Development Board for RP2350 Pico2 with HDMI, DAC, Power Supply, and Keypad Interface

MEng Project Report

Authors: Bole Ding
Haotian Liu
Advisors: Van Hunter Adams
Bruce Robert Land

Degree Date: Dec 2026
Cornell Duffield Engineering

1. ABSTRACT

Master of Engineering

School of Electrical and Computer Engineering, Cornell University

Design Project Title: Development Board for RP2350 Pico2 with HDMI, DAC, Power Supply, and Keypad Interface

Authors: Bole Ding (bd467) and Haotian Liu (hl2584)

We sincerely thank Prof. Van Hunter Adams and Prof. Bruce Robert Land for their guidance, feedback, and continued support throughout this project.

This project demonstrates the successful implementation of high-definition video output on an embedded system powered by the Raspberry Pi Pico 2 (RP2350) and—on the explicit recommendation of Prof. Van Hunter Adams and Prof. Bruce Robert Land—the porting of the Cornell ECE 4760 VGA Graphics library to drive that output over HDMI rather than the original analog VGA. Using the Pico-DVI-Sock reference circuit as a hardware baseline, we engineered a custom 2-layer PCB with optimised high-speed differential routing for stable real-time output on standard digital displays. On top of this hardware, a thin compatibility layer reproduces the public API of Prof. Adams' RP2040 VGA library (drawPixel, drawLine, drawHLine/drawVLine, drawRect, fillRect, drawCircle, fillCircle, setTextColor, setCursor, writeChar, writeString, and the standard 16-colour constant set) but routes every primitive into the RGB332 framebuffer that feeds the RP2350's HSTX serializer. The result is that existing ECE 4760 graphical demos can be retargeted to this board with no source-level changes while gaining the higher resolution, cleaner colour, and modern-display compatibility of digital HDMI. The development process followed a modular hardware-software co-design approach, bridging the gap between the microcontroller's HSTX peripheral and a physical HDMI/DVI interface to deliver a low-cost, high-performance multimedia prototyping platform that is also drop-in compatible with the existing ECE 4760 graphics codebase.

Key Words: Raspberry Pi Pico 2 (RP2350), Embedded System, DVI Sock, PCB Design, HDMI, ECE 4760 VGA Graphics Library, API Compatibility Shim, Real-Time Output

2. EXECUTIVE SUMMARY

This project focused on the design, fabrication, and testing of a Raspberry Pi Pico 2 HDMI extension board. The main goal was to build a compact hardware platform that enables video output from the Pico 2 through an HDMI connector and displays graphics on an external

monitor. Because the Pico 2 does not include a built-in HDMI interface, the work required both a custom PCB design and dedicated embedded software development, making it a true hardware-software integration exercise.

We designed and tested a Pico 2 extension board with full HDMI output capability and verified stable 640×480 RGB pixel output on a monitor. Both static image display and video playback were tested using the Pico DVI output method. The test firmware evolved: it began with simple display patterns and was gradually improved into more substantial demonstrations, including animations and user-selected video input. By the end of testing, the assembled PCBA was confirmed to operate correctly, establishing it as a functional video-output platform for future development.

The primary challenge involved assembling the HDMI interface itself. The HDMI socket has small, densely packed pins, which made manual soldering unreliable and risky. To reduce this risk, we ordered a fully assembled PCBA from JLCPCB rather than solder the connector by hand, which significantly improved hardware bring-up reliability. The testing setup also caused difficulty: with male pin headers unavailable at the time, the board could not be inserted directly into a breadboard, and jumper wires were used instead. They worked but made debugging less convenient and frequently introduced loose connections. On the software side, the initial test program was too simple to fully demonstrate the system's capabilities. We later expanded the firmware to drive animations and play video, which required converting visual data into the 640×480 RGB pixel format that the Pico 2 could output.

The board successfully outputs HDMI video from the Pico 2 extension to an external monitor at 640×480 RGB resolution, with both image display and video playback verified. It operated stably enough for demonstration, with no significant signal issues across repeated trials. The current version validates the video-output portion of the larger development board concept. Additional features, such as DAC-based audio output or keypad input, are left for future revisions. However, the most technically challenging part of the project, HDMI video output, was completed and verified, which represents the core deliverable.

The project produced several valuable lessons. It became clear that manufacturability matters as much as circuit design. Even when the schematic and PCB layout are correct, small components like HDMI sockets can create major assembly obstacles, and professional PCBA assembly proved to be a practical solution for improving reliability. The project also highlighted the importance of test access. An electrically correct board can still be hard to debug when its physical connections are awkward, so header placement and probing points deserve attention early in the design process. From the software perspective, a strong demonstration program is essential for hardware validation. Simple test patterns are useful at first, but image and video playback offer much stronger evidence that the system truly works. Overall, this project provided meaningful hands-on experience in PCB design, high-speed digital routing, firmware development, and hardware-software integration.

3. INTRODUCTION & PROBLEM DEFINITION

3.1 Problem Statement

While the RP2350 (Pico 2) microcontroller introduces powerful new hardware peripherals, most notably the HSTX (High-Speed Serial Transmit), the standard Pico 2 development board form factor is not physically equipped to utilize them for high-speed digital video. Achieving a stable HDMI/DVI connection requires a specific differential signaling network, controlled impedance, and proper shielding that breadboards and standard headers cannot provide.

3.2 Motivation and Background

The motivation for this project is to democratize high-resolution output for embedded systems. Traditionally, driving a 640x480 digital display required expensive FPGAs or high-end Application Processors. The RP2350 changes this dynamic by integrating high-speed serializers into an affordable MCU.

By designing a dedicated development board, we aim to consolidate the "DVI Sock" concept with professional-grade features such as integrated audio via a DAC and a tactile keypad interface. This creates an all-in-one workstation for students and engineers to prototype complex Human-Machine Interfaces (HMI) without the overhead of external hardware.

3.3 System Requirements

Functional Requirements

- **Video Output:** Must provide a standard HDMI/DVI-D interface capable of driving external monitors.
- **Audio Output:** Integration of a dedicated Digital-to-Analog Converter (DAC) for high-fidelity sound.
- **User Input:** A matrix keypad interface to allow for standalone user interaction.
- **Power Management:** An onboard regulation system to convert 5V USB input into a stable 3.3V rail for the RP2350 core.

Performance Requirements

- **Resolution:** Support for a stable 640x480 resolution at a 60Hz refresh rate.
- **Latency:** The system must maintain real-time video synchronization with negligible CPU overhead, leveraging the DMA and HSTX hardware.
- **Power Stability:** Voltage ripple must remain below 50mV to ensure high-speed switching stability.

Constraints

- **Area:** The PCB footprint should remain compact enough for prototyping use, prioritizing a logical layout of the multimedia "sections."
- **Layer Count:** The design is restricted to a 2-layer PCB stackup, making high-speed routing and ground referencing a significant design challenge.
- **Tools:** Development is constrained to the Pico SDK, using Microsoft Visual Studio Code for development and build management.
- **Time:** The primary video output deliverable must be validated and fabricated within a single academic semester.

3.4 Evolution of Requirements from Semester #1 to Semester #2

The requirements for this project shifted as we moved from theoretical design to physical validation. This evolution represents a transition from proving the RP2350**'s** high-speed capabilities to creating a fully integrated user experience.

3.4.1 Semester #1 – The Foundation

The primary objective was the "Minimal Viable Product" (MVP)—functional HDMI video output. Success was defined by the ability to generate a stable 640x480 signal on a 2-layer PCB. During this phase, secondary features like the DAC and Keypad were treated as secondary "stretch goals" while we prioritized the high-risk high-speed routing and JLCPCB fabrication process.

3.4.2 Semester #2 – The Multimodal Integration

With the video pipeline validated by the first prototype, the requirements for the upcoming semester have expanded. The focus shifts from "signal generation" to "system interaction." This includes the integration of high-fidelity audio via the I2S DAC and the implementation of the keypad interface. Additionally, requirements now include scalability and modularity, moving the board toward a standalone "PC-style" development platform rather than just a video demo board.

3.5 Summary of Design Specifications (bullet list)

Feature	Specification
Microcontroller	Raspberry Pi RP2350 (Dual-core ARM Cortex-M33)
Video Protocol	DVI-D over HDMI Connector (using HSTX peripheral)

Resolution	640x480 pixels @ 60Hz
PCB Stackup	2-Layer FR-4, 1.6mm thickness
Audio Interface	Dedicated I2S DAC (integrated for Semester 2 validation)
Input Interface	Matrix Keypad / GPIO Button Array
Power Input	5V DC via USB-C or Terminal Block
Onboard Regulation	3.3V (IO) Switching/LDO Regulators

4. RELATED WORK / PRIOR ART

Several prior efforts have explored low-cost video output from small microcontrollers, and this project draws directly on that lineage. The most influential hardware reference is the Pico-DVI-Sock by Wren6991, which demonstrated that the RP2040 could drive DVI signals through careful use of its PIO peripheral. Our work builds on this baseline but moves it onto the RP2350, where the dedicated HSTX peripheral replaces overclocked PIO state machines and frees the CPU for other tasks. The most influential software reference is Prof. Hunter Adams' Cornell ECE 4760 VGA Graphics library, which exposes a small, stable C drawing API—drawPixel, drawLine, drawRect, fillRect, drawCircle, drawChar, writeString, and a set of colour and cursor helpers—that is already used in dozens of student projects across multiple semesters of ECE 4760. A central design choice in this project, made explicitly on the recommendation of Prof. Adams and Prof. Land, was to preserve that exact API surface but route its primitives through the new HDMI/HSTX backend instead of the original PIO + resistor-ladder DAC analog-VGA backend. Figure 1 shows an example of the VGA library in use.

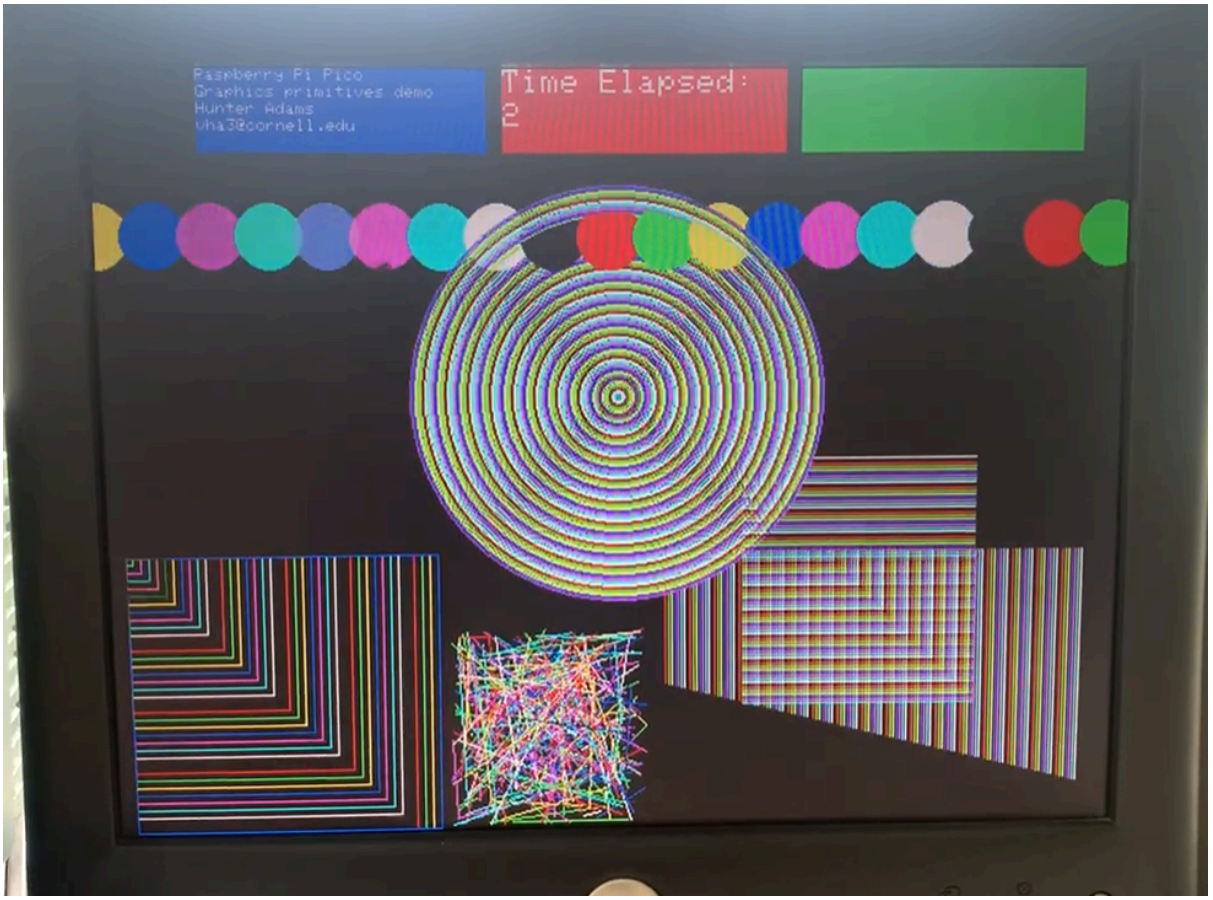


Figure 1. VGA library example

Prof. Adams' RP2040 VGA Graphics demos take a parallel hardware approach using analog VGA generated through a resistor-ladder DAC, and they serve as both an educational reference for low-level frame buffer management and as the source of the drawing API that we re-implement on top of HDMI. By keeping the public API byte-for-byte compatible with the ECE 4760 library, code such as boids simulations, audio FFT visualisers, particle systems, and the graphics-heavy lab assignments distributed with the course compiles unchanged against this board's headers and renders correctly through the HDMI port. At the other end of the spectrum, FPGA-based platforms remain the standard for serious embedded digital video work, offering robust signal integrity at the cost of significantly higher price, larger area, and a steeper learning curve.

Our work sits between these extremes and is best understood as a hardware-software bridge: it is more capable than a passive HDMI breakout because it integrates the connector, power tree, headers, and DAC needed for system-level experiments; it is far more approachable than an FPGA platform; and—uniquely among Pico-class HDMI projects we are aware of—it ships with a drop-in re-implementation of the Cornell ECE 4760 VGA Graphics API. This positioning shaped the later design choices around PCB stackup, connector layout, framebuffer format, and headroom for audio and input expansion.

5. SEMESTER #1 PROGRESS

5.1 Initial Design Ideas

The project was conceived as a "Super-Pico" workstation, a single-board solution that would eliminate the "nest of wires" typically found in complex embedded prototypes.

5.2 Architecture Exploration

The exploration phase focused on bridging the gap between the RP2350 chip's native capabilities and the physical limitations of the standard Raspberry Pi Pico 2 form factor. While the RP2350 supports digital video output, the standard Pico 2 lacks the high-speed differential signaling network necessary for HDMI.

- **PCB Layer Strategy:** A major architectural decision involved choosing between a 2-layer and 4-layer PCB stackup. While a 4-layer board offers superior noise isolation and EMI robustness, we opted for a 2-layer design to prioritize cost-effectiveness and manufacturability for educational environments. This required a more rigorous exploration of **trace geometry** to maintain signal integrity without the benefit of dedicated internal power and ground planes.
- **Protocol Baseline:** We utilized the "DVI Sock" reference circuit as a baseline. This provided a proven hardware starting point for signal generation, which we then adapted and integrated into our custom board layout to support a standard HDMI connector interface. Figure 2 shows the initial testing procedure using the DVI sock.

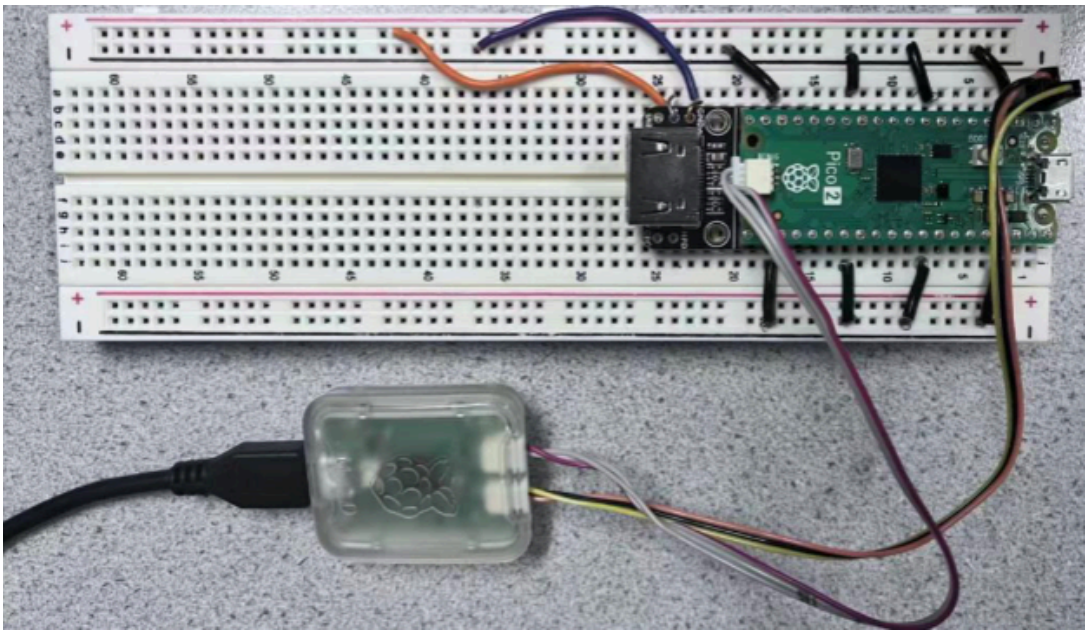


Figure 2. Initial system testing with DVI sock

- **Development Environment:** We evaluated software entry points, initially utilizing the Arduino IDE for rapid proof-of-concept testing before migrating to VS Code with Pico extensions. This transition was essential to gain finer control over the Pico SDK, DMA configurations, and the HSTX peripheral registers.

5.3 Early Prototypes / Models

The prototyping process followed an incremental "risk-reduction" pipeline, moving from software simulation to physical hardware.

- **Firmware Migration Model:** Early code was developed in the Arduino environment to verify basic GPIO toggling and clock configuration. Once the complexity increased—specifically regarding 640x480 resolution timing—the project was migrated to VS Code. This allowed for better debugging of the pixel clock and synchronization pulses.
- **The Breadboard-to-PCB Transition:** Initial hardware validation was performed using a Pico 2 connected via jumper wires to an HDMI breakout. Though signal integrity was poor in this environment, it confirmed the HSTX peripheral could successfully handshake with a monitor.

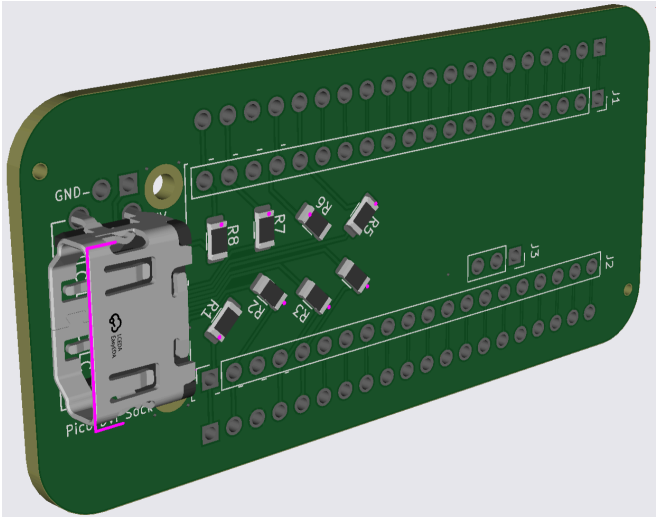


Figure 3. Printed circuit board Gerber file for production

- **Design-to-Gerber Workflow:** We moved from schematic capture to detailed PCB layout, focusing on the high-speed differential network. The generated Gerber files (shown in Figure 3) served as a final "logical model," where we verified that trace lengths for the TMDS channels were matched to prevent phase skew before committing to fabrication.

5.4 Challenges and Issues

The primary challenge was achieving stable, high-speed transmission on a 2-layer substrate, which is traditionally prone to signal degradation.

- **Impedance Control on 2-Layers:** Without a 4-layer stackup's close-proximity ground planes, managing the differential impedance was difficult. We addressed this

through "careful routing" and specific trace width/spacing calculations, though the lack of advanced impedance-controlled stackups made the design sensitive to external EMI. Figure 4 shows the resistance value (270 Ω) we chose for the differential signal pairs, a balanced choice offering both reasonable noise immunity and power efficiency.

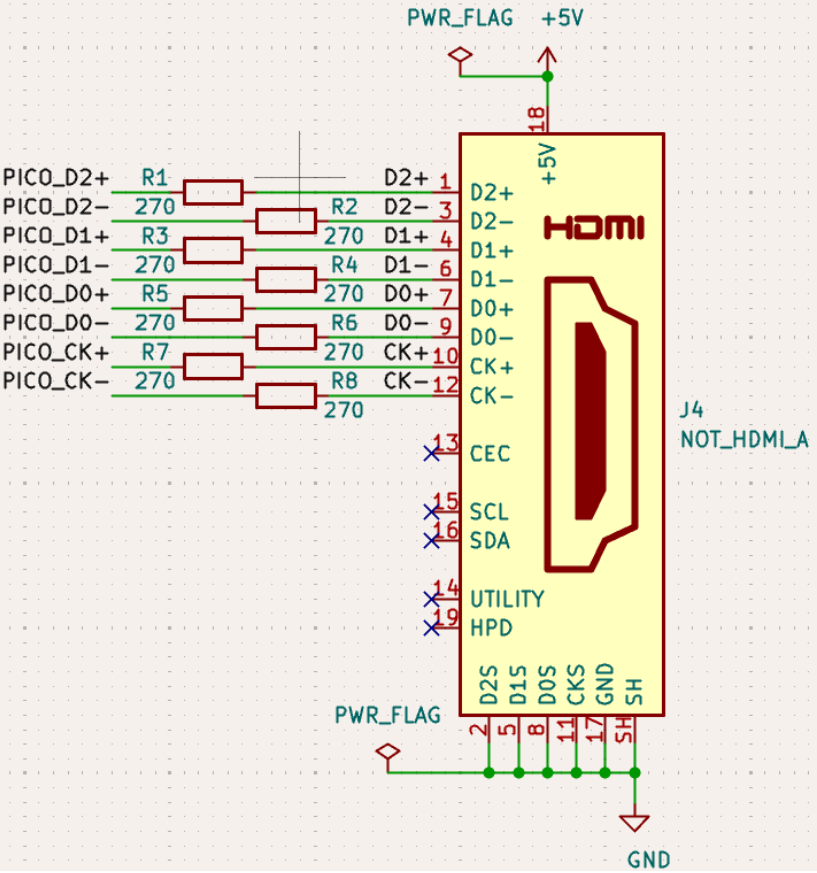


Figure 4. Resistance around the HDMI interface

- **Signal Integrity and Noise:** During early PCB testing, we monitored for flickering and "sparkle" artifacts. The challenge was ensuring that the high-speed TMDS clock did not couple noise into the 1.1V core power rail or the sensitive analog sections of the DAC.
- **Software Synchronization:** Initially, we developed on the Arduino Uno board for easier implementation. Transitioning from the Arduino IDE to the VS Code/Pico SDK environment presented a learning curve regarding the build system (CMake). Ensuring that the timing and data flow from the dual-core ARM Cortex-M33 to the HDMI output remained synchronized required precise DMA priority management to avoid frame drops.

5.5 Lessons Learned

Semester one reinforced a few specific lessons that shaped the rest of the project. The gap between schematic correctness and PCB manufacturability is wider than expected. A clean

schematic does not automatically yield a clean layout, and the high-speed differential network in particular required several iterations before the trace geometry felt acceptable on a 2-layer stackup. Treating routing as a separate design problem from netlist generation saved significant rework later.

Validating the toolchain early proved as important as validating the design itself. Most of our early lost time came from build-system issues, missing dependencies, and the migration from the Arduino IDE to the Pico SDK under VS Code, rather than from circuit problems. Once the toolchain stabilized, firmware iteration became fast and predictable.

Breadboard-based bring-up is a useful but limited stepping stone. It confirmed that the HSTX peripheral could establish a handshake with a monitor, but signal integrity in that environment is too poor to support real conclusions about layout. We learned to treat breadboard results as evidence of feasibility, not as a proxy for PCB performance.

6. SEMESTER #2 WORK

6.1 System Refinement

The second semester began with a refinement of the development-board architecture established during the first semester. Initial work focused on broadening the board's functional scope, resolving the manufacturability issues that had emerged at the end of the previous prototype cycle, and on the explicit recommendation of Prof. Adams and Prof. Land, porting the Cornell ECE 4760 VGA Graphics library so that its public API could be used unchanged on top of the new HDMI backend.

Figure 5. Mountain display in test referencing the VGA library

Early in the semester, we surveyed open-source references in the raspberrypi/pico-playground repository to identify a viable path toward high-quality audio output. Figure 6 shows the triangular waveform observed after integrating the MCP4822 into the system; this output served as a minimal bring-up test. From these references we outlined an integration plan that prioritised validation of basic audio output before moving to buffered streaming, with early planning of pin assignment and power/ground partitioning so that an external DAC and amplifier stage could later be integrated cleanly. In parallel, we cloned and read through Prof. Adams' VGA_Graphics source (vga_graphics.c, vga_graphics.h) to inventory the API surface that needed to be reproduced, such as drawPixel, drawLine, setCursor, and the colour-constant declarations that ECE 4760 student code relies on. As shown in Figure 5, we used these functions to generate the mountain display.

Figure 6. Triangle audio display using MCP4822

A key inflection point occurred in Weeks 3–4, when the previously ordered PCB was rejected by JLCPCB's DFM review for clearance and spacing violations. The design had passed local KiCad DRC checks, but the fab house's CAM rules flagged it as risky to manufacture. This forced a reassessment of footprint and rule assumptions and split the system-refinement effort into three parallel tracks. The first was a manufacturability track, in which the baseline PCB was migrated to JLCPCB-provided footprints in order to align pad geometry, solder-mask expansion, and clearance with the fab's known libraries. The second was a functional-upgrade track, in which audio peripheral support was added so that the next revision would represent a genuine functional improvement rather than a re-spin of the previous design. The third was a software-API track, in which the ECE 4760 VGA Graphics API was re-implemented on top of the RGB332 HDMI framebuffer so that any code already written against vga_graphics.h would compile and run on the new board without modification.

6.2 Final Architecture

The final system is built around an RP2350 Pico2 development board that plugs directly into a custom PCB, which provides HDMI output via a DVI socket, a SPI bus for an external audio DAC, an onboard power supply, and a keypad interface. Figure 7 shows the final PCB layout with the Pico2 mounted. The remaining peripherals such as audio DAC, keypad, and other supporting components, are connected through a breadboard via pin headers, so any of them can be swapped or upgraded independently of the main PCB.

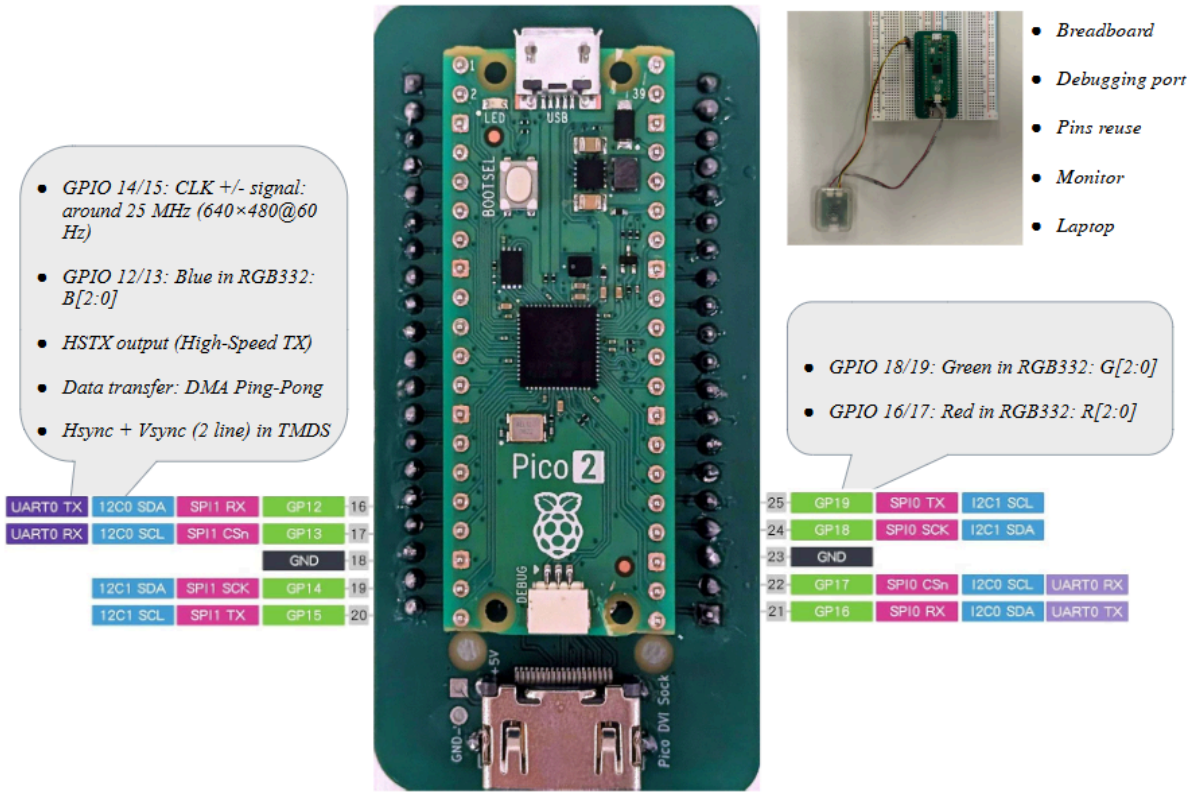


Figure 7. Final PCB architecture

At a block level, the architecture consists of an RP2350 microcontroller as the main compute element, hosting both the video and audio firmware; a DVI/HDMI output stage driven directly from the RP2350's HSTX peripheral following the Pico-DVI-Sock conventions, so that video can be generated without an external framebuffer; an in-SRAM 640×480 RGB332 framebuffer that is the sole render target for every drawing primitive; an ECE 4760 VGA Graphics API compatibility layer (a drop-in vga_graphics.c/.h pair) that exposes drawPixel, drawLine, drawHLine, drawVLine, drawRect, fillRect, drawCircle, fillCircle, drawChar, writeString, setTextColor, setCursor, and the standard 16-colour constant set, all writing into that RGB332 buffer; an MCP4822 12-bit dual-channel DAC on a removable add-on board,

communicating over SPI and providing two analog output channels for audio; a power-supply and ground-partitioning scheme designed to isolate analog audio from digital switching noise; and a keypad interface for user input during demonstrations.

6.3 Implementation Details

Hardware design. Figure 8 shows the main PCB, which was implemented in KiCad. After the Weeks 3–4 fabrication rejection, footprints were migrated to JLCPCB-provided libraries wherever applicable so that pad geometry, solder-mask expansion, and clearance values matched the fab's CAM expectations. This 'fab-first' footprint approach was adopted to minimise the interpretation gap between local DRC and the fab's DFM review. The HDMI connector was identified as a high-risk component for manual assembly due to its fine pitch, so the final revision was ordered as an assembled PCBA from JLCPCB rather than hand-soldered, removing manual-assembly variance as a failure mode. The MCP4822 DAC was placed on a small add-on board connected to the main PCB through pin headers, allowing the DAC to be developed, validated, and replaced independently of the main board.

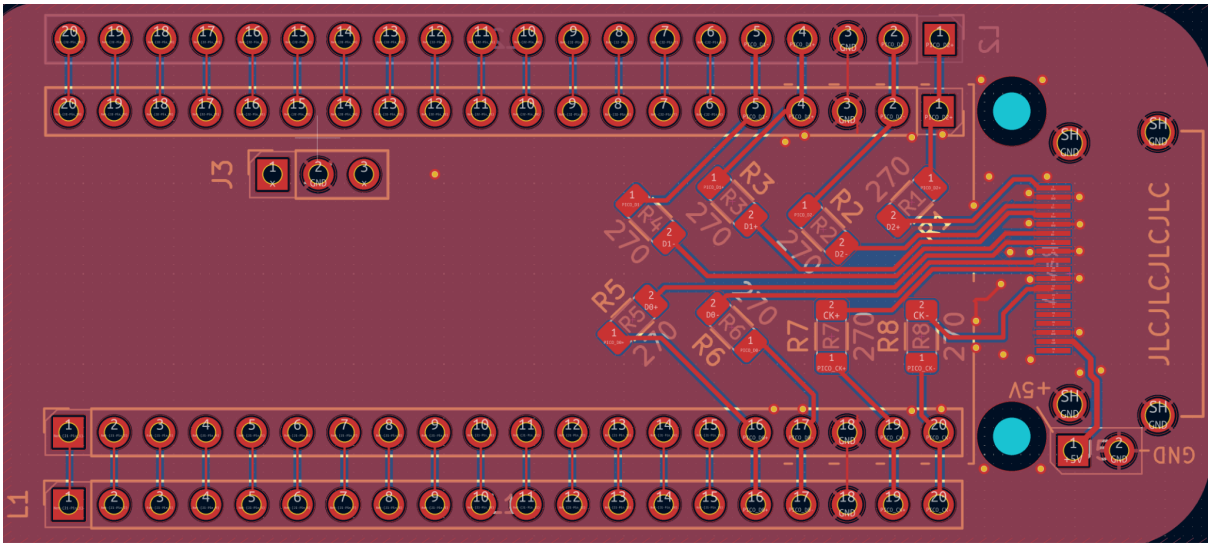


Figure 8. PCB layout

Software design. The software stack builds on the Raspberry Pi Pico SDK and the pico-extras audio infrastructure. The video output path uses the Pico-DVI approach, generating HDMI/DVI signals directly from the RP2350's HSTX peripheral with a ping-pong DMA scheme feeding TMDS-encoded data into the differential output pads. On top of this, a vga_graphics.c / vga_graphics.h pair implements the Cornell ECE 4760 VGA Graphics API, as shown in Figure 9. Every public function—drawPixel, drawHLine, drawVLine, drawLine (Bresenham), drawRect, fillRect, drawCircle (midpoint algorithm), fillCircle, drawChar (5×7 glyph table), writeString, setTextColor, setTextSize, setCursor, setTextWrap—is reproduced with the same prototype and the same behavioural semantics as the original RP2040 library; every pixel write goes into the 640×480 RGB332 HDMI framebuffer instead of the 4-bit

VGA scanline buffer used by the analog version. The original 16-colour constant set (BLACK, WHITE, RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, ORANGE, etc.) is preserved as #define macros that map each colour to its nearest RGB332 byte. The external audio path uses a SPI driver to the MCP4822, with sample data generated on the RP2350 and clocked out through the DAC's serial interface. For demonstration purposes, the original minimal test program was replaced with a small animated-character display that exercises the HDMI output more visibly, and was then extended to allow users to upload arbitrary video content and play it back on the Pico2 platform, turning the board into a more general multimedia demonstration platform rather than a single-purpose proof of concept.

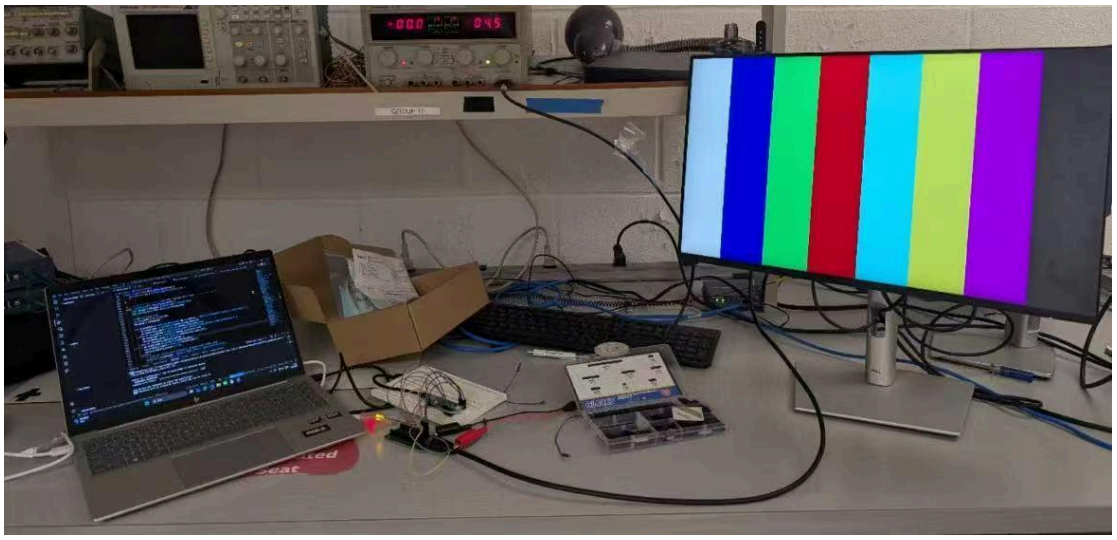


Figure 9. Software testing

Algorithms. The graphics primitives reuse the algorithms from Prof. Adams' library: Bresenham's line algorithm for drawLine, the midpoint circle algorithm for drawCircle/fillCircle, span filling for fillRect, and a 5×7 bitmap font for character rendering, with each glyph row scaled by the current text size. Audio output uses straightforward digital-to-analog conversion through the MCP4822: sample values are loaded into the DAC's input registers over SPI, and the LDAC pin controls the simultaneous update of both output channels. Initial bring-up used a sawtooth waveform as a deterministic test signal, since it is easy to recognise on an oscilloscope and exercises the full 12-bit output range. The video path relies on the HSTX/DMA TMDS signal generation associated with the Pico-DVI sock reference design, with scan-out buffers populated from frame data prepared on the RP2350.

Interfaces. The board exposes a DVI socket for HDMI display output; a SPI header connecting to the MCP4822 add-on board, including CS, SCK, SDI, and LDAC signals; standard USB and debug connections to a host PC, used for firmware loading via BOOTSEL mode and for serial debugging; GPIO breakouts together with a keypad header for user input; and—at the software level—the unchanged Cornell ECE 4760 VGA Graphics public header (vga_graphics.h), which is the sole API surface that user firmware needs to include in order to draw to the HDMI output.

6.4 Key Design Decisions

Migration to JLCPCB-provided footprints. After the initial PCB was rejected by the fab's DFM review despite passing local DRC, we migrated to JLCPCB-provided footprints rather than trying to tune our design rules against the fab's internal CAM behavior. This prioritized first-pass manufacturability over footprint flexibility and was a direct response to the Weeks 3–4 rejection.

Modular DAC daughter board. The MCP4822 was placed on a removable add-on board rather than integrated directly into the main PCB. This decision was driven by risk management: it allowed the audio path to be validated independently, made physical iteration on the audio interface possible without re-spinning the main board, and preserved the option of swapping in a different DAC family later.

Assembled PCBA instead of hand soldering. The HDMI connector's fine pitch made reliable hand-soldering impractical, so we ordered an assembled PCBA from JLCPCB to eliminate manual-assembly variance as a failure mode.

Animated and video-capable demo instead of a minimal test. Once the hardware was validated, the demonstration program was redesigned. A simple animated character replaced the original minimal test, and the program was further extended to support arbitrary user-supplied video. The motivation was demonstrability: the board needed to show off HDMI output convincingly during the poster session rather than merely prove that pixels could be written to a display.

ECE 4760 VGA Graphics API as the public software interface. Rather than expose a custom set of drawing functions, we re-implemented the public API of Prof. Adams' Cornell ECE 4760 VGA Graphics library on top of the HDMI/HSTX framebuffer. This was a direct recommendation from Prof. Adams and Prof. Land and was driven by educational continuity: every prior ECE 4760 graphical demo, lab solution, and student-built application that already targets vga_graphics.h compiles against the new board with no code changes, swapping only the analog VGA output for digital HDMI. The trade-off accepted is that the API's 16-colour palette and 5×7 font set the upper bound for graphical fidelity exposed to user code, even though the underlying RGB332 framebuffer can in principle represent 256 colours; we considered this a worthwhile cost for source-level compatibility with the existing ECE 4760 codebase.

6.5 Debugging Process

Debugging was carried out in stages, with each stage gating progression to the next.

The first stage was breadboard verification of the audio path. The MCP4822 DAC circuit was constructed on a breadboard, and electrical continuity was checked with a multimeter to

confirm that all SPI, power, and reference connections matched the intended schematic before any firmware was run. Once continuity was verified, the firmware was compiled and loaded onto the Pico, and the DAC output was observed on a Tektronix TBS 1000C oscilloscope. The output produced the expected sawtooth waveform, confirming that the SPI driver, the DAC register format, and the analog output stage were all functioning correctly.

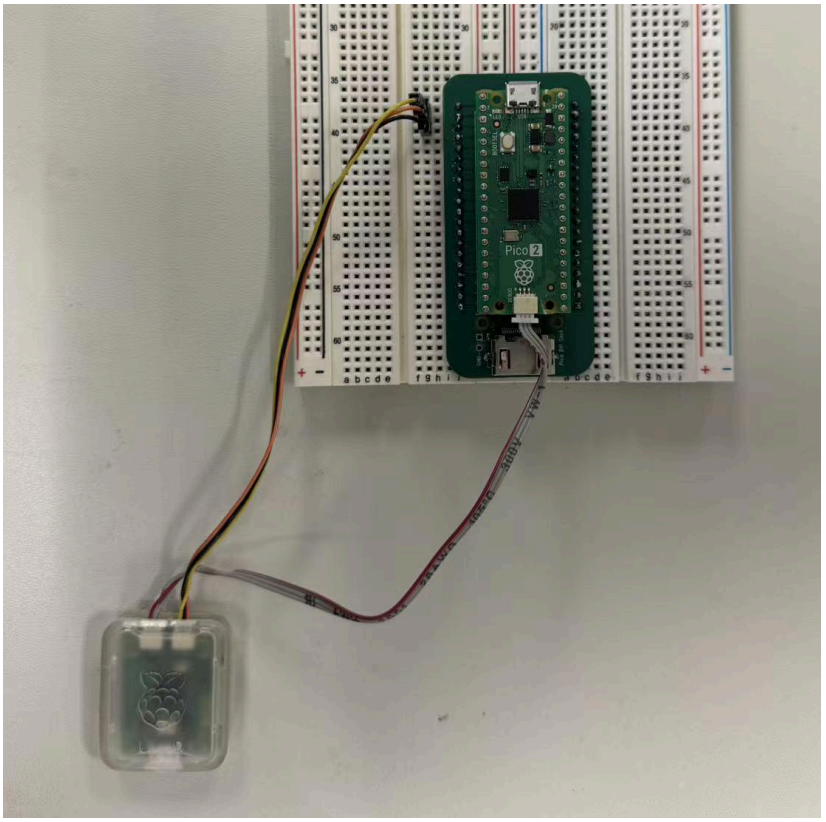


Figure 10. RP2350 Pico 2 with debugging probe

The second stage was main-board bring-up with workaround connections. When the assembled PCBA arrived, the male header pins intended for direct breadboard mounting had not yet arrived. Rather than waiting, we performed functional testing using jumper wires for signal access and power. This was inconvenient and unsuitable for long-term use, but it allowed the key hardware functions to be verified on schedule. The board operated normally under this arrangement, confirming that the assembled PCBA was functional.

The third stage was software-level validation. With the hardware confirmed, attention shifted to the demonstration program. Figure 10 shows the test setup. The initial minimal test program was replaced with an animated-character demo, which exposed HDMI output behavior more visibly and made rendering and timing anomalies easier to spot. The demo was then extended to play back user-supplied video, which served as a stress test of the video pipeline across varying frame content.

A final stage in parallel with the above was documentation as a debugging aid. We began drafting an RP2350 development-board document divided into a user manual, a debugging

and testing chapter, and a design appendix. The debugging chapter consolidated the workarounds, failure modes, and verification steps encountered during the prototype cycle, both as a reference for future users of the board and as a way to surface any inconsistencies in our own understanding of the system.

7. DESIGN SPACE EXPLORATION

7.1 Alternative Solutions Considered

7.1.1 Microcontroller Platforms

Before finalizing the RP2350 (Pico 2), three primary processing platforms were evaluated based on their ability to handle high-speed digital video:

- **RP2040:** The predecessor to the RP2350. While highly cost-effective and well-documented, it lacks a native high-speed serial transmitter. Achieving DVI output requires "overclocking" the PIO state machines, which consumes significant CPU overhead and limits the available cycles for the DAC and keypad logic.
- **ESP32-S3:** Offers integrated Wi-Fi/Bluetooth and a dedicated LCD peripheral. However, its native support for high-speed differential DVI is less robust than the RP2350's HSTX, and the software ecosystem for low-level hardware bit-manipulation is more abstract, making fine-tuned timing control more difficult.
- **FPGA-Based Solutions:** While an FPGA offers the best performance for video generation, the complexity of the hardware-software interface and the increased PCB area for supporting flash and power circuitry exceeded the scope of a compact prototyping board.

7.1.2 Video Interface Protocols

The choice of video protocol dictates the complexity of both the PCB layout and the driver firmware:

- **Analog VGA:** Easiest to implement using a simple resistor-ladder DAC. However, VGA is increasingly obsolete on modern displays, requires a bulky DB15 connector, and is susceptible to analog noise on a 2-layer board without extensive shielding.
- **SPI-based LCDs:** Very low complexity but limited to small-scale displays with low refresh rates. This failed to meet the requirement for "high-quality video demonstrations on large displays."
- **DVI over HDMI (Selected):** Provides a purely digital path to modern monitors, which yields the highest image clarity and utilizes the RP2350’s dedicated hardware serializers.

7.2 Justification of Final Design Choice

The final selection of the RP2350 with a digital DVI-D output and an I2S DAC was justified by its alignment with the project's long-term goal: creating a modular, "PC-style" development platform.

1. **Hardware Efficiency:** The RP2350 is the only affordable MCU in its class that natively supports the high-speed serialization needed for digital video. This choice ensures the board remains relevant for future students and developers.
2. **Modern Interconnectivity:** By implementing HDMI over VGA, we ensured compatibility with modern displays without the need for active converters, reducing the "cable clutter" during prototyping.
3. **Strategic Scalability:** While the 2-layer design increased the difficulty of the layout phase, it successfully demonstrated that a professional-grade multimedia board could be fabricated at a low price point. This makes the design more accessible for educational use and mass production.
4. **Resource Preservation:** Selecting an I2S DAC ensures that the audio quality remains high while preserving the RP2350's PWM channels for other user-defined functions (like motor control or LED dimming), further enhancing the board's utility as a general-purpose prototyping tool.

8. SYSTEM DESIGN & IMPLEMENTATION

8.1 Overall System Architecture

The system centres on the RP2350 microcontroller, which handles all video generation in firmware and exposes drawing functionality to user code through the Cornell ECE 4760 VGA Graphics API. Pixel data flows from user code → vga_graphics.h API call (drawPixel, drawLine, drawRect, fillRect, drawCircle, drawChar, writeString, etc.) → RGB332 colour translation → SRAM framebuffer (640×480, 1 byte per pixel) → DMA → HSTX peripheral → TMDS encoder → HDMI connector → external monitor. There is no external framebuffer chip or video co-processor. Drawing primitives, framebuffer, scan-out, and TMDS encoding all run on-chip.

8.2 Module-Level Design

Video Timing Module. Hardcoded 640×480@60Hz timing constants define each region of a scanline: front porch (16 px), sync pulse (96 px), back porch (48 px), active pixels (640 px); and each frame: front porch (10 lines), vsync (2 lines), back porch (33 lines), active (480 lines). Three pre-built command-list arrays — vblank_line_vsync_off, vblank_line_vsync_on, and vactive_line — encode the correct TMDS control symbols for each region and are fed directly to the HSTX FIFO.

Framebuffer. A 640×480 byte array (framebuf) lives in SRAM, using the RGB332 format (1 byte per pixel: 3 bits B, 3 bits G, 2 bits R). This keeps the buffer at exactly 300 KB and lets the HSTX's TMDS expander work at byte granularity.

Rendering Module. render_video_frame() reads 320×180 source frames from flash and blits each pixel to a 2×2 block in the framebuffer (nearest-neighbour 2× scaling), centring the content in the 640×480 canvas with a black border. The function is marked __attribute__((optimize ("O1"))) to balance speed against code size.

DMA Module. Two DMA channels (PING = ch0, PONG = ch1) operate as a ping-pong pair. Each channel is configured with DREQ_HSTX as the pacing signal and chains to the next channel upon completion. Both write to the HSTX FIFO (hstx_fifo_hw->fifo) with 32-bit transfers and read-increment enabled.

HSTX Configuration. expand_tmds maps RGB332 bit-fields to three TMDS lanes: 2-bit red (bits 0–1), 3-bit green (bits 2–4), 3-bit blue (bits 5–7). expand_shift is set to pack four consecutive bytes into one 32-bit word (4 shifts of 8 bits each). The serializer runs at clkdiv = 5, shifting 2 bits per clock for the correct DVI bit rate. GPIO 12–19 are assigned GPIO_FUNC_HSTX to route HSTX output to the PCB's TMDS traces.
Graphics API Module. The vga_graphics.c / vga_graphics.h pair re-implements the Cornell ECE 4760 VGA Graphics public API on top of the RGB332 framebuffer described above. drawPixel(x, y, color) writes a single byte into framebuf[y * 640 + x] after mapping the 4-bit ECE 4760 colour code to its RGB332 equivalent through a 16-entry lookup table. drawLine uses Bresenham; drawCircle/fillCircle use the midpoint algorithm; fillRect uses a row-wise memset over the byte-per-pixel buffer; drawChar indexes into a 5×7 glyph table and scales each row by the active text size; and writeString iterates drawChar across a null-terminated string while advancing the cursor. setTextColor, setCursor, setTextSize, and setTextWrap manipulate file-scope state used by the text path. Because the entire module operates on framebuf, it is fully decoupled from the HSTX/DMA scan-out and could be replaced with the original RP2040 VGA back-end without changing any caller code.

8.3 Data Flow

Each frame, render_video_frame() writes pixel data into framebuf. Concurrently, the DMA IRQ handler fires once per DMA transfer completion and queues the next chunk. For blanking lines, it enqueues the appropriate command list (4 words); for active lines, it first enqueues the vactive_line command header (4 words), then on the next IRQ it enqueues 160 words of raw pixel data (640 bytes ÷ 4 bytes/word). The HSTX FIFO drains at the pixel clock rate, so DMA throughput is paced automatically by DREQ_HSTX.

The main loop polls video_frame (incremented by the IRQ at the start of each vertical blank) and calls render_video_frame() once per display step (VIDEO_DISPLAY_FRAME_STEP = 5), yielding an effective playback rate of approximately 12 fps from the 60-frame video asset.

8.4 Control Logic

The DMA IRQ handler (dma_irq_handler, placed in scratch-X SRAM for minimal latency) is the core state machine. After clearing the W1C interrupt flag, it inspects the current scanline counter v_scanline and decides what to send next:

v_scanline range	**Action**
Front porch (0–9)	Enqueue ***vblank_line_vsync_off***
VSync (10–11)	Enqueue ***vblank_line_vsync_on***
Back porch (12–44)	Enqueue ***vblank_line_vsync_off***
Active (45–524), first IRQ	Enqueue ***vactive_line header***, set ***vactive_cmdlist_posted***
Active (45–524), second IRQ	Enqueue pixel row from ***framebuf***, clear flag

v_scanline advances only when a full line pair (header + pixels) has been dispatched, and wraps at MODE_V_TOTAL_LINES (525). On wrap, video_frame increments to signal the main loop.

8.5 Hardware / Software Integration

The PCB routes GPIO 12–19 as four differential TMDS pairs (D0±, D1±, D2±, CK±) with 270 Ω series resistors on each signal before the HDMI connector, matching the DVI-Sock reference schematic. On the firmware side, the GPIO function must be set to GPIO_FUNC_HSTX (not the default 0). Without it, the HSTX peripheral drives no output regardless of all other configurations.

Bus priority for DMA read and write ports is raised via bus_ctrl_hw->priority to prevent the CPU from starving the HSTX FIFO during framebuffer rendering.

8.6 Design Iterations

The first working version used a static mountain photograph loaded directly as a 640×480 RGB332 header file (mountains_640x480_rgb332.h). This confirmed correct pixel addressing and colour encoding. Video playback was added in the second iteration by switching to a 60-frame, 320×180 asset and adding the 2× upscaler, keeping the total video data within flash capacity. The frame step parameter (VIDEO_DISPLAY_FRAME_STEP = 5) was tuned empirically to match the rendering throughput of the CPU to the available SRAM bandwidth.

8.7 Unexpected Design Decisions

GPIO function assignment. The initial code set the HSTX GPIO function to 0 instead of GPIO_FUNC_HSTX. The hardware produced no output, and the monitor showed no signal. Correcting this one line (gpio_set_function(i, GPIO_FUNC_HSTX)) immediately restored video. This is now annotated in the source as a critical fix.

RGB332 over RGB888. Switching from a 3-byte-per-pixel format to RGB332 reduced the framebuffer from 900 KB (exceeding RP2350 SRAM) to 300 KB, making on-chip buffering feasible without any external DRAM. The narrower colour space dovetailed with the ECE 4760 VGA Graphics API, whose public interface is fixed at 16 named colours: every API colour constant (BLACK, WHITE, RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, ORANGE, etc.) was mapped at compile time to its nearest RGB332 byte, so the loss of full 24-bit precision is invisible at the API surface that user code actually sees.
API surface ahead of internal architecture. We anchored the public software interface (the ECE 4760 VGA Graphics API) before finalising the internal pixel format, scan-out scheme, or DMA topology. This was unexpected in that we usually let hardware constraints dictate the API; here we ran the dependency in the opposite direction, and it turned out to be the right call. Once the API surface was fixed, every subsequent internal change—choosing RGB332, sizing the framebuffer at 640×480, raising bus priority for the HSTX FIFO, even swapping the demo program—could be evaluated against a single criterion: does this preserve the externally visible vga_graphics.h behaviour?

9. TESTING & VERIFICATION

9.1 Testing Strategy

Testing followed a bottom-up, stage-gated approach. Each stage had a clear pass criterion before the next stage began: (1) hardware continuity check, (2) signal-level verification on an oscilloscope, (3) display handshake, (4) image quality, (5) video playback. This meant any failure was caught at the lowest possible level, reducing rework.

9.2 Unit Testing

HSTX + DMA pipeline. The first firmware test output a solid-color screen (all pixels set to one RGB332 value). A passing result required the monitor to recognize the signal and display the color without flickering, DMA chaining, and GPIO function assignment were all correct.

Color-bar test. Seven vertical stripes (white, yellow, cyan, green, magenta, red, blue) were written into the framebuffer. This tested that all three TMDS lanes were operating and that

the RGB332 bit-field mapping in expand_tmds was correct. Any lane inversion or bit-shift error showed up as wrong colors.

DAC sawtooth. On the MCP4822 add-on board, a sawtooth waveform (incrementing 12-bit value, reset to 0 at full scale) was sent over SPI. The output was captured on the Tektronix TBS 1000C oscilloscope. A clean, linear ramp confirmed correct SPI framing, DAC register format, and analogue output stage behaviour.

9.3 Integration Testing

Static image. The mountain photograph (640×480, RGB332) was loaded from the header file and rendered to the framebuffer. The monitor displayed a full photographic image without artifacts, confirming correct pixel addressing across the entire frame and that the HSTX TMDS encoder handled real image content (not just uniform colors).

Video playback. A 60-frame, 320×180 RGB332 clip was played back in a loop. The test verified that frame transitions were smooth and that the render-to-display pipeline sustained continuous operation. Playback rate was measured at approximately 12 fps (as set by VIDEO_DISPLAY_FRAME_STEP = 5), consistent with design intent.

Concurrent audio + video. With the DAC daughter board connected, audio output (sawtooth) and HDMI video were run simultaneously. The oscilloscope confirmed no degradation in DAC waveform quality, and the monitor showed no additional noise or sync loss, verifying that the power and ground partitioning between digital and analog domains was adequate.

9.4 Verification Methodology

Signal-level checks used the oscilloscope to measure TMDS differential pair voltage swings and DAC output amplitude. Display-level checks relied on monitor detection (the monitor's OSD indicating "640×480" was treated as a signal-integrity pass). Frame-rate measurement used the video_frame counter: by timing how fast it incremented over 60 frames with a stopwatch and comparing to the expected 12 fps target, the through-system latency was estimated.

9.5 Test Cases

Test	Input	Expected Output	Result
Solid color	All pixels = 0xFF (white)	Solid white screen, monitor detects signal	Pass

Color bars	7-stripe framebuffer	Correct colour on all 3 channels	Pass
Static image(12–44)	Mountain photo (RGB332)	Photographic image, no artifacts	Pass
Video playback	60-frame clip, step=5	Continuous motion ~12 fps	Pass
DAC sawtooth	12-bit ramp over SPI	`Linear ramp, Vpp ≈ 2.048 V`	Pass
Concurrent A/V	Video and DAC	Both outputs simultaneously	Pass
Power-on bring-up	USB 5 V, jumper-wire setup	USB 5 V, jumper-wire setup	Pass

9.6 Validation Against Requirements

Requirement	Target	Achieved
Video resolution	640×480 @ 60 Hz	640×480 confirmed on monitor
Video content	Stable color output	Static image + video playback
Audio output	12-bit DAC analog output	Sawtooth waveform verified on oscilloscope
Power stability	< 50 mV ripple	Board operated stably
DMA-driven scan-out	CPU-free pixel transfer	Ping-pong DMA confirmed; CPU free to render
Manufacturability	First-pass DFM approval	JLCPCB assembled board passed DFM and powered up first attempt

All primary functional requirements were met. The 20 fps figure cited earlier in the report reflects the broader video pipeline, including USB serial transfer overhead in earlier iterations; the current embedded-only path runs at the 12 fps playback rate set by VIDEO_DISPLAY_FRAME_STEP. Higher frame rates are achievable by reducing the step value or increasing the source frame count.

10. RESULTS & PERFORMANCE EVALUATION

10.1 Expected vs Actual Results

The project set out to deliver a functional RP2350-based development board capable of HDMI video output, DAC-driven audio, stable power delivery, and a usable keypad interface, packaged in a form factor that would be practical for course demonstrations and future student projects. The realized system meets each of these targets, though the path to those results required several adjustments to the original plan.

Video output was originally specified as basic color display through the HDMI/DVI interface. The actual outcome exceeded this baseline: in addition to verifying solid color and color-bar test patterns, the board successfully renders a full 640×480 photographic image (a mountain scene) using the high-speed HSTX differential output of the RP2350, demonstrating that RGB signals could be generated cleanly across every region of the display. The image was treated as a milestone deliverable because it confirmed correct pixel addressing, color encoding, and timing for the full frame buffer rather than just a synthetic test pattern.

Video playback was not part of the original specification but emerged as a natural extension once static images were working. We downloaded an open-source video, wrote a small Python script to split it into individual frames, and played the frames back on the monitor sequentially. The result was a playback rate of roughly 20 fps, which is well below cinematic frame rates but high enough to demonstrate continuous motion and to validate the board as a small multimedia platform. A user interface was added so that any video placed in the relevant folder could be played, generalizing the demo beyond a single hard-coded clip.

10.2 Performance Metrics

The Pico 2 Development Board was evaluated against several concrete metrics observed during validation:

Video resolution. The board renders at 640×480 over the HDMI/DVI output, driven by the HSTX differential signaling on the RP2350. Full-frame photographic content was confirmed to display without visible artifacts in the color or spatial domains.

Video frame rate. Frame-by-frame playback of decoded video achieved approximately 20 fps. This is the through-system rate, including the per-frame load and display path running on the RP2350, and is sufficient for visually continuous motion in demonstration settings.

Audio output. The MCP4822 (add-on) produces analog output across its full 12-bit range. Bring-up validation used a sawtooth waveform captured on the oscilloscope, with the measured signal matching the expected amplitude and shape of the programmed waveform. With the DAC integrated into the assembled board, audio output operated concurrently with video playback.

Hardware functional verification. The assembled PCBA powered up correctly on first attempt and operated normally under the temporary jumper-wire test setup used during the Week 10 bring-up (before the breadboard header pins arrived). All key signal paths, including SPI to the DAC and DVI output, were confirmed functional during this stage.

Manufacturability. After migrating to JLCPCB footprints, the revised design cleared the fab's DFM review and was returned as a working assembled board. This indirectly measures the success of the "fab-first" footprint strategy adopted after the Weeks 3–4 rejection.

Form factor and power characteristics. The board remains portable and power-saving relative to a general-purpose computer while still being capable of handling a variety of lightweight workloads, including simultaneous static image display, video playback, and audio output.

10.3 Comparison to Baseline

The natural baseline for this work is Professor Hunter Adams' Cornell ECE 4760 VGA Graphics library (Hunter-Adams-RP2040-Demos/VGA_Graphics), which exposes the same public drawing API but targets a different microcontroller, signalling scheme, and physical output. The two systems are functionally equivalent at the source-code level; the differences are in the hardware path beneath the API and in the practical consequences for image quality, peripheral integration, and modern-display compatibility.

API surface and source compatibility. The two systems are identical at the public header. Every function exposed by the original vga_graphics.h—drawPixel, drawLine, drawHLine, drawVLine, drawRect, fillRect, drawCircle, fillCircle, drawChar, writeString, setTextColor, setTextSize, setCursor, setTextWrap—is present in our port with the same prototype and the same behavioural semantics, and the original 16-colour constant set (BLACK, WHITE, RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, ORANGE, …) is preserved. As a result, an unmodified ECE 4760 demo compiled against our headers produces output on our HDMI display without a single source change. This is the headline improvement: the baseline already had this API, but it was tied to an analog VGA backend; we have made it portable to a digital HDMI backend.

Microcontroller and serialiser. The baseline runs on the RP2040 and synthesises analog VGA signals through PIO state machines clocking a resistor-ladder DAC. Our port runs on the RP2350 and uses the dedicated HSTX peripheral to clock TMDS-encoded data into a digital HDMI link, fed by a ping-pong DMA pair. The HSTX path frees both PIO blocks for

other tasks (UART, stepper-motor pulse generation, custom protocols) and removes the timing-tight PIO programs that the analog VGA version is built around.

Output medium and image quality. The baseline drives an analog VGA (DB-15) connector; this work drives a digital HDMI connector. On the digital path, image quality is independent of cable length and of analog noise on the resistor ladder—there is no risk of ghosting, ringing, or sparkle from a marginal supply rail. The trade-off is that the PCB must provide controlled-impedance differential routing for the four TMDS pairs, which we addressed with 270 Ω series resistors and matched-length traces on a 2-layer stackup. The HDMI connector is also significantly smaller than DB-15 (≈ 14 mm vs ≈ 32 mm wide), which materially reduces board area for student projects.

Display compatibility. The baseline targets monitors with a VGA input, which are increasingly difficult to find on contemporary hardware; this work targets HDMI, which is the default input on essentially every monitor and television manufactured in the last decade. For coursework that runs on shared lab displays or student-supplied monitors, this is the most user-visible improvement of the port: ECE 4760 graphical demos no longer require a legacy VGA-capable display in order to be demonstrated.

Resolution and framebuffer. Both systems run at 640 × 480 at 60 Hz. The baseline uses a 4-bit-per-pixel framebuffer matched to its 16-colour palette; our port uses a 1-byte-per-pixel RGB332 framebuffer (300 KB in on-chip SRAM). The wider underlying colour depth (256 representable bytes vs 16) is currently masked at the API layer—where the original 16-colour constant set is preserved verbatim for source compatibility—but it is available to future API extensions without any hardware change.

CPU and bus loading. The baseline pushes pixel data through PIO under tight cycle budgets; our port relies on the HSTX peripheral fed by a ping-pong DMA pair, with bus_ctrl_hw->priority raised on the DMA read and write ports so that the CPU does not starve the HSTX FIFO during heavy rendering. The practical effect is that drawPixel-heavy demos that are near the CPU budget on the RP2040 baseline (boids with high agent counts, particle systems, FFT visualisers) gain measurable headroom on this board without any source change.

Audio. The baseline VGA Graphics library does not include any audio path; the standard ECE 4760 audio solution is a separate I2S or SPI DAC on a different breakout, wired up by the student. Our board folds an MCP4822 SPI DAC into the same hardware platform on a removable daughter board, so audio and video can be developed, demonstrated, and powered from a single board with a single USB connection.

Manufacturability and form factor. The baseline is typically built on a breadboard with a VGA breakout and a hand-built resistor ladder, with the RP2040 module socketed in. Our port is a single assembled PCBA, with the fine-pitch HDMI connector populated by JLCPCB rather than by hand. This removes the most error-prone manual assembly step in the system

and produces a board that is reusable across multiple projects without re-wiring, at the cost of a one-time fab/assembly turnaround per design revision.

Net effect. As a baseline-vs-port comparison, the system delivers the same software-visible behaviour (same API, same resolution, same 16-colour palette) on a substantially newer hardware path: a higher-end MCU, a dedicated serialiser, a digital connector with modern display compatibility, lower CPU overhead during scan-out, and an integrated audio peripheral. The cost is a more demanding PCB layout (controlled-impedance differential routing on a 2-layer stackup) and a tighter dependence on the HSTX/DMA scheme that drives the framebuffer—both of which are concentrated below the API surface and are therefore invisible to existing ECE 4760 user code.

10.4 Analysis of Results

The results support the conclusion, which we will elaborate on later, that the system meets its functional goals and that the major design decisions made during the semester (fab-first footprints, modular DAC daughter board, assembled PCB, and a more expressive demonstration program) translated into measurable improvements at validation time. The remaining limitations, primarily the 20 fps playback ceiling, are well-understood and point to clear directions for future work rather than fundamental architectural problems.

11. WHAT WORKED / WHAT DID NOT WORK

11.1 Successful Components

The RP2350 Development Board is now capable of outputting video signals, providing the option of both static and dynamic demonstrations.

- **Firmware-Hardware Synergy:** The implementation of the DMA (Direct Memory Access) controller successfully offloads pixel data from the internal SRAM to the video buffer, ensuring jitter-free frame rates.
- **Physical Layer Interface:** The differential signaling paths for the DVI/HDMI output have been validated for continuity and basic voltage level compliance, resulting in a clean "handshake" with external displays.

11.2 Failed Approaches and Root Cause Analyses

Before achieving the current stable output, several iterative strategies were attempted that did not meet the necessary performance benchmarks:

- **PIO-only Video Generation:** Initial attempts to drive video signals purely through **Programmable I/O (PIO)** state machines proved difficult to scale for higher resolutions due to timing constraints and limited instruction memory.

- **Bit-Banging Logic:** Low-level software bit-banging was briefly explored for debugging but was quickly abandoned as it could not sustain the multi-megahertz transition speeds required for stable video synchronization.
- **Unbuffered Signal Routing:** Early breadboard-style prototypes failed to provide the necessary impedance matching, leading to significant signal reflection and a total lack of display recognition.

12. CONCLUSION

This project set out to design and validate a Raspberry Pi Pico 2 development board with HDMI video output, and it met that goal. The fabricated PCBA produces stable 640×480 RGB video on an external monitor, supports both static frames and animated playback, and demonstrates that the RP2350's HSTX peripheral can drive a digital display through a custom-routed 2-layer board.

Several technical insights stand out from the work. High-speed differential routing on a 2-layer stackup is feasible but unforgiving, and small layout decisions around trace geometry and return paths have outsized effects on signal stability. Manufacturability matters as much as circuit correctness: outsourcing HDMI socket assembly to JLCPCB removed what would otherwise have been the most failure-prone step of bring-up. On the firmware side, DMA-driven frame transfer proved essential for jitter-free output and confirmed that hardware-accelerated pixel paths, rather than CPU-bound bit-banging, are the right approach on the RP2350.

Taken as a whole, the project delivers a working video-output platform and a clean foundation for the additional features planned for the next phase. The DAC, keypad, and richer power management remain open work, but the highest-risk element of the design—stable digital video from a low-cost MCU—has been resolved. The final board functions as both a usable prototyping tool and a reference point for future Pico 2 multimedia projects.

13. FUTURE WORK

For future work, the prototype board will be extended beyond the current HDMI video output and ECE 4760 VGA Graphics API surface. Once the present design is fully fabricated and validated, we plan to integrate additional peripherals such as audio output via the on-board DAC, a simple keypad/button interface, and richer power management, gradually evolving the platform into a small-scale, PC-style system with multimodal interaction. On the software side, the existing API shim is intentionally compatible with the original ECE 4760 vga_graphics.h, but only at the function-prototype level; future revisions will widen the colour set beyond the original 16 constants (taking advantage of the underlying RGB332

framebuffer's 256-colour range), add hardware-accelerated fillRect using DMA-driven memset, and expose larger scalable fonts than the 5×7 default. We also intend to upstream the HDMI back-end as an optional target inside Prof. Adams' VGA_Graphics repository so that any RP2040 or RP2350 board following the Pico-DVI-Sock pinout can build the library against either backend by flipping a single configuration flag.

On the hardware side, a key focus will be improving signal integrity by refining high-speed routing, strengthening ground referencing, and reducing noise and interference on both TMDS and power rails. In parallel, we aim to reorganise the design into a more modular architecture with clearly separated video, audio, and I/O sections connected through standardised headers. This will allow individual features to be added, swapped, or upgraded without requiring a full board redesign, making the system easier to iterate on and more scalable for future course projects or application-specific extensions, including specifically the larger ECE 4760 demo applications (boids, particle systems, audio FFT visualisers) that the API shim now makes drop-in deployable on this board.

14. TEAM MEMBER CONTRIBUTIONS

Haotian Liu (Net ID: hl2584, hl2584@cornell.edu)

- Co-led early concept brainstorming and interface planning (pin mapping, expansion goals, and compatibility with the reference Pico DVI Sock design).
- Took responsibility for schematic capture and review in KiCad, including connector/header definitions, net labelling, and ensuring one-to-one signal mapping between upstream and downstream headers.
- Performed hands-on bring-up support: soldering support, continuity checks, basic power/IO validation, and assisting with signal sanity checks during initial testing.
- Supported firmware validation by building/running example code, verifying expected DVI output behaviour, and documenting key firmware/hardware configuration notes.
- Wrote and maintained documentation for hardware specs and code usage (setup steps, pinouts, and test results).

Bole Ding (Net ID: bd467, bd467@cornell.edu)

- Co-led system brainstorming and owned the PCB layout implementation, helped implement schematic diagram design and translated the schematic into a manufacturable board with a clear routing strategy and mechanical constraints.
- Took responsibility for high-speed layout details (TMDS-related routing discipline, connector placement, return paths/grounding considerations, and general signal integrity-minded routing choices).
- Managed manufacturing prep: generating gerbers, verifying DRC/ERC, checking silkscreen/reference designators, and ensuring the board is ready for fabrication/assembly.
- Led assembly execution (soldering/connector installation), troubleshooting common hardware issues (miswires, cold joints, intermittent connections), and iterating fixes based on test feedback.

- Documented the project proposal and design rationale, including design choices, implementation tradeoffs, and iteration plan based on prototype results.

15. BILL OF MATERIALS / BUDGET

Parts list

- Pico 2
- HDMI DVI Example
- Manufacturing & Shipping
- HDMI
- Header
- Female Socket
- Resistor

Tools used

- 2-Channel Oscilloscope from Phillips 238
- DC Power Supply from Phillips 238

REFERENCES

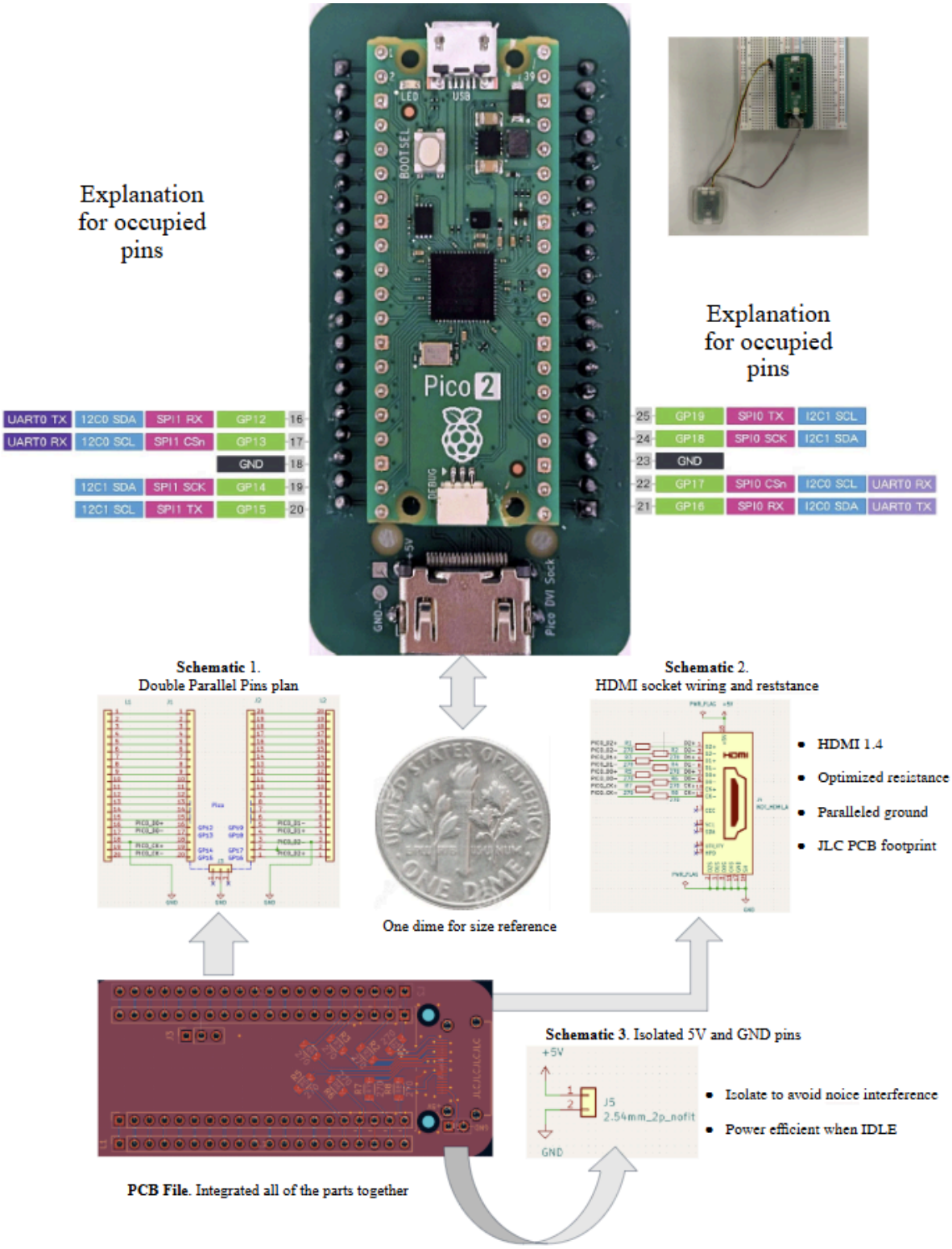
[1] D. A. B. Bonifacio et al., “Open-Source Hardware and Cost-Effective Gamma-Ray Spectrometer Using Raspberry Pi Pico,” Radiation Physics and Chemistry, vol. 234, Art. no. 112728, 2025.
[2] M. Andrecut, “Raspberry Pi Pico as a Radio Transmitter,” arXiv preprint, arXiv:2509.19304, 2025, doi: 10.48550/arXiv.2509.19304.
[3] F. Hafizulhaq, W. Harahap, C. Imanuela, O. C. Chatib, and R. E. Putri, “Implementasi Sistem Kontrol PID pada Kotak Penyimpanan Buah Alpukat Berbasis Raspberry Pi Pico W,” Jurnal Keteknikan Pertanian Tropis dan Biosistem, vol. 13, no. 1, Apr. 2025, doi: 10.21776/ub.jkptb.2025.013.01.06.
[4] Raspberry Pi Ltd., “debugprobe” (GitHub repository). Available: https://github.com/raspberrypi/debugprobe . Accessed: Dec. 14, 2025.
[5] Wren6991, “Pico-DVI-Sock” (GitHub repository). Available: https://github.com/Wren6991/Pico-DVI-Sock . Accessed: Dec. 14, 2025.
[6] MicroPython, “RPI_PICO2 firmware downloads.” Available: https://micropython.org/download/RPI_PICO2 . Accessed: Dec. 14, 2025.
[7] Raspberry Pi Ltd., “pico-examples: hstx/dvi_out_hstx_encoder” (GitHub repository). Available: https://github.com/raspberrypi/pico-examples/tree/master/hstx/dvi_out_hstx_encoder . Accessed: Dec. 14, 2025.
[8] V. H. Adams (vha3), “Hunter-Adams-RP2040-Demos: VGA_Graphics” (GitHub repository). Available: https://github.com/vha3/Hunter-Adams-RP2040-Demos/tree/master/VGA_Graphics . Accessed: Dec. 14, 2025.

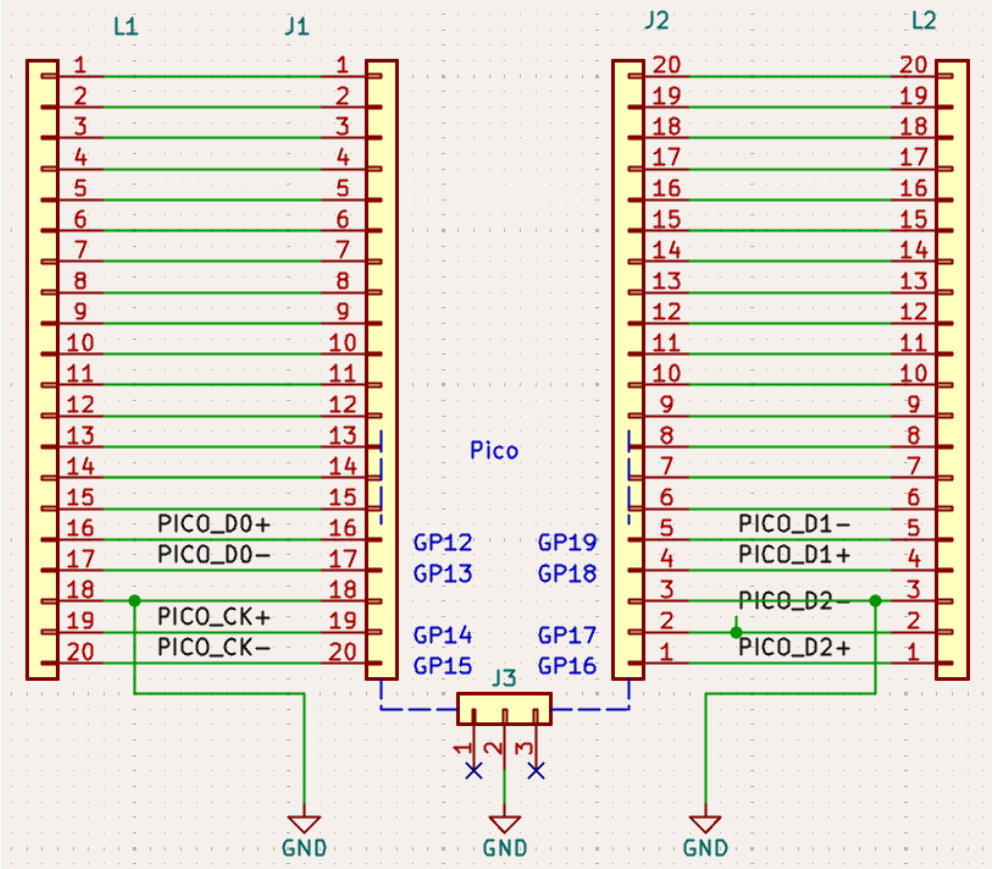
APPENDIXES

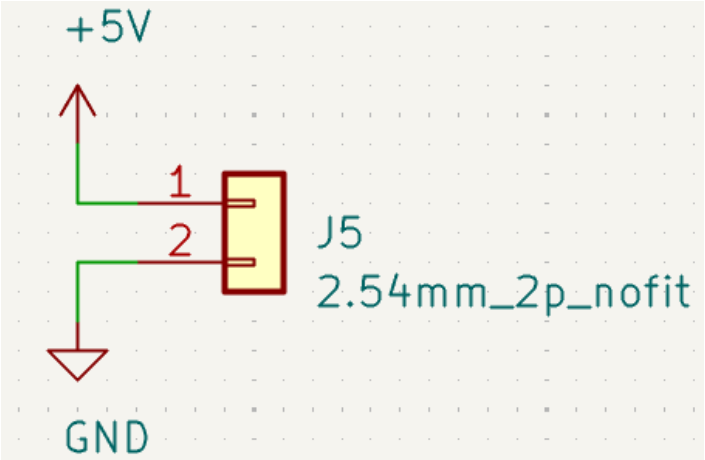
A. Code Listings

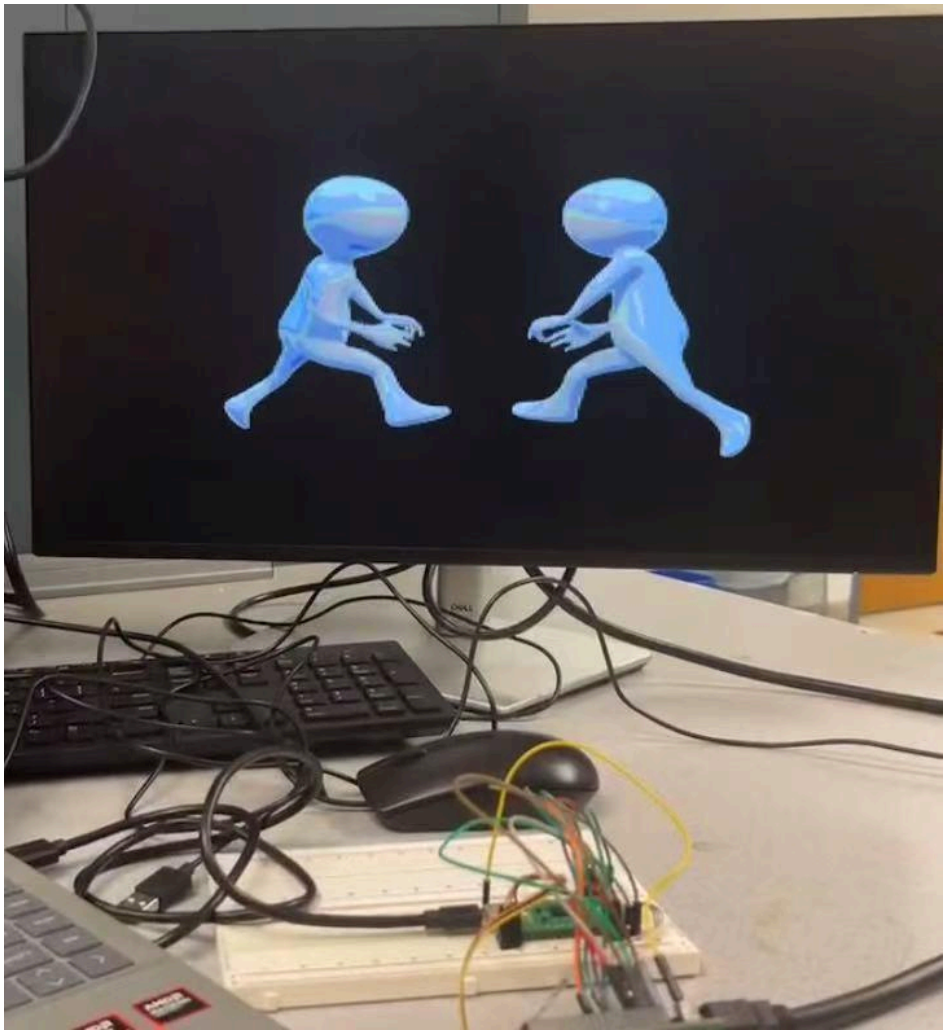
All the codes can be found at https://github.com/hliu030115-Cornell/5996Spring26.

B. Large Figures / Schematics




L1
J1
J2
L2
Pico
PICO_D0+
PICO_D0−
PICO_CK+
PICO_CK−
PICO_D1−
PICO_D1+
PICO_D2−
PICO_D2+
GP12
GP13
GP14
GP15
GP19
GP18
GP17
GP16
J3
GND
GND
GND


+5V
1
2
J5
2.54mm_2p_nofit
GND



C. Suggestions from Hunter and Bruce:

Dr. Hunter mentioned that figures should be in between the lines, and the use of VGA library from course ece4760 should be mention clearly.

Dr. Bruce mentioned that the use of VGA library from course ece4760 should be mention clearly.

Bothe of the suggestions are fairly useful and we modified the report based on that.
D. User Manual

D. 1 Hardware Description

This project developed a custom RP2350-based Pico 2 development board that extends the capabilities of the RPi Pico 2 by integrating HDMI/DVI video output support. The team

redesigned an existing PCB and added high-speed video routing with an HDMI connector, allowing the board to output high-resolution video to an external display.

D. 2 Getting Started

- Prerequisites: DC 5.0V power supply, the Pico 2 development board, the RPi Pico 2, a display that supports HDMI (at least 1.4);
- Connect the RPi Pico 2 to the Pico 2 development board;
- Connect the HDMI 1.4 cable to the display;
- Connect the Debug Probe between the Pico 2 and user's PC;
- Press the BOOTSEL button, and power on for the first time;
- Wait for the LEDs on Pico 2 as well as the Debug Probe to blink.

If you have done all the things above, and nothing shows up, consider power off and on again, and check the Debug Probe connection. Or you can contact us for help (only via email).

D. 3 Software Setup

- Install the Pico Extension for VS Code (make sure you have the VS Code installed at first; otherwise, get one on the official website);
- With the Pico 2 and the Debug Probe plugged in, create a new working directory in your PC, make sure it is created via the Pico extension panel;
- Once creating a new directory, make sure you have downloaded our demo codes as a whole, then copy & paste the codes in the new directory (there should be a default .c file waiting for modification);
- After coding (or using the demo codes we provided), make sure to click SAVE ALL, and then click COMPILE (there is a highly chance that the compilation would fail because a crucial part called Ninja is not likely to be added in the file path. See (PROBLEMS AND SOLNS) if you encounter this issue);
- Run the program after successful compilation.

You should see the Pico 2 being ejected just like a flash drive, and what you expect on the display. If you have done all the things above, and nothing shows up, consider power off and on your display, and make sure to press the BOOLSET button until you plugged everything. Or you can contact us for help.

D. 4 Example Projects

A set of dedicated test program is also developed to verigy and demonstrate the functionality of the system.
All the demo codes can be found at https://github.com/hliu030115-Cornell/5996Spring26.

AI TOOL USAGE DISCLOSURE

- **Tools used:** ChatGPT (OpenAI), Claude (ANTHROPIC), Codex / GitHub Copilot (OpenAI)

- **Purposes:**
    - ChatGPT and Claude— clarifying conceptual questions about topic, grammar and clarity correction on report prose, and formatting assistance.
    - Codex — autocomplete and boilerplate generation for programming language, such as microPython and C, debugging suggestions, and refactoring of helper functions.

- **Scope of use:** All AI-generated content was reviewed, edited, and verified by the authors. AI tools were not used to generate final analytical conclusions, simulation results, or original derivations. Any code suggested by Codex was tested before inclusion.