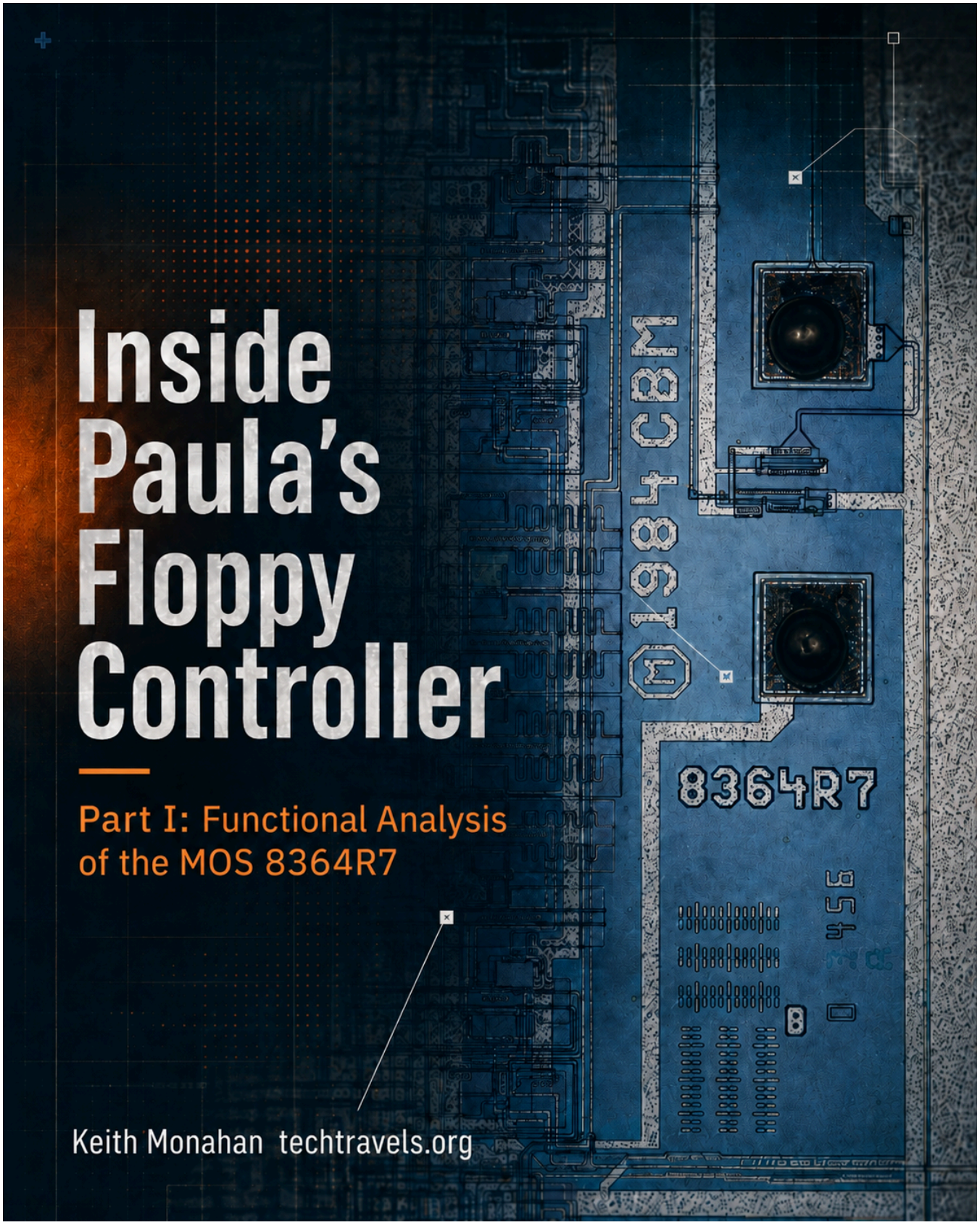



Contact: insidepaula@techtravels.org

# Contents

# 1. Introduction

The Amiga uses a custom chip called Paula for audio, interrupts, serial communications, and part of the floppy-disk subsystem. Amiga documentation describes how to use Paula's disk registers. The floppy logic inside the chip has remained largely unexplained.

This document explains what that logic does, why each major part is required, and how the parts cooperate during a disk read or write. It begins with the complete floppy subsystem and then examines Paula's read and write paths as a series of functional activities. This independent technical report is written for technically curious readers. Its scope is the functional architecture of the controller; the detailed transistor circuits are the subject of Part II.

## 1.1 Why this paper exists

Like many of you, I grew up in the 1980s. My interest in the Amiga began with an Amiga 500 in my childhood bedroom. Hard drives were not yet affordable, and floppy disks were an inseparable part of the experience. Using them was tactile and noisy: the drive clicking (thank God for those no-click utilities), the satisfying clunk of a disk seating fully in the drive, and the stepper motor's scanning noises. I had boxes and boxes of disks stored in cases from mall stores such as Electronics Boutique and Charles Babbage's (before it became GameStop!). They held games, applications, demos, `.MOD` music files, early copies of my resume, and AmigaBASIC source code I had written.

By the second half of the 1990s, PCs had become much more powerful, and I had made the fateful switch to the dark side (a.k.a. PCs). It wasn't until 2005 that I became concerned about preserving my digital childhood. I built an `.ADF` creator called the *Amiga Floppy Project*. This was before KryoFlux, Greaseweazle, Gotek drives, and today's widely available disk tools. The project combined a cross-platform Java application with a microcontroller-based disk interface; I later recreated the interface with an FPGA and wrote its logic in Verilog.

I had long known how to work with the practical effects of the system's disk reads and writes, yet I wanted to understand more. Preserving my own files had left me with an itch to scratch: what was Paula *actually* doing inside the machine?

Commodore's documentation describes the programming interface, but it does not go deeply enough to explain how Paula actually does the work. In July 2023, Frank Wolf and ttlworks made it possible to investigate the circuitry itself by publishing their dissection of the 8364R7 Paula. Their work turns the physical silicon into vectorized layouts, connected schematics, and notes that can be followed signal by signal.

This paper is my attempt to answer that question. It follows a read from the pulses arriving from the drive to the words placed in chip RAM, then follows a write from chip RAM to the pulses sent back to the drive. Along the way, it explains clock recovery, sync detection, word framing, buffering, DMA, serialization, write precompensation, and transfer control, and identifies the physical modules that perform each job.

Part I explains these operations at the system and functional-block level. Part II will stand on its own as the circuit-level account, covering transistor paths, logic equations, PLA reductions, timing proofs,

and explicitly open evidence gaps. Chapter 6, *Three Paula Mysteries* uses the same model to examine the `DSKLEN` double write, the final lost write bits, and the final read word that may not arrive.

## 1.2 From documentation to dissection

The *Amiga Hardware Reference Manual* explains how software controls the disk hardware: which registers to use, how to start DMA, how to configure sync detection, and which limitations software must accommodate. It does not describe the circuits that perform those operations.

Glenn Keller, Paula's designer, is also the inventor named on US patent 4,780,844, *Data Input Circuit with Digital Phase Locked Loop*. The patent describes a digital floppy input circuit that closely matches the main principles of Paula's data separator. It is especially useful for understanding why the circuit tracks the timing of incoming disk pulses and how it applies phase and frequency corrections.

The patent's scope excludes Paula's DMA control, transfer-length counter, buffering, sync interrupt, register interface, and write precompensation. It presents a preferred implementation instead of a circuit diagram of the shipped 8364R7. The manual supplies operating behavior, the patent supplies design context, and the dissection connects both to physical hardware.

### The work of Frank Wolf and ttlworks

Frank Wolf and the researcher known online as ttlworks published the Paula dissection project, images, notes, and extensive schematics through 6502.org; the project thread is at 6502.org/forum/viewtopic.php?t=7681. Their work exposes physical circuits that had previously been unavailable for study.

Their project notes give some indication of the scale: Frank arranged the decapsulation and microscope photography, then spent a year stitching and vectorizing the images; ttlworks reports spending about 800 hours extracting the schematic. The floppy controller is one of Paula's most complex sections.

The dissection notes identify the floppy section's main structures and its common whole-track use, while noting that the programmed transfer can be shorter. They leave the detailed operation open and invite further analysis. This paper builds on that foundation. The reproduced dissection images are used with the authors' permission.

This paper would not have been possible without their work. I cannot analyze what I cannot see, and they deserve tremendous credit for the time, money, and effort they invested. Great job, guys!

## 1.3 What this project is examining

The chip examined here is the MOS/CSG 8364R7 Paula. In the dissection, Paula is divided into numbered sections according to function and physical location. Section 10 contains the floppy-controller circuits. It is further divided into labeled regions 10a through 10v.

These labels locate circuits in the dissection. A single function can extend across several labeled regions, and one region can contain logic used by several activities. This document therefore starts with functions such as input conditioning, clock recovery, buffering, and serialization. The module labels are included as locations readers can use to return to the source material.

The analysis asks three questions:

- How does the Amiga turn the pulses received from a floppy drive into data in memory?

- How does it turn data in memory into correctly timed write pulses for the drive?
- Which parts of those operations are handled by Paula, and which are handled by the drive, the CIA chips, Agnus, or software?

Paula's interface is a timed stream of raw disk bits. Software applies the disk-format rules and locates the requested data within that stream.

## 1.4 What is included in the dissection

The dissection provides several views of the same chip. Die photographs show the physical structures on the silicon. Traced layout images make those structures easier to follow. Schematic sheets redraw the circuits using transistors and familiar logic symbols. An EAGLE schematic file connects the sheets into a source that can also be queried by software. Written notes identify the broad purpose of each numbered section and record the dissection authors' observations.

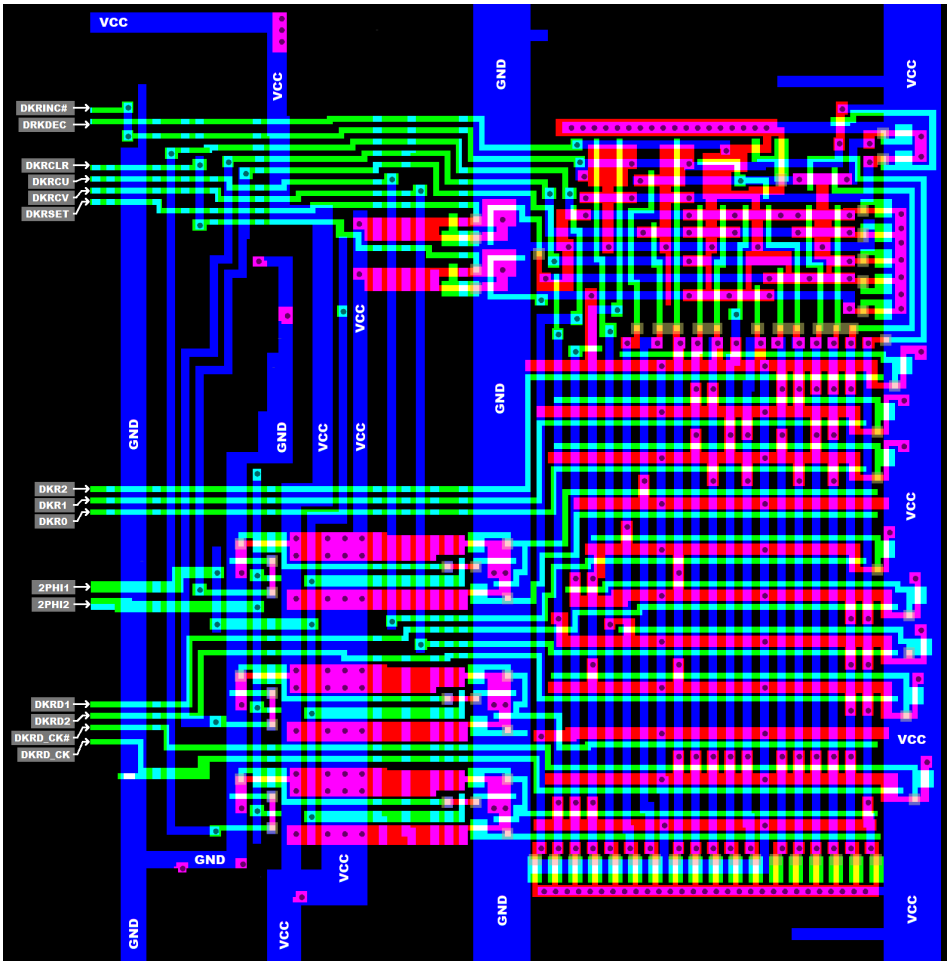
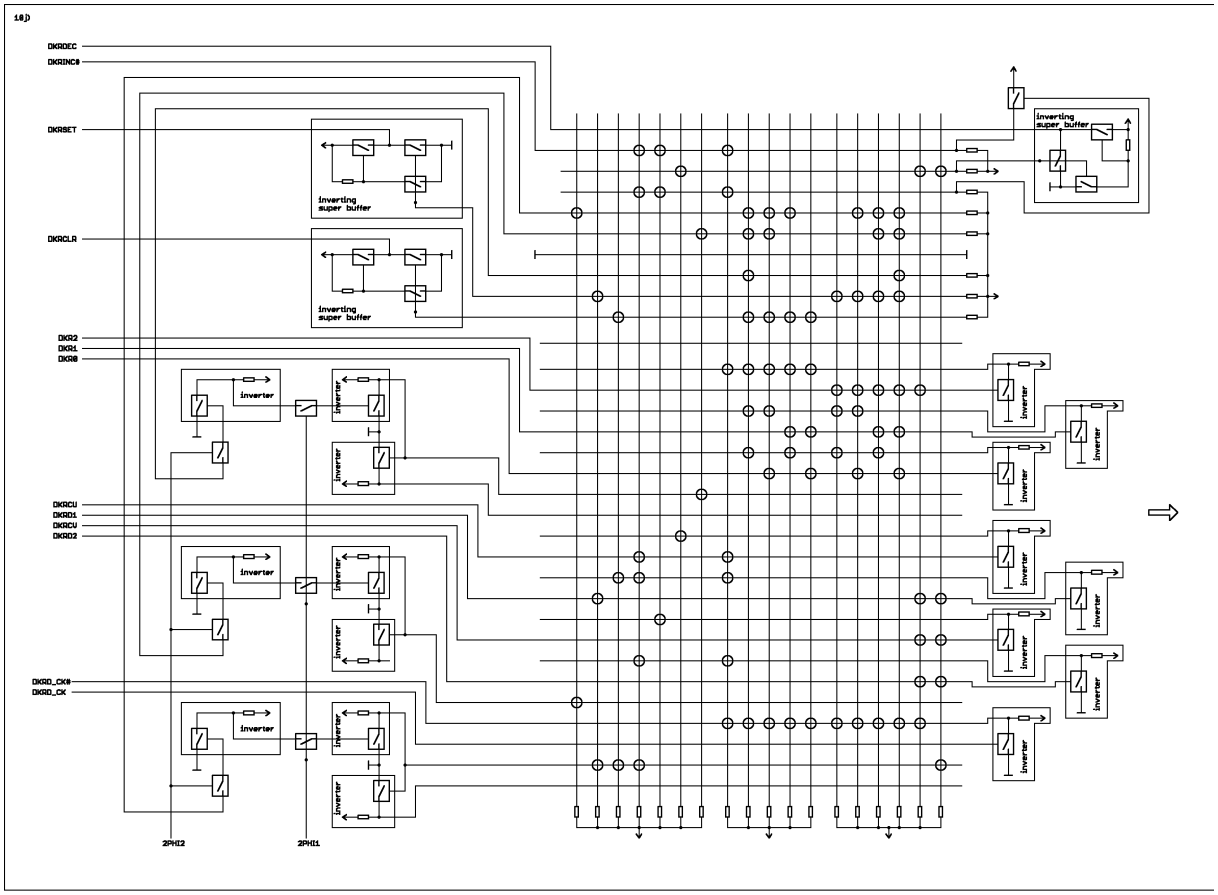
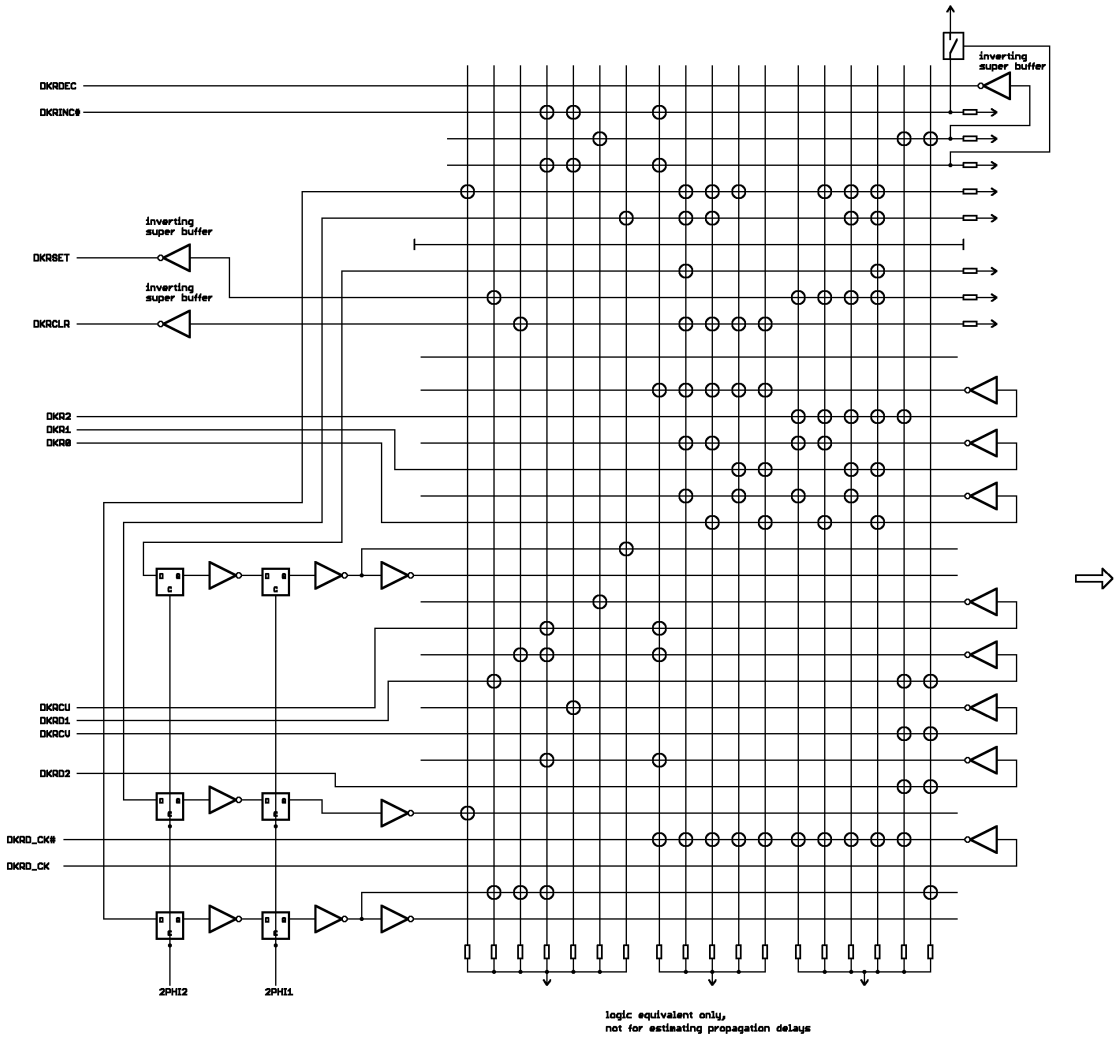


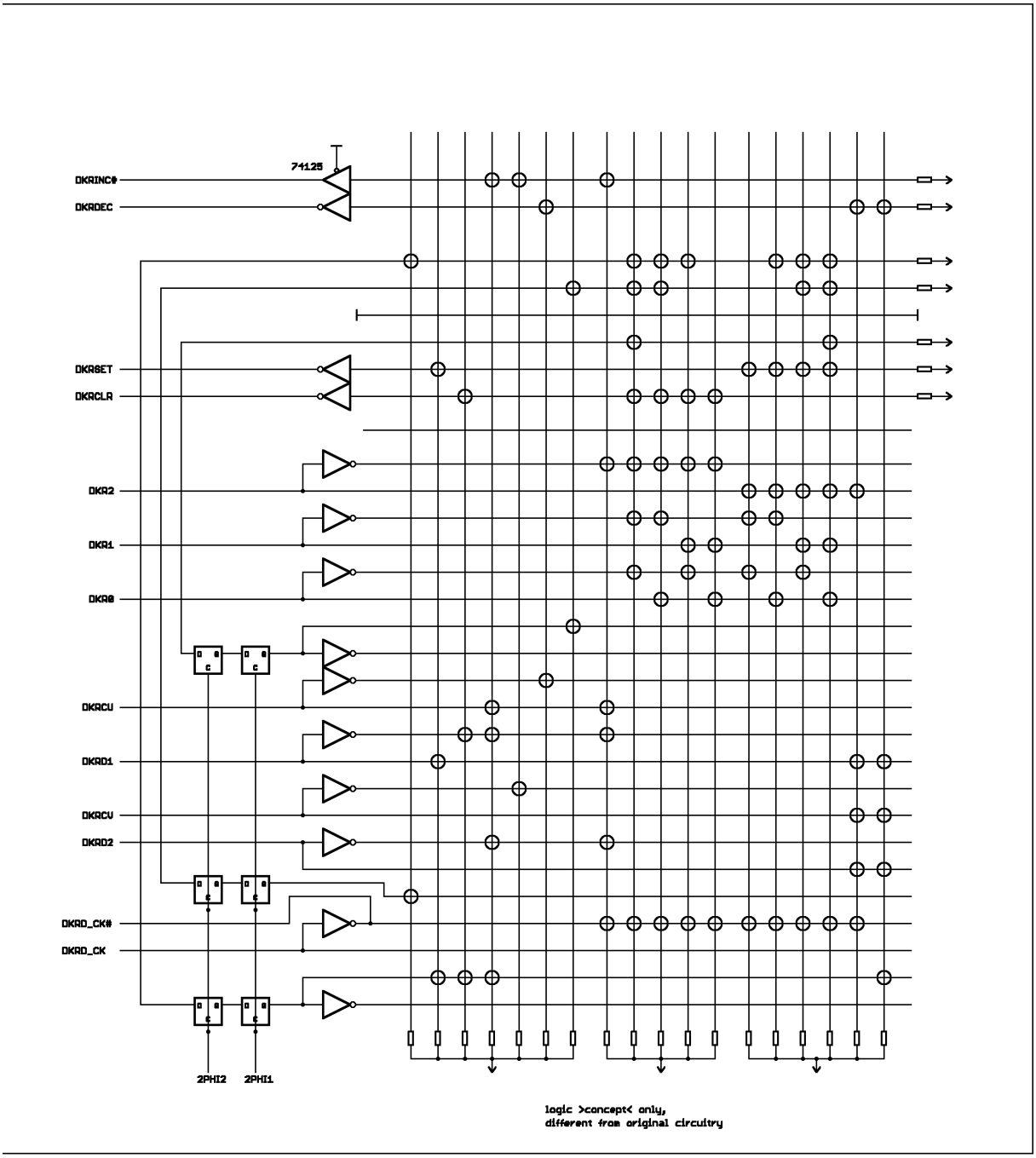

**Figure 1-1. Vectorized physical layout of module 10j.** This is the large programmable-logic region used by the read data separator. The colors distinguish physical layers and connections in the traced chip layout; the labels around the edges identify signals entering or leaving the region. At this stage, the image establishes where devices and conductors exist. Functional explanation requires the connections to be traced and tested. Reproduced unchanged from the 8364R7 dissection materials published by Frank Wolf and ttlworks, used with permission.




**(a) Transistor-level extraction.**




**(b) Logic-symbol redraw.**




**(c) Conceptual simplification.**

**Figure 1-2. Three views of the same schematic extraction.** Panel (a) preserves the transistor-level structure, panel (b) redraws it with logic symbols, and panel (c) simplifies it conceptually. The source warns that the conceptual version differs from the original circuitry, so this analysis checks transistor connectivity rather than treating the simplification as an exact replacement. Reproduced from sheet 48 of the 8364R7 dissection published by Frank Wolf and ttlworks, used with permission; the original side-by-side panels are separated here and their strokes strengthened for legibility.

Section 10 contains two programmable logic arrays, a transfer-length counter, shift registers, a sync comparator, a multi-level data buffer, and probable write-precompensation logic. The dissection identified those structures but left many of their relationships open.

The analysis compares the dissection with the *Amiga Hardware Reference Manual* and Keller's patent to separate physical findings from plausible interpretations. Because the schematic is a hand transcription rather than an original Commodore design document, the die images control when a connection is unclear or conflicts with observed operation.

## 1.5 From unknown to reproducible

A circuit can be physically present in the source material and still not be understood. This analysis treats understanding as a progression of questions in place of a vague confidence score. A claim begins with no supported information. Locating the relevant source establishes where to look; extraction establishes what connects; explanation supplies the causal mechanism. An independent test checks that account. A sufficiently complete record lets another investigator repeat the evidence chain.

The six labels are project-specific vocabulary built from ordinary engineering activities. They make no claim to industry-standard usage. Their purpose is to make the evidence path readable without requiring prior knowledge of how one specialized confidence term supposedly ranks against another.

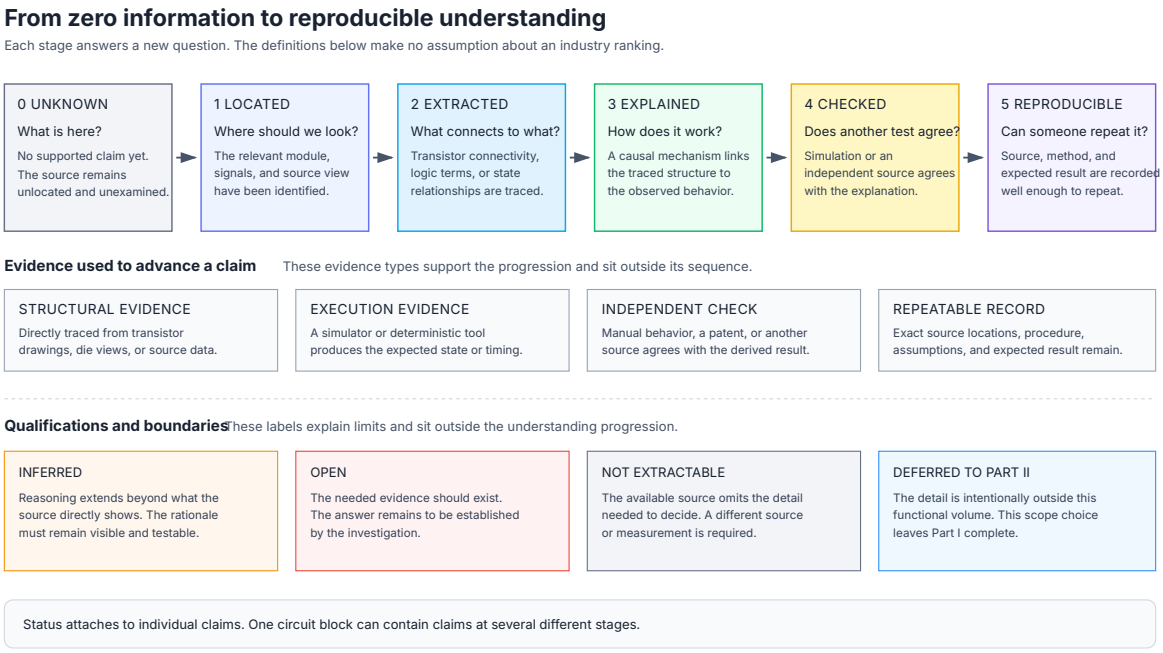



**Figure 1-3. Path from zero information to reproducible understanding.** Each stage states the new question that has been answered. Structural extraction, simulation, and independent corroboration are kinds of evidence used to move a claim forward. They sit outside the sequence of stages. Likewise, *inferred*, *open*, *not extractable*, and *deferred to Part II* qualify a claim or explain why it stops. More than one may apply. Deferral records a scope boundary and leaves the completeness of this volume intact.

The method begins with the raw transistor copy. Identify the externally visible signals and trace until every device in the selected path has either an explained role or an explicitly named steering function. Reduce that connectivity to logic or state relationships. Then check the result by deterministic simulation or by an independent source such as the *Amiga Hardware Reference Manual*, the die image, or the Keller data-separator patent. Finally, retain the source locations, assumptions, procedure, and expected result. Here, "reproducible" means that another investigator can repeat the stated chain and see whether it holds. Every result remains open to revision when better evidence appears.

Part I uses the plain-language stages in Figure 1-3 and states open boundaries directly. It stops at functional architecture and supporting evidence. Part II covers the transistor schematics, detailed equations, PLA reductions, and cycle-level proofs, preserving enough material for a separate circuit-level account.

For the principal mechanisms, Part I reaches checked functional understanding. Part II will include the source locations, equations, device accounting, replay procedures, and expected results needed to reproduce the circuit proofs.

## 1.6 What this analysis adds

This volume brings together an explanation from source material that previously existed mainly as separate drawings, annotations, and behavioral descriptions. Commodore documentation remains the authority for externally specified behavior. The analysis:

- traces the complete read and write operations across section 10′s twenty-two modules, 10a through 10v, and marks where the drive, Agnus, and CIA take over;
- explains the three-stage data path and the parallel token path as the common machinery behind receive buffering, transmit buffering, and DMA handoff;
- explains why word-sync capture begins with the word *following* the match;
- connects the selectable write-delay paths to the documented precompensation magnitudes while leaving the exact pattern-selection rules open;
- derives the four-state transfer sequencer and connects it to the documented double write to `DSKLEN`; and
- uses Paula's read and write end effects to test and bound the functional model.

## 1.7 The functional picture

The drive handles the magnetic medium and presents digital transition pulses. The CIA chips handle selection, motor control, side selection, stepping, and status. Agnus owns chip-RAM addresses and bus cycles, while software supplies the disk-format rules.

Paula occupies the timed data path between them. During a read it recovers bit-cell timing, reconstructs serial bits, finds a programmable sync pattern, forms words, and buffers them for DMA. During a write it buffers prepared words, serializes them, generates transition timing, applies write precompensation, and drives the disk interface. Both directions share a three-word data cascade, a parallel token queue, the transfer sequencer, and the length counter.

# 2. Functions of a Floppy Disk Controller

A floppy disk stores information as a sequence of magnetic changes along a circular track. The computer cannot use those changes directly. It needs electronics that can turn the signal from the drive into timed bits when reading, and turn timed bits into write commands when recording new data.

Those operations involve more than one component. The drive handles the head, motor, head-positioning mechanism, and the small electrical signals associated with the head. The controller handles time-critical digital processing and transfers. The computer's software decides what the recorded bits mean as files, sectors, headers, and error-checking values. Different computer systems place the boundary between controller hardware and software in different places.

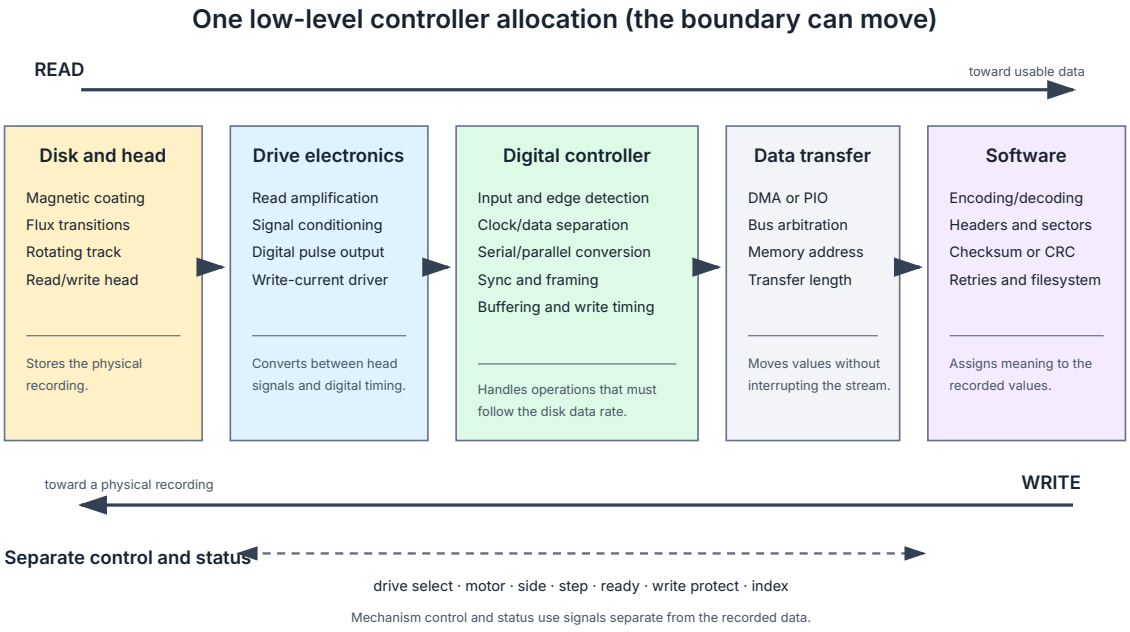



**Figure 2-1. One low-level allocation of floppy-disk responsibilities.** The read path runs from the magnetic medium toward software. The write path runs in the opposite direction. Drive-control and status signals are separate from the recorded data path. The boundaries are illustrative rather than Paula's exact partition: Paula, for example, counts the transfer length while Agnus supplies its memory address. More integrated sector controllers also move encoding, framing, and error checking from software into controller hardware.

## 2.1 The drive and its analog electronics

The surface of a floppy disk is coated with magnetic material. The drive's read/write head is positioned close to that surface. As the disk rotates, the head passes over a track containing changes in magnetic direction.

During a read, a change in magnetic flux produces a small voltage in the head. That voltage is not suitable for direct use by a digital controller. Its amplitude and shape depend on the head, disk, speed,

recorded pattern, and electrical noise. The drive therefore amplifies and conditions the signal before presenting a digital read-data pulse to the computer. The exact analog design differs among drive models, but the purpose is the same: each detected magnetic transition must become a reliable digital event at the drive interface. A false pulse can become an extra bit; a missed pulse can remove recorded data. The drive performs this first stage while the head signal is still small and analog.

During a write, the direction is reversed. The computer supplies digital transition timing and a write-enable signal. Electronics in the drive switch current through the selected head so that the magnetic material is recorded in the required direction. The controller does not drive the head coil directly.

The drive also contains the motor and the mechanism that moves the head between tracks and selects a recording surface. Signals such as drive select, motor on, step, step direction, side select, track zero, ready, write protect, disk change, and index belong to this mechanical and status interface. Which of these signals are present and their exact meanings vary among drive standards.

## 2.2 The read path

The drive supplies a timed series of pulses, with one pulse for each detected magnetic transition. Several activities are required before the computer can place useful data in memory.

### 2.2.1 Input conditioning and event detection

The drive signal arrives asynchronously: its transitions are not aligned to the controller's internal clock. The input may also have edge shapes or short disturbances that ordinary internal logic should not receive directly. Input conditioning gives the signal defined digital thresholds, and synchronization brings it safely into the controller's clock domain. Edge-detection logic then produces one internal event for each accepted drive pulse.

### 2.2.2 Why a data separator is required

The drive provides read-data pulses but no separate clock that marks the position of every recorded bit. The read-data line is held high by a pull-up. When the drive detects a magnetic flux reversal, its open-collector output pulls the line low, so the falling edge begins the pulse that reports that reversal. The absence of a pulse during a bit cell can also carry information. The controller must therefore determine where each bit cell begins and ends before it can decide whether that cell contains a transition.

Fixed timing is not sufficient. The rotational speed is controlled within a range, not at one perfectly constant value. Disks and drives also introduce smaller timing changes. A controller that sampled only at fixed intervals could gradually sample too early or too late, even when every transition had been detected correctly.

The data separator solves this problem. It maintains an estimate of the current bit cell timing, compares incoming transitions with that estimate, and corrects its phase or rate when the transitions arrive earlier or later than expected. It then produces the regular timing events used to decide the value of each recorded cell.

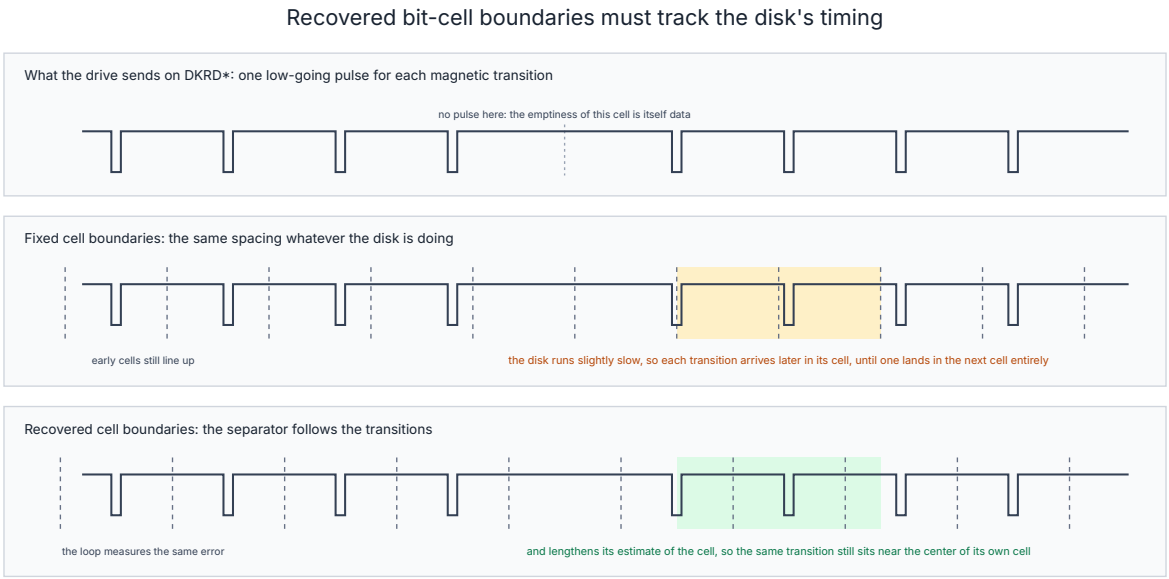



**Figure 2-2. Recovered boundaries track the disk.** The external `DKRD*` line idles high and the drive reports each magnetic transition as a low-going pulse. The falling edge marks the start of that pulse; a cell that contains no transition has no pulse. If the cell boundaries are fixed, a disk running slightly slow pushes each transition later within its cell until one crosses into the next cell and the reconstructed data is wrong. Recovered boundaries that track the transition cadence keep each pulse within its intended cell.

The name reflects its two outputs: recovered timing and reconstructed data. The incoming wire carries both kinds of information in the timing of one pulse stream, so the controller has to separate them before the data can be shifted into a register.

### 2.2.3 Serial-to-parallel conversion

The disk surface presents one recorded cell after another. The computer's memory and processor normally transfer bytes or words. A shift register is therefore required to collect the recovered serial bits into a parallel value.

This conversion changes the form in which the same bits are held. Once a complete byte or word has been assembled, it can be buffered and transferred over the computer's data bus.

### 2.2.4 Sync detection and framing

A continuous bitstream does not inherently identify the first bit of a byte or word. If reception begins at an arbitrary point on a rotating track, the controller needs some way to establish a useful boundary. Disk formats place known patterns in the stream for this purpose.

Some controllers recognize fixed address marks defined by the formats they support. Others, including Paula, can compare the incoming stream with a pattern selected by software. When the pattern is found, the controller can report the event or align the following transfer, giving later bytes and words the intended starting point.

### 2.2.5 Buffering and memory transfer

The disk continues rotating while the controller waits for access to memory. The incoming stream cannot normally be paused while another device uses the memory bus. A buffer holds completed bytes or words until the computer can accept them.

Data can then be transferred by programmed input/output or DMA. With programmed I/O, the processor reads or writes the controller at the required times. With DMA, control logic requests a memory cycle and the system transfers the data with less immediate processor involvement. Either method needs a way to detect overrun: if the buffer fills before a value is removed, incoming data will be lost.

## 2.3 The write path

Writing does not require the controller to recover a clock from the disk. The controller chooses the output timing. It still needs several stages to convert data in memory into signals the drive can record.

### 2.3.1 Parallel-to-serial conversion

Memory supplies bytes or words, while the drive expects transitions in time order. A write shift register converts each parallel value into a serial bitstream. A buffer keeps additional values ready so that a delay in memory access does not interrupt the recording.

### 2.3.2 Write timing

Each output bit must occupy a defined time interval. The controller generates this interval from a stable clock and advances the write shift register at the selected data rate. A later read reconstructs the bit cells from that transition spacing.

### 2.3.3 Encoding

The raw data bytes cannot always be written directly as transitions. Recording codes such as FM, MFM, and GCR impose rules on where transitions may appear. Those rules ensure that the read path receives enough timing information and avoids patterns the recording system cannot reproduce reliably.

A sector-oriented controller may generate the encoded form in hardware. A lower-level controller may expect software to provide data that is already encoded. This is one of the main ways floppy-controller designs differ.

### 2.3.4 Write precompensation

Closely spaced magnetic transitions can be detected at slightly displaced times when the track is read back. This effect is most important where bit density is greatest. Write precompensation moves selected transitions slightly earlier or later during the write so that their expected readback positions are closer to the intended timing.

Precompensation improves the timing accuracy of the physical recording after the data has been selected. It may be performed inside the controller or by additional circuitry, depending on the design.

### 2.3.5 Write data and write enable

The drive needs both the transition stream and a signal that authorizes writing. The write-enable or write-gate signal limits head current to the intended part of the operation. Without that control, selecting the drive or sending ordinary interface activity could alter data outside the requested write.

## 2.4 Disk control and data transfer

The read and write data paths do not position the head by themselves. Before a transfer, the system must select a drive, choose a side, start the motor when necessary, move the head to the required track, and check status such as ready and write protect. The index signal can identify a particular rotational position, but not every controller uses it in the same way.

The system must also know where data will be placed in memory and how much data to transfer. Some controllers contain address or sector registers and exchange bytes directly with the processor. Others cooperate with a separate DMA controller. Paula requests memory service and counts words, but Agnus supplies the memory address and performs the actual chip-RAM cycles.

## 2.5 Sectors, encoding, and error checking

Files are not stored as an unlabeled sequence of bytes. A disk format defines how a track is divided into sectors and how each sector is identified. It also defines sync patterns, headers, gaps, recording code, and a checksum or CRC used to detect damaged or incorrectly read data.

These rules must be implemented somewhere. A high-level floppy controller can search for an address mark, read a sector header, compare the requested track and sector, decode the data, verify the CRC, and return a detailed result status. Software gives such a controller a command describing the requested operation.

A lower-level controller transfers a less interpreted representation. Software must then locate headers, decode the recording format, identify sectors, verify their contents, and decide whether to retry. This requires more processor work, but it also allows software to support disk formats that were not built into the controller.

## 2.6 Three ways to divide the work

Floppy-controller designs become easier to compare when we ask the same questions of each one: what does the host request, which time-critical operations happen in hardware, and which disk-format rules remain in software?

| Design approach | Hardware handles | Software handles |
|---|---|---|
| Sector-command controllers, including the Western Digital WD17xx/WD177x family, NEC µPD765A / Intel 8272, and Intel 82077AA | The host requests operations such as reading or writing a sector. The controller recognizes address marks and sector fields, performs FM/MFM encoding, generates or checks CRCs, and reports a result. Later devices integrate more of the data separator, buffering, | Software selects the operation, supplies or receives sector data, and responds to the reported status. The controller's command set and format logic define the sector formats it can handle directly. |

| Design approach | Hardware handles | Software handles |
|---|---|---|
| | precompensation, and drive interface. | |
| Apple Disk II | A small controller card and its firmware provide a low-level connection between the processor and drive. | The 6502 and controller firmware cooperate on low-level transfers; Apple DOS supplies the sector and file organization. |
| Paula | Paula recovers read timing, reconstructs the serial stream, detects a programmable sync word, forms and buffers words, requests DMA, serializes writes, and applies write precompensation. | Software builds or interprets the recorded format, including encoding, sector layout, headers, and checksums. |

Paula's boundary keeps the operations that must pace the rotating disk in hardware while leaving the meaning of the recorded stream to software. Compared with Apple Disk II, Paula provides much more autonomous timing, buffering, and DMA support. Compared with the WD, NEC, and Intel controllers, it leaves more of the sector format open to software.

Software made practical use of that boundary. Commodore documented support for double-density MFM layouts beyond the native Amiga format. CrossDOS supplied the DOS filesystem and sector interpretation for 720 KB PC disks, while A-Max supplied the Macintosh interpretation for compatible 720 KB MFM media. Paula handled the timed MFM stream in both cases. The changed software supplied the meaning of the recorded data.

# 3. The Amiga floppy subsystem

The complete Amiga floppy operation is divided among software, the CIA chips, Agnus, Paula, chip RAM, and the drive. Paula handles the time-critical bitstream in the middle of that larger operation.

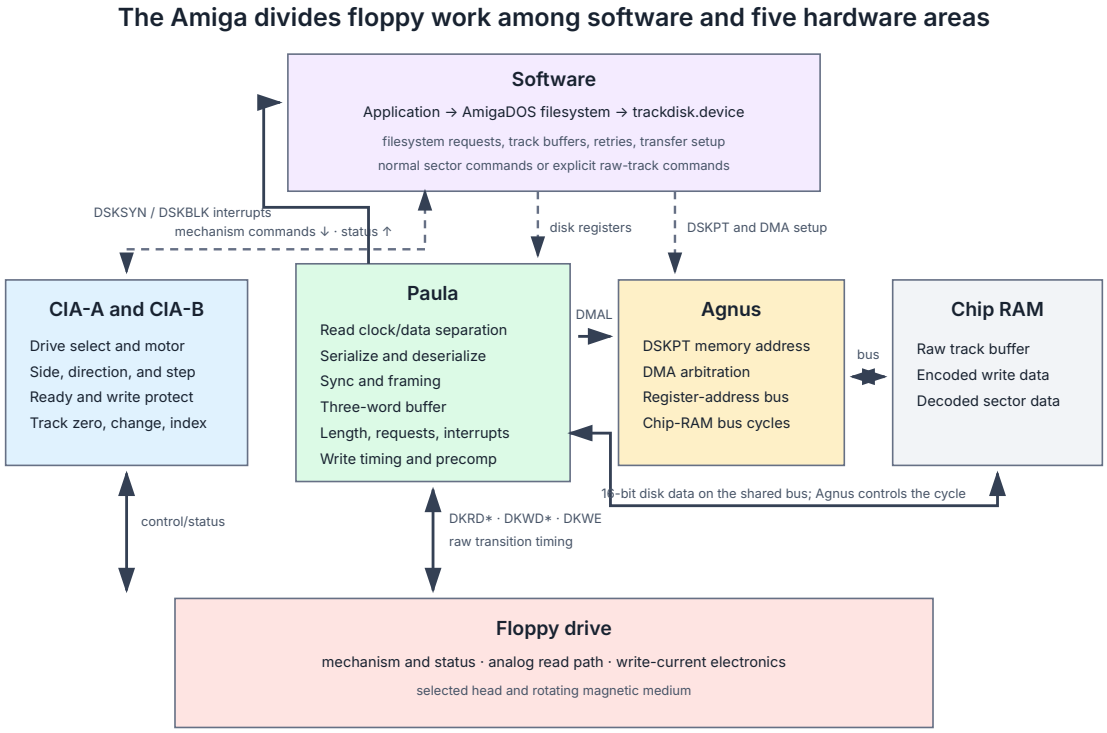



**Figure 3-1. The Amiga floppy subsystem.** The CIAs carry drive-control and status signals. Paula carries the raw read and write signals. Agnus controls DMA addressing and access to chip RAM. Software initiates and completes the request.

## 3.1 The system-level parts

### 3.1.1 Applications and the filesystem

An application normally asks AmigaDOS to read or write file data. It does not need to know the current track, the sector encoding, or the custom-chip register sequence. The filesystem determines which disk blocks contain the requested bytes and sends the appropriate requests to the disk device.

### 3.1.2 `trackdisk.device`

`trackdisk.device` is the normal low-level AmigaOS interface to the built-in floppy drives. Its central job is to present sector-oriented I/O on top of a mechanism and DMA engine that normally move recorded data a track at a time. It accepts device I/O requests, controls the drive, interprets the Amiga track format, and hides track boundaries from ordinary callers.

Each drive has its own track buffer. If that buffer already represents the track containing a requested sector, an ordinary read can be satisfied from memory without starting Paula. If another track is needed, `trackdisk.device` first commits the buffer if it has been changed, then positions the drive and reads the required track. It locates and validates the sectors in the captured representation and retains the result for later requests to the same track.

An ordinary `CMD_WRITE` likewise changes sector data in the buffered track and marks the buffer dirty; it does not necessarily cause an immediate physical write. A `CMD_UPDATE` request writes a changed buffer to disk, and a dirty buffer must also be committed before it can be reused for another track. `CMD_CLEAR` instead invalidates the buffered track and forces a later request to read it again. Only the track-fetch and track-writeback cases cross the Paula/Agnus DMA path examined in this paper.

For ordinary access it supports commands such as `CMD_READ` and `CMD_WRITE`, which refer to one or more sectors. Lower-level commands such as `TD_RAWREAD` and `TD_RAWWRITE` transfer recorded track data without the normal sector processing, so the caller assumes responsibility for the representation. Extended versions of several commands use the disk-change count to reject stale requests. These interfaces are documented in the Amiga ROM Kernel Reference Manual and the `trackdisk.device` documentation.

### 3.1.3 The CIA chips

The two 8520 CIA chips handle most signals associated with the drive mechanism. CIA-B port B provides drive select, motor control, side select, step direction, and the step pulse. CIA-A port A receives ready, track-zero, write-protect, and disk-change status. The index pulse is connected separately to CIA-B's `FLAG` input.

Software uses the CIAs to select the drive and side, place the head over the correct track, and confirm that the drive is in a suitable state.

### 3.1.4 Agnus and chip RAM

Agnus owns the chip-RAM bus and the disk DMA pointer. Software writes the destination or source address to `DSKPTH` and `DSKPTL`, which are Agnus registers. When Paula needs a word transferred, it raises a request on `DMAL`, the line it also uses to ask for audio DMA cycles. Agnus decides when the memory cycle can occur and supplies the address.

Disk DMA shares chip RAM with the processor, display, audio channels, sprites, blitter, and other DMA users. Paula reports when a disk word is ready or needed; Agnus arbitrates the shared bus and supplies the address.

The transfer buffer must be in chip RAM because the custom chips do not have direct access to Fast RAM. Software can copy or process the result elsewhere after the DMA operation if necessary.

### 3.1.5 Paula

Paula handles the operations that must follow the disk bit rate. On a read it accepts the drive's transition pulses, recovers timing, reconstructs bits, detects a selected sync pattern, forms words, and buffers them for DMA. On a write it accepts words from DMA, serializes them, generates the output timing, applies write precompensation, and controls the write-data and write-enable pins.

Paula also meters the programmed number of words and produces the disk sync and block completion interrupts.

### 3.1.6 The floppy drive

The drive positions the head, rotates the disk, detects magnetic transitions during a read, and records transitions during a write. Its analog electronics present a digital read-data signal to Paula and accept write-data timing from Paula.

The CIAs and Paula meet different parts of the drive's electrical interface: the CIAs handle mechanism control and status, while Paula exchanges recorded-data timing with the drive's analog electronics.

## 3.2 Three software viewpoints

### 3.2.1 Normal file and sector access

For ordinary use, an application asks the filesystem for file data. The filesystem translates that request into block or sector access. `trackdisk.device` serves the request from its buffered track when possible. A buffer miss causes a track read; a dirty track is physically written when the device commits it.

### 3.2.2 Raw-track access

Some software needs the representation below ordinary sectors. Disk utilities, format-specific tools, and software working with nonstandard recordings may issue raw-track requests through `TD_RAWREAD`, `TD_RAWWRITE`, or their applicable extended forms. These commands still use the operating-system device interface, but they expose data closer to the stream transferred by Paula.

A program that uses a device command prepares an I/O request, opens `trackdisk.device`, and submits it through the normal Exec device interface. Raw access preserves information that sector commands may discard, but the caller must understand the returned representation and the limits of the drive and OS version.

### 3.2.3 Direct hardware access

Software can program the custom chips directly, but it must not do so while the operating system is also using the floppy hardware. The Amiga Hardware Reference Manual identifies `trackdisk.device` and `disk.resource` as the system components responsible for this hardware. Software that needs exclusive low-level control must first obtain it through the appropriate system mechanism or take complete control of the machine.

Drive-control lines, disk DMA registers, interrupts, and buffers are shared state. Two independent users cannot safely move the head or change a DMA transfer at the same time.

## 3.3 Division of work in the Amiga

| Activity | Main owner | Why it belongs there |
|---|---|---|
| File and directory interpretation | Filesystem and application software | These are operating-system data structures, not properties of the recorded signal. |
| Sector interface, per-drive track buffering, format processing, validation, and retries | `trackdisk.device` | These operations turn Paula's format-independent transfer into ordinary disk-device requests. |

| Activity | Main owner | Why it belongs there |
|---|---|---|
| Drive selection, motor, side, direction, and stepping | CIA-B under software control | These are relatively slow mechanism-control signals and are separate from the data path. |
| Ready, track zero, write protect, and disk change | CIA-A | These status inputs describe the drive and inserted disk rather than the bitstream. |
| Index pulse | CIA-B `FLAG` | The signal reports disk rotation; it does not enter Paula's floppy logic. |
| DMA address and chip-RAM arbitration | Agnus | Agnus already owns the shared memory bus and coordinates all custom-chip DMA. |
| Read timing and serial-to-parallel conversion | Paula | These operations must follow incoming transition timing without processor delays. |
| Write serialization, timing, and precompensation | Paula | These operations must produce precisely timed output while DMA continues. |
| Analog read and write electronics | Floppy drive | The head signals require circuitry located close to and designed for the drive mechanism. |

The Amiga floppy controller is therefore a subsystem whose time-critical data component is Paula.

## 3.4 Paula's disk registers and controls

Software and Agnus configure Paula through a small set of registers.

| Register or control | Owner | Purpose |
|---|---|---|
| `DSKPTH` / `DSKPTL` | Agnus | Address of the chip-RAM track buffer used by disk DMA. |
| `DSKLEN` | Paula | Direction, DMA enable, and transfer length in sixteen-bit words. |
| `DSKDAT` | Paula | DMA data port for words being sent toward the write path. |
| `DSKDATR` | Paula | DMA data port for words arriving from the read path; the manual lists it as an early-read dummy address rather than a register software drives. |
| `DSKBYTR` | Paula | Most recently received byte and selected status flags. |
| `DSKSYNC` | Paula | Sixteen-bit pattern watched by the read-path comparator. |
| `ADKCON` | Shared Paula control | Disk rate, sync behavior, and write-precompensation choices. |
| `DMACON` | Shared DMA control | Enables disk DMA as part of the system-wide DMA controls. |

| Register or control | Owner | Purpose |
|---|---|---|
| `INTREQ` / `INTENA` | Paula interrupt control | Reports and enables sync-found and block-complete interrupts. |

## 3.5 Paula's internal functional map

At the level used in the next two chapters, Paula's floppy circuits form four groups:

1. **Read path:** input conditioning, transition-event detection, clock and data separation, deserialization, sync comparison, and byte/word framing.
2. **Write path:** serialization, fixed write timing, transition classification, precompensation, and the write-data/write-enable outputs.
3. **Shared data path:** a three-word buffer used in both directions, plus the data registers and bus connections.
4. **Control path:** transfer arming, direction, length counting, buffer-state tracking, DMA requests, and interrupts.

These four functional groups overlap on the die. Several modules participate in more than one group because Paula reuses registers, counters, and control signals between reads and writes.

## 3.6 From architecture to transfer paths

These boundaries define the two paths examined next. A read moves transition timing from the drive through Paula and Agnus into chip RAM; a write moves prepared words in the opposite direction. The operating system surrounds both transfers with drive control, track preparation or decoding, validation, and error handling.

# 4. Reading a track with Paula

On a read, the floppy drive sends Paula a sequence of digital pulses. The timing of those pulses describes the magnetic transitions recorded on the track, but the signal does not include a separate clock or framing. Paula must recover the timing, reconstruct the serial bits, form words, and keep the data moving into memory.

The discussion begins with a normal software read, then follows the signal through the circuits identified in the dissection. Labels such as `10c` and `10d` name regions of related circuitry; the complete read path crosses many of them.

## 4.1 The read operation from software's point of view

For an ordinary file read, the filesystem identifies the required blocks and `trackdisk.device` checks its per-drive track buffer. If the required track is already buffered, the device can return the sector data without starting disk DMA. The path shown here is the buffer-miss case, in which the requested track must be brought in from the drive.

If the current buffered track is dirty, the device commits it before reusing the buffer. It then selects the drive and side, positions the head, checks status through the CIAs, prepares a DMA buffer in chip RAM, and configures the disk transfer. Paula and Agnus capture the recorded track representation; `trackdisk.device` locates, decodes, and validates its sectors, updates the buffered-track state, and returns the requested data or retries the operation. A raw-track request uses the same drive and DMA hardware but returns the lower-level representation without normal sector processing.

A buffer miss brings the track through the complete system

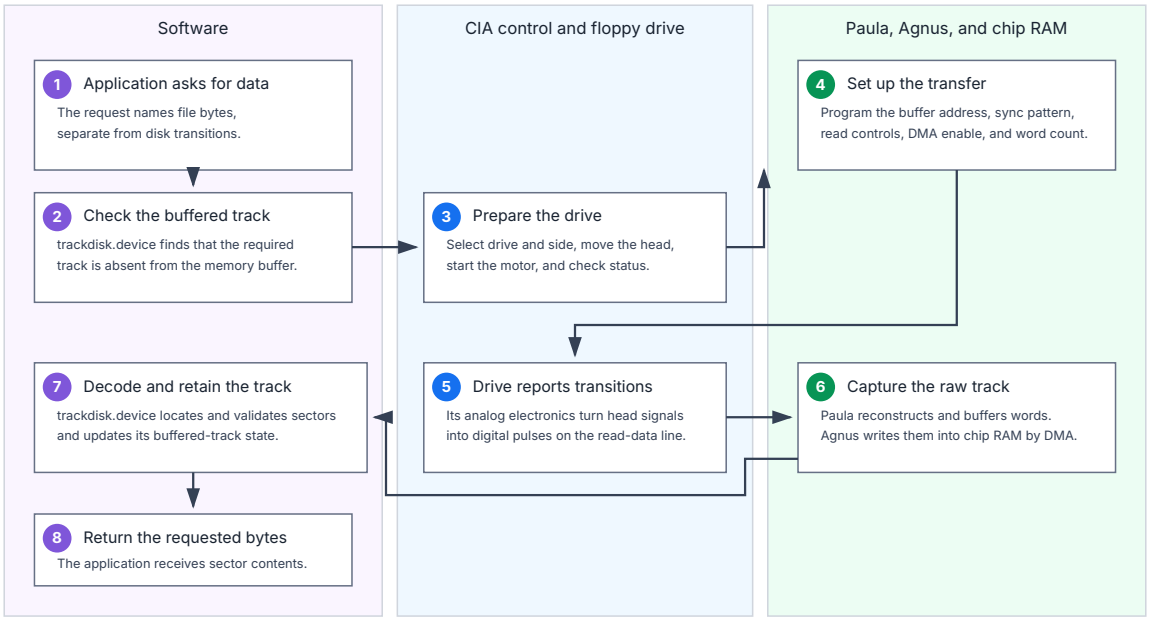


**Figure 4-1. A buffer miss is a complete-system operation.** A buffer hit remains in software. On a miss, the CIAs and drive prepare the mechanism, Paula performs the continuous bitstream work, Agnus transfers the words to chip RAM, and `trackdisk.device` validates and retains the track data before completing the request.

## 4.2 The complete read path

Paula's read path performs a fixed sequence of activities:

1. accept and synchronize the drive's read-data signal;
2. produce one internal event for each accepted transition;
3. recover bit-cell timing and reconstruct the serial data;
4. shift the serial data into a sixteen-bit value;
5. compare that value with the programmed sync word;
6. establish byte and word boundaries;
7. hold completed words until memory service is available; and
8. request DMA cycles so Agnus can write the words into chip RAM.

The stages solve different failure modes. A correctly detected transition can still be assigned to the wrong bit cell; correct bits can be grouped from the wrong starting position; and correct words can be lost while the memory bus is busy. The path must preserve timing, alignment, and flow control in that order.

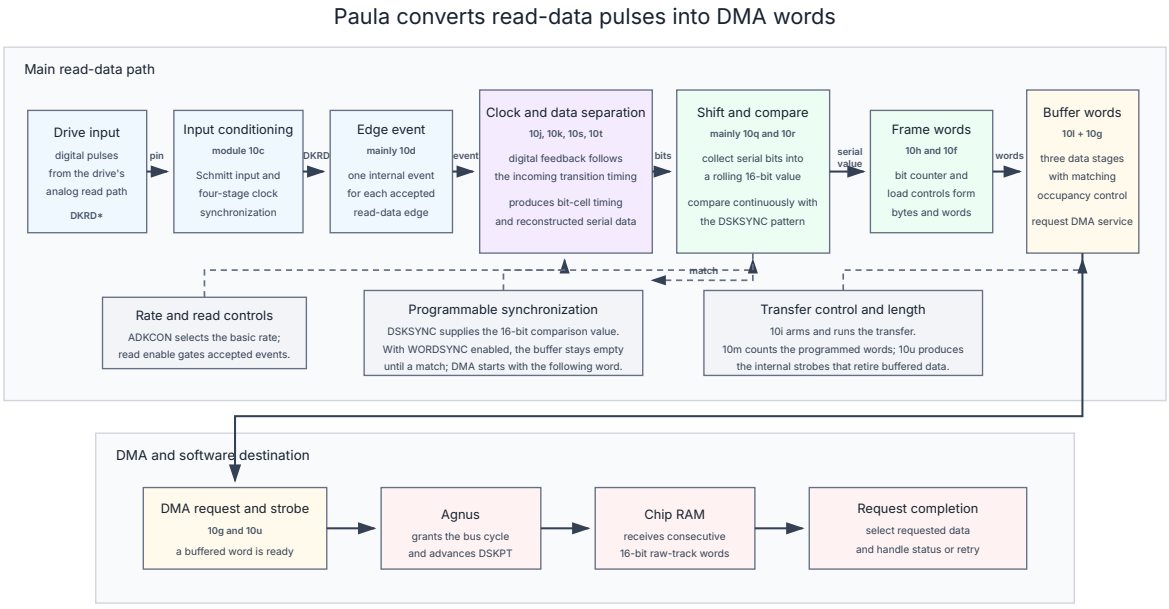



**Figure 4-2. Paula's high-level read path.** Solid arrows show the data moving from the drive toward chip RAM. Dashed arrows show settings and transfer controls. Each box represents a functional activity, which may span more than one physical circuit block.

## 4.3 Drive readback

The drive's read/write head produces a small analog signal as it passes a change in magnetic flux. Drive electronics amplify and condition that signal into a digital pulse for Paula. A missing, extra, or displaced pulse can change the reconstructed data, but the drive-specific analog path is outside the Paula dissection.

## 4.4 Input conditioning: module 10c

The read-data line is held high by the Amiga's pull-up. When the drive detects a flux reversal, its open-collector output pulls the line low. The falling edge at Paula's external `DKRD*` pin therefore marks the start of the read-data pulse.

The pin signal is asynchronous to Paula's internal clock. Module 10c first passes it through an inverting Schmitt-trigger stage, then through four clocked latch stages. The inversion turns the low-going external pulse into a high-going internal `DKRD` pulse, which is then used by the next part of the controller.

The Schmitt input provides defined switching behavior at the chip boundary, and the four clocked stages bring the asynchronous signal into Paula's clock domain. The dissection establishes both that structure and the inversion between external `DKRD*` and internal `DKRD`.

## 4.5 Edge detection: mainly module 10d

The synchronized input remains high or low for a period of time. The timing-recovery logic needs an event that identifies when a new transition was received, rather than a level that may remain asserted across several internal clock periods. Module 10d compares the current `DKRD` value with a delayed copy and produces `DKRD_CK`.

`DKRD_CK` produces one pulse for each rising edge of internal `DKRD` when read-event detection is enabled. Because module 10c inverts the pin signal, this internal rising edge is the conditioned and synchronized form of the falling edge on external `DKRD*`. Both edges refer to the same flux reversal. When the drive releases the line and the pull-up returns external `DKRD*` high, internal `DKRD` falls. That return edge does not create a second read event.

This one-pulse representation gives the separator one timing comparison per received transition. Processing the sustained input level would apply false corrections.

## 4.6 Clock and data separation: modules 10j, 10k, 10s, and 10t

The drive does not provide a separate bit clock. Paula must determine the bit-cell timing from the positions of the read-data transitions themselves. This operation is called clock and data separation.

Paula implements it as a digital feedback system. A phase counter and programmable rate state predict the current bit timing. Logic in module 10j examines where an incoming edge occurred relative to that prediction and selects phase or rate corrections. Modules 10k and 10t retain and apply those corrections, then feed the updated timing back into the next measurement. Module 10s supplies the signals that qualify those corrections and the enable that decides when read events are admitted at all. The `FAST` control selects the basic two- or four-microsecond bit-cell range documented for the Amiga disk hardware.

Because disk speed and individual transition intervals vary, a fixed sampling schedule would drift away from the recorded data. Paula instead adjusts its timing estimate so later logic can decide every expected cell, including cells with no transition.

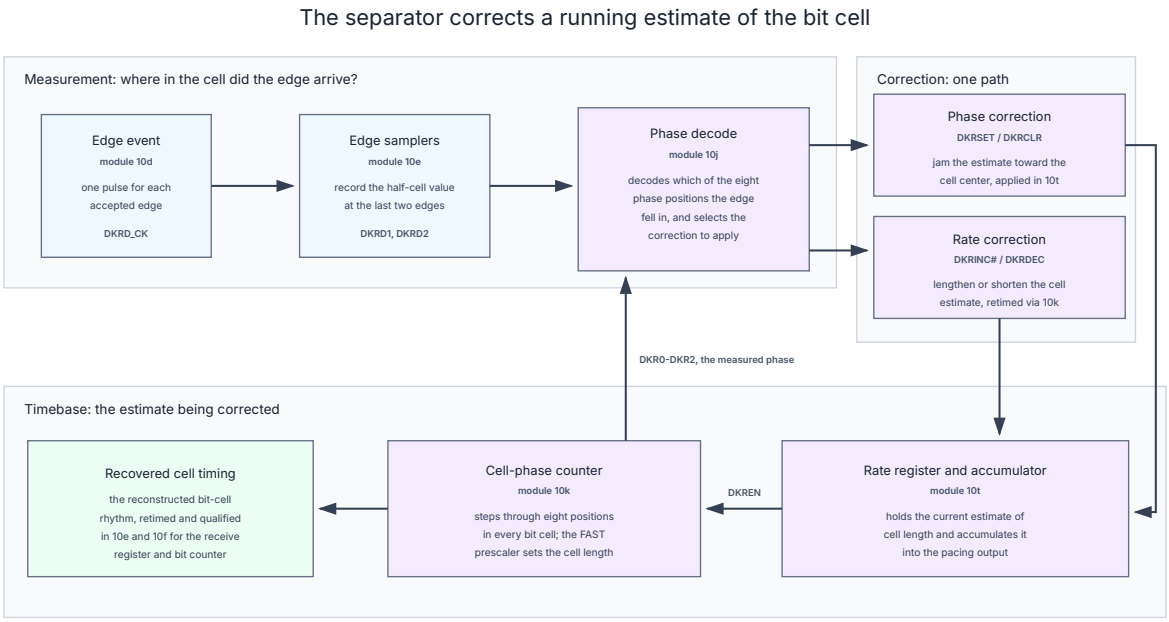



**Figure 4-3. The separator corrects a running estimate.** Each accepted edge is sampled and decoded against the current phase counter. The decode selects either a phase correction, which pulls the estimate toward the incoming transitions, or a rate correction, which changes the estimated length of a cell. Both reach the rate register and accumulator, whose pacing output drives the phase counter, which in turn supplies the recovered cell timing and the measurement for the next edge.

The dissection and Keller's patent establish the broad loop and the roles of its correction paths. The expression for one recovered-clock signal contains two inputs that are not identified in the available schematic, so its complete equation remains to be recovered.

## 4.7 Recovered-bit timing

The separator's timing result must be converted into events that advance the receive logic. Those events define when the current bit cell has been evaluated and when the next serial bit can enter the shift register. Supporting logic in modules 10e and 10f retimes and qualifies the recovered timing before it reaches the bit counter and data path.

Transition detection alone cannot produce the bitstream because an encoded stream can contain cells with no transition. The receive register therefore advances on recovered cell timing; incoming pulses alone cannot supply the missing cells.

## 4.8 Deserialization: mainly modules 10q and 10r

Disk data arrives one bit at a time, while the Amiga's disk DMA path transfers sixteen-bit words. The receive shift register collects each reconstructed serial bit and advances it through a sixteen-bit value.

Parts of this path and its associated comparison logic are shown in modules 10q and 10r, with related upper-bit and status circuitry in 10p.

## 4.9 Sync detection: modules 10q, 10r, and 10e

The receive register can begin at an arbitrary point on a rotating track. Software therefore programs a sixteen-bit value in `DSKSYNC`. A comparator checks the current sixteen-bit receive value against that pattern continuously. Its match output is also visible as the `WORDEQUAL` status in `DSKBYTR`.

> **Why `$4489` was chosen**
>
> The choice of `$4489` follows from the raw bit pattern of the conventional MFM `$A1` address mark. Encoding `$A1` normally produces `$44A9`. For the sync mark, the clock transition associated with the sixth data bit is deliberately omitted:
>
> `$A1` data → normal MFM `$44A9` → omit one clock transition → sync mark `$4489`
>
> The omitted transition violates the normal MFM clocking rule, so the resulting word cannot occur in a correctly encoded data field. It therefore provides a distinctive pattern from which Paula can establish the sixteen-bit word boundary.

With `WORDSYNC` enabled, Paula can remain armed with an empty transfer queue until the selected pattern establishes a bit boundary. Transfer begins with the word following the sync pattern. The comparator continues operating, so a later match can establish alignment again.

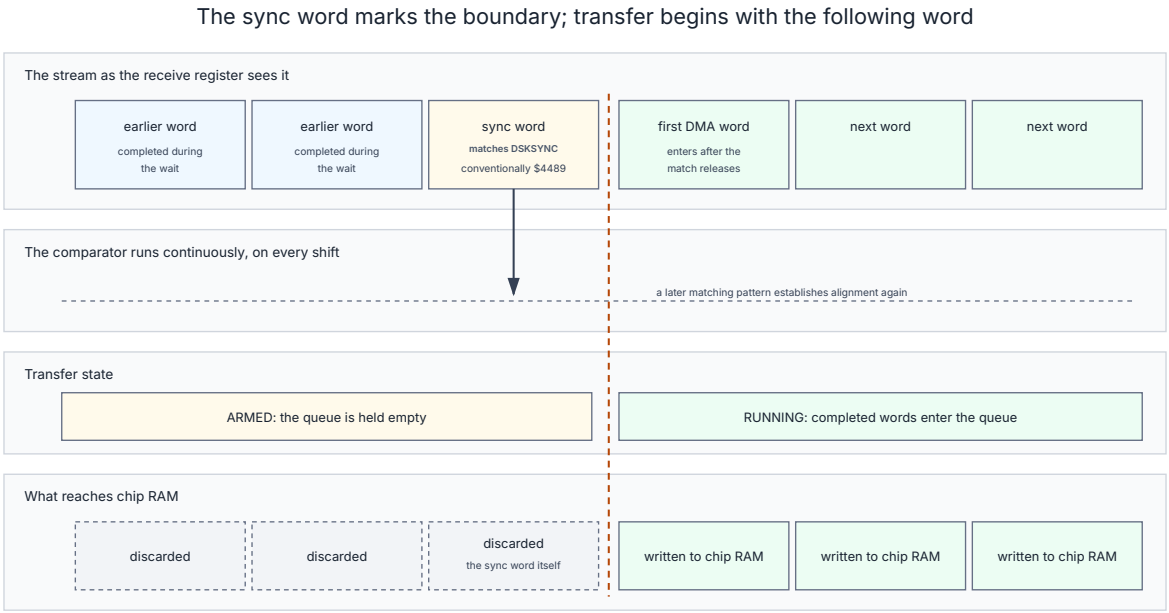



**Figure 4-4. The sync word sets the boundary; transfer begins after it.** While the controller is armed, the transfer queue is held empty. Every word completed during the wait is discarded. The sync word is still in flight at the moment of the match and is among those discarded words. The match releases the controller, and the first word to traverse the queue is the one that enters afterward.

## 4.10 Byte and word framing: modules 10h, 10f, 10n, 10o, and 10p

Once a useful boundary has been established, Paula must count recovered bit cells so that it knows when a byte and a sixteen-bit word are complete. Module 10h contains the four-stage bit counter, while module 10f decodes counter states into load and shift controls. The byte latches and `DSKBYTR` output circuitry appear in modules 10n, 10o, and 10p.

The counter maintains the byte and word cadence after synchronization; an offset of even one bit would produce different parallel values from the same serial sequence.

`DSKBYTR` gives software the most recently received byte and selected status bits. The dissection also confirms that its `DSKBYT` flag, which is generated in module 10q rather than with the byte latches, is set when a byte is received and cleared when the register is read. `DSKBYTR` therefore provides a byte-at-a-time read path for software, separate from the main sixteen-bit DMA flow, although both depend on the same received stream.

## 4.11 The three-word buffer: modules 10l and 10g

A completed word cannot always be written to chip RAM immediately. Agnus allocates disk DMA a fixed group of three memory cycles in each horizontal scan line, and the recorded stream does not pause between them. Paula therefore contains three data stages in module 10l and matching three-stage occupancy control in module 10g.

Module 10l holds the sixteen-bit values. Module 10g holds tokens that record which stages contain usable words and generates the corresponding DMA requests. As a word moves between data stages, its occupancy token moves through the control stages.

The three stages correspond to the three disk DMA cycles available in each scan line, a connection also made in the dissection notes. When the controller is armed and waiting for a `WORDSYNC` match, the control logic holds this queue empty; the first queued word is therefore the one following the match.

## 4.12 DMA into chip RAM

When a word is ready at the output of the buffer, module 10g asserts the appropriate disk DMA request. Module 10u produces the internal strobes that read and retire the buffered value. Agnus grants a chip-RAM bus cycle, uses the address held in `DSKPT`, writes the sixteen-bit word, and advances the pointer.

Paula determines when a word needs service; Agnus determines when and where the shared-memory transfer occurs. This separation keeps processor latency out of the continuous disk stream.

Paula has no overrun status. `DSKBYTR` carries only the byte-ready, DMA-enabled, direction, and sync-match flags, and no disk register indicates that a completed word was lost because the buffer had not been emptied in time. The Amiga prevents the condition instead of detecting it. The *Amiga Hardware Reference Manual* assigns disk DMA the highest priority and dedicated cycles in every scan line because a missed cycle would lose data.

Module 10m contains the transfer-length counter, which limits the DMA block to the programmed word count. Its terminal-count signal `DSKL_C` is not drawn in the available section-10 schematic; Section 6.3 examines the resulting final-word timing question.

## 4.13 Sync and completion interrupts

Paula can report when the programmed sync pattern has been seen and when the disk block transfer has completed. The comparator match is the source of the disk-sync indication, and the block interrupt reports the controller's end of the requested transfer. The interrupt bit positions and the block-complete path are established. The transistor copy omits the sync pulse-forming circuit, so that smaller mechanism remains open.

The sync interrupt remains useful when `WORDSYNC` is not delaying DMA. After the block interrupt, device code can decode the captured data or handle an error; as discussed in Chapter 6, completion does not guarantee that every requested final word reached memory on affected hardware.

# 5. Writing a track with Paula

Writing begins with a different division of work from reading. The read path receives an unknown transition stream and has to recover its timing. The write path starts with data that software has already prepared, so Paula can generate the output timing from its own clocks. Paula's job is to keep the prepared stream moving at the selected rate, apply write precompensation, and control the drive's write interface.

## 5.1 The write operation from software's point of view

An application normally writes file data through the filesystem. The filesystem determines which disk block must change and sends a request to `trackdisk.device`. Ordinary device users can issue a command such as `CMD_WRITE`; software that needs lower-level control can use an applicable raw-write command such as `TD_RAWWRITE`.

For a normal device write, `trackdisk.device` first changes the sector data in its per-drive track buffer and marks the track dirty. The sector request can therefore be completed without immediately recording the disk. The physical path described below begins when the device commits the dirty track, for example in response to `CMD_UPDATE` or because another track must replace it in the buffer.

At writeback, `trackdisk.device` constructs the encoded track representation in DMA-reachable chip RAM. The programmed word count can cover a whole track or a shorter transfer. Device code selects and positions the drive, checks write protect, and programs `DSKPT`, `DSKLEN`, and `ADKCON` for the source address, direction, length, cell rate, and precompensation. Starting DMA requires the documented double write to `DSKLEN`, whose circuit mechanism is examined in Section 6.1. A raw-write command reaches the same hardware path but supplies caller-prepared recorded data instead of using normal sector processing.

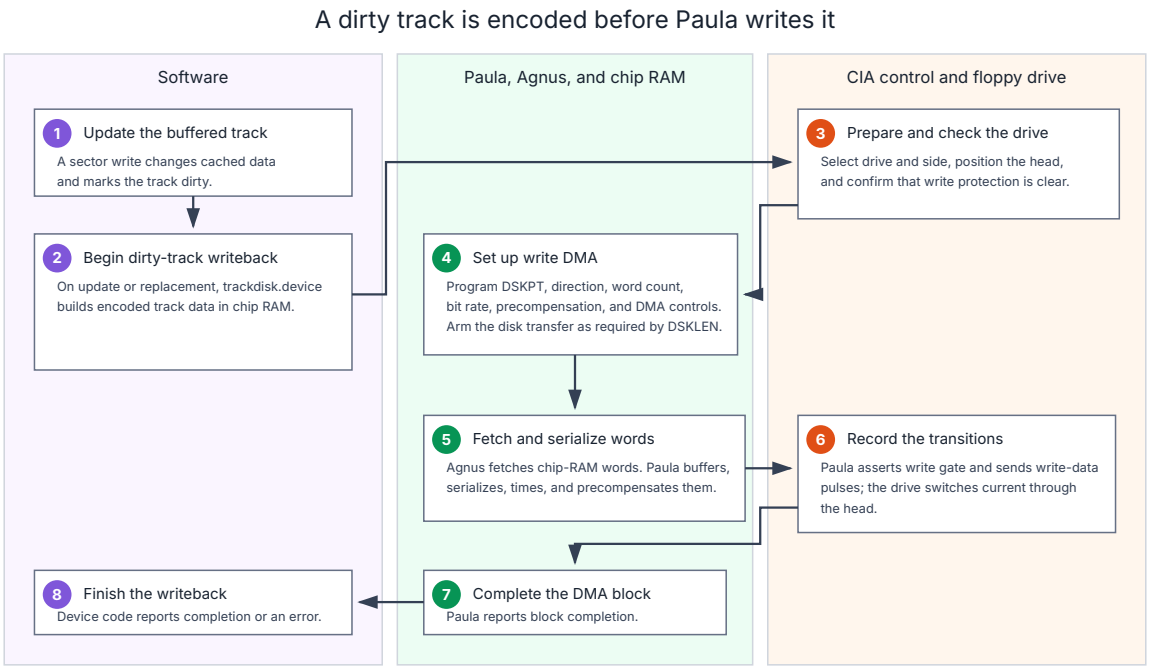



**Figure 5-1. A dirty track is encoded and committed as a complete track.** The preceding sector request changes the buffered representation. At writeback, Agnus fetches the encoded track from chip RAM, Paula converts it into timed drive signals, and the drive performs the physical recording.

## 5.2 The complete write path

The write path performs the following activities:

1. software prepares the track image in chip RAM;
2. Agnus fetches sixteen-bit words when Paula requests them;
3. Paula holds several words in its shared transfer buffer;
4. a shift register serializes one word at a time;
5. write timing advances the serializer at the selected bit-cell rate;
6. precompensation moves selected transitions slightly early or late;
7. output circuitry sends write-data timing and write gate to the drive; and
8. the drive switches current through the head to record magnetic transitions.

The stages isolate three constraints: variable DMA service cannot interrupt the fixed serial cadence, and the fixed cadence must reach the drive only while write gate authorizes head current.

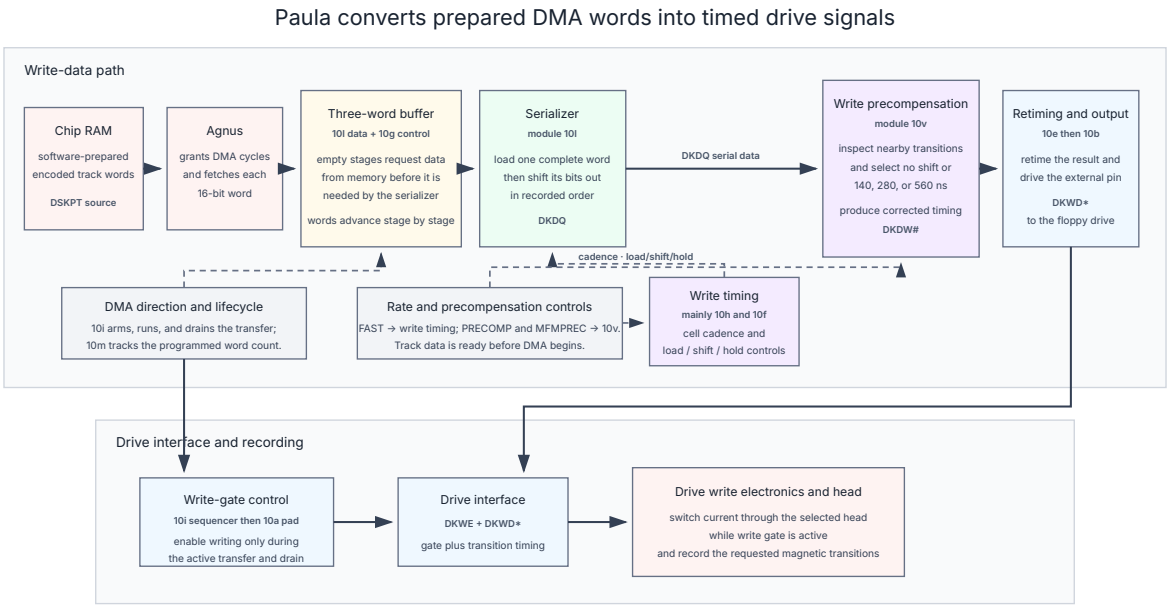



**Figure 5-2. Paula's high-level write path.** The main data path runs from chip RAM to the drive's write-data input. The write-gate path is separate because it authorizes recording but does not carry the transition pattern.

## 5.3 Prepared track data

Before DMA begins, software stores the encoded words to be written in chip RAM. During the transfer, Paula preserves their order while applying the selected cell timing and precompensation. The path can write standard or nonstandard sequences within the electrical limits of the drive, media, and Paula timing.

## 5.4 DMA from chip RAM

When the write engine is active, the buffer control in module 10g requests DMA service for empty stages. Read mode requests service for full stages. Agnus grants a chip-RAM cycle, reads the next sixteen-bit value from the address in `DSKPT`, and advances the pointer.

Paula reports when it needs data, while Agnus schedules the fetch on the shared memory bus. Processor latency never enters the fixed-rate output path.

## 5.5 The shared three-word buffer: modules 10l and 10g

The write path uses the same three-stage structure as the read path. Module 10l holds the sixteen-bit data, and module 10g tracks which stages are occupied. In write mode, the request logic asks Agnus to fill empty positions so additional words are ready before the serializer reaches them.

The buffer decouples the three disk DMA slots in each scan line from the fixed-rate serial output. Several ready words carry the serializer across gaps in chip-RAM service.

The dissection shows a cascading pipeline rather than an addressed three-entry memory. Words and their corresponding occupancy tokens move from one stage to the next when the following position becomes available.

## 5.6 Serialization: module 10l

Module 10l also contains the write shift register attached to the shared data path. When a buffered word reaches the output position, control logic loads it into the shift register. The register then shifts one bit at a time and produces the serial signal `DKDQ`.

The shift register preserves the prepared bit order while converting sixteen-bit bus transfers into a continuous serial stream.

## 5.7 Write timing: mainly modules 10h and 10f

Writing uses Paula's configured timing because no incoming disk clock needs to be followed. The `FAST` control selects the documented two-microsecond cell normally used for MFM or the four-microsecond cell normally used for GCR.

The circuit reuses the separator's cell-boundary cadence for writing. A retimed form of that cadence reaches module 10f, which generates mutually exclusive load, shift, and hold controls for the write register. Module 10h counts serial progress and selects the write-side word boundary. This reuse matters: Paula has one bit-cell timebase serving both directions. Incoming disk edges steer it during read; its rollover cadence paces the locally generated stream during write. The final expression that exports the cell boundary contains two unlabeled inputs in the available schematic, so Part I leaves its exact equation open.

Transition spacing becomes part of the recording. If the serializer advances too early, too late, or twice during one cell, a later read can reconstruct different data.

The analysis establishes the counter, the write-side terminal-count selection, and the load/shift/hold control relationship.

## 5.8 Write precompensation: module 10v

The serial output from 10l enters module 10v. The circuit contains outgoing-bit history taps, two mirrored pattern groups, and a selected delay path for the outgoing change. Depending on the `PRECOMP1:0` setting, the available correction is none, 140 nanoseconds, 280 nanoseconds, or 560 nanoseconds. The structure and magnitudes are established. The exact MFM and GCR pattern rules remain open.

Closely spaced magnetic transitions can be detected at displaced times during readback, an effect commonly called **pulse crowding** or **peak shift**. Each transition produces a response with finite width; neighboring responses overlap, so the detected time moves in a pattern-dependent direction. A later floppy-drive write-precompensation patent describes the same interference between adjacent reproduced waveforms. Moving selected transitions during recording brings their expected readback times closer to the intended positions without changing the intended sector data.

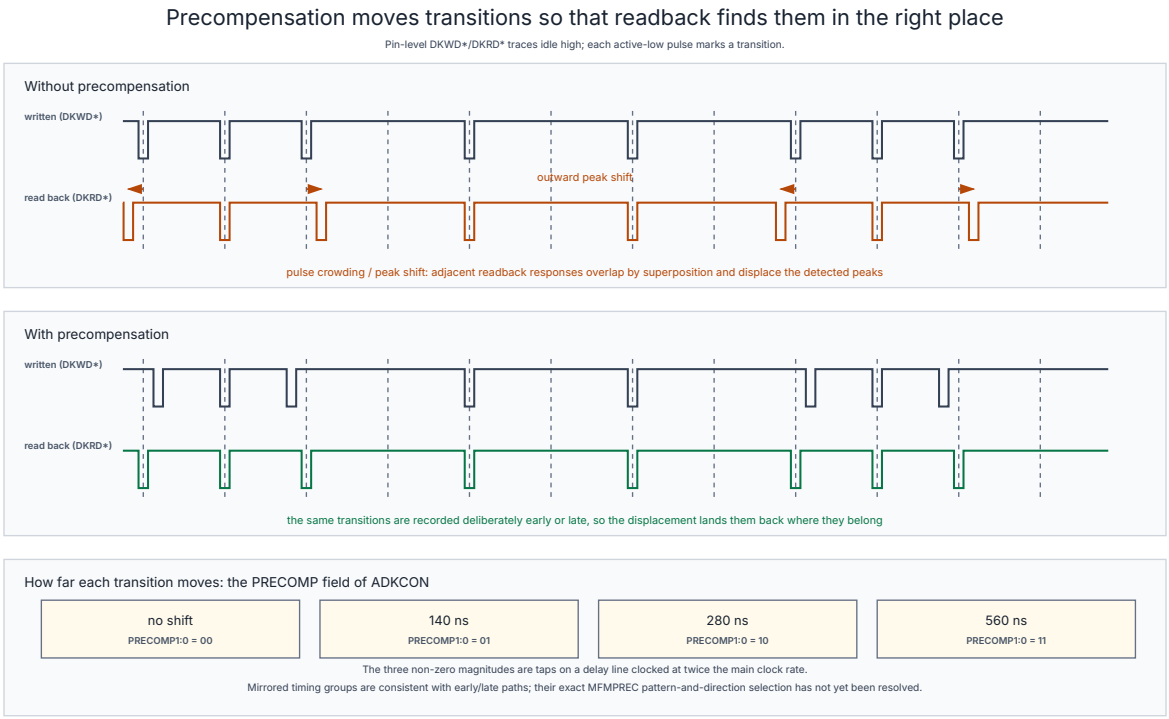



**Figure 5-3. The correction is applied before the disk introduces the error.** The waveforms use the active-low external `DKWD*` and `DKRD*` polarity. They show the early/late correction concept: closely spaced transitions push each other apart, so an offset during recording can move readback toward the intended positions. `PRECOMP1:0` establishes the available correction magnitude. The mirrored circuitry is consistent with two timing directions. Its exact `MFMPREC` pattern-and-direction selection remains unresolved.

The dissection establishes the magnitude-selection structure. Module 10v contains a clocked delay chain and three nonzero selections in the same 1:2:4 relationship as the documented values. The selector contains two mirrored paths for each magnitude, consistent with adjusting transitions in either timing direction, plus an unshifted path when precompensation is disabled. The mirrored direction assignment remains an inference from structure and function.

`MFMPREC` selects between the documented MFM and GCR precompensation modes. The relevant transistors have been located and the correction magnitudes are established; the exact pattern rules that choose a correction remain to be derived.

## 5.9 Write-data output and write gate: modules 10v, 10e, 10b, 10i, and 10a

Module 10v produces the corrected timing signal `DKDW#`. Module 10e retimes it into the internal `DKWD` signal, and the strong pad driver in 10b sends the result to the external active-low `DKWD*` pin.

Write gate follows a separate path. The sequencer in 10i considers the selected write direction, active DMA state, buffer state, bit-counter progress, and reset. Its output passes through module 10a to the external `DKWE` line. The dissection and the hardware manual's drive timing diagram show this interface low during active writing.

The separate gate prevents write-data activity from altering the disk outside an active write and remains asserted while the final active word is shifted.

## 5.10 Drive write electronics

The drive accepts the write-data transition timing and the write-gate signal. While write gate is active, its electronics switch current through the selected head in response to the incoming timing. The rotating disk records the resulting magnetic changes.

Drive electronics supply the head-coil current, polarity switching, head selection, and analog-path protection. Paula supplies the transition timing and authorized write interval.

## 5.11 Transfer completion

Module 10m holds the programmed word count, and the transfer-control state machine in 10i moves through idle, armed, running, and draining states. The draining state lets words already accepted by Paula finish the buffer and serializer lifecycle after the programmed count is reached. It does not by itself prove that write gate remains open until every downstream 10v/10e delay has cleared.

The main lifecycle and the `DSKBLK` completion result are established. The section-10 schematic omits the terminal-count signal formation for `DSKL_C`. Section 6.3 examines its role in the final-word behavior.

The Amiga Hardware Reference Manual also documents a hardware bug in which the last three bits sent to disk are lost. Software must allow for that behavior when preparing the end of a write. Section 6.2 traces the likely mechanism through the delay pipeline in 10v.

# 6. Three Paula Mysteries: The Double Write, Lost Bits, and the Missing Word

The *Amiga Hardware Reference Manual* requires software to write `DSKLEN` twice and warns about two end-of-transfer defects. It gives the required workarounds; the circuit reasons are absent. The dissection reveals the `DSKLEN` protection mechanism, suggests a specific cause for the lost write bits, and identifies terminal-count and buffer-drain timing as the key to the possibly absent final read word.

These three behaviors test the functional reconstruction at different levels of closure. The model explains and checks the double-write mechanism. For the two end effects it narrows the possible cause to a specific circuit boundary and states the decisive missing evidence. Both end effects remain open.

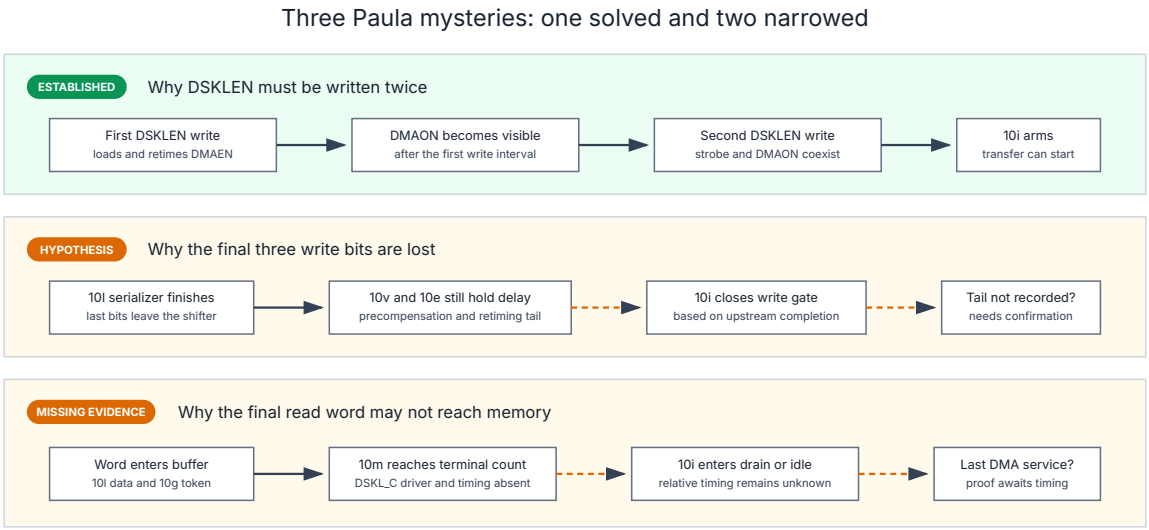



**Figure 6-1. Current status of the three behaviors.** The double-write mechanism is explained and checked against the documented programming sequence. The two end effects remain open, but each has been localized to a specific part of the transfer-ending logic and given a decisive next test.

## 6.1 Why `DSKLEN` must be written twice

The manual presents the double write as protection against an accidental disk write:

> "The DMAEN bit in the DSKLEN register must be turned on twice in order to actually enable the disk DMA hardware."

Source: *Amiga Hardware Reference Manual*, third edition, printed page 247, "Disk DMA Channel Control."

The required programming sequence first forces disk DMA off with `$4000`, writes the desired `DSKLEN` value, writes the same value again to start DMA, and restores `$4000` after completion.

### What the dissection shows

The upper `DSKLEN` bits are captured on dissection sheet 54, module 10p. The `DMAEN` value passes through a `PHI1` retiming stage before it reaches the transfer sequencer. Module 10i on sheet 47 contains the transfer-control state machine. Its arming condition requires two things at the same time:

- a write strobe for `DSKLEN`; and
- `DMAON` already asserted.

The write that first enables `DMAEN` cannot meet both conditions because its new value has not yet passed through the retiming stage. When software writes the value again, `DMAON` is already visible and the new `DSKLEN` strobe can move the sequencer from idle to armed.

The same analysis identifies four sequencer states: idle, armed, running, and draining. The extracted state equations show that one write never leaves idle and that the second write permits the transfer to start.

**Current conclusion:** the double-write mechanism is established. One retimed enable register and the 10i arming condition form the interlock.

**Dissection sources:** Frank Wolf and ttlworks, 8364R7 dissection, sheet 47, module 10i, and sheet 54, module 10p (`DSKLEN` upper-bit circuitry).

## 6.2 The final three write bits are lost

The manual gives a direct warning:

> "the last three bits of data sent to the disk [are] lost."

Source: *Amiga Hardware Reference Manual*, third edition, printed page 247, "Disk DMA Channel Control."

Software writing a track must include enough trailing data that losing the final three bits does not remove required track content.

### What the dissection shows

The last bit shifted out of module 10l on sheet 50 does not reach the drive immediately. The serial output `DKDQ` first passes through the write-precompensation delay circuitry in module 10v on sheet 60. The result is then retimed in module 10e on sheet 43 before the pad driver sends it to the external `DKWD*` pin.

The write-gate condition comes from module 10i. It is based on the active write state, DMA enable, buffer output, and the write-side bit-counter boundary. These conditions describe completion at or near the serializer, not completion of every downstream precompensation and retiming stage.

### Current hypothesis

The likely explanation is an incomplete pipeline drain. When the final word has left the serializer, 10i can close write gate while several delayed transition decisions remain in 10v and 10e. Once write gate becomes inactive, those remaining transitions cannot be recorded by the drive.

The candidate sequence is specific: the serializer finishes, 10i closes write gate, and delayed transition decisions still in 10v and 10e reach a drive that can no longer record them. This explains a lost tail. The current work has not established why the documented length is exactly three bits. The existing 10v and 10e drawings contain enough information for the next test: count and replay every post-serializer stage against the write-gate shutdown point.

**Dissection sources:** Frank Wolf and ttlworks, 8364R7 dissection: sheet 50, module 10l; sheet 60, module 10v; sheet 43, module 10e; and sheet 47, module 10i.

## 6.3 The final read word may not arrive

The second hardware warning immediately follows the first:

> "the last word in a disk-read DMA operation may not come in."

Source: *Amiga Hardware Reference Manual*, third edition, printed page 247, "Disk DMA Channel Control."

The manual's *may* marks a conditional failure: software can receive one fewer word than the requested length, but it does not happen on every transfer.

### What the dissection shows

A completed read word enters the three-stage data cascade in module 10l on sheet 50. The matching token queue in module 10g on sheet 45 records its position and asks Agnus for DMA service. The word reaches chip RAM only after its token reaches the output stage and Agnus grants the corresponding memory cycle.

The programmed word count is held in the length counter on sheet 51, module 10m. Module 10i uses its terminal-count signal, `DSKL_C`, when leaving the running state and draining the buffer.

`DSKL_C` appears in the schematic only as a labeled interface signal. The gates forming it and the end of the length counter's borrow chain are not drawn, so its timing relative to the last buffer token cannot be recovered from the schematic pages.

### Current hypothesis

The candidate failure sequence is that `DSKL_C` moves 10i out of the running state before the last token reaches the buffer output and receives a DMA slot. The word then remains inside Paula instead of reaching chip RAM. Whether that sequence occurs depends on the missing `DSKL_C` timing.

The current schematic ends at `DSKL_C` as a labeled interface signal. The next step is to trace its generating circuit from the existing 10m die material if the available layers support an unambiguous reconstruction, then compare its timing with the last buffer token. A source limitation at that point would require a clearer die trace or a hardware measurement. Either route can distinguish a race from a fixed pipeline condition.

**Dissection sources:** Frank Wolf and ttlworks, 8364R7 dissection: sheet 45, module 10g; sheet 50, module 10l; sheet 51, module 10m; and sheet 47, module 10i.

## 6.4 What would resolve the open cases

The double-write mechanism is established. The other two cases require one decisive timing result each:

| Behavior | Decisive remaining test |
|---|---|
| Final three write bits are lost | Count and replay the complete post-serializer delay in the existing 10v/10e source against write-gate shutdown. |
| Final read word may not arrive | Recover `DSKL_C` from the 10m die material if its layers suffice; otherwise measure the last buffer and DMA events. |

# 7. What the dissection establishes

The analysis covers every module in section 10 and checks the connected results against the *Amiga Hardware Reference Manual*, Keller's patent, the register decoder, and the DMA and interrupt circuits. Those cross-checks support the functional architecture below and expose the circuit details that remain unresolved.

## 7.1 Functional result and current status

Paula is a DMA-coupled serial controller. On a read it accepts digital transition pulses from the drive, recovers timing, reconstructs serial data, detects a programmable sync word, forms bytes and words, buffers them, and requests transfers to chip RAM. On a write it accepts software-prepared words from chip RAM, buffers and serializes them, generates the output timing, applies write precompensation, and drives the write-data and write-gate signals.

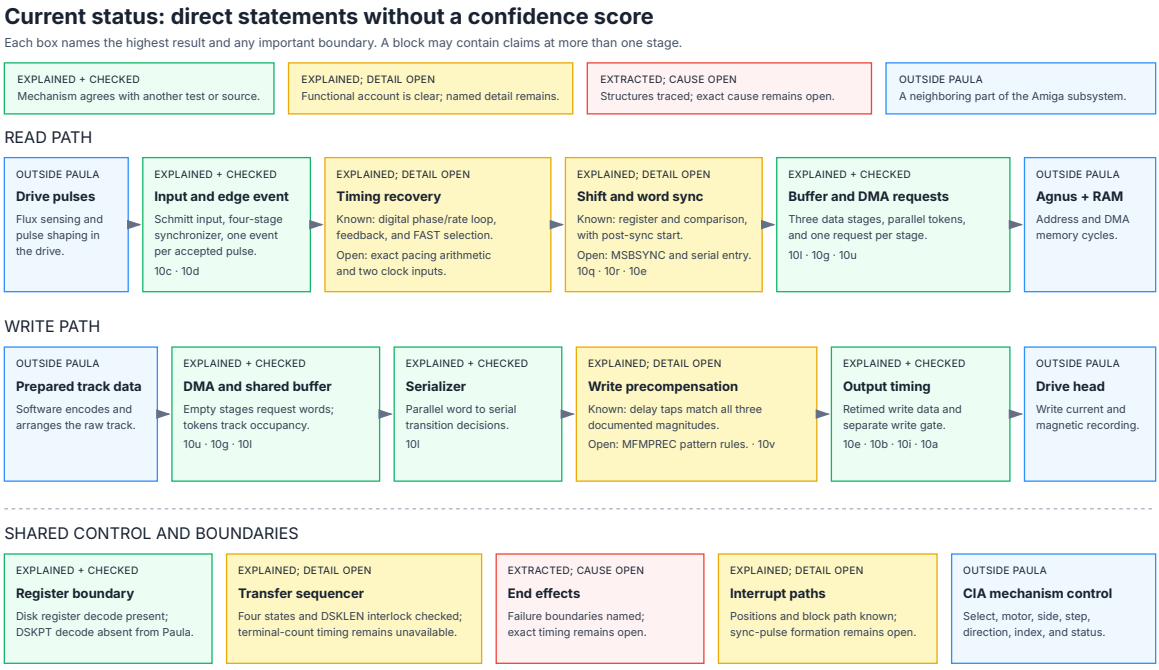



**Figure 7-1. Status map for the current circuit analysis.** The boxes use the progression defined in Figure 1-3 and spell out both the result and its boundary. *Explained and checked* means that a causal account has an execution or independent cross-check. *Explained; detail open* means that the functional mechanism is established while a named detail remains unresolved. *Extracted; cause open* means that the relevant structures have been traced while the exact causal timing still awaits proof. The phrases carry no numeric score. Blue boxes belong to the drive, CIAs, Agnus, or software outside Paula.

Status is assigned to individual claims. Figure 7-1 summarizes the dominant result in each block and names important exceptions. For example, the `DSKLEN` double-write interlock is explained and checked even though other parts of the transfer-ending logic remain open.

## 7.2 What the circuit evidence adds

### A physical system boundary

The manuals assign drive control to the CIAs and memory addressing to Agnus. The die implements the same division. Paula's drive interface consists of read data, write data, and write gate. Its register decoder contains the disk data and control registers but omits `DSKPTH` and `DSKPTL`. Paula holds disk words and raises requests; Agnus supplies their chip-RAM addresses and performs the bus cycles.

### A connected read mechanism

Modules 10c and 10d convert each accepted external read pulse into one synchronized event. The digital feedback loop recovers bit-cell timing, and the receive register maintains the rolling sixteen-bit value compared with `DSKSYNC`. At the sync boundary, the `WORDSYNC` setting causes the armed state to hold the queue empty until a match. The following word becomes the first queued word, and later matches can establish alignment again.

### One data cascade and one token cascade

Paula contains three sixteen-bit data stages in 10l and a matching token queue in 10g. The queue generates one request per stage, corresponding to Agnus's three disk DMA cycles per scan line. Direction changes the meaning of occupancy: full stages request service on a read, while empty stages request data on a write. Keeping data and tokens separate lets the same cascade derive requests from state and direction without interpreting the word values.

### A shared timebase and a measured precompensation path

The shared buffer loads the 10l serializer, whose `DKDQ` output passes through 10v precompensation, 10e retiming, and the 10b pad driver. A separate 10i/10a path produces write gate. The write controls reuse the separator's cell-boundary cadence. In 10v, the three nonzero `PRECOMP1:0` choices select paths through a doubled-clock delay chain in the same 1:2:4 relationship as the manual's 140, 280, and 560 ns values. The magnitude path is established while the `MFMPREC` pattern rules remain open.

### One transfer lifecycle

The four-state 10i sequencer connects direction, DMA gating, sync wait, buffer state, write gate, and block completion. Its arming condition also explains the double write to `DSKLEN`: the newly written enable cannot arm the sequencer in the same interval, but it is visible when the second write strobe arrives. The same transfer lifecycle serves read and write.

## 7.3 The Part II boundary and open evidence

Part I's functional account is complete at its chosen level. The table distinguishes three kinds of work beyond that boundary. *Deferred to Part II* identifies evidence ready for the circuit-level treatment intentionally kept out of Part I. *Open* marks an answer the investigation has yet to establish. *Not extractable* identifies a source that omits the information required to decide; a different source or a measurement would be needed.

| Area | What Part I establishes | Status beyond Part I | Reason |
|---|---|---|---|
| Data-separator arithmetic | Phase measurement, phase and rate correction paths, feedback, and basic rate selection are connected. | **Not extractable from the drawn schematic** | Two thirds of the rate register are intentionally abbreviated, so the exact pacing arithmetic cannot be recovered from sheet 58 alone. |
| Recovered-clock formation | Its source region and downstream consumers are known. | **Not extractable from the drawn schematic** | Two significant inputs in the available expression are unlabeled. |
| `MSBSYNC` | The `ADKCON` storage and readback mechanism is established. | **Open** | Its exact effect belongs in the 10q/10r receive and comparison path and has not been reduced. |
| MFM/GCR precompensation choice | The `MFMPREC` devices are located, and the correction magnitudes are established. | **Open** | The bit-pattern logic that chooses early, late, or unchanged timing is not finished. |
| Bit-counter dynamics | The four-stage structure, read/write taps, and control connections are known. | **Deferred to Part II; open** | Its charge-dependent sequence requires a switch-level dynamic analysis using the existing source. That circuit and timing treatment belongs in Part II. |
| Transfer terminal count | The 14-bit counter's load, decrement, and hold roles are clear. | **Not extractable from the drawn schematic** | The gates forming `DSKL_C` and the borrow-chain terminus are absent, so exact final-word timing cannot be decided from those pages. |
| Interrupt internals | The `DSKBLK` and `DSKSYN` positions, purposes, and block-complete path are established. | **Open** | The sync-interrupt pulse formation is present only in conceptual redraws, not in the transistor copy. |
| Documented end effects | The two failure sites are bounded to the write-output drain and the read terminal-count/buffer boundary. | **Open** | The exact three-bit loss and conditional missing-word mechanisms have not been demonstrated. |

## 7.4 What Part II takes forward

Part II has three distinct jobs beyond Part I's functional mission:

1. present the transistor circuits, PLA equations, state reductions, and timing proofs behind mechanisms that Part I already explains functionally;
2. test open leads such as the `MFMPREC` pattern rules, `MSBSYNC`, and the two end effects without promoting them to conclusions prematurely; and
3. mark true source limits, such as the missing `DSKL_C` formation and abbreviated separator slices, where die-layout recovery or hardware measurement would be required.

Appendix A maps the modules, Appendix B lists the registers, pins, and recurring signals, and Appendix C defines the recurring terms. Appendix D connects the principal conclusions to their evidence and present boundary.

## 7.5 Conclusion

The three opening questions now have connected answers. On a read, Paula conditions drive pulses, recovers bit-cell timing, reconstructs and frames the serial stream, then moves complete words through a three-stage data/token cascade that requests Agnus DMA. On a write, the same cascade accepts memory words, feeds a serializer, selects precompensated transition timing, and drives write data and write gate. Across both directions, the drive handles the magnetic medium, the CIAs handle mechanism control, Agnus handles memory cycles, and software supplies the disk format.

The dissection now connects the documented behavior of Paula's disk registers to one physical mechanism across all of its floppy modules. It explains the word-after-sync behavior and `DSKLEN` double write, identifies the common read/write machinery, and turns the two end effects into bounded engineering questions with named evidence requirements.

The functional mission is complete. Named transistor questions remain open. The complete Amiga floppy controller spans the drive, CIAs, Paula, Agnus, memory, and software, with Paula at its timing-critical center. Part II can stand on that foundation and present the detailed circuits and proofs as its own contribution.

# Appendices

These tables collect the names used in the preceding chapters. They support the main text and give readers direct routes back into the dissection.

## Appendix A. Paula module map

Section 10 of the dissection is drawn one module per schematic sheet, except sheet 41, which holds three. The listed functions come from the analysis rather than from the module names. The last column gives the section of this document in which each module is discussed.

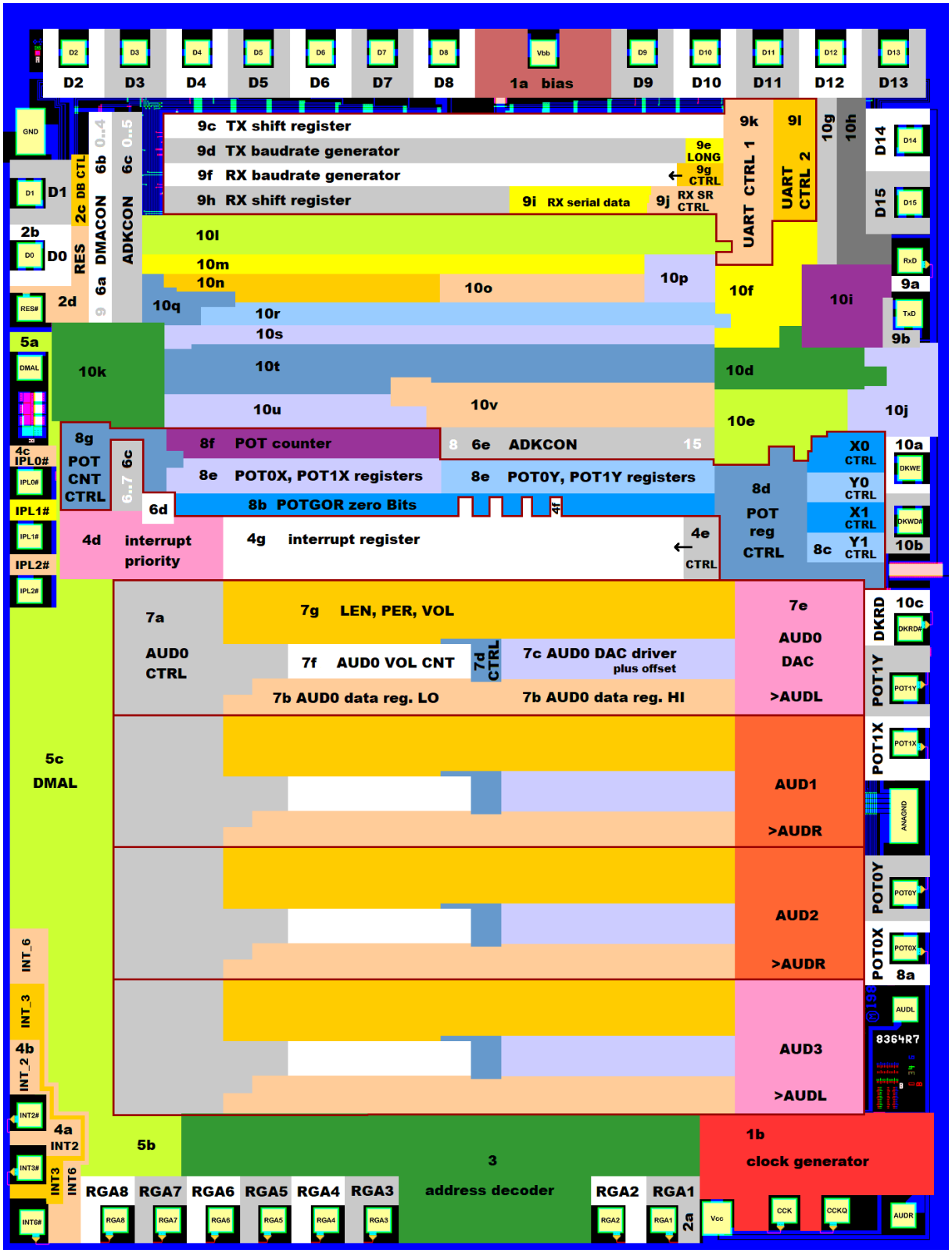



**Figure A-1. Annotated floorplan of the 8364R7 Paula die.** Regions 10a through 10v identify the physical locations of the floppy-controller modules within Paula. Reproduced unchanged from the 8364R7 dissection materials published by Frank Wolf and ttlworks, used with permission.

| Module | Sheet | Function | Discussed in |
|---|---|---|---|
| 10a | 41 | Write-gate pad driver; a three-stage inverting super buffer | §5.9 |
| 10b | 41 | Write-data pad driver; the same cell as 10a | §5.9 |
| 10c | 41 | Read-data input: protection, an inverting Schmitt trigger, and a four-stage synchronizer | §4.4 |
| 10d | 42 | Edge detection: one internal event for each accepted read-data edge | §4.5 |
| 10e | 43 | Edge samplers, the read-event flag, the sync-wait level, write-data retiming, and synchronized reset | §4.7, §4.9, §5.9 |
| 10f | 44 | Decodes counter states into the load, shift, and hold controls; rebuffered clocks | §4.10, §5.7 |
| 10g | 45 | Three-stage occupancy queue; per-stage transfer strobes and the three DMA requests | §4.11, §4.12, §5.4, §5.5 |
| 10h | 46 | Four-stage bit counter with separate read-side and write-side carry taps | §4.10, §5.7 |
| 10i | 47 | Transfer sequencer: the idle, armed, running, and draining states, and the write gate | §5.11, §6.1 |
| 10j | 48 | Data-separator logic array: phase decode and selection of phase or rate corrections | §4.6 |
| 10k | 49 | Cell-phase counter, the `FAST` divide-by-two prescaler, and correction retiming | §4.6 |
| 10l | 50 | Sixteen-bit data register: the three-deep buffer cascade plus the attached shift register | §4.11, §5.5, §5.6 |
| 10m | 51 | Fourteen-bit transfer-length down counter | §4.12, §5.11 |
| 10n | 52 | One `DSKBYTR` bus-driver slice | §4.10 |
| 10o | 53 | Eight-bit byte latch with its `DSKBYTR` bus drivers | §4.10 |
| 10p | 54 | `DSKBYTR` status bits; the `DSKLEN` direction and enable latches | §4.10, §6.1 |
| 10q | 55 | Receive shift register and sync-comparator slices; the `DSKBYT` flag and its clear on read | §4.8, §4.9, §4.10 |
| 10r | 56 | Receive shift register and sync comparator, continued; the buffered match output | §4.8, §4.9 |
| 10s | 57 | `DKRCU` / `DKRCV` frequency-path qualifiers or decodes (exact limit role provisional), plus the read-event enable | §4.6 |
| 10t | 58 | Rate register, addend multiplexer, and accumulator; produces the pacing output | §4.6 |
| 10u | 59 | Transfer strobes: byte latch, buffer read, token retire, and the DMA request interface | §4.12 |
| 10v | 60 | Write precompensation: the clocked delay line and its tap selection | §5.8 |

Several modules serve both directions, which is why the read and write chapters name some of the same regions. The labels identify locations in the dissection; they do not each correspond to one independent function.

Logic outside section 10 that the floppy controller depends on:

| Sheets | Function |
|---|---|
| 3 | Clock generator |
| 6 | Register address decoder. All seven Paula disk registers decode at their documented addresses, and there is no decode for `DSKPTH` or `DSKPTL` |
| 11 | DMA request logic and the `DMAL` pin |
| 12, 13 | `DMACON` and `ADKCON` bit cells, including the set/clear mechanism |
| 7-10 | Interrupts. The block-complete and sync bits sit at `INTREQ` bits 1 and 12 |

## Appendix B. Registers, pins, and signals

### B.1 Disk registers

The *Amiga Hardware Reference Manual* gives the addresses, access rules, and functions listed below. `DSKDAT` and `DSKDATR` are the ports used by the DMA transfer itself rather than registers software normally drives.

| Register | Address | Access | Owner | Purpose |
|---|---|---|---|---|
| `DSKPTH` / `DSKPTL` | `$DFF020` / `$DFF022` | write | Agnus | Chip-RAM address for the transfer, written as one long word |
| `DSKLEN` | `$DFF024` | write | Paula | Bit 15 `DMAEN`, bit 14 `WRITE`, bits 13-0 length in words |
| `DSKDAT` | `$DFF026` | write | Paula | DMA data port toward the write path |
| `DSKDATR` | `$DFF008` | early read | Paula | DMA data port from the read path; an early-read dummy address |
| `DSKBYTR` | `$DFF01A` | read | Paula | Bit 15 `DSKBYT`, 14 `DMAON`, 13 `DISKWRITE`, 12 `WORDEQUAL`, 7-0 the received byte |
| `DSKSYNC` | `$DFF07E` | write | Paula | The sixteen-bit pattern the read comparator watches for |
| `ADKCON` / `ADKCONR` | `$DFF09E` / `$DFF010` | write / read | Paula | Bit 15 set/clear, 14-13 `PRECOMP1:0`, 12 `MFMPREC`, 10 `WORDSYNC`, 9 `MSBSYNC`, 8 `FAST` |
| `DMACON` | `$DFF096` | write | shared | Bit 4 `DSKEN` and bit 9 `DMAEN` must both be set for disk DMA |
| `INTREQ` / `INTENA` | `$DFF09C` / `$DFF09A` | write | Paula | Bit 1 `DSKBLK` block complete, bit 12 `DSKSYN` sync found |

The `ADKCON` disk fields in full:

| Field | Value | Meaning |
|---|---|---|
| `PRECOMP1:0` | 00 / 01 / 10 / 11 | No precompensation / 140 ns / 280 ns / 560 ns |
| `MFMPREC` | 0 / 1 | GCR precompensation / MFM precompensation |
| `WORDSYNC` | 1 | Hold the transfer off until the `DSKSYNC` pattern is found |

| Field | Value | Meaning |
|---|---|---|
| `MSBSYNC` | 1 | Synchronize on the most significant bit of each byte, normally for GCR |
| `FAST` | 1 / 0 | Two-microsecond bit cell, normally MFM / four-microsecond cell, normally GCR |

## B.2 Pins

Paula's entire electrical involvement with the drive is three pins. Everything else in the drive interface reaches the CIAs.

| Pin | Name | Direction | Function |
|---|---|---|---|
| 37 | `DKRD*` | input | Pulled-high read data from the drive; a falling edge begins each low-going pulse for a detected flux reversal |
| 38 | `DKWD*` | output | Write data to the drive, the timed transition stream |
| 39 | `DKWE` | output | Write gate, which authorizes the drive to record |
| 12 | `DMAL` | output | The request line on which Paula asks Agnus for an audio or disk DMA cycle |

This document follows the manual's asterisk convention for active-low external pins. The dissection writes the same convention with `#`, so its `DKRD#` and `DKWD#` labels correspond to `DKRD*` and `DKWD*` here. Internal signal names retain the spelling used in the dissection. The write gate reaches the drive connector low while recording.

## B.3 Internal signals named in this document

These are the dissection's names for signals inside Paula. The list helps a reader find them on the schematic sheets; software cannot observe them directly.

| Signal | Made in | Meaning |
|---|---|---|
| `DKRD` | 10c | The inverted, conditioned, and synchronized form of external `DKRD*` |
| `DKRD_CK` | 10d | One pulse for each rising edge of internal `DKRD`, corresponding to a falling edge on external `DKRD*` |
| `DKR0` - `DKR2` | 10k | The cell-phase counter that tracks position within a bit cell |
| `DKRSET`, `DKRCLR` | 10j | Phase corrections, applied when an edge arrives early or late |
| `DKRINC#`, `DKRDEC` | 10j | Rate corrections, which adjust the estimated cell length |
| `DKREN` | 10t | The pacing output that advances the phase counter |
| `DKRD_CKR` | partly drawn | A recovered-clock signal whose expression contains two unidentified inputs |
| `FDC0` - `FDC15` | 10q, 10r | The receive shift register's outputs; the low eight are the `DSKBYTR` byte |
| `ISSYNC` | 10q, 10r | The comparator's match output, reported as `WORDEQUAL` |
| `DSKSY1` | 10e | The sync-wait level derived from the match |
| `DSKDE` | 10i | Set while the transfer engine is running |
| `DSKDG` | 10i | Holds the buffer queue empty while the controller is armed |

| Signal | Made in | Meaning |
|---|---|---|
| `DSKDL0` - `DSKDL2` | 10g | The per-stage strobes that move a word through the buffer cascade |
| `DL3` , `DL5` , `DL7` | 10g | The three disk DMA requests, one per buffer stage |
| `DSKL_C` | source omits driver | The length counter's terminal count, absent from the section-10 schematic |
| `DKDQ` | 10l | The serial output of the write shift register |
| `DKDW#` | 10v | The write data after precompensation |
| `DKWD` | 10e | The retimed write data that reaches the output pad |
| `DKWE#` | 10i | The internal write-gate condition; the August 2026 schematic revision adds a previously missing inverter to this path and labels the resulting line `DKWE` |

## Appendix C. Glossary

**Address mark.** A recorded pattern that identifies the start of a sector header or data field in formats that define one. `DSKSYNC` provides a programmable sixteen-bit comparison.

**Bit cell.** The interval of time allotted to one recorded bit. Whether a cell contains a flux transition is what carries the information, so a controller must know where each cell begins before it can decide the cell's value.

**Chip RAM.** The memory the custom chips can reach. Disk buffers must be in it, because Agnus cannot perform DMA into any other memory.

**Data separation.** Recovering both the bit timing and the data values from a pulse stream that carries them together. The name reflects the two outputs.

**DMA.** Direct memory access: a transfer performed by dedicated logic rather than by the processor. For disk, Paula decides that a word needs service and Agnus performs the memory cycle.

**Flux transition.** A change in the direction of magnetization along a track. The drive reports each one it detects as a digital pulse.

**GCR.** Group coded recording, a family of codes that map data groups onto recorded patterns obeying the medium's run-length limits. Used by Apple and Commodore drives.

**MFM.** Modified frequency modulation, the recording code of Amiga and PC double-density disks. Each data bit becomes a clock/data pair of channel bits, so the encoded stream contains twice as many bits as the unencoded data:

| Data bit | Recorded pair | Condition |
|---|---|---|
| 1 | `01` | always |
| 0 | `10` | when it follows a 0 |
| 0 | `00` | when it follows a 1 |

**Peak shift.** The tendency of closely spaced transitions to be detected at displaced times on readback because neighboring reproduced waveforms overlap. Write precompensation moves selected recorded transitions to counter the expected shift.

**Raw track.** The recorded bitstream of a track as it appears before any format-specific decoding: sync patterns, headers, gaps, encoded data, and checksums all still in their recorded form.

**Sync word.** A sixteen-bit pattern software programs into `DSKSYNC` so the controller can establish a word boundary in a continuous stream. Amiga MFM conventionally uses `$4489`.

**Terminal count.** The point at which a counter reaches the end of its programmed range. For disk transfers this is where the length counter stops the DMA block.

**Write precompensation.** Writing selected transitions slightly early or late so that peak shift moves them back toward their intended positions when the track is read. It changes when transitions are recorded, never which data is recorded.

**Write gate.** The signal that authorizes the drive to pass current through the head. Without it, activity on the write-data line does not alter the disk.

## Appendix D. Claim and evidence map

The table below connects the paper's principal conclusions to the progression in Figure 1-3. *Stage reached* names the last question the current work answers. The boundary column separately states whether further detail is intentionally deferred, still open, or absent from the available source. The table provides a compact provenance map. Part II carries the circuit derivations.

| Claim | Current result | Stage reached | Main evidence | Boundary |
|---|---|---|---|---|
| Paula's system boundary | Paula owns disk timing and data; the CIAs own mechanism control, and Agnus owns the memory address and DMA cycles. | **Checked** | AHRM pin/register descriptions, sheet 6 register decode, section-10 interfaces, and sheet 11 DMA request logic | Circuit-level decode and interface proof deferred to Part II. |
| Read input and edge formation | The input pad, Schmitt trigger, four-stage synchronizer, and edge detector produce one internal event per accepted drive pulse. | **Checked** | Modules 10c/10d, transistor copy, die view, and deterministic event replay | Detailed device sequence deferred to Part II. |
| Data-separator loop | Paula measures pulse phase and applies phase and rate corrections to | **Checked** | Modules 10j/10k/10s/10t, AHRM operating | Exact 10t pacing arithmetic is not extractable from |

| Claim | Current result | Stage reached | Main evidence | Boundary |
|---|---|---|---|---|
| | a fully digital timing loop. | | modes, and Keller patent cross-check | the abbreviated schematic. |
| Word after sync | With `WORDSYNC`, the token queue is held empty while armed and begins accepting data after the sync match, so DMA starts with the following word. | **Checked** | 10q/10r comparator, 10e sync level, 10i armed state, 10g queue equations, and combined state replay | Transistor equations and replay details deferred to Part II. |
| Shared data/token cascade | Three sixteen-bit data stages and a parallel occupancy queue create read-side full-stage requests and write-side empty-stage requests. | **Checked** | Modules 10l/10g/10u, extracted queue equations, and simulated push/retire sequences | Device-level proof deferred to Part II. |
| Write precompensation magnitude | The three nonzero `PRECOMP1:0` choices select doubled-clock delay paths in the same 1:2:4 relationship as the documented 140, 280, and 560 ns corrections. | **Checked** | Module 10v structure and clock chain, sheet 3 clocks, and AHRM values | `MFMPREC` pattern-selection rules remain open. |
| `DSKLEN` double write | A write strobe can arm 10i only when the retimed enable is already visible, so the first write enables and the second starts the transfer. | **Checked** | Modules 10p/10i, extracted state equations, simulated one-write/two-write sequences, and AHRM procedure | Circuit derivation deferred to Part II. |
| Final write-bit loss | The post-serializer pipeline and the write-gate closing terms are extracted; incomplete drain is the leading inference. | **Extracted** | Modules 10l/10v/10e/10i and the AHRM warning | Exact three-bit mechanism is open. |

| Claim | Current result | Stage reached | Main evidence | Boundary |
|---|---|---|---|---|
| Possibly missing final read word | The last word must traverse the token queue and receive Agnus service while 10i responds to terminal count. | **Extracted** | Modules 10g/10l/10m/10i and the AHRM warning | Exact result is open because `DSKL_C` formation is not extractable from the schematic. |

The stage belongs to the individual claim, regardless of how well the surrounding block is understood. Part I takes its principal functional conclusions through the checked stage. Part II will present the equations, device accounting, and replay details needed for a reader to reproduce the circuit proofs from the source.

## Appendix E. Sources

The circuit conclusions in this paper come from the dissection, checked against the manual and patent where their coverage overlaps. The controller and software sources below support the broader comparisons and operating examples.


1. Commodore-Amiga, Inc., *Amiga Hardware Reference Manual*, third edition, Addison-Wesley, 1991, ISBN 0-201-56776-8.
2. Glenn Keller, *Data Input Circuit with Digital Phase Locked Loop*, US patent 4,780,844, October 25, 1988.
3. Frank Wolf and ttlworks, MOS 8364R7 Paula dissection project and discussion, 6502.org, begun 2023. This analysis used the schematic release current in late July 2026. A revision of August 4, 2026 corrects inverter polarities in three 10i output paths (the length-count enable, the write gate, and the completion interrupt) and redraws four 10h carry-tap cells as direction-selected multiplexers, leaving every PLA contact pattern unchanged. The revised schematic reproduces the transfer lifecycle described in this paper as drawn; no conclusion stated here required change against it.
4. Western Digital, *Storage Management Products Handbook*, WD17xx/WD177x material, 1986.
5. NEC, *μPD765A Floppy Disk Controller Datasheet*.
6. Intel, *82077AA CHMOS Single-Chip Floppy Disk Controller*.
7. Apple Computer, *Disk II Installation and Operating Manual*, 1982, and *Apple II DOS User's Manual*, 1983.
8. AmigaOS documentation, `trackdisk.device` and CrossDOS.
9. ReadySoft, *A-Max IV Manual*, version 4.0.
10. Kunihiro Hashimoto, Chiharu Tsukamoto, and Chiharu Kawakubo, *Write Data Write-Pre-Compensation System in a Floppy Disk Drive Unit and Apparatus Therefor*, US patent 5,187,614, February 16, 1993.