Commodore PET 4032 Startup and KERNAL Initialization Analysis

Report ID: PET4032-SR-20260210
Date: 2026-02-10
Status: Final

Executive Summary

This report provides a detailed technical analysis of the Commodore PET 4032's startup sequence, from power-on to the display of the BASIC ready prompt. The PET 4032, equipped with BASIC 4.0, relies on a coordinated process between its 6502 CPU and multiple ROM chips—the Kernal, Editor, and BASIC ROMs—to initialize system hardware and prepare the user environment.

The startup process begins with the CPU executing a reset routine in the Kernal ROM. This routine orchestrates the initialization of critical hardware, including the I/O chips (PIA, VIA) and, most importantly, the 6545 Cathode Ray Tube Controller (CRTC). Unlike earlier PETs with hardwired video logic, the 4032 requires the CPU to actively program the CRTC to generate a video signal. Failure in this step results in a blank screen, a common symptom of boot failure.

Following hardware setup, the Editor ROM routines are called to initialize the screen. This involves clearing the 1KB screen memory block at address `$8000`, setting up a line-linking table in zero-page RAM for the 40-column display, and homing the cursor. Once the screen is prepared, control is passed to the BASIC 4.0 interpreter, which displays the familiar " **COMMODORE BASIC 4.0** " banner and awaits user input.

This report details the specific ROM routines involved, the hardware conditions required for a successful boot, the intricacies of video and memory initialization, and common failure points that can prevent the system from starting, providing a foundational understanding for emulation, repair, and historical analysis.

---

1. Complete Startup Sequence: Power-On to BASIC Prompt

The boot process of the Commodore PET 4032 is a sequential execution of routines stored in its Read-Only Memory (ROM). This sequence initializes the hardware, prepares the screen, and loads the BASIC interpreter.

1.1. Power-On Reset and Kernal Initialization

1. **Power-On Reset (POR):** When the PET is switched on, the 6502 CPU's reset line is triggered. The CPU reads the reset vector address from memory locations `$FFFC` and `$FFFD`. This vector points to the main initialization routine within the `$F000` Kernal ROM.
2. **Kernal Hardware Setup:** The Kernal's reset routine begins by disabling interrupts and performing low-level hardware initialization. This includes setting up the data direction and control registers for the system's primary I/O chips:
    - **PIA1 (`$E810`):** Configured for keyboard matrix scanning and cassette tape interface.

    - **PIA2 (`$E820`):** Configured for the IEEE-488 peripheral bus.
    - **VIA (`$E840`):** Configured to manage the user port, tape motor control, and a timer that will later be used to generate the 60Hz system interrupt (IRQ).
3. **Editor ROM Entry:** After basic hardware setup, the Kernal routine jumps to the main entry point of the Editor ROM at address `$E000`. The Editor ROM is responsible for all screen and keyboard handling.

1.2. Editor ROM and Screen Initialization

1. **Editor Reset:** The jump to `$E000` is immediately redirected to the `RESET_EDITOR` routine at `$E036`. This routine orchestrates the entire screen setup.
2. **CRTC Initialization:** The first critical step is a call to a routine (`L_E617`) that initializes the MOS 6545 CRTC. The CPU sends a sequence of values to the CRTC's registers at `$E880` and `$E881`. This programs the CRTC with the correct timings and parameters for a 40-column by 25-line display, enabling it to generate horizontal and vertical sync signals for the monitor.
3. **Audible Feedback:** The system then calls a routine to "ring the bell" twice, producing the characteristic startup chirp via the internal piezo speaker. This provides audible confirmation that the CPU and ROMs are executing correctly.
4. **Screen Clearing:** Control is passed to the clear screen routine at `$E042`. This routine performs two key functions:
    - It initializes the **Screen Line Link Table** at zero-page addresses `$00E0` to `$00F8`. This 25-byte table stores the high byte of the starting memory address for each of the 25 screen lines, used to manage logical lines that wrap across physical lines.
    - It writes a space character (`$20`) to all 1024 bytes of the screen memory, located from `$8000` to `$83FF`, effectively clearing the display.
5. **Cursor Homing:** The routine concludes by setting the cursor's logical and physical position to the top-left corner of the screen (line 0, column 0).

1.3. BASIC Interpreter and Ready Prompt

1. **Control Transfer to BASIC:** With the hardware and screen initialized, the Kernal passes execution control to the BASIC 4.0 ROMs, located from `$B000` to `$DFFF`.
2. **BASIC Initialization:** The BASIC ROMs perform their own setup, which includes initializing pointers for program text storage (`$0028`), variable storage (`$002C`), and string memory (`$0030`).
3. **Display Banner:** BASIC then uses the Kernal's "Output to Screen" routine to print the startup message: `*** COMMODORE BASIC 4.0 ***`.
4. **Ready Prompt:** Finally, BASIC prints the amount of free memory and the `READY.` prompt, places the blinking cursor on the line below, and enters its main input loop to await commands from the user.

---

2. Analysis of Specific KERNAL/Editor Routines

The PET's functionality is built upon a set of routines at fixed addresses. The following addresses are critical to the system's operation.

- **`$E204` (Part of Output to Screen):** This address falls within the `L_E202` routine, which is the primary entry point for writing a character to the display. When called, this routine takes the character from the accumulator, interprets it, and places it in the correct screen memory location (`$8000-$83FF`) based on the current cursor position. It handles special control characters for curs-

or movement (up, down, left, right, home), reverse video, color changes, and screen clearing. It is also responsible for advancing the cursor and triggering a screen scroll when the cursor moves off the bottom of the display.

- `$E44C` **(Part of Main IRQ Handler):** This address is part of the `L_E442` routine, the main Interrupt Request (IRQ) handler. The 6502's IRQ vector at `$FFFE-$FFFF` is hardwired to point here. When a hardware interrupt occurs (typically from the VIA timer), the CPU jumps to this routine. Its primary job is to save the processor's state (A, X, Y registers) on the stack and then jump to the actual interrupt service routine at `$E455`.
- `$E46C` **(Part of Actual IRQ Service):** This address is inside the `L_E455` routine, which performs the substantive work for each system tick (60 times per second). Its key tasks include:
  1. **Updating the Jiffy Clock:** It increments the 3-byte system clock counter at `$008D` (`TIME`).
  2. **Keyboard Scan:** It scans the 10x8 keyboard matrix via PIA1 (`$E810`) to detect key presses and places them in the keyboard buffer queue (`$026F`).
  3. **Cursor Blinking:** It manages the software-driven cursor blink by decrementing a counter (`$00A8`) and, when it reaches zero, toggling the character under the cursor between its actual value and a solid block character.
- `$F481` **(Kernal I/O Routine):** This address is located in the main Kernal ROM (`$F000-$FFFF`). This region contains the device-independent I/O routines for peripherals like the cassette tape and IEEE-488 disk drives. While the exact function of `$F481` is not specified in the provided data, it is part of the cluster of routines responsible for file operations such as `LOAD`, `SAVE`, and `VERIFY`. For example, the `VERIFY` routine in BASIC 4.0 is located nearby at `$F4F6`. This code handles the low-level protocol for communicating with external storage devices.
- `$78F6` **(Not a ROM Address):** This address is located in the user RAM area, which on a 32KB machine extends up to `$7FFF`. It is not part of the permanent KERNAL or Editor ROM. An instruction at this address would be part of a program loaded by the user, a temporary data buffer, or an expansion ROM if one were mapped into this space. It plays no role in the standard boot sequence.

---

3. Hardware Conditions and Dependencies During Boot

The KERNAL's boot process is contingent on the correct functioning and response of several key hardware components. It actively waits for or initializes these components.

- **Interrupts:** Upon reset, interrupts are disabled. After the initial hardware and screen setup is complete, the KERNAL executes a `CLI` (Clear Interrupt Disable) instruction. This allows the periodic 60Hz interrupt signal from the VIA to be recognized by the CPU. This interrupt is the "heartbeat" of the system, driving the clock, keyboard scanning, and cursor blinking. The system does not wait for an interrupt during boot, but enables them as one of the final steps before passing control to BASIC.
- **Timers:** The primary timer is located within the 6522 Versatile Interface Adapter (VIA) at `$E840`. During initialization, the KERNAL loads one of the VIA's 16-bit timers with a value that will cause it to count down and generate an IRQ every 1/60th of a second. This timer must be configured correctly for the system to operate properly post-boot.

- **I/O Devices:** The KERNAL does not wait for I/O devices to respond during boot, but it actively initializes them.
  - **6545 CRTC:** This is the most critical dependency. The KERNAL must successfully write a series of configuration bytes to the CRTC registers at `$E880-$E881`. If the CRTC chip is faulty or does not respond, no video signal will be generated.
  - **6521 PIAs:** The KERNAL sets the data direction registers of the two PIA chips to configure their ports for input (e.g., reading the keyboard columns) or output (e.g., selecting keyboard rows).

---

4. Video and Screen Initialization

The PET 4032's use of a programmable 6545 CRTC makes its video initialization more complex than that of earlier models.

- **CRTC Programming:** The process is driven by a table of values stored in the Editor ROM. The initialization routine (`L_E61D`) iterates through this table, writing each value to the appropriate internal register of the 6545. These registers define parameters such as:
  - Horizontal and vertical total characters/rows.
  - Horizontal and vertical displayed characters/rows (e.g., 40x25).
  - Sync pulse widths.
  - Screen memory start address (pointing the CRTC to `$8000`).
  - Cursor start/end scan lines and cursor mode.
- **Timing Requirement:** This initialization must occur very early in the boot sequence. Without a programmed CRTC, the monitor receives no valid sync signals, resulting in a blank screen. The CPU is solely responsible for generating this signal via the CRTC; there is no hardware fallback.
- **Screen Memory:** The 1KB video RAM is located at `$8000-$8FFF`. The KERNAL's clear screen routine (`$E042`) directly accesses this memory range, writing a space character (`$20`) to each of the 1024 locations to ensure a clean display.
- **Software Cursor:** Although the 6545 CRTC supports a hardware-generated cursor, the PET 4032's design does not use it (the relevant CRTC pin is typically unconnected). Instead, the cursor is implemented entirely in software as part of the 60Hz IRQ service routine. This routine saves the character at the cursor's current screen memory address, writes a block character (`$A9`), and then restores the original character on the next cycle, creating a blinking effect.

---

5. Memory Initialization and Checks

The PET 4032's memory initialization during boot is minimal and focused on preparing the system for the BASIC environment.

- **Zero-Page Initialization:** The KERNAL and Editor ROMs initialize numerous critical variables in zero-page RAM (`$0000-$00FF`). The disassembly of the Editor ROM shows the setup of pointers and flags essential for screen management, including:
  - `PNT` (`$00C4`): Pointer to the current screen line's memory address.

    - `PNTR` (`$00C6`): The current cursor column (0-39).
    - `TBLX` (`$00D8`): The current physical screen line number (0-24).
    - `LNMX` (`$00D5`): The physical screen line length (40).
    - `LDTB1` (`$00E0`): The 25-byte screen line link table.
- **Screen RAM Clearing:** As detailed previously, the 1KB of screen RAM at `$8000` is cleared by writing spaces to every byte.
- **Absence of RAM Test:** The PET KERNAL does not perform a comprehensive RAM test on startup. It assumes the RAM is working correctly. If there is a fault in the RAM chips that the CPU needs for its stack or zero-page variables, the boot process will fail unpredictably, typically resulting in a hang or crash. This often manifests as a static, garbled screen display as the CPU loses execution control. After the KERNAL passes control to BASIC, the BASIC ROM initializes its own set of pointers (`TXTTAB`, `ARYTAB`, `FRETOP`) that define the boundaries for user programs, variables, and strings within the main RAM block.

---

6. Causes for KERNAL Boot Failure

A failed boot on a PET 4032 typically presents as a blank, static, or garbled screen. The underlying causes are almost always hardware-related.

- **Blank Screen (No Display):** This is the most common failure mode for CRTC-based PETs. It indicates that the 6545 CRTC is not being initialized.
    - **CPU or ROM Failure:** A dead 6502 CPU, a faulty Kernal ROM, or a faulty Editor ROM will prevent the CRTC initialization code from ever running.
    - **Power Supply Failure:** Missing +5V or +12V supply lines will prevent the CPU, ROMs, or CRTC from functioning.
    - **Clock or Reset Circuitry Failure:** If the CPU does not receive a valid clock signal or is held in a permanent reset state, it cannot execute any instructions.
- **Garbled or Static Screen (Looping/Hanging):** This symptom suggests the CPU started executing code but crashed.
    - **RAM Failure:** A faulty RAM chip is a primary cause. If the CPU attempts to use a bad RAM location for the stack (page 1) or zero-page variables, it will lead to a crash. The video RAM may still be partially functional, displaying the random data present at power-on.
    - **Address/Data Bus Fault:** A short or open circuit on one of the address or data bus lines can cause the CPU to read incorrect instructions or data, leading to a hang.
    - **Faulty I/O Chip:** A faulty PIA or VIA that interferes with the data bus can also cause the system to crash during the initialization sequence.

In these scenarios, the KERNAL is not “looping” in a controlled manner but is typically halted or executing nonsensical instructions from invalid memory locations, leaving the screen in an uninitialized state.

---

7. Common Emulator Implementation Issues

Accurately emulating the PET 4032 boot sequence requires precise replication of its hardware and timing. Common challenges include:

- **CRTC Emulation:** The 6545 CRTC is a complex, programmable chip. An emulator must accurately model its internal registers, its timing relative to the CPU clock, and how it accesses screen RAM to generate the video display. An inaccurate CRTC model can lead to display artifacts, incorrect screen geometry, or a complete failure to produce an image.
- **CPU and I/O Timing:** The interaction between the 1MHz 6502 CPU and the I/O chips (PIAs, VIA) is timing-sensitive. The 60Hz IRQ from the VIA must be emulated precisely, as it drives the software-based cursor and keyboard input. Incorrect timing can cause a non-functional keyboard, a static cursor, or programs that rely on the jiffy clock to fail.
- **Correct ROM Versions:** The PET 4032 specifically uses BASIC 4.0 ROMs, which are tied to the CRTC-based hardware. Using an incorrect ROM set (e.g., from an earlier non-CRTC PET) will fail, as the code will not match the hardware it is trying to control. Emulators must pair the correct Kernal, Editor, and BASIC ROM images for the 4032 model.
- **Memory Map and I/O Mirroring:** The PET's memory map, with ROMs, RAM, and I/O registers at specific locations, must be replicated perfectly. Emulators must also handle the "mirroring" of I/O registers, where a chip responds to multiple addresses within its designated block. For instance, the CRTC's two registers at `$E880` and `$E881` are mirrored throughout the `$E880-$E88F` range. Failure to model this can cause compatibility issues with software that uses non-canonical addresses.

References

1. Commodore PET - Wikipedia (https://en.wikipedia.org/wiki/Commodore_PET)
2. Commodore PET 4032 startup sound download - Reddit (https://www.reddit.com/r/Commodore/comments/16m6klr/commodore_pet_4032_startup_sound_download/)
3. Commodore PET 4032 computer - oldcomputers.net (https://oldcomputers.net/pet4032.html)
4. Commodore PET FAQ - 6502.org (http://www.6502.org/users/andre/petindex/local/petfaq.html)
5. Commodore PET 4032 startup sound download - Reddit (https://www.reddit.com/r/commodorepet/comments/16m6l0i/commodore_pet_4032_startup_sound_download/)
6. Commodore PET Programming Model - 6502.org (http://www.6502.org/users/andre/petindex/progmod.html)
7. DAVES OLD COMPUTERS - Commodore PET - dunfield.classiccmp.org (http://dunfield.classiccmp.org/pet/)
8. PET: editor kernal routines? - Lemon64.com (https://www.lemon64.com/forum/viewtopic.php?t=65731&sid=ea137efb04ba68104a3c471e1a276ca8)
9. Commodore PET ROM descriptions - 6502.org (http://www.6502.org/users/andre/petindex/roms.html)
10. C64 KERNAL ROM: Making Sense - C64os.com (https://c64os.com/post/c64kernalrom)
11. Kernal - C64-Wiki (https://www.c64-wiki.com/wiki/Kernal)
12. KERNAL - Wikipedia (https://en.wikipedia.org/wiki/KERNAL)
13. Commodore KERNAL History - Pagetable.com (https://www.pagetable.com/?p=926)

14. Commodore PET repair info - 6502.org (http://www.6502.org/users/andre/petindex/repairs.html)
15. A dumb (but obscure) Commodore PET video implementation question: Hardware cursor? | Vintage Computer Federation Forums (https://forum.vcfed.org/index.php?threads/a-dumb-but-obscure-commodore-pet-video-implementation-question-hardware-cursor.79194/)
16. PET Emulators - 6502.org (http://www.6502.org/users/andre/petindex/emus.html)
17. GitHub - DLehenbauer/commodore-pet-clone: Open hardware clone of the Commodore PET 4032 (https://github.com/DLehenbauer/commodore-pet-clone)
18. Masswerk's Commodore PET 2001/4032 Online Emulator (https://www.masswerk.at/pet/)
19. PET index - 6502.org (http://6502.org/users/andre/petindex/index.html)
20. Troubleshooting Common Problems with the Commodore PET 2001 - dasarodesigns.com (http://www.dasarodesigns.com/projects/troubleshooting-common-problems-with-the-commodore-pet-2001/)
21. Commodore PET 2001 stuck on boot displaying garbled characters - Reddit (https://www.reddit.com/r/Commodore/comments/z0xbtw/commodore_pet_2001_stuck_on_boot_displaying/)
22. CBM PET 2001 garbled screen - Retrocomputing Stack Exchange (https://retrocomputing.stackexchange.com/questions/16483/cbm-pet-2001-garbled-screen)
23. Commodore PET 4000 Series Troubleshooting Guide - Retro Tech Collection Wiki (https://wiki.retrotechcollection.com/Commodore_PET_4000_Series_Troubleshooting_Guide)
24. Commodore PET 2001 Repair - Garbage on screen - Tynemouth Software (http://blog.tynemouthsoftware.co.uk/2017/07/commodore-pet-2001-repair-garbage.html)
25. Commodore PET 4032 no boot no beep - AmiBay (https://www.amibay.com/threads/commodore-pet-4032-no-boot-no-beep.33987/)
26. Commodore PET 2001 Repair Part 1 - Tynemouth Software (http://blog.tynemouthsoftware.co.uk/2024/02/commodore-pet-2001-repair-part-1.html)
27. Disassembly of: edit-4-40-n-60Hz-901499-01.bin - zimmers.net (https://www.zimmers.net/anonftp/pub/cbm/firmware/computers/pet/edit-4-40-n-60hz-901499-01.dis.txt)