Research Report: Commodore PET 4032 PIA and Editor ROM Initialization

Publication Date: 2026-02-10
Report ID: PET4032-PIA-2026-02-10

Executive Summary

This report provides a detailed analysis of the Commodore PET 4032's hardware and software initialization processes, with a specific focus on the Peripheral Interface Adapter (PIA) chips and the Editor ROM. The PET 4032, a cornerstone of early personal computing, relies on a set of standardized integrated circuits for its input/output (I/O) operations, primarily the MOS 6520 PIA. This analysis covers the memory-mapped I/O architecture, the distinct roles of the two primary PIA chips, the system startup sequence from KERNAL reset to the BASIC `READY` prompt, and the unique screen management techniques employed by the 40-column Editor ROM.

Key findings indicate that the PET's I/O functionality is concentrated in a 256-byte memory region from $E800 to $EFFF, which houses registers for the PIAs, the VIA, and the CRTC. PIA 1 is dedicated to human interface tasks such as keyboard scanning and cassette tape operations, while PIA 2 manages the IEEE-488 peripheral bus. The Editor ROM, located at $E000, plays a pivotal role in the boot sequence, receiving control from the KERNAL ROM to initialize the screen, configure the CRTC, and establish a sophisticated "line linking" system for text display on 40-column models. This report also examines the challenges inherent in emulating these hardware components, emphasizing that accurate simulation requires more than isolated functional replication; it demands a holistic, system-level approach that accounts for inter-chip signal timing and dependencies.

1.0 System Architecture and Memory Map

The Commodore PET's architecture is a classic example of a 6502-based microcomputer system from its era. Its memory map provides a structured framework that allocates specific address ranges for RAM, ROM, and I/O devices. Understanding this map is fundamental to comprehending how software interacts with the underlying hardware. For PET models up to the 8032, the memory is organized as follows:

- **$0000 - $7FFF:** This range is dedicated to Random Access Memory (RAM). The total amount of available RAM varies by model, from 4KB to 32KB. This area holds user programs written in BASIC, machine language routines, variables, and system data structures in the zero page ($0000-$00FF).
- **$8000 - $8FFF:** This is the Screen RAM. On the PET 4032, this is typically a 1KB block of memory that is mirrored to fill the 4KB address space. The contents of this memory are directly translated into characters on the display by the video circuitry.
- **$9000 - $BFFF:** These three 4KB blocks are designated as Expansion ROM sockets. They allow for the addition of third-party software, language extensions like the "Programmer's Toolkit," or custom utilities that integrate with the operating system. On systems with BASIC 4.0, the block at $B000 contains the first part of the BASIC ROM.
- **$C000 - $DFFF:** This 8KB space contains the core of the Commodore BASIC interpreter, provided by Microsoft.

- **$E000 - $E7FF:** This is the Editor ROM. It contains the routines responsible for screen editing, keyboard input, and cursor management. It works in concert with the KERNAL to provide a complete user interface.
- **$E800 - $EFFF:** This crucial 256-byte block is the **I/O Area**. It serves as the control center for all hardware peripherals, containing the memory-mapped registers for the PIA, VIA, and CRTC chips.
- **$F000 - $FFFF:** This is the KERNAL ROM, which contains the fundamental operating system routines for the PET. It handles interrupts, resets, and low-level I/O operations, forming the foundational layer upon which BASIC and the Editor run.

When a PET is powered on, the 6502 processor begins execution with the reset vector, which points to an initialization routine within the KERNAL ROM. This routine performs basic hardware checks and then transfers control to the Editor ROM to prepare the system for user interaction, ultimately leading to the familiar `READY` prompt.

2.0 The I/O Area: A Hub of Hardware Control

The memory range from $E800 to $EFFF is the nerve center of the PET's hardware interaction. By reading from (PEEK) or writing to (POKE) specific addresses within this block, software can control peripherals, read sensor data, and manage communication. The primary components mapped to this area are the Peripheral Interface Adapter (PIA), the Versatile Interface Adapter (VIA), and the Cathode Ray Tube Controller (CRTC).

2.1 The Peripheral Interface Adapter (PIA)

The PIA is a general-purpose I/O integrated circuit that provides a bridge between the microprocessor's data bus and external devices. The Commodore PET uses the MOS 6520 PIA, which is functionally identical to the Motorola MC6821. This chip offers two 8-bit bidirectional parallel ports (Port A and Port B) and four control lines for handshaking and interrupt management. The PET 4032 architecture incorporates two such PIA chips, each assigned a distinct set of responsibilities.

2.1.1 PIA 1 ($E810 - $E81F): User and Tape Interface

PIA 1 is primarily responsible for managing direct user interaction and the Datasette tape drive. Its registers are mapped to the following addresses:

- **$E810 (Port A):** This port is configured for input and serves multiple functions. Its bits are used to select a specific row of the keyboard matrix for scanning, read the cassette sense line (detecting if tape buttons are pressed), and monitor the IEEE-488 EOI (End or Identify) line.
- **$E811 (Control Register A):** This register controls the CA1 and CA2 lines. CA1 is used as an input for the cassette read signal, while CA2 is an output used for screen blanking.
- **$E812 (Port B):** This port reads the state of the keyboard columns. By systematically selecting a row via Port A and reading the columns via Port B, the KERNAL can determine which key, if any, is being pressed.
- **$E813 (Control Register B):** This register controls the CB1 and CB2 lines. CB1 acts as an interrupt input, often tied to the vertical retrace signal from the video circuitry to time screen updates. CB2 is an output that controls the cassette tape motor, turning it on or off as required for load and save operations.

2.1.2 PIA 2 ($E820 - $E82F): IEEE-488 Bus Interface

PIA 2 is dedicated entirely to handling the IEEE-488 bus, a parallel communication standard used to connect peripherals like disk drives and printers.

- **$E820 (Port A):** Configured as an input port to read data from the IEEE-488 data lines.

- **$E821 (Control Register A):** Manages the ATN (Attention) input line and the NDAC (Not Data Accepted) output line, which are critical for the IEEE-488 handshaking protocol.
- **$E822 (Port B):** Configured as an output port to send data to the IEEE-488 data lines.
- **$E823 (Control Register B):** Manages the SRQ (Service Request) input line and the DAV (Data Valid) output line, also essential for the bus protocol.

2.2 The Versatile Interface Adapter (VIA) and CRTC

While the PIAs handle the keyboard and IEEE-488 bus, other I/O tasks are managed by the MOS 6522 VIA and the Motorola 6845 CRTC.

- **VIA ($E840 - $E84F):** The VIA is a more advanced I/O controller that includes timers, a shift register, and two 8-bit ports. On the PET, it is used for various functions, including generating sound, controlling the user port, and managing additional cassette and IEEE-488 control signals. For example, POKEing address 59468 ($E84C), the VIA’s Peripheral Control Register, is the standard method for switching between the uppercase/graphics and lowercase/uppercase character sets. Sound is generated by manipulating the VIA’s timers and shift register via POKEs to addresses like 59464 ($E848) and 59466 ($E84A).
- **CRTC ($E880 - $E88F):** The Cathode Ray Tube Controller is a specialized chip that generates the video timing signals and memory addresses needed to display text on the screen. It is programmed by writing a register index to address $E880 and the desired data value to $E881. The CRTC is responsible for defining the screen geometry (characters per line, number of lines) and controlling the hardware cursor.

3.0 Editor ROM Initialization and Screen Management

The path from a cold boot to a usable BASIC prompt is a well-orchestrated sequence that passes from the KERNAL ROM to the Editor ROM. The disassembly of the 40-column Editor ROM (part number 901499-01) provides a clear view of this process.

3.1 The Initialization Path: From KERNAL to READY

1. **Power-On Reset:** When the PET is turned on, the 6502 processor executes the reset vector located at $FFFC-$FFFD. This vector points to an initialization routine inside the KERNAL ROM, which begins at address `$FD16`.
2. **KERNAL Initialization:** The KERNAL routine performs low-level hardware setup.
3. **Jump to Editor ROM:** After its initial tasks, the KERNAL calls the main initialization entry point of the Editor ROM. This is the first jump in the Editor ROM’s jump table at **$E000**, which directs the processor to the `RESET_EDITOR` routine at address **$E036**.
4. **Editor Initialization (`$E036`):** The `RESET_EDITOR` routine executes a series of subroutines to prepare the user environment:
    - It first calls a routine at `$E683` to initialize core editor variables.
    - Next, it calls `$E617` to initialize the CRTC, setting it to the default text/graphics mode.
    - It then calls the “ring bell” routine at `$E654` twice, producing the familiar startup chime.
5. **Screen Clear and Line Link Table Setup (`$E042`):** Following the initial setup, the code proceeds to the screen clearing routine at `$E042`. This is more than a simple memory-fill operation; it establishes a critical data structure for the 40-column display known as the **Screen Line Link Table**.
    - This table resides in zero page memory from **$00E0 to $00F8**, with one byte for each of the 25 screen lines.

  - The routine at `$E042` iterates backwards from the last line of the screen, calculating the high byte of the starting memory address for each line (e.g., $80 for the first line, which starts at $8000). This high byte is stored in the corresponding entry of the link table.
  - The high bit of each byte in this table is used as a **link flag**. When set (the default state), the line is considered independent. When cleared, it signifies that the line is logically “linked” to the line above it, a feature used for word wrap.
  - The low byte of each line’s starting address is fetched from a static table in ROM at `$E798`.
  - After building the table, the routine fills the entire 1KB screen RAM ($8000-$83FF) with space characters and homes the cursor to the top-left position.
6. **Return to BASIC:** Once the screen is initialized, control is passed to the BASIC ROM, which prints its startup message (e.g., `*** COMMODORE BASIC 4.0 ***`) and the `READY.` prompt, awaiting user input.

3.2 The Code at $E4E7 and Keyboard Scanning

The prompt requested an analysis of the code at address `$E4E7`. While the provided disassembly focuses on the initialization routines starting at $E000, the Editor ROM’s jump table gives crucial context. The entry at `$E027` is a jump to `$E4BF`, the start of the `Scan Keyboard` routine. Therefore, the code at `$E4E7` is part of this keyboard scanning logic.

This routine is responsible for polling the keyboard matrix via PIA 1. It would systematically write values to the row select register at `$E810` and read the corresponding key presses from the column register at `$E812`. The code at and around `$E4E7` would contain the logic to decode this matrix, handle key repeats, manage the shift key status, and place detected key codes into the keyboard buffer queue located at `$026F`. This routine is called repeatedly by the main IRQ handler (vectored through `$E00C` to `$E442`) to ensure the keyboard is responsive.

4.0 Challenges in PIA Emulation

Emulating a vintage computer like the PET presents unique challenges, particularly with I/O chips like the 6520 PIA. The goal of modern emulation is often not just to replicate functionality but to achieve cycle-accurate simulation for maximum compatibility.

One developer’s experience in creating a 6502-based system emulator highlights the core difficulties. The early PET was chosen as an ideal target specifically because it uses off-the-shelf components like the 6520 PIA and 6522 VIA, avoiding the complexity of proprietary, undocumented custom chips. The development process involved implementing the PIA’s functions based on its datasheet and verifying them with isolated unit tests.

However, a critical insight emerged: **unit testing a chip in isolation is insufficient.** The developer noted that issues with the PIA emulation only became apparent when it was integrated into a larger system with an emulated CPU, LCD, and keypad. The complex, time-sensitive interactions between chips—the handshaking signals, interrupt requests, and data transfers—exposed subtle flaws that were not caught by simple functional tests.

This underscores a fundamental principle of hardware emulation:
* **System-Level Integration is Key:** The true test of a component’s emulation is its ability to function correctly within the context of the entire simulated system.
* **Cycle Accuracy Matters:** To correctly emulate the behavior of peripherals, especially during I/O handshaking, the emulator must track the state of every signal on a clock-cycle-by-clock-cycle basis. This requires a simulation core that can process all chip states “simultaneously” for each time step, often using double-buffering techniques to ensure that all components read from a consistent state

before calculating the next state.
* **Datasheets Are Not the Whole Story:** While datasheets describe a chip’s intended functions, they do not always capture the nuances of its behavior when interacting with other specific components in a real-world circuit. Accurate emulation often requires observing the hardware’s behavior directly or relying on the findings of other researchers.

5.0 Conclusion

The Commodore PET 4032 stands as a testament to the elegant and effective design principles of early microcomputers. Its architecture, while simple by modern standards, demonstrates a clear and logical separation of concerns. The memory-mapped I/O space at $E800-$EFFF provides a centralized and straightforward mechanism for software to control a host of peripherals through the 6520 PIA and 6522 VIA chips. PIA 1’s role in handling the keyboard and cassette drive, and PIA 2’s dedication to the IEEE-488 bus, create a robust I/O subsystem.

The system’s startup sequence reveals a sophisticated interplay between the KERNAL and Editor ROMs. The Editor ROM’s initialization routine is not merely a perfunctory setup; it implements the unique and clever Screen Line Link Table, a data structure essential for the 40-column PET’s text editing capabilities. This process, from the initial KERNAL reset to the final BASIC `READY` prompt, is a highly optimized path to prepare the machine for user interaction.

Finally, the challenges faced in emulating the PET’s hardware serve as a valuable lesson. The successful emulation of a component like the 6520 PIA depends not only on a faithful implementation of its documented features but also on a cycle-accurate simulation of its interactions within the complete system. This highlights the intricate, time-dependent nature of vintage hardware and the level of detail required to preserve its behavior in a modern software environment.

References

1. Commodore PET Programming Model - André Fachat (http://www.6502.org/users/andre/petindex/progmod.html)
2. Disassembly of: edit-4-40-n-60Hz-901499-01.bin - Steve J. Gray (https://www.zimmers.net/anonftp/pub/cbm/firmware/computers/pet/edit-4-40-n-60hz-901499-01.dis.txt)
3. Growing the 6502 emulator. - JustCode.be (http://justcode.be/2022/03/01/minimal_6502.html)
4. Peripheral Interface Adapter - Wikipedia (https://en.wikipedia.org/wiki/Peripheral_Interface_Adapter)
5. PET/CBM FAQ - PROGRAMMING - Port Commodore (https://portcommodore.com/dokuwiki/doku.php?id=larry:comp:commodore:pet:pet_faq-programming)