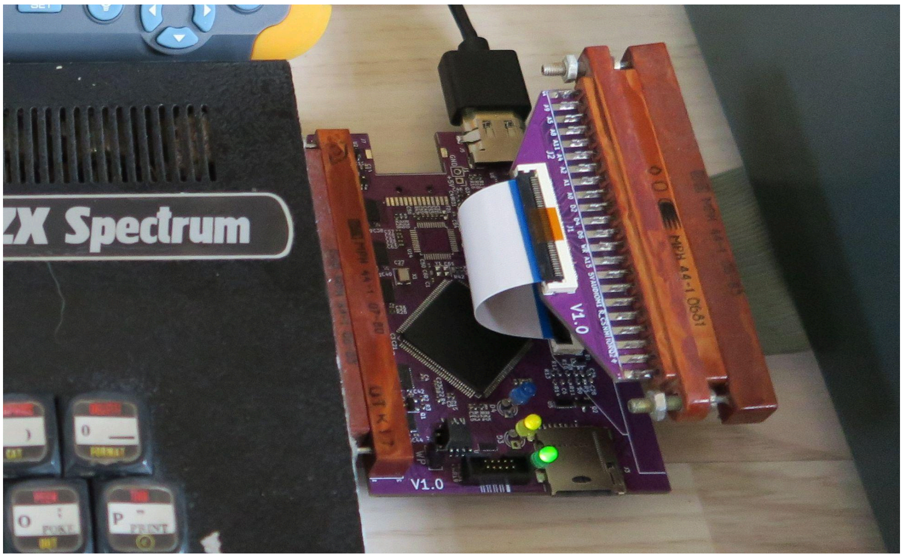


Figure 4. Add-on with floppy drive connected

3.1 Power Supply

A stable power supply is essential for ensuring reliable operation of the FPGA and other high-speed digital components. The legacy ZX Spectrum provides a 5V input, which must be converted for modern circuitry. A stable power supply minimizes the risk of digital errors, which is especially important in applications where high-speed signal integrity is essential.

3.1.1 3V Supply

The 3.3V rail is generated using the XC6210B332MR, a low dropout regulator chosen for its excellent transient response, low noise, and high efficiency - qualities essential for powering high-speed FPGAs and digital ICs [20]. It provides an output voltage with an accuracy of ±2% and has a typical quiescent current (Iq) of 35μA. With a standard 1μF ceramic capacitor at its output, the regulator delivers a stable and low-noise 3.3V supply that is crucial for the proper operation of the FPGA and other digital circuits. Additionally, the XC6210B332MR is available in a standard SOT23-5 package, making it easy to handle and integrate into the compact PCB layout required by the design. This regulator is also low cost and it was readily available from local suppliers.

3.1.2 1.2V Supply

The 1.2V supply for the HX4K FPGA core is derived from the previously regulated 3.3V rail using the TLV75512PDBV [21]. This regulator converts 3.3V to a stable 1.2V output with high precision, offering an output accuracy of ±1%. It has a low quiescent current of 25μA, ensuring minimal power loss and efficient operation. The TLV75512PDBV is designed for optimal performance even in compact applications, making it an excellent choice for powering the FPGA's core logic reliably. The regulator also comes in a standard SOT23-5 and is also cheap and widely available.

3.2 FPGA

The Lattice HX4K FPGA was chosen because it is cost effective, available in a TQFP package (simplifying assembly compared to BGA devices), and benefits from open-source tool support. The HX4K-TQ144 variant was selected over HX8K models because it offers a higher number of EBR and pins in a TQFP package, whereas HX8K devices are only available in BGA packages and are significantly more expensive.

The specific resources available on the iCE40 HX4K variant are key constraints for this project:

- Logic Cells (LCs): According to the datasheet the HX4K provides 3520 LCs. Each LC is the fundamental building block for logic implementation and typically consists of a 4-input Look-Up Table (LUT4), a D-style Flip-Flop (DFF), and dedicated carry logic for arithmetic operations. The LUT4 can implement any arbitrary 4-input combinational logic function or act as a small 16x1 ROM.
- Embedded Block RAM (EBR): The device contains a total of 80kb of EBR. This memory is physically organized into 20 independent 4Kbit blocks (referred to as RAM4K blocks). Each 4Kbit block is highly configurable and can be arranged in various aspect ratios (depth x width), such as 256x16, 512x8, 1024x4, or 2048x2. These EBR blocks support dual-port operation, allowing simultaneous access (one read and one write) from two independent ports.
- Other Resources: The HX4K includes 2 PLLs for frequency synthesis and clock phase/duty cycle adjustment. The number of available general-purpose I/O pins is 95 plus dedicated pins.

During the routing process using nextpnr-ice40, the tool reported 7680 LCs and 32 blocks of EBR. This initially led to the assumption that there was a bug in nextpnr-ice40. However, a community post clarified that the iCE40 HX4K device actually contains 7680 LUTs, as it shares the same die as the HX8K. Lattice disables half of these resources in their proprietary programming software, but the full capacity can be unlocked using the IceStorm open-source toolchain [22].

3.3 Level Shifting

All inputs and outputs between the legacy ZX Spectrum (operating at 5V) and the modern FPGA (operating at 3.3V) are interfaced using SN74LXC8T245 [23] and 74AXP4T245BQ [24] level shifters. A total of five such devices are deployed to ensure proper voltage matching, which is crucial for robust signal transmission and to protect the FPGA from overvoltage conditions.

3.4 HDMI Output

The PTN3366 integrated circuit is utilized to convert AC-biased signals to DC-biased TMDS signals for HDMI output. The rationale for selecting this component, as well as the implementation details, are discussed in Section 4.

3.5 SD-Card Interface and Protection

The SD-card socket is connected to the FPGA through an IP4251 filtering and protection IC. The IP4251 [25] provides effective filtering, transient voltage suppression, and surge protection between the SD card and the FPGA, ensuring reliable data transfer and safeguarding sensitive circuitry from electrical disturbances. The SD card is used to load ZX Spectrum snapshots in .sna and .z80 formats, which represent complete memory dumps, including both system memory contents and CPU register states, into the computer's memory.

3.6 Memory Expansion

To upgrade the ZX Spectrum clone’s limited memory, a 512 kB IS65WV5128EBLL SRAM was included in the design [26]. This expansion effectively takes the system to the performance level of a ZX Spectrum 128K model, while still leaving room for even larger configurations as seen in other variants (e.g., models with 256 kB or 512 kB of RAM).

The selected SRAM has an access time of approximately 45 µs. In typical operation, it takes about one clock cycle from the FPGA to establish the address and start a read cycle; then, after the address is established, the inherent access time of the SRAM (45µs) determines the delay before the good data appears on the bus. All together, the complete read cycle - address latching, waiting for the 45µs access time, and reading the data - can be accomplished within two pixel clock cycles after optimization using pipelined or asynchronous interfacing techniques.

This performance is necessary for the seamless integration of expanded memory into the vintage computing environment, enabling fast data retrieval and efficient memory paging. Additionally, the relatively moderate access time of the IS65WV5128EBLL enables more complex applications and higher performance software, which were not possible on the original low-memory 48K ZX Spectrum.

Overall, the memory expansion not only brings the system nearly to the level of a ZX Spectrum 128K mark and also provides a platform for further upgrades in RAM capacity, in line with the numerous ZX Spectrum variants with much more than 128kB of memory.

4 HDMI Output Implementation and Image Processing

HDMI is widely used to transmit uncompressed digital video and audio. At its core, HDMI uses TMDS for data transmission, which ensures high-quality signal integrity over relatively long cable runs. To encode pixel information with a low error rate, the overall signal is organized across four channels: three TMDS data channels (for red, green, and blue pixel data) and one TMDS clock channel. For a 640×480 at 60Hz video mode (commonly referred to as DMT0659 [27]), the pixel clock is approximately 25.175MHz while the TMDS clock operates at about 251.75MHz (10× the pixel clock) [28] and horizontal frequency is 31.469kHz [29] ensuring that each frame and line adheres to precise horizontal and vertical synchronization intervals. These timings are critical, especially when interfacing with digital displays that expect tightly regulated video signals.

Although HDMI and DVI both utilize TMDS signaling, HDMI offers several enhancements, including the encapsulation of I2S audio within data islands. Given that DVI is much easier to implement but lacks native audio support, this work will target HDMI to meet audio transmission requirements.

This chapter outlines the design and implementation of an HDMI output module, focusing on timing synchronization, signal encoding, and image enhancement algorithms tailored to the FPGA add-on architecture.

4.1 Signal Composition

The HDMI signal is composed of three main periods: the Active Video Period, the Control Period, and the Data Islands. An example of each period placement in a 640x480p video frame is shown in Figure 5 [30].

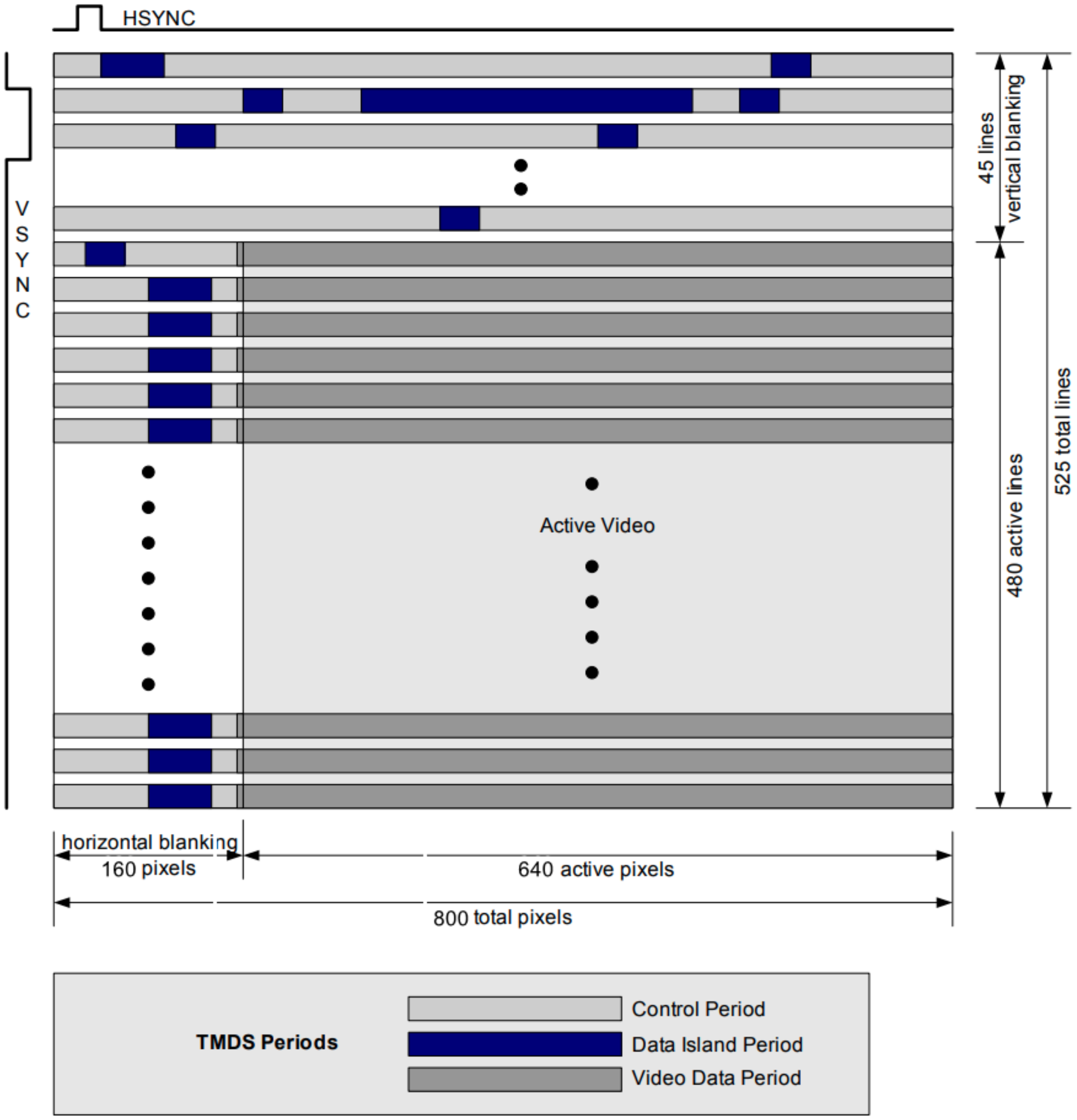
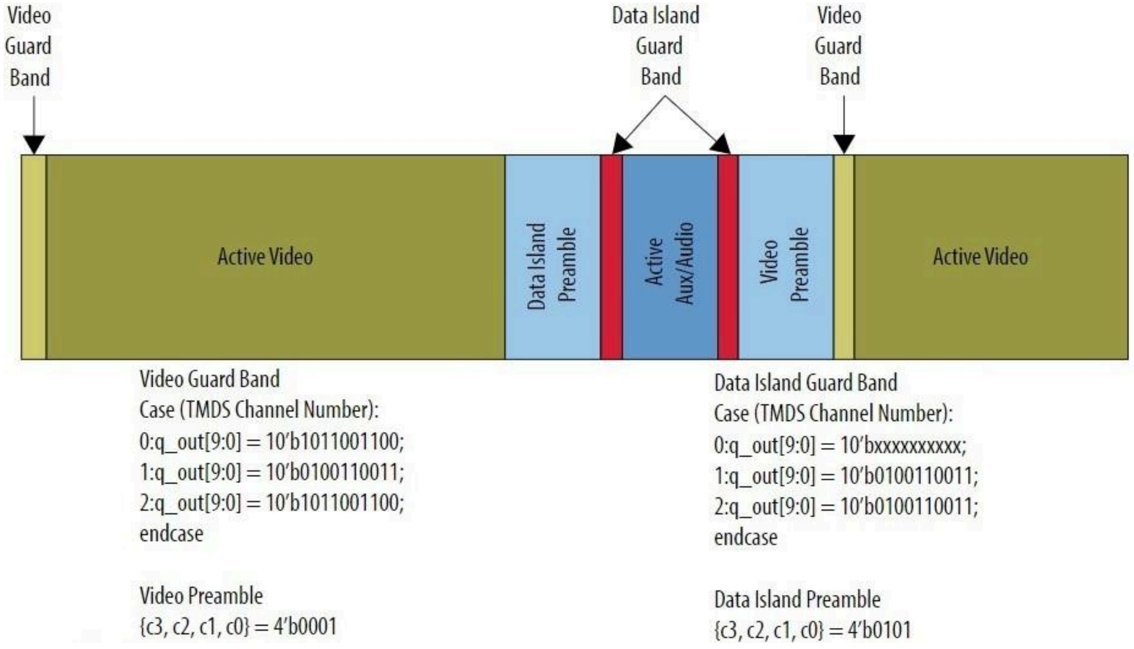


Figure 5. : TMDS periods in 640x480p video frame

The Active Video Period carries the visible image data, transmitting pixel values in RGB or YCbCr format. This phase aligns with the display resolution and refresh rate, delivering data line by line and frame by frame without compression. Each pixel is encoded using an 8b/10b scheme, where every 8-bit video data byte is mapped to a corresponding 10-bit TMDS symbol. This encoding minimizes signal transitions and ensures DC balance. In Figure 6 is shown a 10-bit fixed-pattern guard band, inserted between horizontal lines to mark the boundary between video and non-video intervals and to prevent false synchronization [31].



```
Video Guard Band
Case (TMDS Channel Number):
0:q_out[9:0] = 10'b1011001100;
1:q_out[9:0] = 10'b0100110011;
2:q_out[9:0] = 10'b1011001100;
endcase

Video Preamble
{c3, c2, c1, c0} = 4'b0001
```

```
Data Island Guard Band
Case (TMDS Channel Number):
0:q_out[9:0] = 10'bxxxxxxxxxx;
1:q_out[9:0] = 10'b0100110011;
2:q_out[9:0] = 10'b0100110011;
endcase

Data Island Preamble
{c3, c2, c1, c0} = 4'b0101
```

Figure 6. Guard Bands fixed pattern

The Control period is used when no video, audio, or auxiliary data needs to be transmitted. A Control Period is required between any two periods that are not Control Periods [30]. The Control Period occurs during the blanking intervals and carries synchronization signals including vsync, hsync, and two control bits (C0 and C1). These are encoded using a specialized Control Period Coding scheme, where the 2-bit control signal is translated into a predefined 10-bit TMDS word. This period also includes a preamble for line and frame synchronization. The limited number of control values allows for simple and reliable decoding on the receiver side.

Data Islands are used to transmit non-video auxiliary data such as audio samples, InfoFrames, and metadata. The data payload is encoded using TERC4 coding, which converts 4-bit nibbles into 10-bit symbols. This 4-to-10 bit mapping is to ensure high transition density for reliable clock recovery. A fixed 10-bit guard band is also inserted before and after each data island to separate it from the surrounding control or video data. TERC4 encoding is defined by a lookup table where each 4-bit input has a unique 10-bit output, designed to maintain transition properties and simplify decoding.

Figure 7 shows how different periods are distributed between TMDS channels [32].

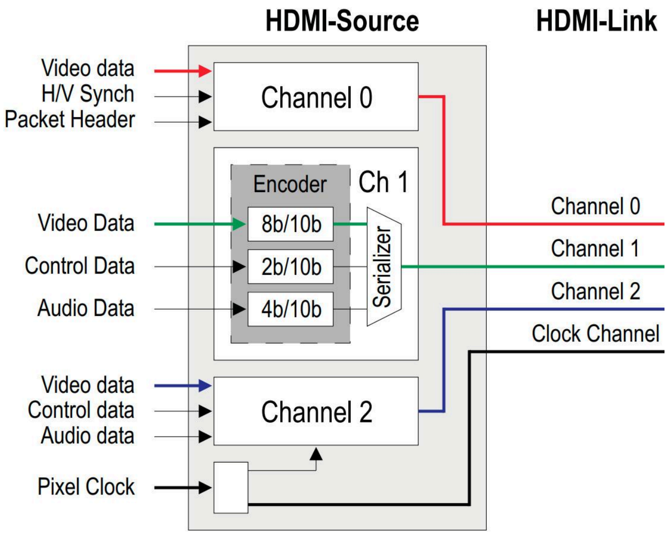


Figure 7. TMDS system

4.2 TMDS Algorithm

TMDS converts the original data into 10-bit symbols that are optimized for reliable transmission over differential pairs. This encoding achieves two essential objectives: minimizing signal transitions and maintaining DC balance. To achieve these, the TMDS algorithm performs two sequential steps:Transition Minimization to reduce the number of bit transitions in the encoded signal, which helps minimize electromagnetic interference (EMI) and DC Balancing to ensure an equal distribution of ones and zeros over time to prevent baseline wander and allow for accurate clock recovery.

4.2.1 Transition Minimization

Given an 8-bit input, the algorithm generates a 9-bit intermediate value by using a conditional XOR or XNOR operation across the bits. The process starts by copying the first bit as-is (q[0] = d[0]). For subsequent bits (i = 1 to 7), the algorithm compares the number of 1s and 0s already accumulated and applies either XOR or XNOR depending on which will result in fewer transitions [33] :

- If the number of 1's in the original byte is greater than 4 (more than half), XNOR is used: q[i] = q[i-1] XNOR d[i]
- Otherwise, XOR is used: q[i] = q[i-1] XOR d[i]

This decision is based on the statistical properties of the data sequence, ensuring that the resulting encoded stream contains fewer transitions compared to the original bit sequence. The additional ninth bit acts as a flag that indicates which encoding method (XOR vs. XNOR) was chosen for that particular data block.

4.2.2 DC Balancing

The second stage in the TMDS process is DC balancing. Even though transition minimization reduces high-frequency components, the encoded data stream can still exhibit an imbalance in the number of ones and zeros over time. DC balancing is applied to maintain a near-zero DC bias in the transmitted signal, which is essential for preserving signal integrity over differential pairs in high-speed interfaces.

After the 9-bit transition-minimized word is computed, a 10th bit is added to signal whether the output should be inverted to preserve balance. The encoder keeps a running disparity counter, tracking whether more 1's or 0's have been transmitted.

- If the running disparity becomes positive, the encoder may invert the current code to introduce more 0's.
- If negative, it may introduce more 1's.
- The 10th bit (q[9]) indicates whether inversion occurred.

This process ensures that the long-term average of the transmitted signal stays centered, avoiding any low-frequency voltage offsets.

The full TMDS video data encoding algorithm, including the control logic for both transition minimization and DC balancing, is illustrated in Figure 8 [30]. This flowchart provides a step-by-step visualization of the encoder's decision-making process for generating the final 10-bit symbol.

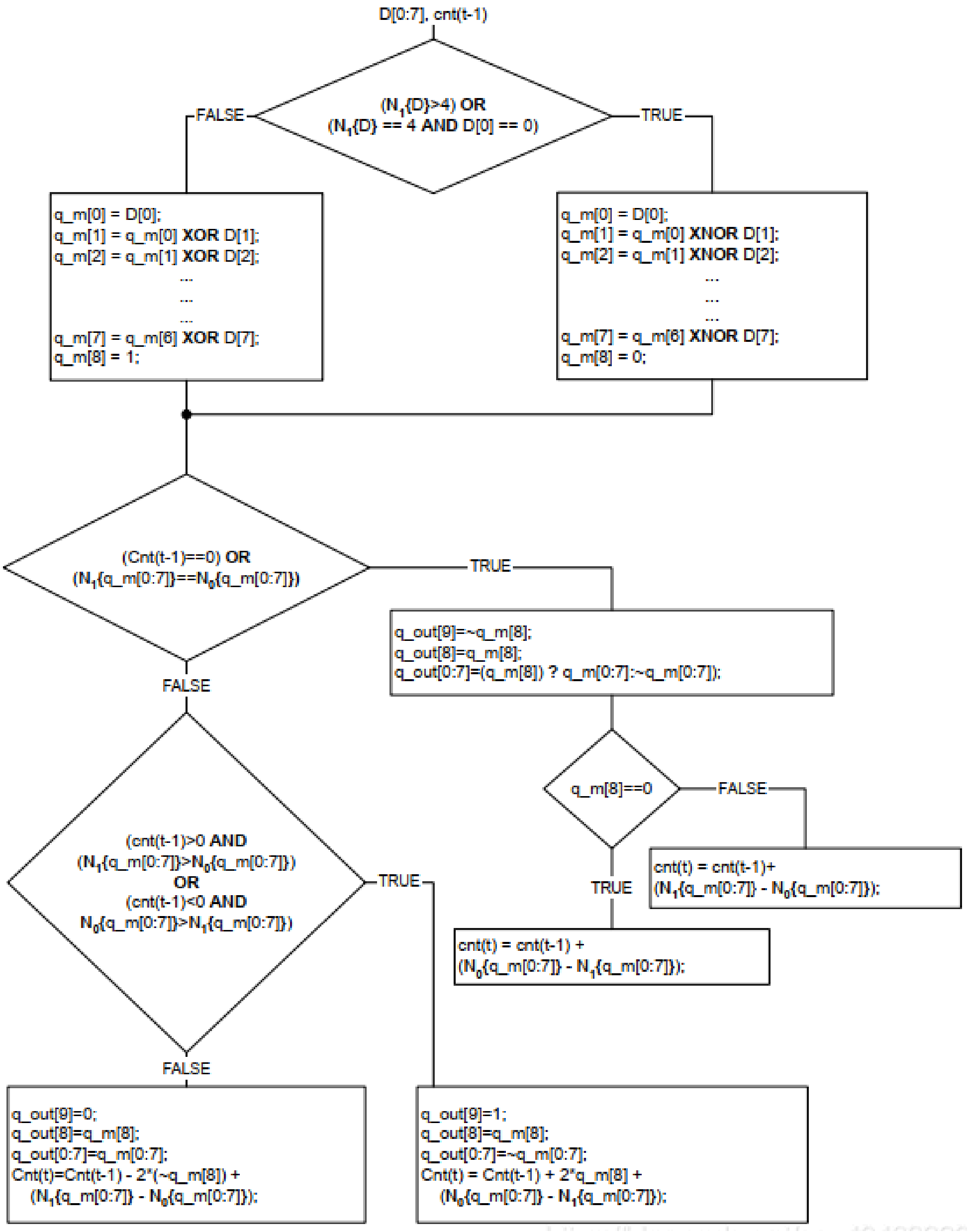

D[0:7], cnt(t-1)
(N_1{D}>4) OR
(N_1{D} == 4 AND D[0] == 0)
FALSE
TRUE
q_m[0] = D[0];
q_m[1] = q_m[0] XOR D[1];
q_m[2] = q_m[1] XOR D[2];
...
q_m[7] = q_m[6] XOR D[7];
q_m[8] = 1;
q_m[0] = D[0];
q_m[1] = q_m[0] XNOR D[1];
q_m[2] = q_m[1] XNOR D[2];
...
q_m[7] = q_m[6] XNOR D[7];
q_m[8] = 0;
(Cnt(t-1)==0) OR
(N_1{q_m[0:7]}==N_0{q_m[0:7]})
TRUE
q_out[9]=~q_m[8];
q_out[8]=q_m[8];
q_out[0:7]=(q_m[8]) ? q_m[0:7]:~q_m[0:7]);
FALSE
q_m[8]==0
FALSE
cnt(t) = cnt(t-1)+
(N_1{q_m[0:7]} - N_0{q_m[0:7]});
TRUE
cnt(t) = cnt(t-1) +
(N_0{q_m[0:7]} - N_1{q_m[0:7]});
(cnt(t-1)>0 AND
(N_1{q_m[0:7]}>N_0{q_m[0:7]})
OR
(cnt(t-1)<0 AND
N_0{q_m[0:7]}>N_1{q_m[0:7]})
TRUE
FALSE
q_out[9]=0;
q_out[8]=q_m[8];
q_out[0:7]=q_m[0:7];
Cnt(t)=Cnt(t-1) - 2*(~q_m[8]) +
(N_1{q_m[0:7]} - N_0{q_m[0:7]});
q_out[9]=1;
q_out[8]=q_m[8];
q_out[0:7]=~q_m[0:7];
Cnt(t) = Cnt(t-1) + 2*q_m[8] +
(N_0{q_m[0:7]} - N_1{q_m[0:7]});

Figure 8. TMDS Video Data Encode Algorithm

4.3 HDMI Chip Selection

When designing an HDMI output module for retro systems, signal integrity, ease of integration, cost, footprint, and full audio support are key concerns. In this project, two approaches were evaluated for generating the HDMI signal:

1. AC-Coupled LVDS Approach

In the first version, the HDMI port was connected to the FPGA output via a network of capacitors and resistors configured for AC-coupled LVDS as seen in Figure 9. This method relies on discrete external components to convert the FPGA's LVDS output to an HDMI-compatible signal. Although this approach can be very cost-effective (since it uses generic passive components) and has a very small footprint, it typically requires careful PCB layout to avoid signal degradation [34]. In practical implementation, the AC-coupled LVDS solution showed stable operation on all tested devices with HDMI input interfaces.

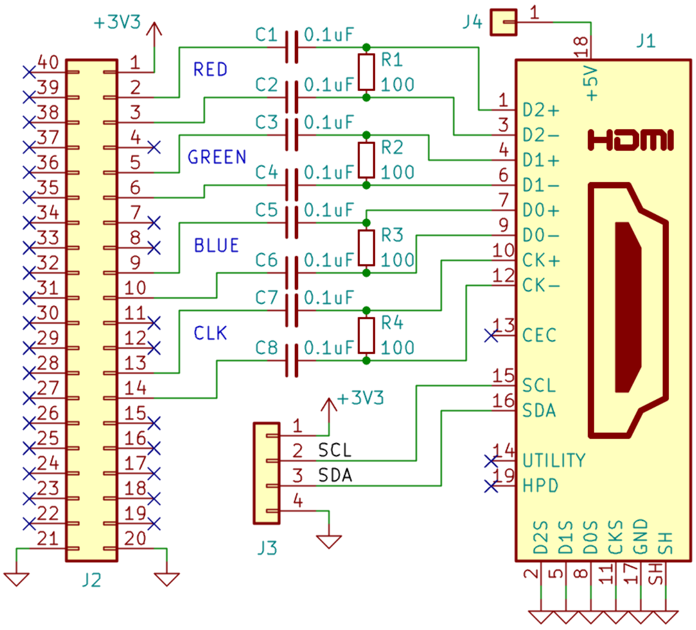


Figure 9. Minimal AC-coupled LVDS schematics for iCE40HX-8K Breakout Board

2. Dedicated HDMI Transmitter Chip – PTN3366

In the alternate approach, a dedicated HDMI transmitter chip, the PTN3366, was used. PTN3363 is a low power, high-speed level shifter device which converts four lanes of low-swing AC-coupled differential input signals to DVI v1.0 and HDMI v1.4b compliant differential output signals. The difference between AC-coupled and DC-coupled signals is illustrated in Figure 10 [35]. Each of these lanes provides a level-shifting differential active buffer, with built-in Equalization, to translate from low-swing AC-coupled differential signaling on the source side, to TMDS-type DC-coupled differential current-mode signaling terminated into 50Ω to 3.3V on the sink side [36].

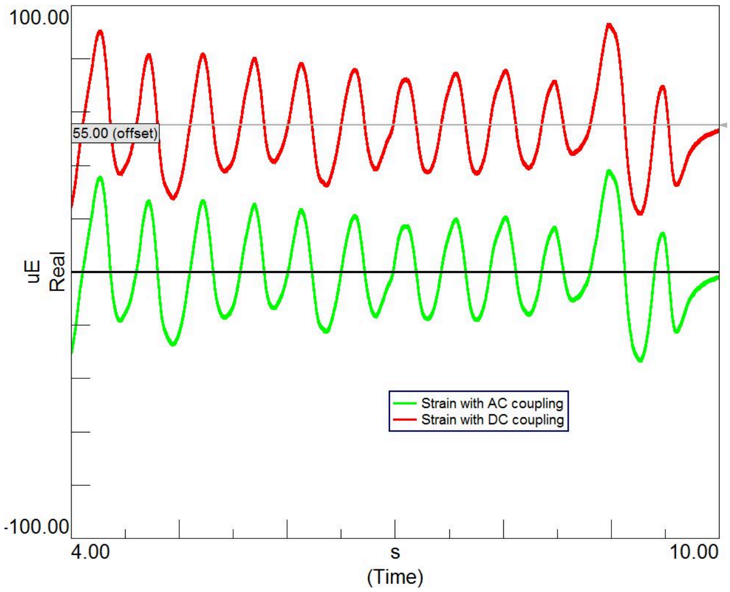


Figure 10. AC-coupling vs DC-coupling

Both implementations required a TMDS clock of 251.75 MHz for data lines. Since this frequency exceeds the Lattice HX40's direct capabilities, DDR mode was utilized to effectively double the 125.798 MHz main clock. Because Lattice HX series FPGAs lack dedicated differential pair outputs as mandated by the HDMI standard, the AC-coupled LVDS design used two SB_LVCMOS primitives [37], while the PTN3366 solution required only one SB_LVCMOS primitive per channel.

For HDMI output testing, two TV sets (Samsung UE43NU7100 and Sharp 49PUS6482), a Dell U2719DC monitor, and a budget HDMI tester from AliExpress [38] were used. Output was evaluated with a 3.2 mm thin, 20 cm long HDMI cable, as well as with an inexpensive 1.5 m cable. Both implementations showed stable performance at 640×480p at 60 Hz on all tested HDMI receiver devices. However, the PTN3366 was ultimately selected because it requires fewer FPGA pins, eliminates the risk of back-powering the FPGA via the TMDS lines and provides built-in equalization and pre-emphasis for improved signal integrity.

Table 3 lists alternative chips that were considered. In the "Price" column, the lowest available price is provided - sourced from Mouser [39] when available, or from the Octopart [40] search engine otherwise. Availability is "Good" when multiple vendors have part in stock and in large quantities. In the "Minimum FPGA Pin Count" column, the required pin count in dual-edge or dual-rate mode is indicated in brackets. For devices with a clock rate requirement of 27 MHz, it is also possible to achieve 1080p@25fps using a pixel clock of 74.25 MHz.

It is evident that with additional spare FPGA I/O pins, some HDMI transmitter ICs could be used to support resolutions up to 1080p@25. Such implementations would require extra pins dedicated to data, audio, and control - typically a total of 17 pins when using dual-rate or dual-edge modes. An alternative to further reduce pin usage is to implement YCbCr 4:2:2 mode with the SiI9022, which employs an 8-bit wide data bus with embedded sync signals.

Currently, only one FPGA pin is completely unconnected, with two additional pins available as spares. If the Ethernet port were removed, up to 9 pins could be freed; omitting the SRAM interface could conserve 30 pins. These constraints illustrate that although higher resolution outputs are feasible with certain HDMI transmitter ICs and increased FPGA resources, the present design’s limited spare I/O availability restricts this capability.

Table 3. Comparison of HDMI Transmitter ICs

Method / Chip	Cost	Availability	min. FPGA Pin Count	Clock Rate for 480p	Footprint	Audio	Configuration Interface	Comments
AC-Coupled LVDS	-	-	RGB: 8	251.75 MHz	N/A	-	-	Lowest cost; very compact per component. Highly sensitive to PCB layout
PTN3366 [36]	1.50€	Moderate	RGB: 4	251.75 MHz	HVQFN-32	-	I²C, pin-strap	AC-Coupled to TMDS level shifter redriver. pin-configurable equalization.
TDP158 [41]	4.01€	Good	RGB: 4	251.75 MHz	WQFN-40	-	I²C	AC-Coupled to TMDS level shifter redriver. 6Gbps. pin-configurable equalization, pre-emphasis, swing and slew rate
SN75DP129 [42]	2.82€	Good	RGB: 4	251.75 MHz	VQFN-32	-	I²C, pin-strap	AC-Coupled to TMDS level shifter redriver.
SiI9022 [43]	5.22€	Low	RGB: 28 (16) I2S 3 I2C: 2	27MHz	QFN-72	I2S, S/PDIF	I²C	HDMI Transmitter. Single clock dual edge and Dual clock single edge modes
TDA19988 [44]	10€	Bad / EoL	RGB: 28 (16) I2S 3 I2C: 2	27MHz	HVQFN-64	I2S, S/PDIF	I²C	Automotive-grade HDMI transmitter; higher cost. Single clock dual edge and Dual clock single edge modes
TFP410 [45]	7.12€	Good	RGB: 28 (16)	27MHz	HTQFP-64	-	I²C, pin-strap	DVI Transmitter. Single clock dual edge and Dual clock single edge modes. DVI only. Pin-configurable.
SiI164 [46]	5€	Low / EoL	RGB: 28	27MHz	TQFP-64	Yes	I²C, pin-strap	similar or same as TFP410
ADV7513 [47]	4€	Low	RGB: 28	27MHz	LQFP-64	I2S, S/PDIF	I²C	HDMI Transmitter.

4.4 HDMI Implementation in the FPGA

The realization of an HDMI source (transmitter) on an FPGA involves several key stages:

1. Video Data Input: Receiving pixel data (RGB) and timing signals (Horizontal Sync - HSYNC, Vertical Sync - VSYNC, Data Enable - DE) corresponding to the desired output resolution and refresh rate.
2. TMDS Encoding: Encoding the 8-bit R, G, and B data channels into 10-bit symbols according to the TMDS specification. This encoding minimizes signal transitions and maintains DC balance. Control signals are also encoded during blanking periods.
3. Serialization: Serializing the 10-bit TMDS symbols onto the high-speed differential output pins. For standard HDMI (like 640x480p@60Hz), the bit rate on each data channel is 10 times the pixel clock frequency.
4. Audio Data Input & Packetizing: Receiving audio samples (typically PCM), buffering them, and formatting them into HDMI audio sample packets. Timing information is conveyed using Audio Clock Regeneration packets.
5. Data Island Transmission: Inserting audio and auxiliary data packets into the HDMI stream during video blanking intervals (Data Island periods).
6. Clock Generation and Distribution: Generating and distributing the necessary clocks: the pixel clock, the high-speed serialization clock (TMDS clock), and the audio sample clock.

To accelerate development, the open-source HDMI encoder core [48] was chosen as a foundation. This core provides a modular implementation of a DVI/HDMI video source, including TMDS encoding and basic packet insertion capabilities.

The Lattice iCE40 family, known for its low power consumption and small footprint also have limited logic resources (LUTs, registers) and EBR compared to mainstream FPGAs. This presents specific challenges and requires slight modifications to the original core like rewriting it from SystemVerilog to Verilog, limiting color depth and reducing count of wires.

The core expects pixel and TMDS clocks, parallel RGB pixel data and optionally audio samples. It handles the 8b/10b encoding, DC-balancing and serialization required for the TMDS data channels. Figure 11 shows the data flow within the HDMI core. Video and control signals from the main HDMI module are sent directly to three TMDS channels. Concurrently, audio and timing signals are processed by a ‘Packet Picker’ and ‘Packet Assembler’ to create data island packets. These packets, along with the video and control signals, are handled by the TMDS channels and then combined by a ‘Serializer’ to generate the final TMDS output signal and clock.

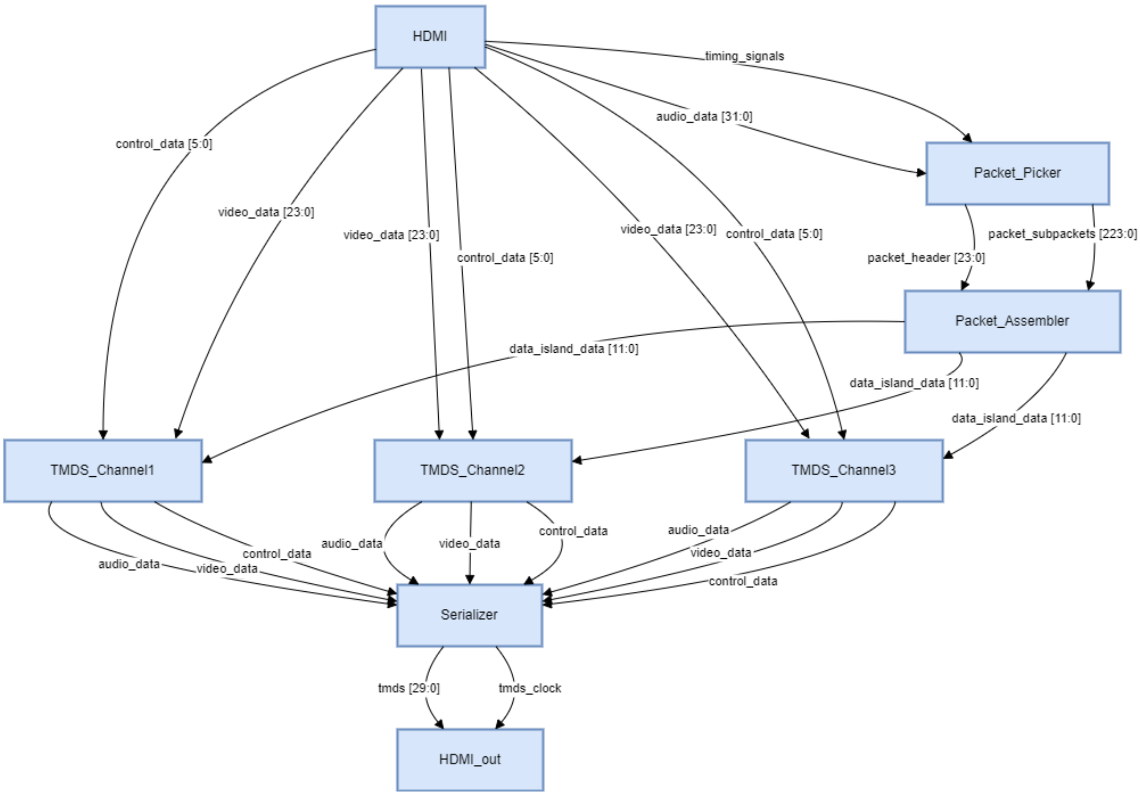


Figure 11. HDMI core module interconnection diagram

4.3.1 Clocking Scheme

A robust clocking strategy is essential for HDMI generation. The implementation utilizes three primary clock domains:

1. Pixel Clock (f_{pixel}): This clock runs at 25.16MHz. It dictates the rate at which pixel data is processed and corresponds to the standard pixel clock for a 640x480p @ 60Hz VGA timing, often used as a base for simple HDMI

implementations. This clock governs the video data path logic before serialization, including interfacing with the ZX Spectrum video generation logic and fetching pixel data. It is derived from a main system clock dividing the f_{TMDS_serial} clock by 5 (251.596MHz / 5 = 25.16MHz).

2. TMDS Serialization Clock (f_{TMDS_serial}): This clock drives the final output serializers. It runs at 125.798 MHz. Crucially, this implementation uses DDR output, meaning data is output on both the rising and falling edges of this clock. The effective serialization rate is therefore 2×125.798MHz = 251.596MHz. This matches the required $10 \times f_{pixel}$ (10×25.16MHz = 251.6MHz) bit rate per TMDS data channel needed for 8b/10b encoding. This f_{TMDS_serial} clock is generated by a PLLfrom a main system clock. The slight discrepancy between 125.8 and 125.798 might be due to PLL generation constraints or slight variations in the base reference clock, but is generally within acceptable tolerances for HDMI synchronization. This high-frequency clock requires careful routing on the FPGA.
3. Audio Clock (f_{audio}): A standard 48kHz sampling clock is used for audio processing. This clock governs the sampling of the ZX Spectrum's audio output (Beeper and potentially AY-3-8912 sound chip) and the feeding of these samples into the HDMI audio packetizer module. As this clock is asynchronous to the pixel and TMDS clocks, proper Clock Domain Crossing (CDC) techniques (e.g., using asynchronous FIFOs) are essential when passing audio data or control signals between the audio clock domain and the HDMI core's clock domain f_{pixel}. The HDMI standard uses Audio Clock Regeneration packets based on the TMDS clock to allow the receiver to accurately reconstruct the original 48 kHz audio sampling rate.

4.3.2 Integration with ZX Spectrum Core

The HDMI module interfaces with the main ZX Spectrum logic implemented on the FPGA.

The source of the video image is the simulated ZX Spectrum's dedicated screen memory area. To decouple the asynchronous memory accesses of the simulated Z80 CPU from the synchronous pixel generation required by HDMI, a shadowing approach using FPGA EBR is employed.

- Screen Memory Shadowing: A bus monitoring module continuously observes the simulated ZX Spectrum's address bus (`zx_a[15:0]`) and data bus (`zx_d[7:0]`). Whenever a memory write or read operation occurs targeting the Spectrum's screen memory region – identified by the address lines `zx_a[15:13]` being equal to 3'b010 (corresponding to addresses 0x4000 - 0x5FFF) – the data present on zx_d along with its address is captured. This captured data is immediately written into a dual-port EBR within the FPGA, effectively creating a real-time shadow copy of the Spectrum's video RAM (both the 6144 bytes of pixel bitmap data and the 768 bytes of attribute data). This write operation is synchronized to the Spectrum's CPU clock domain.
- Pixel Generation for HDMI: The HDMI timing generator, operating synchronously with the pixel clock (f_{pixel} = 25.16 MHz), dictates the pixel stream sent to the HDMI encoder. For each pixel clock cycle corresponding to the active display area, the generator calculates the current pixel's coordinates on the screen (cx, cy). These coordinates are translated into read addresses for the second port of the dual-port EBR framebuffer.
- Spectrum Data Retrieval: Based on the calculated EBR address derived from cx and cy, the corresponding pixel bitmap byte (containing data for 8 horizontal pixels) and the relevant attribute byte (defining foreground/background colors, brightness, and flash status for an 8x8 character cell) are read from the EBR. This read operation occurs four times slower than the HDMI pixel clock because every output pixel is doubled.
- Color Conversion: A dedicated combinatorial logic block, implemented as a Look-Up Table (LUT), takes the fetched bitmap bit (for the specific cx position within the 8-pixel byte) and the attribute byte. It interprets the Spectrum's 3-bit foreground/background color, 1-bit brightness modifier, and 1-bit flash attribute to generate the final color. This logic translates the Spectrum's specific color representation into a standard 24-bit RGB value (rgb[23:0], providing 8 bits per color channel). The flashing effect is also handled here by alternating between foreground and background colors.

- Optional HQ2x Scaling/Filtering: Before being sent to the final output stage, the rgb[23:0] pixel data is passed through an intermediate processing core. This core implements an optional HQ2x scaling and filtering algorithm. Based on a runtime selection (determining if HQ2x enhancement is active), this core performs one of two operations: if HQ2x is disabled, it can perform simple pixel doubling (nearest neighbor scaling) to increase the image size; if HQ2x is enabled, it applies the HQ2x algorithm itself, which provides edge smoothing and interpolation during scaling. The output pixel data, still in rgb[23:0] format, represents the potentially scaled image data. The specifics of this HQ2x core and other implemented upscaling techniques will be detailed in Chapter 5: Pixel scaling.
- Output to HDMI Core: The final 24-bit RGB pixel value is registered and output to the main HDMI encoder module, synchronized precisely with the f_{pixel} clock and the accompanying timing signals (HSYNC, VSYNC, DE) which define the 640x480 frame structure. The actual Spectrum display area is centered within this larger frame.

The audio output from the Spectrum's sound sources (Beeper logic, AY chip simulation) are sampled at 48 kHz. This might involve simple digital sampling of a simulated output or processing of audio samples generated by an AY core. These PCM samples are then passed to the audio input stage of the HDMI core, likely via an asynchronous FIFO to handle the clock domain crossing from f_{audio} to f_{pixel}.

5 Pixel scaling

The display of graphics generated by vintage computer systems, such as the ZX Spectrum with its native resolution of 256x192 pixels, on modern high-resolution monitors presents a significant challenge. Directly displaying the original low-resolution image results in an impractically small picture area. Consequently, image scaling is required to enlarge the image to occupy a reasonable portion of the contemporary display.

Retro graphics, particularly from the 8-bit and 16-bit eras, often rely heavily on the precise placement of individual pixels and constrained color palettes to define shapes, textures, and artistic style [49]. Simple scaling algorithms, especially those based on traditional interpolation methods developed for continuous-tone images like photographs, tend to disregard this deliberate pixel-level design. Methods like bilinear or bicubic interpolation can cause excessive blurring, soften sharp edges, and introduce intermediate colors not present in the original limited palette, thereby compromising the distinct visual identity of the source material [50]. The goal when scaling such graphics is typically integer magnification (e.g., 2x, 3x) while enhancing, or at least preserving, the perceived visual quality and respecting the original artistic intent encoded in the pixel data.

Pixel scaling algorithms can be broadly categorized into basic interpolation methods and algorithms specifically designed for pixel art.

5.1 Basic Interpolation Methods

These methods treat the image as a continuous signal sampled at discrete points and attempt to reconstruct the signal at a higher sampling rate (resolution).

- Nearest Neighbor (NN): This is the simplest scaling technique. Each pixel in the source image is simply replicated into a block of pixels in the destination image [50]. For 2x scaling, each source pixel becomes a 2x2 block of the same color. NN interpolation is computationally trivial and has the property of preserving the original color palette exactly. However, it introduces severe blockiness, particularly visible as jagged edges ("jaggies") on diagonal lines and curves [51]. While often considered aesthetically poor for general viewing, its preservation of sharp pixel boundaries can sometimes be preferred in specific retro contexts where maintaining the original blocky look is desired.

- Bilinear Interpolation: This method calculates the value of each output pixel by performing a linear interpolation based on the four nearest neighboring pixels (a 2x2 area) in the source image. It produces smoother results than NN, reducing blockiness, but at the cost of significant blurring and softening of edges [50]. Furthermore, the averaging process can introduce new colors not present in the original palette. Its tendency to blur sharp details makes it generally unsuitable for pixel art.
- Bicubic Interpolation: Bicubic interpolation extends the concept of bilinear interpolation by considering a larger neighborhood of 16 pixels (4x4) from the source image and fitting a cubic spline function to determine the output pixel value. This generally yields smoother gradients and less noticeable interpolation artifacts compared to bilinear interpolation for photographic images [50]. However, it introduces even more blurring than bilinear interpolation and can sometimes produce "ringing" artifacts (overshoot/undershoot near sharp edges). Its strong smoothing effect is detrimental to the sharp, stylized appearance typical of pixel art.

The fundamental issue with applying basic interpolation methods to pixel art is their inherent averaging nature. Pixel art often relies on sharp transitions between distinct colors and the deliberate placement of individual pixels to convey detail. Interpolation algorithms, designed for continuous-tone images, work by averaging or fitting curves through neighboring pixel values. This process inevitably blurs the sharp transitions and obscures the individual pixel details that define the pixel art style, making these methods generally inappropriate unless a deliberately softened effect is sought [52].

5.2 Pixel-Art Specific Algorithms

Recognizing the shortcomings of traditional interpolation for retro graphics, a class of algorithms has been developed specifically to scale low-resolution, low-color-count images while attempting to preserve sharpness and intelligently smooth jagged lines. These algorithms operate by analyzing local pixel patterns to infer the intended geometric features (lines, curves, corners) and rendering them smoothly at the target resolution.

- ScaleNx Family (e.g., Scale2x, EPX): Algorithms like EPX (Eric's Pixel Expansion) and its derivatives (Scale2x, AdvMAME2x) analyze the immediate 3x3 neighborhood of a pixel [53]. They detect edges based on color differences between the central pixel and its direct neighbors (up, down, left, right). The central pixel is then expanded (e.g., into a 2x2 block for Scale2x) conditionally, using the colors of the neighbors to smooth detected edges. These algorithms typically follow simple rules, are relatively computationally inexpensive, and often preserve the original color palette.
- HQx Family (e.g., hq2x, hq3x, hq4x): Developed by Maxim Stepin, these algorithms (High Quality Scale) also analyze the 3x3 neighborhood but employ a more sophisticated pattern matching approach. The configuration of colors in the neighborhood is compared against a large set of 256 pre-computed patterns stored in a lookup table (LUT). Each LUT entry contains a pre-rendered, smoothed representation (e.g., 2x2 pixels for hq2x) corresponding to that specific neighborhood pattern [54]. Hqx algorithms are known for producing high-quality smoothing, particularly on diagonal lines and curves, significantly reducing jaggies compared to NN, though they might introduce slight blurring.
- xBR Family (e.g., xBR, xBRZ): The xBR ("scale by rules") family, initiated by Hyllian, also uses pattern recognition on the pixel neighborhood but combines it with a set of explicit interpolation rules rather than relying solely on a LUT [53]. These algorithms are designed to support more complex features, such as anti-aliased lines and fine details, potentially preserving the sharpness of the original image better than hqx in some cases. xBR variants are often praised for producing very smooth results, although this can sometimes lead to more noticeable alterations in the original shapes compared to other algorithms [55].

- Other Methods: More advanced techniques exist, including various forms of Edge-Directed Interpolation (EDI) like NEDI and EEDI, vectorization (image tracing) which converts the bitmap to a vector representation before rendering at the target size, and recent approaches using deep convolutional neural networks [52] [53]. However, these methods are generally significantly more computationally complex and memory-intensive, making them unsuitable for implementation on resource-constrained FPGAs like the iCE40 HX4K.

Pixel-art specific algorithms represent a fundamentally different approach compared to basic interpolation. Instead of merely calculating weighted averages of pixel values, they attempt to interpret the intent behind the arrangement of pixels within a local neighborhood. By recognizing patterns that correspond to features like edges, corners, or curves, they apply transformations (conditional pixel expansion, LUT-based replacement or rule-based interpolation) specifically designed to render these inferred features smoothly at the higher target resolution [50]. This context-sensitive analysis allows them to achieve results that are often perceived as superior for preserving the characteristic look and feel of pixel art.

5.3 Rationale for Algorithm Selection in this Project

The primary requirement for this project is to implement 2x integer scaling suitable for enhancing the visual presentation of ZX Spectrum-like graphics (256x192 resolution, limited color palette) on a modern display, using a Lattice iCE40 HX4K FPGA.

Given this context, two algorithms were selected for implementation:

1. Pixel Doubling (Nearest Neighbor 2x): This serves as a baseline implementation. It requires minimal hardware resources (essentially just adjusted address calculation) and provides a direct reference point for the visual output of the original, unscaled pixels presented at a larger size. It is included as a user-selectable option via a hardware switch.
2. HQ2x: This algorithm was chosen as the primary enhancement technique based on subjective visual preference compared to the blockiness of NN as seen in figure 12 [56].

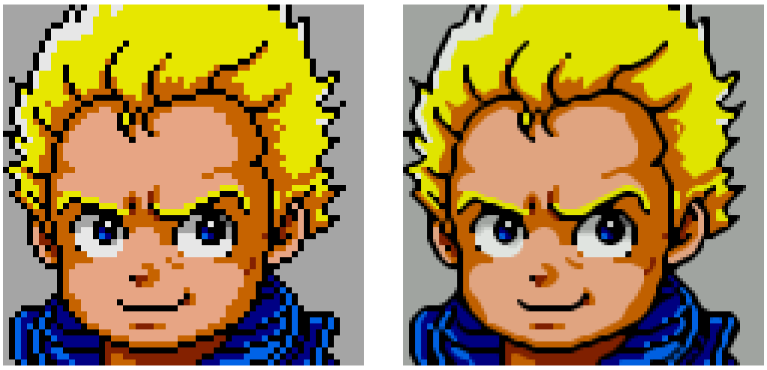


Figure 12. nearest neighbour 2x vs hq2x

HQ2x also potentially offers a better balance of visual improvement versus implementation complexity compared to alternatives like xBR. While xBR might offer better detail preservation in some cases, its rule-based nature can translate to more complex logic. The LUT-based approach of hq2x, while requiring storage for the LUT, potentially maps more straightforwardly to hardware logic or memory blocks, assuming the LUT itself is manageable [54]. Higher-order variants like hq3x/hq4x were not considered due to their significantly increased computational and memory requirements. Table 4 provides a comparative summary of the discussed algorithms, as illustrated in Figure 13 [57], highlighting the trade-offs underlying this selection.

Table 4. Comparison of pixel scaling algorithms

Algorithm	**Core Logic**	**Visual Output (Sharpness, Smoothness, Artifacts)**	**Color Preservation**	**Relative Cost**	**Suitability for Pixel Art**
Nearest Neighbor (NN)	Pixel Replication	Very Sharp, Blocky (Jagged Edges)	Perfect	Very Low	Low (Preserves pixels)
Bilinear Interpol.	Linear Average (2x2 Neighborhood)	Blurred, Soft Edges, Smooth	Poor (New Colors)	Low	Very Low (Blurring)
Bicubic Interpol.	Cubic Spline Average (4x4 Neighborhood)	Very Blurred, Smoother, Ringing Artifacts	Poor (New Colors)	Medium	Very Low (Blurring)
Scale2x (EPX)	Edge Detection (3x3), Conditional Expansion	Sharp Edges, Smooth Diagonals, Simple Patterns	Good	Low - Medium	Medium

hq2x	Pattern Matching (3x3), LUT Interpolation	Sharp but Smoothed Edges/Curves, Reduced Jaggies	Good (Original)	Medium - High	High
xBR	Pattern Recognition (3x3), Rule-Based Interpol.	Very Smooth Edges/Curves, Detail Preservation (Variable)	Good	High	High

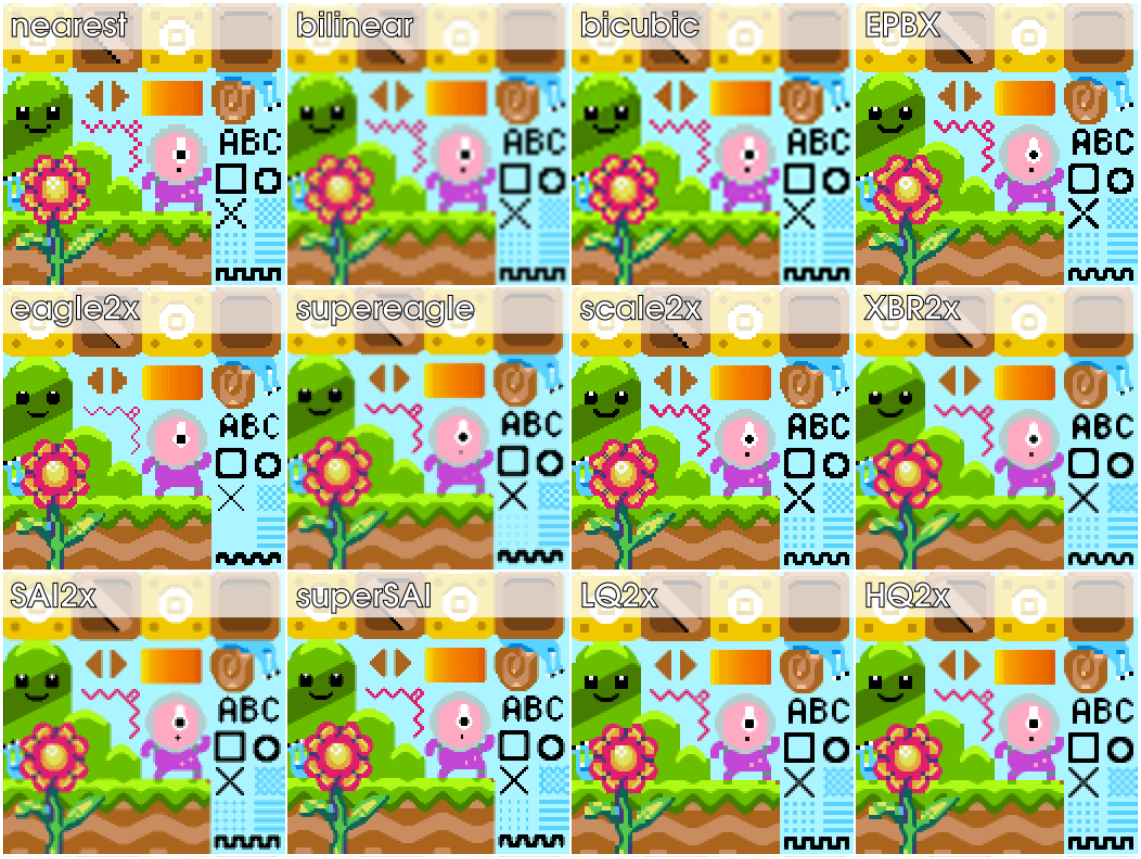


Figure 13. Comparison of pixel scaling algorithms

5.4 The hq2x Algorithm

The hq2x algorithm operates by examining the local neighborhood around each pixel in the source image and using a pattern-matching approach to determine the corresponding 2x2 block of pixels in the scaled output image [58].

5.4.1 Core Logic: Neighborhood Analysis and Pattern Matching

The algorithm iterates through the source image, processing one pixel at a time. For each source pixel, designated as P, it considers the 3x3 square of pixels centered on P, which includes P itself and its eight immediate neighbors.

A key step is determining the similarity between the color of the central pixel P and each of its eight neighbors. The original hqx algorithm specification employs the YUV color space for this comparison. RGB color values are first converted to YUV. The difference between the neighbor's color and P's color is then calculated, typically using a weighted distance formula that gives more significance to the difference in luminance (Y component) than chrominance (U and V components). This weighting is intended to better align the notion of color similarity with human visual perception. If the calculated color difference falls below a predefined threshold, the neighbor is considered "similar" to P; otherwise, it is considered "dissimilar".

The binary outcome (similar/dissimilar) for each of the eight neighbors effectively creates an 8-bit pattern index, ranging from 0 to 255. This index represents the specific configuration of color similarities and differences within the 3x3 neighborhood.

This 8-bit index is then used to query a pre-computed lookup table (LUT). This LUT is the core component of the hq2x algorithm. It contains 256 entries, one for each possible neighborhood pattern. Each entry stores the specific interpolation pattern required to generate the 2x2 block of output pixels that corresponds to the single input pixel P given its neighborhood context. The LUT entry essentially dictates how the colors of P and its relevant neighbors (those determined to be part of the pattern) should be selected or blended to produce the four output pixels. This process effectively replaces the single pixel P with a 2x2 representation that is smoothed according to the detected local pattern [58] [54]. A visual representation of these neighborhood patterns and their interpolated results is partially illustrated in Figure 14 [59].

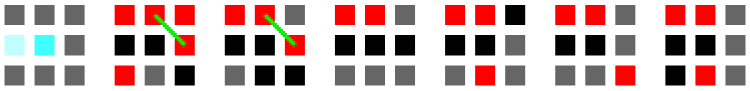


Figure 14. Fragment of visual representation of hq2x

5.4.2 Edge Handling and Smoothing

Edge detection in hq2x is not performed using explicit gradient operators [60]. Instead, it is an implicit result of the pattern matching process. Specific configurations of similar and dissimilar neighbors in the 8-bit pattern index naturally correspond to different types of local features, such as horizontal, vertical, or diagonal edges, as well as corners and areas of uniform color.

The interpolation patterns stored within the 256-entry LUT are carefully designed, pre-rendered versions of these features at the target 2x resolution. According to the algorithm's author, these patterns are generated by determining the "most probable vector representation" for each neighborhood configuration, rasterizing this vector representation at a higher resolution using anti-aliasing, and storing the resulting 2x2 pixel pattern in the LUT. The design constraints ensure that continuity of lines is preserved across adjacent pixel expansions, and the resulting output is optimized for smoothness. For instance, if the 3x3 neighborhood pattern indicates a diagonal line passing through P, the corresponding LUT entry will contain a 2x2 pixel pattern that renders a segment of that diagonal line smoothly [54].

5.4.3 Color Comparison Implementation Detail (YUV vs. Simplified)

While the original hqx algorithm specifies color difference calculation in the YUV color space, implementing this directly in hardware, especially on a resource-constrained FPGA, presents significant challenges. The conversion from RGB to YUV involves matrix multiplications or complex lookups and the subsequent weighted difference calculation adds further arithmetic complexity. This would consume a non-trivial amount of logic resources (LUTs, potentially DSP blocks if available) and likely increase the clock cycles required for processing each pixel [54].

Given the target platform (iCE40 HX4K) and the origin of the reference verilog code (fpganes, optimized for FPGA implementation), the implemented version deviates from the original algorithm by using a simplified color comparison method and removal of the YUV calculations. Hardware optimizations include using a simpler metric in the RGB space, such as the sum of absolute differences between the R, G, and B components and a direct equality check, performed on reduced bit-depth color values.

This represents a practical trade-off, sacrificing the potentially higher perceptual accuracy of the YUV comparison for a significant reduction in hardware complexity and improved performance, which is often necessary to meet the timing and resource constraints of real-time FPGA applications [61].

5.5 Hardware Platform and Constraints: Lattice iCE40 HX4K

The target hardware for this implementation is the Lattice iCE40 HX4K FPGA. Understanding its architecture and resource limitations is crucial for evaluating the feasibility of the hq2x algorithm and guiding the necessary optimizations.

5.5.1 Implications for Memory-Intensive Algorithms

The resource profile of the iCE40 HX4K presents specific challenges for implementing memory-intensive algorithms like hq2x, particularly when combined with other system components that also require memory.

- EBR Capacity: A total of 16KB of on-chip EBR is relatively modest for image processing applications, which often benefit from buffering multiple image lines or even entire frames to facilitate neighborhood operations or random access patterns. The primary video buffer for the ZX Spectrum display (storing pixel and attribute data) already consumes approximately 6.75 KB (6144 bytes pixels + 768 bytes attributes). This pre-allocation significantly reduces the EBR available for other modules.
- EBR Granularity: While having 20 independent 4Kbit blocks provides flexibility, it can also lead to inefficient utilization if the memory structures required by the design do not align well with the 4Kbit (512 Byte) block size. For instance, a memory need slightly exceeding a single EBR block may require allocation of an additional block, leading to fragmentation. In the case of a 6912-byte memory requirement, addressing this space necessitates a 13-bit wide address bus, effectively reserving 8192 bytes. As a result, 16 EBR blocks are consumed, despite the actual demand being lower.
- LCs vs. EBR Trade-off: A critical aspect of FPGA design is how memory structures described in HDL are mapped during synthesis. The synthesis tools

attempt to infer whether such structures should be implemented using the dedicated EBR blocks or using the general-purpose logic cells (LCs) configured as distributed RAM. Implementing memory using LCs is significantly less efficient in terms of resource usage; it consumes a large number of LUTs (for storage and addressing logic) and flip-flops (for registered outputs), and places heavy demands on the routing resources, often leading to slower performance and longer synthesis times.

The combination of these factors - limited total EBR, the specific 32x4K block structure, and the significant EBR usage by the main video buffer - creates a critical resource constraint for the hq2x implementation. This scarcity strongly suggests that the synthesis tool may struggle to map the relatively large line buffers required by hq2x entirely into the remaining EBR. This explains the extremely high LC utilization reported in the synthesis results (Section 5.6.2), even after color depth reduction, despite the algorithm's memory footprint seemingly fitting within the total available EBR capacity. The dedicated EBR blocks are likely being prioritized for the main video buffer, forcing the hq2x line buffers onto the general-purpose logic fabric.

5.6 hq2x Implementation and Optimization on iCE40 HX4K

The implementation of the hq2x algorithm on the iCE40 HX4K required significant adaptation from the reference fpganes code due to the hardware resource constraints outlined previously.

5.6.1 Baseline fpganes hq2x Verilog Implementation

The starting point for this work was the hq2x Verilog module developed by Ludvig Strigeus for the fpganes NES emulator project [62].

The original implementation utilizes register arrays to buffer input and output pixel data, facilitating access to the required pixel neighborhood for the hq2x calculations.

- Input Buffer (`inbuf`): Defined as `reg [14:0] inbuf[0:511];`. This array stores pixel data for two lines of the source image in RGB 5:5:5 format. The memory required for `inbuf` is 512 words * 15 bits/word = 7680 bits.

- Output Buffer (outbuf): Defined as `reg [14:0] outbuf[0:2047];`. This array stores pixel data for four lines of the 2x scaled output image. For a source width of 256, the output width is 512 pixels. Thus, this buffer holds 4 lines * 512 pixels/line = 2048 entries. As color depth is 15 bits, the memory required for outbuf is 2048 words * 15 bits/word = 30720 bits.
- 256 element LUT (hqTable): Defined as `reg [5:0] hqTable[256];`. This array is used to quickly determine whether two pixels are similar or different, based on quantized color indices, which in turn informs edge detection and the upscale pattern selection. Memory required for hqTable is 256 words * 6 bits/word = 1536 bits.

Total Memory Requirement (Original): The combined memory footprint of these two buffers in the original 15-bit implementation is 7680 bits + 30720 bits + 1536 = 39936 bits, which is approximately 4.88 KB. Attempting to synthesize 39936 bits of memory using LC will fail, because it exceeds 7680, the total LC count available on the HX4K as each bit stored in a register requires at least one flip-flop (FF). Additionally, implementing the read and write access logic for such large arrays using LUTs would require extensive multiplexing structures.

In practice, the synthesis results were inconsistent. In some cases, the synthesis tool mapped the outbuf to EBR, despite the memory pattern not conforming to the supported EBR configurations [63]. This occasionally produced a working bitstream. However, in other instances, the tool attempted to map all buffer structures to logic cells, which exceeded the available logic resources (e.g., 114% utilization reported by nextpnr-ice40) and ultimately caused the place-and-route stage to fail. The following is an excerpt from the nextpnr-ice40 tool, demonstrating a successful placement and routing process in which the router allocated the outbuf (occupying 8 blocks) and hqTable (1 block) registers to embedded block RAM (EBR):

```
Info:            ICESTORM_LC:  7268/ 7680    94%
Info:           ICESTORM_RAM:    25/   32    78%
Info:                  SB_IO:    43/  256    16%
Info:                  SB_GB:     8/    8   100%
Info:           ICESTORM_PLL:     1/    2    50%
Info:            SB_WARMBOOT:     0/    1     0%
```

Table 5 shows a comparison of the memory requirements of the line buffers for the original and modified versions.

Table 5. Line buffer sizes

	inbuf		outbuf		
Version	dimensions	size (kB)	dimensions	size (kB)	Total KB
Original 15-bit	512 * 15	0.94	2048 * 15	3.75	4.69
6-bit	512 * 6	0.38	2048 * 6	1.5	1.88
4-bit	512 * 4	0.25	2048 * 4	1	1.25

5.6.2 Design changes and synthesis results

To make the hq2x algorithm fit within the iCE40 HX4K's resource constraints, the primary strategy employed was to reduce the color depth, thereby drastically decreasing the memory footprint of the line buffers.

- Modification 1: 6-bit Color Depth: The first modification reduced the color representation from 15 bits per pixel to 6 bits. This corresponds to 2 bits for each of the Red, Green, and Blue channels (RGB 2:2:2). This represents a significant reduction (60%) in memory requirements compared to the 15-bit original, while still providing enough resolution to match the original color palette of the ZX Spectrum.

  Synthesis using the Yosys tool yielded the following resource utilization:

  ```
  Info:            ICESTORM_LC:  8774/ 7680   114%
  Info:           ICESTORM_RAM:    20/   32    62%
  Info:                  SB_IO:    43/  256    16%
  Info:                  SB_GB:     8/    8   100%
  Info:          ICESTORM_PLL:     1/    2    50%
  ```

  Although memory usage was significantly reduced and the nextpnr-ice40 router successfully mapped the `outbuf` structure to EBR, the bitstream generation failed due to excessive utilization of logic cells (114%), exceeding the available 7680 logic elements on the target FPGA.

- Modification 2: 4-bit Color Depth: To further reduce resource usage, the color depth was decreased again, this time to 4 bits per pixel. This specific 4-bit scheme was tailored to the target application (ZX Spectrum graphics), representing 1 bit each for Red, Green, and Blue, plus 1 bit for a 'Highlight' or 'Bright' attribute, mimicking the Spectrum's native color attribute system. This reduced the memory requirement by another 33% compared to the 6-bit version. The synthesis results for the 4-bit version were:

  ```
  Info:            ICESTORM_LC:  3160/ 7680    80%
  Info:           ICESTORM_RAM:    19/   32    59%
  Info:                  SB_IO:    43/  256    16%
  Info:                  SB_GB:     8/    8   100%
  Info:           ICESTORM_PLL:     1/    2    50%
  Info:            SB_WARMBOOT:     0/    1     0%
  ```

  This modification resulted in a significant drop in LC usage. The 80% LC utilization, while high, represents a workable design that fits within the FPGA's resources, although with limited headroom. Again, the `outbuf` and `hqTable` structures were successfully mapped to EBR, occupying two and one blocks, respectively.

5.6.3 Future Work: Optimizing Resource Usage via Line‑Buffer Migration

Synthesis results for the 4-bit hq2x implementation indicate high resource utilization on the target FPGA. Although the design meets timing and functional requirements within the device capacity, it consumes 80% of the available Logic Cells (LCs). Furthermore, the current implementation relies on the synthesis tool's ability to opportunistically map some large register structures into EBR resources.

Analysis of the post-synthesis resource report confirms that the primary contributor to the high LC count is the implementation of the input line buffer (`inbuf`, requiring storage for two lines of pixel data) and the output line buffer (`outbuf`, requiring storage for four lines) using general-purpose logic fabric, specifically LCs configured as registers and multiplexers. This approach is inefficient compared to utilizing dedicated on-chip memory resources. Therefore, optimizing the implementation of these line buffers presents the most significant opportunity for substantially reducing overall LC utilization [64].

Several strategies are proposed to address this limitation:

1. Direct Memory Access from Video Buffer. This strategy involves eliminating the intermediate `inbuf` and `outbuf` line buffers entirely. This could be achieved by designing the hq2x core to fetch pixel data directly from a main video frame buffer. To get pixel RGB values, we need to make two memory calls. One is to get eight pixels and another query to get attributes for the 8x8 box. To calculate the four output pixels corresponding to a single source pixel P, the hq2x algorithm requires access to the 3x3 neighborhood surrounding P. This means that for each calculation cycle (occurring every 4 pixel clocks), the logic must potentially read up to 9 pixel values from the main BRAM. As the pixel clock is only 25MHz, a five times faster main clock could feasibly accommodate the necessary read operations per calculation cycle without becoming a performance bottleneck.
2. EBR-Based Line Buffer Implementation: An simpler approach involves explicitly instantiating the `inbuf` and `outbuf` line buffers using dedicated EBR blocks instead of synthesizing them from general logic. While initial device assessments based on datasheet summaries suggested a limit of 20 EBR blocks for the iCE40 HX4K, practical experience using the open-source IceStorm synthesis toolchain reveal that 32 EBR blocks are, in fact, addressable and usable within the fabric. This revised understanding confirms the feasibility of allocating the necessary EBR blocks (1 for `inbuf` and 2 for `outbuf`) for efficient line buffer implementation.
3. External SRAM-Based Line Buffer Implementation: Furthermore, the utilization of external SRAM is another option for buffer implementation, should on-chip EBR resources prove insufficient or be required for other critical functions. Although external SRAM have higher access latency around 45ns compared to the EBR (capable of operation synchronous to system clocks, e.g. 8ns cycle time at 125 MHz), it could serve as a viable alternative.

This way there will be substantial reduction in Logic Cell utilization. By eliminating the large `inbuf` and `outbuf` register arrays currently consuming thousands of LCs, this optimization could potentially free up a large fraction of the FPGA's logic resources, bringing the total LC usage down from 80% to a much more comfortable level. Successfully implementing such an optimization would shift the design complexity from large storage structures in logic to more sophisticated control logic, ultimately leading to a more efficient utilization of the FPGA's resources and providing valuable headroom for future development.

6 SD Card Access

To provide modern mass storage functionality for the ZX Spectrum, an SD card interface is implemented using the Serial Peripheral Interface (SPI) protocol. SPI is a synchronous serial communication protocol widely supported by SD cards, especially in embedded and low-resource environments. In this project, the SPI master is implemented by the Z80 processor itself, with all low-level communication performed through I/O port reads and writes. The FPGA acts as a transparent intermediary, routing these I/O operations to the SD card lines.

The SD card is operated in SPI mode, which simplifies signal handling and allows direct bit-banged communication from the Z80 CPU. This approach is lightweight and avoids the need for complex DMA or bus mastering logic in the FPGA.

6.1 FPGA Routing of Z80 I/O to SPI Signals

The FPGA continuously monitors Z80 I/O activity and maps specific port interactions to SPI signal lines. When the Z80 writes to I/O port #03, the FPGA captures individual bits from the data bus and maps them to MOSI, SCK, and CS lines connected to the SD card. Reading from I/O port #01 returns the state of the MISO line, allowing the CPU to receive serial data from the SD card.

Although all SD card lines are currently routed to the FPGA, DAT1 and DAT2 are unused in SPI mode. These can be excluded in future hardware revisions to reclaim two FPGA I/O pins.

6.2 SD Card Boot ROM and FAT Filesystem

At power-on, the FPGA disables the built-in ROM and loads a custom boot ROM from internal BRAM. This ROM contains a Z80 program compiled using the SDCC compiler and implements a menu-based loader for *.sna* snapshot files.

The ROM firmware uses Petit FatFs [65], a lightweight FAT16/32 filesystem library, to access files on the SD card. All file access routines (disk_readp, disk_writep) are backed by low-level SPI functions that manipulate the SPI lines using the above-mentioned Z80 I/O operations. This allows reliable sector reads, directory traversal, and file parsing directly from SD storage. SD/MMC card initialization flow for SPI mode is shown in Figure 15 [66].

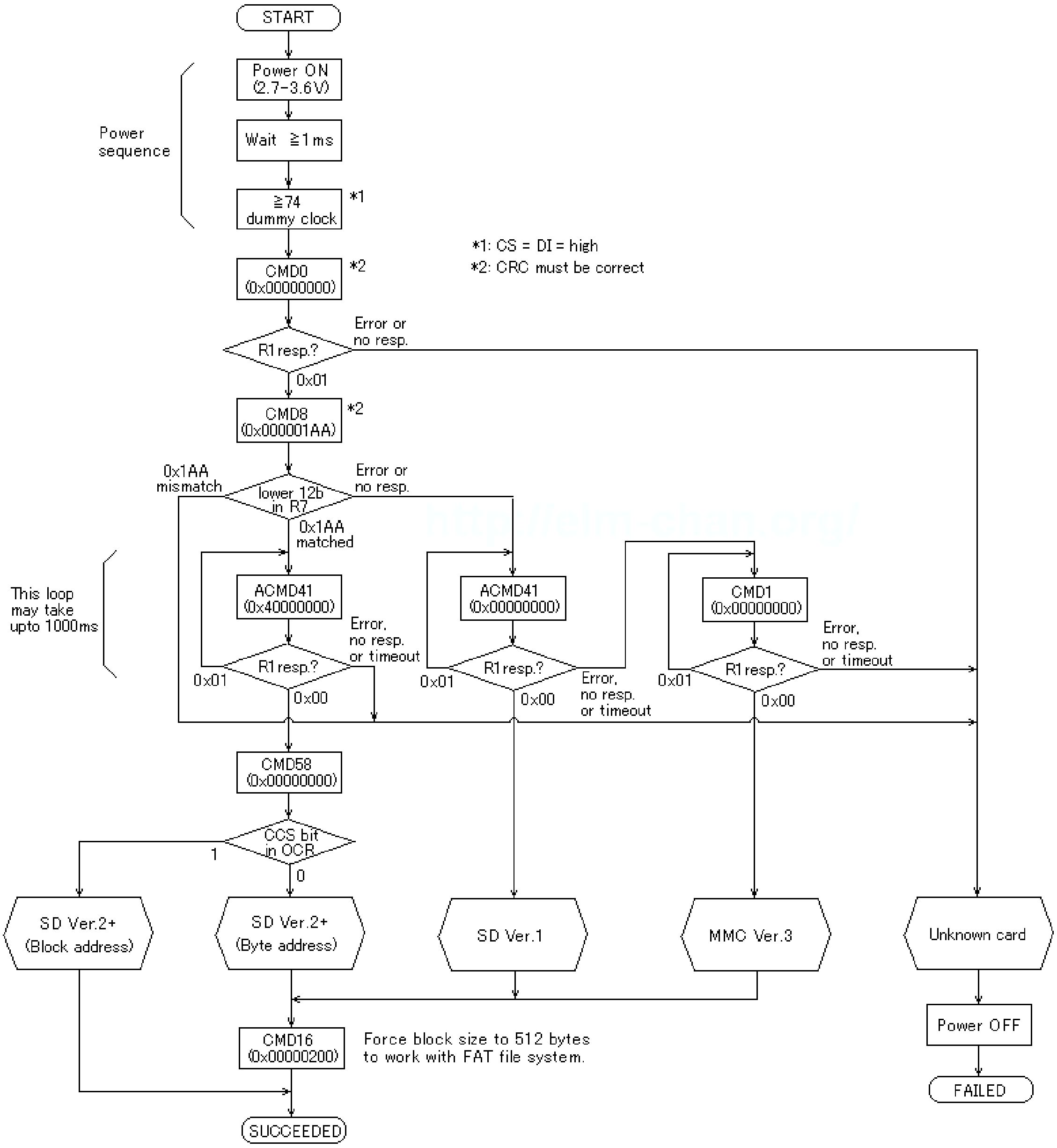


Figure 15. SD/MMC initialization flow

6.3 Snapshot Loading

Upon booting, the ROM displays a menu listing available *.sna* snapshot files on the SD card. When the user selects a snapshot, the loader parses the file, copies memory contents to appropriate RAM pages and sets registers and interrupt mode. The snapshot format includes the CPU stack pointer, which is restored as the last step. Execution is started via a RETN (Return from Non-Maskable Interrupt) instruction, booting into the loaded program image as if it had been cold-started. This works because the process of creating an *.sna* snapshot involves triggering a NMI; the NMI routine pushes the current Program Counter (PC) onto the stack before the registers and memory are saved. The Stack Pointer (SP) value saved in the *.sna* header thus points to this saved PC on the stack [67].

6.4 Summary

This architecture avoids the need for a dedicated SD controller in the FPGA by exploiting the flexibility of the Z80's I/O ports and minimal SPI protocol requirements. Despite the simplicity, it enables reliable loading of full program snapshots with filesystem support and a user-friendly interface, all controlled entirely from the host CPU.

7 AY-3-8912/YM2149 Sound Synthesis

Sinclair ZX Spectrum initial audio capabilities were limited to a single-channel internal beeper, controlled directly by the CPU. This chapter details integration of a popular Programmable Sound Generator (PSG) from the 8-bit era into a ZX Spectrum clone, enabling digital audio output via the HDMI interface.

The PSG chosen for this implementation is based on the General Instrument AY-3-8910/8912 family and its licensed derivative, the Yamaha YM2149. These chips were ubiquitous in the 1980s, providing the characteristic sound for machines such as the later ZX Spectrum 128K, the Amstrad CPC, MSX computers, and the Atari ST. They are known for their three independent square wave tone channels, a flexible noise generator, and hardware envelope control, enabling the creation of complex musical scores and sound effects far beyond the capabilities of the original Spectrum beeper. Yamaha often referred to their YM2149 variant as a Software-controlled Sound Generator (SSG).

7.1 AY-3-8912 / YM2149 Programmable Sound Generator Architecture

7.1.1 Functional Blocks

- Functional Blocks: The core sound generation capability resides in three independent tone generators. Each produces a square wave output. The frequency of each channel is determined by a 12-bit Tone Period (TP) value, loaded across two 8-bit registers which allows precise frequency control over a wide range, typically spanning 8 octaves [68].
- Noise Generator: A single noise generator produces pseudo-random digital noise, often used for percussion sounds or effects like explosions.
- Mixer: The Mixer block determines which sound sources (Tone A, Tone B, Tone C, Noise) are enabled for each of the three output channels (Channel A, Channel B, Channel C).
- Amplitude Control / Level Control (Channels A, B, C): Each output channel has independent amplitude control and each register offers mixed or variable level mode.

- Envelope Generator: A single, shared Envelope Generator provides amplitude modulation capabilities. Its behavior is defined by three registers: RB, RC and RD which select predefined shapes such as single-shot decays, attacks, or repeating triangular or sawtooth patterns.
- Digital-to-Analog Converters (DACs): After mixing and amplitude modulation, the digital signals for each of the three channels are converted to analog voltage levels by internal DACs before being output on the analog channel A, B, and C pins. A notable difference exists here between the AY and YM chips: the YM2149 employs a 5-bit internal DAC mechanism, providing 32 distinct levels for smoother volume transitions, particularly when using the envelope generator. The AY-3-891x series effectively has a 4-bit resolution, resulting in 16 coarser steps [69].
- I/O Ports (A and B): The 40-pin versions (AY-3-8910, YM2149F) include two general-purpose 8-bit parallel I/O ports (Port A, Port B). These were often used for reading joysticks, keyboards, or controlling other peripherals. The FPGA implementation in this project omits these ports as they are not required for the functionality [70].

7.1.2 Register Interface

Interaction with the AY/YM chip is mediated through its 16 internal 8-bit registers, accessible via an 8-bit bidirectional data bus (DA0-DA7) and several control lines.
The standard procedure for accessing a register involves two distinct bus cycles:

1. Address Latch Cycle: The CPU places the desired register address (0 to 15, i.e., 0x0 to 0xF) onto the lower bits of the data bus (typically DA3-DA0) and activates the appropriate control signals to instruct the chip to latch this address internally.
2. Data Read/Write Cycle: The CPU then performs a subsequent bus cycle to either write data to the selected register or read data from it, again using specific control signal combinations.

The primary control signals governing these bus cycles on the original chips are:

- BC1 (Bus Control 1): This input, derived from CPU address lines, select the operational mode.
- BDIR (Bus Direction): This input signal, also derived from the CPU's read/write line, indicates whether the CPU intends to read from (BDIR=0) or write to (BDIR=1) the chip.

7.1.3 Sound Synthesis Methods

The programmability of the registers allows for the synthesis of a wide variety of sounds:

- Tone Generation: The fundamental frequency of each channel's square wave is determined by $f_{MasterClock} / (16xTP)$. The 12-bit TP value allows for $2^{12} = 4096$ distinct period settings per channel, providing fine control over pitch across the audible spectrum and beyond.
- Noise Generation: The noise source is typically implemented using an LFSR. Lower values result in faster clocking and higher-frequency ("whiter") noise, while higher values produce lower-frequency ("rumbling") noise characteristics. The noise output can be mixed into any combination of the three main channels via register R7.
- Envelope Generation: The envelope generator provides dynamic amplitude control. The 16-bit Envelope Period (RB, RC) sets the fundamental frequency of the envelope cycle. The 4-bit Shape control (RD) determines the contour within each cycle [68].

7.2 YM2149 FPGA Core Implementation

The open-source YM2149 core from the ZX_Spectrum-128K_MIST project was chosen because it uses half as many logic cells as other cycle-exact cores [71] [72]. The core outputs three independent, parallel buses, each carrying an 8 bit unsigned integer representing the instantaneous amplitude of the corresponding channel (A, B, or C) which are mixed into 12 bit left and right channels according to ABC scheme. This means that A is mixed to right, B to left and right, and C to left.

In the ZX Spectrum 128K, the YM2149 is mapped into the Z80's I/O space at two fixed ports: 0xFFFD (register‑select port) and 0xBFFD (data port). This minimal two-port scheme allows access to all sixteen PSG registers (three tone generators, noise, envelope, and two 8-bit I/O ports) using only simple address decoding and the two BDIR/BC signals per bus cycle.

Communication is synchronized over two successive bus cycles using the PSG's BDIR and BC control inputs. To write a register, the CPU first asserts /IORQ + /WR with BDIR = 1 and BC = 0 while addressing 0xFFFD, thereby latching the 4-bit register index into the PSG. In the immediately following cycle, /IORQ + /WR are again asserted with BDIR = 1 and BC = 1 while addressing 0xBFFD, which transfers the 8-bit data into the previously selected register. To read back a register, /IORQ + /RD are asserted with BDIR = 0 and BC = 1 on 0xBFFD, causing the PSG to drive the data bus with the contents of the selected register [73].

7.3 I2S Protocol

I2S, first specified by Philips Semiconductor in 1986, is a widely adopted serial bus standard designed explicitly for transferring digital audio data, typically PCM, between integrated circuits within electronic devices. Its primary advantage lies in its synchronous nature, using separate lines for clock and data signals [74].

Generating the I2S clock signals (SCK and WS) requires careful design within the FPGA. The WS frequency must match the target audio sample rate (48 kHz). The SCK frequency must be derived correctly based on the sample rate, bit depth, and channel count ($f_{SCK} = 1.546MHz$ for 48kHz/16-bit/Stereo).

The I2S bus operates in a Master/Slave configuration. The Master device is responsible for generating and distributing the SCK and WS clock signals. The Slave device synchronizes its data transmission or reception to these incoming clocks. In the context of this project, the FPGA logic generating the audio stream for the HDMI core acts as the I2S Master [75].

7.4 Audio Processing Pipeline: From Sound Sources to HDMI

The generation of the final HDMI audio output involves several processing stages within the FPGA, transforming the raw digital outputs from the YM2149 core and the beeper emulation into a correctly formatted and timed audio stream suitable for HDMI encapsulation. A conceptual block diagram illustrating this pipeline is presented in Figure 16.

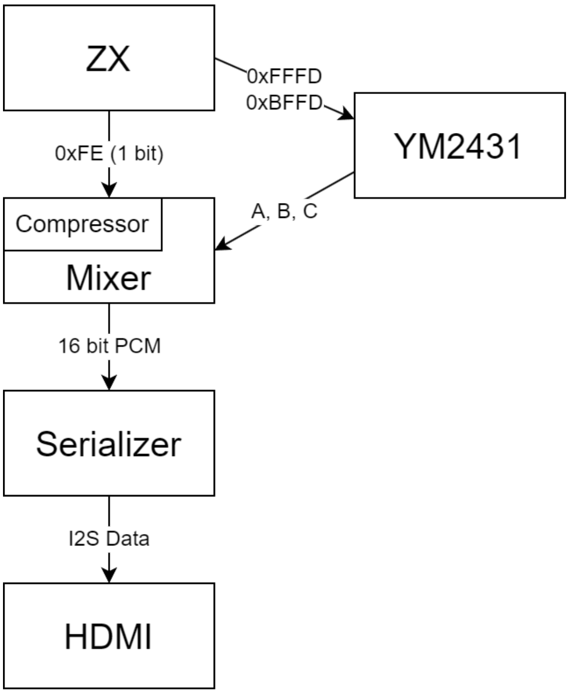


Figure 16. Audio pipeline

1. ZX-Spectrum audio out. Sound path is determined by output port address. 0xFFFD or 0xBFFD writes to YM2431 registers and 0xXXFE bit 4 toggles beeper.
2. Digital Mixing Stage combines the 1 bit beeper and audio generated by the three 8 bit individual channels of the YM2149 core to 16 bit left and right PCM data.
3. Serializer serializes 16-bit left/right PCM samples into the I2S bitstream. It outputs bit clock (i2s_bclk, 1.536 MHz), word-select clock (i2s_lrclk, 48 kHz) and serial data (i2s_data). On each 48 kHz sampling clock it shifts out 16 bits per channel.
4. HDMI core takes the prepared audio data and embeds it into the HDMI data stream alongside the video signal.

7.5 Future Improvements in Audio Quality

To further refine the quality of the audio stream, digital filtering techniques such as Infinite Impulse Response (IIR) or Finite Impulse Response (FIR) filters may be implemented within the FPGA. These filters can smooth out harsh frequency transitions or remove undesirable high-frequency artifacts, particularly useful when mixing multiple sources (e.g., AY-3-8912 and beeper). A basic low-pass IIR filter, can be implemented using minimal logic and can effectively reduce aliasing from the square-wave-based sound generators.

Since the audio generation and output are handled entirely in the digital domain before I2S encoding, these improvements can be developed and integrated incrementally without altering the physical design. This opens the possibility of real-time configuration changes, such as enabling filters or adjusting stereo mapping via FPGA-controlled registers.

8 Conclusions and Future Work

This thesis has presented the design and implementation of a custom FPGA-based expansion module for a specific ZX Spectrum Leningrad computer clone. The solution integrates a wide array of modern functionality, including HDMI video output with pixel enhancement algorithms, SD card-based mass storage, digital audio transport via HDMI using I2S, and support for the AY-3-8912 sound chip.

The work demonstrates that it is possible to modernize a soviet-era 8-bit computer with minimal intrusion. A key technical achievement of the project is the full-featured HDMI output, implemented entirely within a Lattice iCE40 FPGA, including support for digital audio over I2S and HQ2x pixel scaling. This makes the solution stand out among existing video-output add-ons for the ZX Spectrum platform.

Another important feature planned for future software development is LAN (Ethernet) connectivity using the W5500 network controller. On the current hardware revision, both the SD card and LAN interfaces are already fully wired to the FPGA using separate SPI buses. While this approach provides maximum flexibility during development and testing, it also increases FPGA pin usage.

In future revisions, the SPI lines for the SD card and W5500 can be tied together to save pins. This is feasible since both peripherals use the standard SPI protocol and can be selected individually via their dedicated chip select (CS) signals. Such a change would simplify the design and make the expansion module more compact and adaptable to other systems.

To enable LAN functionality, the remaining task is to implement a W5500 SPI driver in the Z80 ROM firmware. This would allow support for useful features such as remote serial debugging, loading *.sna* snapshots over the network, and potentially more advanced networking applications. The groundwork for this functionality is already laid out in the current hardware, and the author plans to continue this development as a next step in the project.

Additionally, the current prototype lacks a finalized enclosure. An improved, custom-designed housing will be created to provide durability, usability, and aesthetics appropriate for a consumer-facing product.

Although the current hardware was tailored to a specific single-copy Leningrad variant with its custom MPH-44 connector, the concept can be extended. With a moderate redesign of the interface logic and connector layout, the same FPGA module can be adapted to support other ZX Spectrum clones, including officially produced 48K and 128K models. In many cases, even simpler connector adapters could enable compatibility, allowing the core FPGA board and firmware to remain unchanged.

In conclusion, this work has demonstrated the technical feasibility and historical relevance of bridging retro computing with modern embedded design. The FPGA-based platform offers both a practical enhancement for users and a flexible development environment for further educational and hobbyist projects. The foundation built here will serve as the basis for continued development and future extensions of the system.

9 References

[1] B. Bertram, “Sinclair 48K ZX Spectrum.” [Online]. Available: https://en.wikipedia.org/wiki/File:ZXSpectrum48k.jpg. [Accessed: 28-Mar-2025]

[2] “List of ZX Spectrum clones.” [Online]. Available: https://en.wikipedia.org/wiki/List_of_ZX_Spectrum_clones. [Accessed: 23-Apr-2025]

[3] T. Gabor, “History of ZX Spectrum cloning.” [Online]. Available: http://users.atw.hu/zxspectrum. [Accessed: 23-Apr-2025]

[4] G. Radan, “ZX-VGA-JOY – ZX Spectrum VGA and Joystick interface.” [Online]. Available: https://web.archive.org/web/20240713051918/https://zx-vga-joy.com/. [Accessed: 22-Apr-2025]

[5] G. Velesoft, “ZX-VGA (SCANDOUBLER).” [Online]. Available: https://velesoft.speccy.cz/zx/zx-vga/. [Accessed: 22-Apr-2025]

[6] V. Trucco, “TK-Pie (a.k.a. ZX-Pie).” [Online]. Available: https://github.com/goloskokovic/ZX-Pie. [Accessed: 22-Apr-2025]

[7] B. Versteeg, “ZX-HD HDMI interface.” [Online]. Available: https://www.bytedelight.com/?page_id=1800. [Accessed: 22-Apr-2025]

[8] “ZX-Uno.” [Online]. Available: https://zxuno.speccy.org/index_e.shtml. [Accessed: 22-Apr-2025]

[9] “ZX Spectrum for MiSTer Platform.” [Online]. Available: https://github.com/MiSTer-devel/ZX-Spectrum_MISTer. [Accessed: 22-Apr-2025]

[10] “ZX Spectrum Next.” [Online]. Available: https://www.specnext.com/. [Accessed: 22-Apr-2025]

[11] B. Bertram, “A Sinclair ZX Spectrum 48K motherboard, issue 3B” [Online]. Available: https://en.wikipedia.org/wiki/File:ZXspectrum_mb.jpg. [Accessed: 28-Mar-2025]

[12] Sinclair Research Ltd., “Sinclair ZX Spectrum Servicing Manual,” Mar. 1984 [Online]. Available: https://worldofspectrum.net/pub/sinclair/technical-docs/ZXSpectrum48K_ServiceManual.pdf. [Accessed: 27-Mar-2025]

[13] D. Stephenson, “Colour Clash: The Engineering Miracle of the Sinclair ZX Spectrum,” *Paleotronic*, pp. 47–48, 2018.

[14] S. Bagan, “Leningrad-48 schematics” [Online]. Available: https://www.cxemateka.ru/v1/leningrad_sch.pdf. [Accessed: 28-Mar-2025]

[15] N. Rodionov and A. Larchenko, *ZX-Spectrum & TR-DOS для пользователей и программистов*. 1994, p. 201.

[16] “Floating bus.” [Online]. Available: https://sinclair.wiki.zxnet.co.uk/wiki/Floating_bus. [Accessed: 28-Mar-2025]

[17] K. Gromov, “System - an article about compatibility and modification issues of domestic ZX Spectrum clones,” *Spectrofon*, 1995.

[18] E. Saukas, “XBifrost* engine - release 1.2,” Jul. 2012 [Online]. Available: https://spectrumcomputing.co.uk/zxdb/sinclair/entries/0027405/BIFROSTENGINEV1.2.txt. [Accessed: 28-Mar-2025]

[19] YosysHQ GmbH, “Yosys Open SYnthesis Suite.” [Online]. Available: https://yosyshq.net. [Accessed: 15-Apr-2025]

[20] Torex Semiconductor Ltd, “XC6210 Series Data Sheet,” no. ETR0317_007 [Online]. Available: https://product.torexsemi.com/system/files/series/xc6210.pdf. [Accessed:

15-Apr-2025]

[21] Texas Instruments, “TLV755P 500mA, Low-IQ, LDO Regulator,” no. SBVS320D, Sep. 2024 [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv755p.pdf. [Accessed: 16-Apr-2025]

[22] Monsonite, “Lattice HX4K resources clarification.” [Online]. Available: https://anycpu.org/forum/viewtopic.php?f=13&t=569#p4074. [Accessed: 21-Apr-2025]

[23] Texas Instruments, “SN74LXC8T245 8-bit Dual-Supply Bus Transceiver Data Sheet,” no. SCES916A, Mar. 2023 [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74lxc8t245.pdf. [Accessed: 16-Apr-2025]

[24] Nexperia B.V., “74AXP4T245 4-bit dual supply translating transceiver Product data sheet,” Feb. 2020 [Online]. Available: https://www.mouser.ee/datasheet/2/916/74AXP4T245-2937302.pdf. [Accessed: 16-Apr-2025]

[25] Nexperia B.V., “IP4251/52/53/54-TTL Product data sheet,” May 2011 [Online]. Available: https://assets.nexperia.com/documents/data-sheet/IP4251_52_53_54-TTL.pdf. [Accessed: 16-Apr-2025]

[26] Integrated Silicon Solution, Inc., “512Kx8 LOW VOLTAGE,ULTRA LOW POWER CMOS STATIC RAM,” Apr. 2017 [Online]. Available: https://www.mouser.ee/datasheet/2/198/62-65WV5128EALL-BLL-737464.pdf. [Accessed: 16-Apr-2025]

[27] CD-R Nederland, “EDID (Extended display identification data).” [Online]. Available: https://www.drhdmi.eu/dictionary/edid.html. [Accessed: 12-Apr-2025]

[28] D. Mayer, “Clarifying HDMI bandwidth,” 12-Aug-2014. [Online]. Available: https://connectedmag.com.au/clarifying-hdmi-bandwidth/. [Accessed: 14-Apr-2025]

[29] T. Verbeure, “Video Timings Calculator.” [Online]. Available: https://tomverbeure.github.io/video_timings_calculator. [Accessed: 12-Apr-2025]

[30] “HDMI Specification Version 1.3a,” 2006 [Online]. Available: https://ez.analog.com/cfs-file/__key/telligent-evolution-components-attachments/00-317-00-00-00-05-21-37/HDMISpecification13a.pdf. [Accessed: 01-Apr-2025]

[31] Altera, “HDMI Intel® FPGA IP User Guide,” p. 11, Mar. 2025 [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/793150?fileName=ug_hdmi-683798-793150.pdf. [Accessed: 13-Apr-2025]

[32] T. Kugelstadt, “Support HDMI 1.3 12-Bit Deep Color With the TMDS341A,” no. SLLA263, p. 1, May 2007 [Online]. Available: https://www.ti.com/lit/an/slla263/slla263.pdf. [Accessed: 12-Apr-2025]

[33] Lattice Semiconductor, “LatticeECP3 HDMI/DVI Interface Reference Design,” no. FPGA-RD-02139-1.6, Jan. 2020 [Online]. Available: https://www.latticesemi.com/-/media/LatticeSemi/Documents/ReferenceDesigns/EI2/FPGA-RD-02139-1-6-HDMI-DVI-Video-Interface-Reference-Design.ashx?document_id=38351. [Accessed: 13-Apr-2025]

[34] Y. Ibrahim, “How to Terminate LVDS Connections,” no. SLAA840, May 2018 [Online]. Available: https://www.ti.com/lit/ab/slaa840/slaa840.pdf. [Accessed: 13-Apr-2025]

[35] C. Rice, “AC versus DC Coupling - What’s the difference?,” 29-Aug-2019. [Online]. Available: https://community.sw.siemens.com/s/article/ac-and-dc-coupling-what-s-the-difference. [Accessed: 17-Apr-2025]

[36] NXP Semiconductors, “PTN3363 Product data sheet,” Aug. 2014 [Online]. Available:

https://www.nxp.com/docs/en/data-sheet/PTN3363.pdf. [Accessed: 14-Apr-2025]

[37] Lattice Semiconductor, “Using Differential I/O(LVDS, Sub-LVDS)in iCE40 LP/HX Devices,” no. TN1253, Jan. 2015 [Online]. Available: http://www.latticesemi.com/~/media/LatticeSemi/Documents/ApplicationNotes/UZ/UsingDifferentialIOLVDSSubLVDSiniCE40Devices.pdf. [Accessed: 14-Apr-2025]

[38] “CCTV Tester with HDMI and VGA Input.” [Online]. Available: https://vi.aliexpress.com/item/1005007046594345.html. [Accessed: 14-Apr-2025]

[39] “ Mouser Electronics.” [Online]. Available: https://www.mouser.ee. [Accessed: 15-Apr-2025]

[40] Altium LLC, “The electronic part search engine.” [Online]. Available: https://octopart.com/. [Accessed: 15-Apr-2025]

[41] Texas Instruments, “TDP158 6Gbps, AC-Coupled to TMDS Level Shifter Redriver,” no. SLLSEX2F, Apr. 2024 [Online]. Available: https://www.ti.com/lit/ds/symlink/tdp158.pdf. [Accessed: 15-Apr-2025]

[42] Texas Instruments, “SN75DP129 Data Sheet,” no. SLAS583A, Mar. 2008 [Online]. Available: https://www.ti.com/lit/ds/symlink/sn75dp129.pdf. [Accessed: 15-Apr-2025]

[43] “SiI9022A/SiI9024A HDMI Transmitter,” *Lattice Semiconductor*, no. SiI-DS-1076-E.01, Aug. 2016 [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Lattice%20PDFs/SiI9022A,24A_Aug-2016.pdf. [Accessed: 14-Apr-2025]

[44] NXP B.V, “TDA19988 Product data sheet,” Jul. 2011 [Online]. Available: https://www.mouser.com/datasheet/2/302/NXP_TDA19988-1189083.pdf. [Accessed: 14-Apr-2025]

[45] Texas Instruments, “TFP410 TI PanelBus^{TM} Digital Transmitter,” no. SLDS145D, Feb. 2024 [Online]. Available: https://www.ti.com/lit/ds/symlink/tfp410.pdf. [Accessed: 18-Apr-2025]

[46] Silicon Image. Inc., “SiI164 PanelLink Transmitter Data Sheet,” no. SiI-DS-0021-E, Jun. 2005 [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/799/SiI_164_PanelLink_Trans_Jun2005.pdf. [Accessed: 14-Apr-2025]

[47] Analog Devices, Inc., “ADV7513 Data Sheet,” Nov. 2011 [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADV7513.pdf. [Accessed: 15-Apr-2025]

[48] S. Puri, “Send video/audio over HDMI on an FPGA.” [Online]. Available: https://github.com/hdl-util/hdmi. [Accessed: 18-Apr-2025]

[49] C. Svensson, “The Effect of CRTs on Pixel Art,” 2024. [Online]. Available: https://datagubbe.se/crt/. [Accessed: 20-Apr-2025]

[50] N. Berry, “Pixel Scalers,” Dec-2013. [Online]. Available: http://datagenetics.com/blog/december32013/index.html. [Accessed: 20-Apr-2025]

[51] J. Conejero, “Interpolation Algorithms in PixInsight.” [Online]. Available: https://pixinsight.com/doc/docs/InterpolationAlgorithms/InterpolationAlgorithms.html. [Accessed: 20-Apr-2025]

[52] Z. Asghar, R. Naeem, and N. K. Jarral, “Correlative analysis of different image scaling algorithms in graphics,” *Int. J. Adv. Res. Comput. Sci. Electron*, vol. 6, pp. 6–10, 2017 [Online]. Available: https://api.semanticscholar.org/CorpusID:64363397. [Accessed: 20-Apr-2025]

[53] “Pixel-art scaling algorithms.” [Online]. Available:

https://en.wikipedia.org/wiki/Pixel-art_scaling_algorithms. [Accessed: 20-Apr-2025]

[54] “hqx (algorithm).” [Online]. Available: https://en.wikipedia.org/wiki/Hqx_(algorithm). [Accessed: 20-Apr-2025]

[55] “Comparison gallery of image scaling algorithms.” [Online]. Available: https://en.wikipedia.org/wiki/Comparison_gallery_of_image_scaling_algorithms. [Accessed: 20-Apr-2025]

[56] M. Stepin, “hq2x Magnification Filter.” [Online]. Available: https://web.archive.org/web/20131029012017/http://www.hiend3d.com/hq2x.html. [Accessed: 22-Apr-2025]

[57] Drummyfish, *Pixel-Art Scaling Comparison*. 2018 [Online]. Available: https://commons.wikimedia.org/wiki/File:Pixel-Art_Scaling_Comparison.png. [Accessed: 22-Apr-2025]

[58] B. Costa, “hqx algorithms implementation.” [Online]. Available: https://github.com/brunexgeek/hqx. [Accessed: 20-Apr-2025]

[59] C. Bœsch, “Butchering HQX scaling filters,” 21-Jun-2014. [Online]. Available: https://blog.pkh.me/p/19-butchering-hqx-scaling-filters.html. [Accessed: 22-Apr-2025]

[60] T. Malche, “Edge Detection in Image Processing: An Introduction,” *Roboflow Blog*, Jun. 2024 [Online]. Available: https://roboflow.com/blog/edge-detection. [Accessed: 20-Apr-2025]

[61] L. Strigeus, “The world’s most compact HQ2X in Verilog?,” 01-Feb-2013. [Online]. Available: https://fpganes.blogspot.com/2013/02/the-worlds-most-compact-hq2x-in-verilog.html. [Accessed: 20-Apr-2025]

[62] L. Strigeus, “FPGA NES.” [Online]. Available: https://github.com/strigeus/fpganes/blob/master/src/hq2x.v. [Accessed: 20-Apr-2025]

[63] YosysHQ GmbH, “Memory mapping.” [Online]. Available: https://yosyshq.readthedocs.io/projects/yosys/en/0.33/CHAPTER_Memorymap.html#simple-dual-port-sdp-memory-patterns. [Accessed: 21-Apr-2025]

[64] G. Stitt, “Optimizing Hardware For FPGAs,” 15-Aug-2024. [Online]. Available: https://stitt-hub.com/optimizing-hardware-for-fpgas. [Accessed: 21-Apr-2025]

[65] E. Chan, “Petit FAT File System Module,” 2013. [Online]. Available: https://elm-chan.org/fsw/ff/00index_p.html. [Accessed: 10-May-2025]

[66] E. Chan, “How to Use MMC/SDC,” 26-Dec-2019. [Online]. Available: https://elm-chan.org/docs/mmc/mmc_e.html. [Accessed: 10-May-2025]

[67] “SNA snapshot format.” [Online]. Available: https://worldofspectrum.net/zx-modules/fileformats/snaformat.html. [Accessed: 10-May-2025]

[68] Yamaha Corporation, “YM2149 SSG Data Sheet,” 1987. [Online]. Available: https://www.ym2149.com/ym2149.pdf. [Accessed: 29-Apr-2025]

[69] E. Brindley, “AY vs YM sound IC differences,” 05-Apr-2018. [Online]. Available: https://maidavale.org/blog/ay-ym-differences. [Accessed: 29-Apr-2025]

[70] O. Poncet, “About the AY-3-8910 and the YM2149,” 21-Sep-2023. [Online]. Available: https://aym-js.emaxilde.net/about. [Accessed: 29-Apr-2025]

[71] MikeJ and A. Melnikov, “YM2149 verilog core,” 2005. [Online]. Available: https://github.com/sorgelig/ZX_Spectrum-128K_MIST/blob/master/ym2149.sv. [Accessed: 29-Apr-2025]

[72] J. Tejada, “JT49 FPGA Clone of YM2149.” [Online]. Available:

https://github.com/jotego/jt49. [Accessed: 29-Apr-2025]

[73] R. S. Walz, “AY-3-8912,” 1995. [Online]. Available: https://web.archive.org/web/20241216035717/http://www.armory.com/~rstevew/Public/SoundSynth/Novelty/AY3-8910/start.html. [Accessed: 29-Apr-2025]

[74] G. Hunter, “I2S Communication Protocol,” 05-Sep-2024. [Online]. Available: https://blog.mbedded.ninja/electronics/communication-protocols/i2s-communication-protocol/. [Accessed: 29-Apr-2025]

[75] Nordic Semiconductor ASA, “I2S — Inter-IC sound interface,” 12-Feb-2025. [Online]. Available: https://docs.nordicsemi.com/bundle/ps_nrf5340/page/i2s.html. [Accessed: 29-Apr-2025]

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

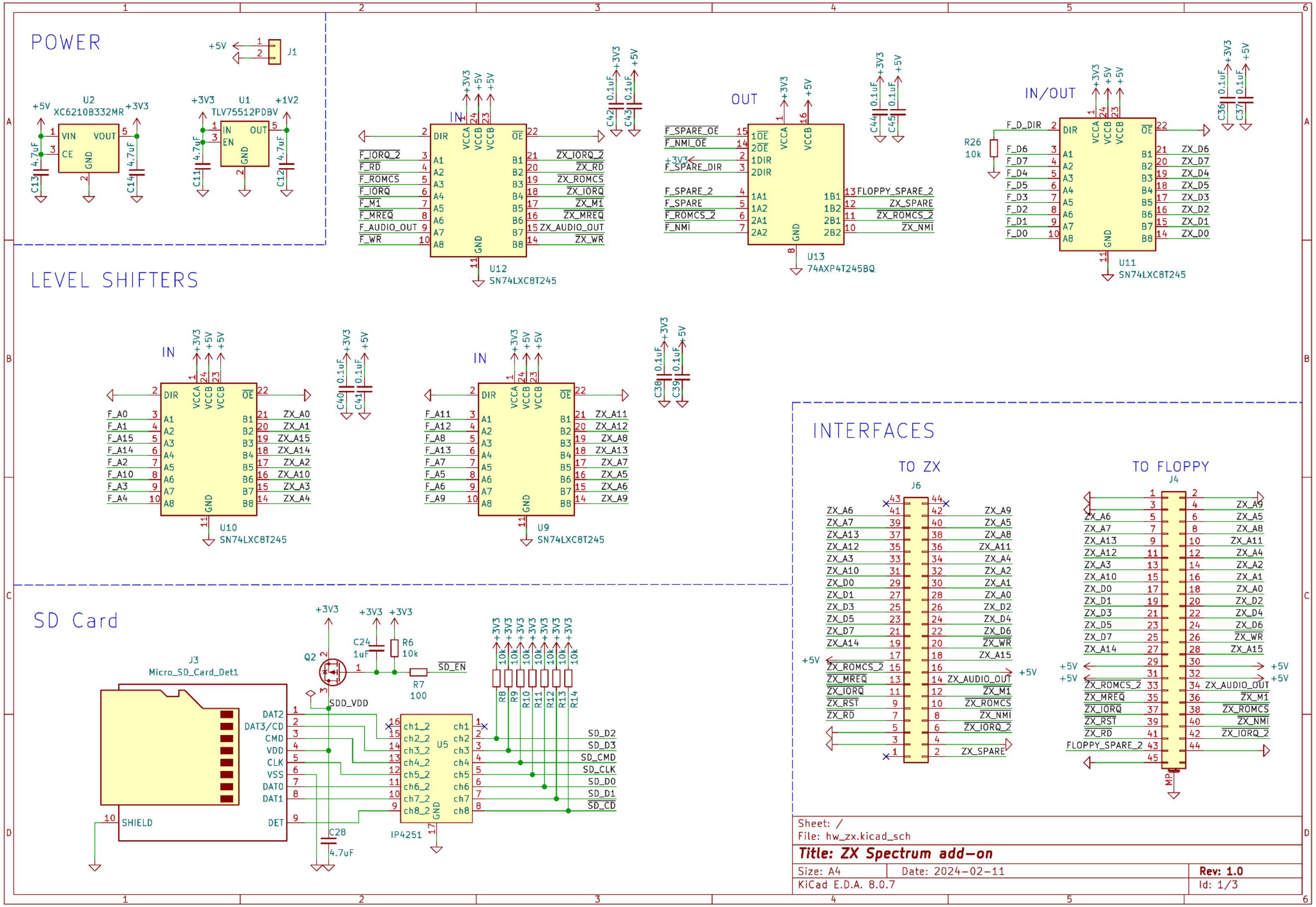
I Siim Salonen

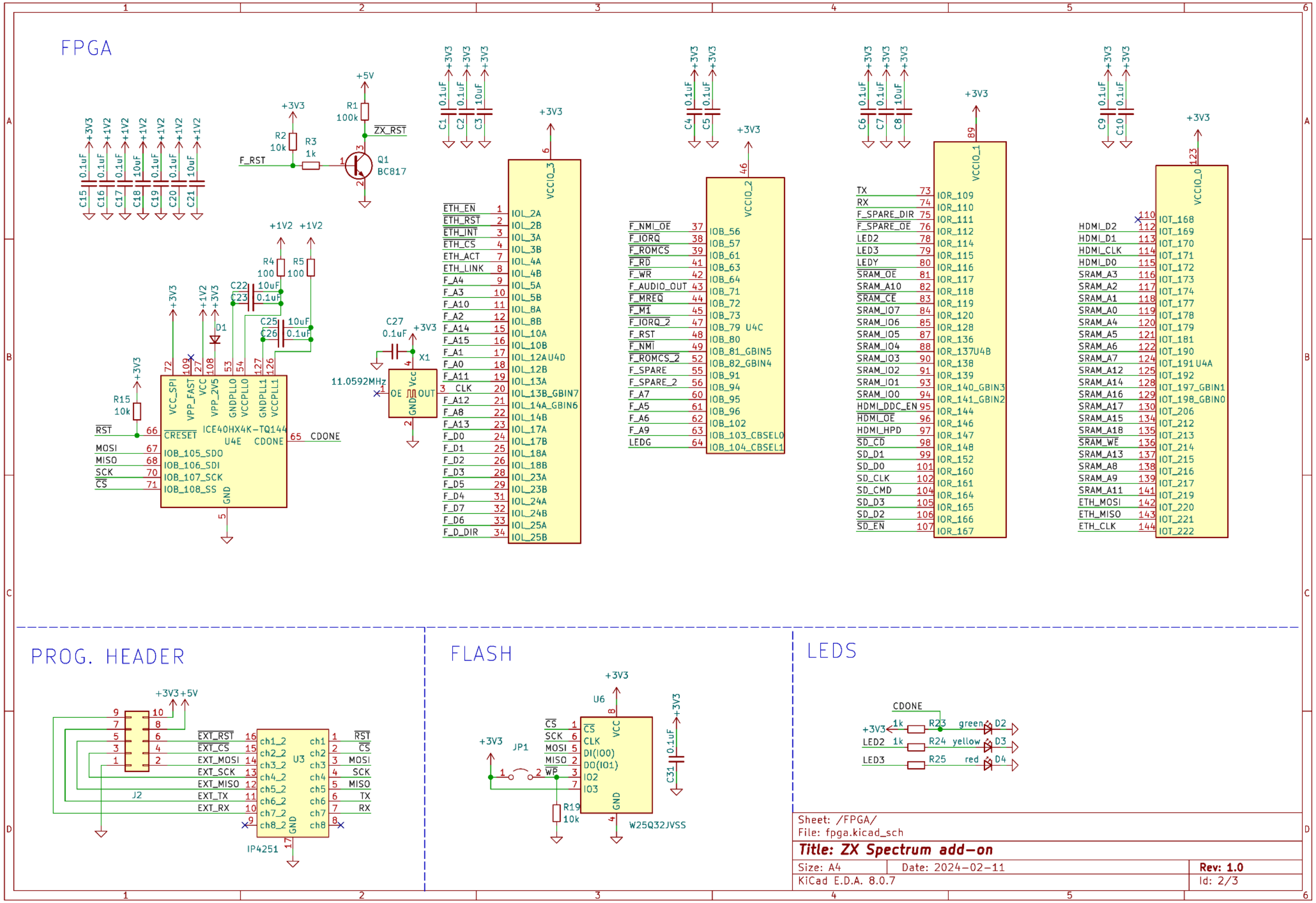
1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “From 8-Bit To Hdmi: Enhancing A Z80-Based Computer Using Fpga Technology”, supervised by Tara Ghasempouri
   1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

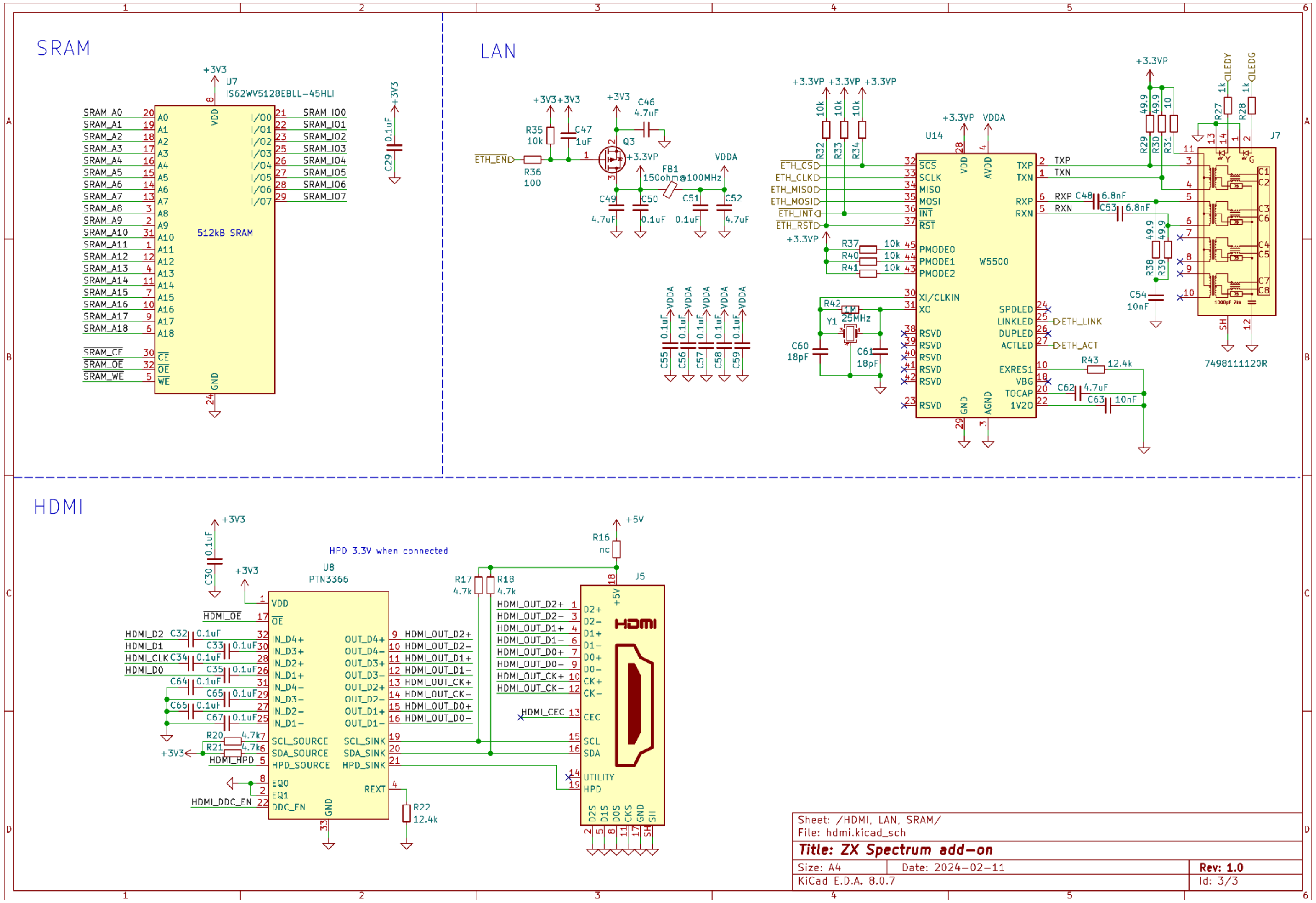
11.05.2025

[1] *The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.*

Appendix 2 – Circuit diagram




FPGA
C15 0.1uF +1V2
C16 0.1uF +1V2
C17 0.1uF +1V2
C18 10uF +1V2
C19 0.1uF +1V2
C20 0.1uF +1V2
C21 10uF +1V2
+3V3
+5V
R1 100k
ZX_RST
R2 10k
R3 1k
F_RST
Q1 BC817
+1V2 +1V2
R4 100
R5 100
C22 10uF
C23 0.1uF
C25 10uF
C26 0.1uF
D1
R15 10k
RST
MOSI
MISO
SCK
CS
VCC_SPI
VPP_FAST
VCC
VPP_2V5
GNDPLL0
VCCPLL0
GNDPLL1
VCCPLL1
CRESET
ICE40HX4K-TQ144
U4E
CDONE
IOB_105_SDO
IOB_106_SDI
IOB_107_SCK
IOB_108_SS
GND
C27 0.1uF
X1
11.0592MHz
OE
OUT
Vcc
GND
CLK
C1 0.1uF
C2 0.1uF
C3 10uF
VCCIO_3
ETH_EN
ETH_RST
ETH_INT
ETH_CS
ETH_ACT
ETH_LINK
F_A4
F_A3
F_A10
F_A2
F_A14
F_A15
F_A1
F_A0
F_A11
F_A12
F_A8
F_A13
F_D0
F_D1
F_D2
F_D3
F_D5
F_D4
F_D7
F_D6
F_D_DIR
IOL_2A
IOL_2B
IOL_3A
IOL_3B
IOL_4A
IOL_4B
IOL_5A
IOL_5B
IOL_8A
IOL_8B
IOL_10A
IOL_10B
IOL_12AU4D
IOL_12B
IOL_13A
IOL_13B_GBIN7
IOL_14A_GBIN6
IOL_14B
IOL_17A
IOL_17B
IOL_18A
IOL_18B
IOL_23A
IOL_23B
IOL_24A
IOL_24B
IOL_25A
IOL_25B
C4 0.1uF
C5 0.1uF
VCCIO_2
F_NMI_OE
F_IORQ
F_ROMCS
F_RD
F_WR
F_AUDIO_OUT
F_MREQ
F_M1
F_IORQ_2
F_RST
F_NMI
F_ROMCS_2
F_SPARE
F_SPARE_2
F_A7
F_A5
F_A6
F_A9
LEDG
IOB_56
IOB_57
IOB_61
IOB_63
IOB_64
IOB_71
IOB_72
IOB_73
IOB_79 U4C
IOB_80
IOB_81_GBIN5
IOB_82_GBIN4
IOB_91
IOB_94
IOB_95
IOB_96
IOB_102
IOB_103_CBSEL0
IOB_104_CBSEL1
C6 0.1uF
C7 0.1uF
C8 10uF
VCCIO_1
TX
RX
F_SPARE_DIR
F_SPARE_OE
LED2
LED3
LEDY
SRAM_OE
SRAM_A10
SRAM_CE
SRAM_IO7
SRAM_IO6
SRAM_IO5
SRAM_IO4
SRAM_IO3
SRAM_IO2
SRAM_IO1
SRAM_IO0
HDMI_DDC_EN
HDMI_OE
HDMI_HPD
SD_CD
SD_D1
SD_D0
SD_CLK
SD_CMD
SD_D3
SD_D2
SD_EN
IOR_109
IOR_110
IOR_111
IOR_112
IOR_114
IOR_115
IOR_116
IOR_117
IOR_118
IOR_119
IOR_120
IOR_128
IOR_136
IOR_137U4B
IOR_138
IOR_139
IOR_140_GBIN3
IOR_141_GBIN2
IOR_144
IOR_146
IOR_147
IOR_148
IOR_152
IOR_160
IOR_161
IOR_164
IOR_165
IOR_166
IOR_167
C9 0.1uF
C10 0.1uF
VCCIO_0
HDMI_D2
HDMI_D1
HDMI_CLK
HDMI_D0
SRAM_A3
SRAM_A2
SRAM_A1
SRAM_A0
SRAM_A4
SRAM_A5
SRAM_A6
SRAM_A7
SRAM_A12
SRAM_A14
SRAM_A16
SRAM_A17
SRAM_A15
SRAM_A18
SRAM_WE
SRAM_A13
SRAM_A8
SRAM_A9
SRAM_A11
ETH_MOSI
ETH_MISO
ETH_CLK
IOT_168
IOT_169
IOT_170
IOT_171
IOT_172
IOT_173
IOT_174
IOT_177
IOT_178
IOT_179
IOT_181
IOT_190
IOT_191 U4A
IOT_192
IOT_197_GBIN1
IOT_198_GBIN0
IOT_206
IOT_212
IOT_213
IOT_214
IOT_215
IOT_216
IOT_217
IOT_219
IOT_220
IOT_221
IOT_222
PROG. HEADER
+3V3 +5V
J2
EXT_RST
EXT_CS
EXT_MOSI
EXT_SCK
EXT_MISO
EXT_TX
EXT_RX
ch1_2
ch2_2
ch3_2
ch4_2
ch5_2
ch6_2
ch7_2
ch8_2
U3
ch1
ch2
ch3
ch4
ch5
ch6
ch7
ch8
GND
RST
CS
MOSI
SCK
MISO
TX
RX
IP4251
FLASH
U6
JP1
CS
SCK
MOSI
MISO
WP
R19 10k
CS
CLK
DI(IO0)
DO(IO1)
IO2
IO3
VCC
GND
W25Q32JVSS
C31 0.1uF
LEDS
CDONE
+3V3 1k R23 green D2
LED2 1k R24 yellow D3
LED3 R25 red D4
Sheet: /FPGA/
File: fpga.kicad_sch
Title: ZX Spectrum add-on
Size: A4
Date: 2024-02-11
Rev: 1.0
KiCad E.D.A. 8.0.7
Id: 2/3

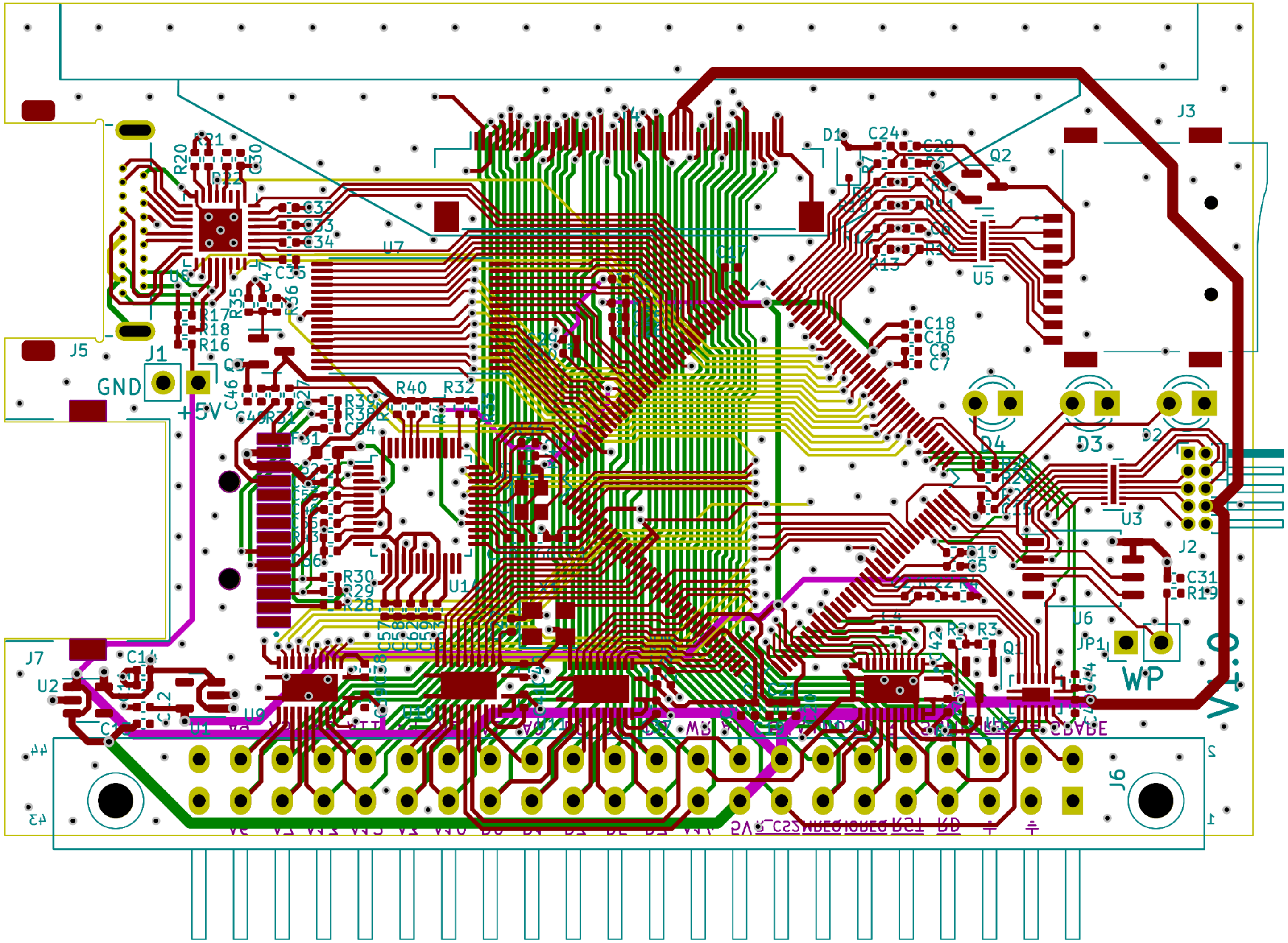


Sheet: /HDMI, LAN, SRAM/
File: hdmi.kicad_sch

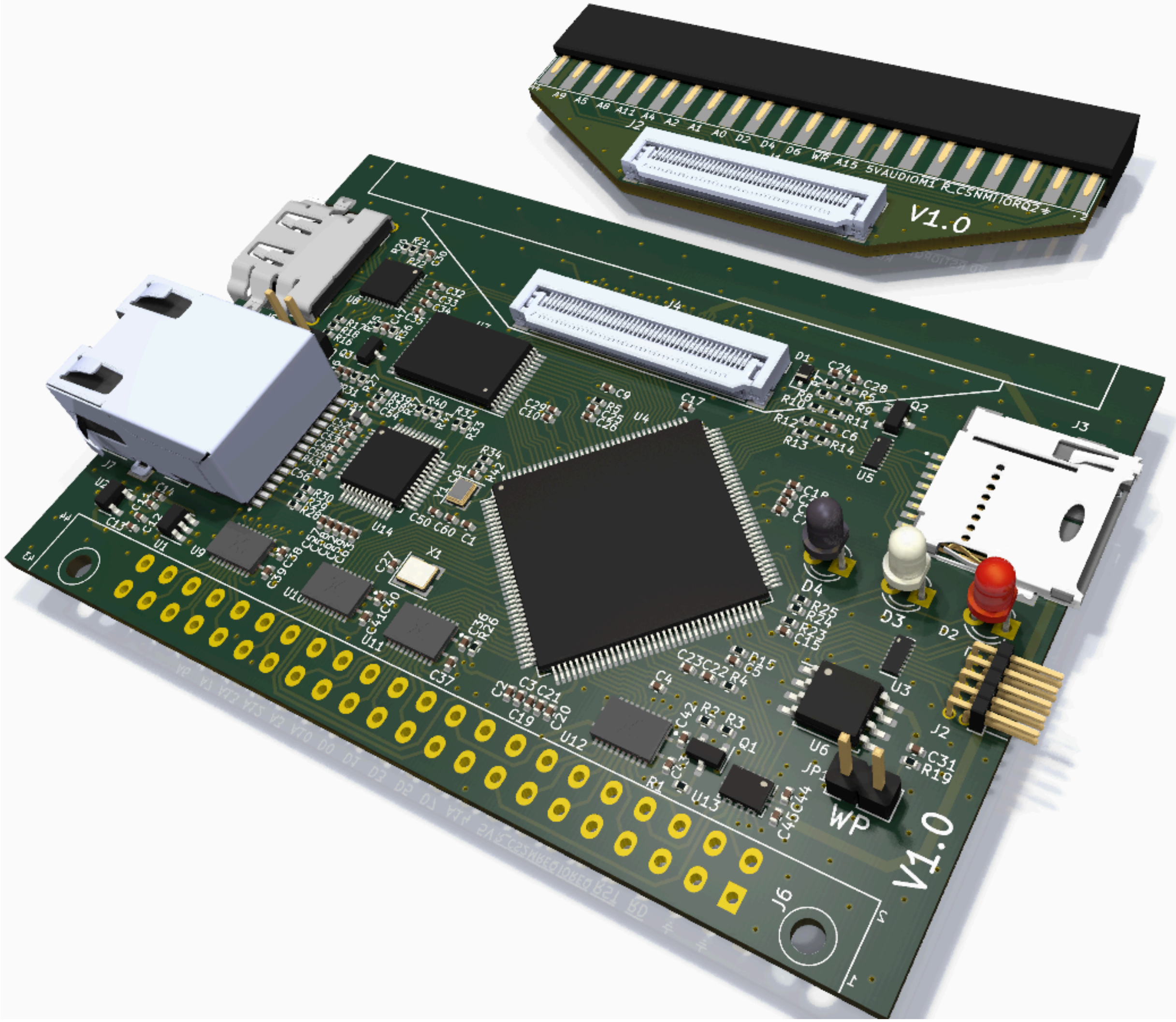
Title: ZX Spectrum add-on

Size: A4 Date: 2024-02-11 **Rev: 1.0**
KiCad E.D.A. 8.0.7 Id: 3/3

Appendix 3 – PCB layout



Appendix 4 – 3D render of ZX Spectrum Add-On


A9 A5 A8 A11 A4 A2 A1 A0 D2 D4 D6 WR A15 5V AUDIO M1 R_CS NMI IORQ2
J2
V1.0
J4
J3
U4
U5
U6
U3
U12
U13
U14
Q1
Q2
D1
D2
D3
D4
X1
J2
JP1
WP
J6
V1.0