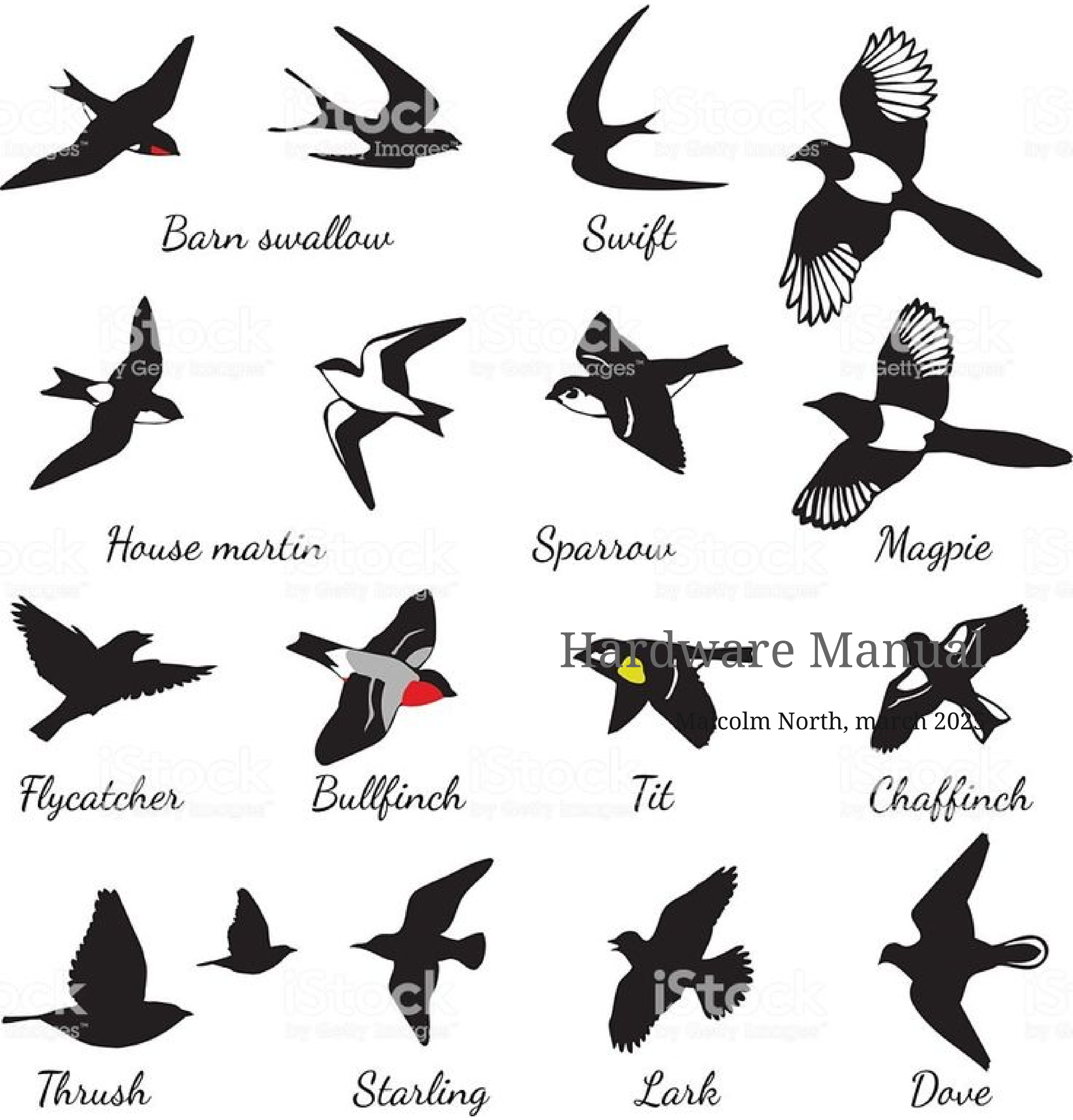
[] | *kingfisher.jpg*



Hardware Manual

Malcolm North, march 2023

Table of Contents

Chapter 1. Abstract

The Aves hardware platform represents a flexible and extensible 8/16-bit computer architecture designed to bridge classic computing with modern interfaces and capabilities. Named after the taxonomic class of birds, Aves embodies both elegance and adaptability in its design.

This system supports multiple processor configurations, from the classic R65C02 to the more advanced W65816, with options for dual-processor arrangements incorporating the V25, V35, Z8K and MC68K. The architecture accommodates various memory configurations, from simple linear addressing to sophisticated bank-switched and segmented memory layouts, providing up to 2MB of addressable space.

At its core, Aves combines the simplicity and reliability of 8-bit systems with some of the expandability of contemporary systems. The platform features a versatile I/O subsystem built around the W65C22 VIA, supporting multiple communication protocols including I2C, SPI, and the custom Aves Serial Bus (ASB).

This hardware reference manual provides comprehensive documentation of the Aves system architecture, memory configurations, bus arrangements, and interface specifications. It serves as the authoritative reference for hardware developers, system programmers, and enthusiasts working with the Aves platform.

Chapter 2. Foreword

The Aves hardware platform represents a thoughtful convergence of classic computing heritage and modern design principles. This project emerged from a deep appreciation for the elegant architectures of early microcomputers, particularly those from Commodore, while embracing contemporary interface standards and manufacturing techniques.

In developing Aves, the preservation of the accessibility and educational value that made 8-bit systems so remarkable, was the highest priority. The result is not merely a recreation of vintage hardware, but rather a carefully considered evolution that maintains the spirit of its predecessors while offering new possibilities.

This reference manual documents the technical implementation of these goals. Whether you're a hardware developer integrating Aves into a larger system, a programmer writing software for the platform, or an enthusiast exploring computer architecture, you'll find detailed information about the system's capabilities and design principles.

The manual is structured to serve both as a comprehensive reference and as an educational resource. While some sections delve into complex technical details, there is a focus on clarity and practical application throughout. The included examples and explanations aim to make the material accessible without sacrificing technical accuracy.

We hope this documentation helps you understand and utilise the full potential of the Aves platform, whether you're building, programming, or simply exploring this bridge between classic and contemporary computing.

Part I: System Architecture Overview

Chapter 3. Architectural Principles

Before delving into specific technical details, it’s important to understand the fundamental design philosophies and architectural decisions that shape the Aves platform. These principles reflect both practical engineering considerations and broader goals for the system’s role in modern retrocomputing.

The architecture of Aves represents a careful balance between preserving the elegant simplicity of classic 8-bit systems and incorporating modern design practices. Rather than simply replicating vintage hardware, each aspect of the system has been thoughtfully evaluated and, where appropriate, enhanced to meet contemporary needs while maintaining the spirit of its inspiration.

At the heart of Aves lies Considered Minimalism, where each component serves multiple purposes through clever design rather than brute-force complexity. This approach yields efficient use of resources while maintaining clarity and serviceability.

The platform embraces Pragmatic Modernization, incorporating contemporary interfaces and manufacturing techniques in ways that preserve the system’s accessibility and educational value. This careful integration of modern elements enhances rather than obscures the fundamental simplicity of the design.

Flexible Evolution stands as another cornerstone of the architecture. The system accommodates various processor and memory configurations while maintaining a consistent programming model, allowing it to grow with user needs without compromising its core principles.

The following sections detail how these principles manifest in specific aspects of the system architecture, from processor selection to memory management and I/O subsystems.

Chapter 4. System Architecture

The Aves platform’s architectural heritage stems from the remarkable family of Commodore 8-bit and 16-bit computers, drawing particular inspiration from the CBM8096, Commodore 64 and Plus/4 machines. Whilst these classic designs laid the groundwork, Aves refines and modernises their most successful elements into a cohesive and elegant architecture.

At the heart of every Aves system lies a philosophy of considered minimalism, where each component serves multiple purposes through clever design rather than brute-force complexity. This approach manifests most notably in the memory management and I/O subsystems, where time-tested concepts from the Commodore machines have been thoughtfully reimagined.

The memory architecture pays homage to the innovative bank-switching schemes of its forebears. Where the CBM8096 introduced upper memory banking, and the Commodore 64 demonstrated the flexibility of ROM overlay techniques, Aves unifies these concepts into a sophisticated yet straightforward memory management system. This arrangement provides remarkable flexibility whilst maintaining compatibility with traditional software design patterns.

Perhaps the most significant departure from tradition lies in the I/O subsystem. Where Commodore machines typically employed multiple interface adapters—CIAs, PIAs, and various custom chips—Aves consolidates these functions into a carefully orchestrated arrangement centred on the W65C22 VIA. This consolidation does not represent a compromise but rather an elegant solution that reduces complexity whilst expanding capabilities.

The VIA implementation demonstrates the platform’s pragmatic approach to modernisation. Through thoughtful programming of this versatile chip, Aves achieves compatibility with contemporary interfaces such as I2C and SPI, whilst also supporting a custom high-speed serial protocol, the Aves Serial Bus (ASB). This bus draws inspiration from the efficiency of Commodore’s serial bus architecture whilst eliminating its notorious timing dependencies.

Throughout the system, one finds this pattern of respectful modernisation. The hardware banking mechanisms, whilst more sophisticated than their predecessors, maintain familiar programming paradigms. The interrupt handling system, though more capable than the original Commodore implementations, remains straightforward and predictable. Even the most advanced configurations, supporting the 16-bit W65816 processor, retain compatibility with their 8-bit siblings through carefully considered hardware abstractions.

This architectural philosophy extends to the physical design as well. The circuit board layouts, component selection, and signal routing all reflect a balance between simplicity and capability. Modern manufacturing techniques and components are employed where beneficial, yet the system remains accessible to hobbyist construction and modification, maintaining the spirit of its predecessors.

I2C Interface

The I2C interface is implemented in software using PA0 of the VIA for SCL and PA7 for SDA. This arrangement provides optimal bit manipulation capabilities, as testing bit 7 with the BIT instruction will directly affect the processor’s N flag for efficient state testing. Both lines require external 4.7kΩ pull-up resistors to VCC to ensure proper signal levels and to meet I2C bus specifications.

The implementation leverages specific characteristics of the 65C02 instruction set. The lines are manipulated using the data direction register to create a pseudo open-drain configuration - setting a pin as input allows it to float high via the pull-up resistor, while setting it as output drives it low. Start conditions are generated by transitioning SDA from high to low while SCL is high, with stop conditions created by the opposite transition. Clock generation employs increment and decrement instructions for precise timing control, while the BIT instruction enables rapid testing of PA7's state through the N flag.

The interface supports standard mode I2C operation at 100 kHz, with timing managed through software delays calibrated to the system clock frequency. Clock stretching is supported through continuous monitoring of the SCL line state, allowing compatibility with slower I2C peripheral devices. Multi-master mode is not supported.

SPI Interface

The SPI interface is implemented in firmware, as modes 0 and 3 only. As with the I2C interface the clock; SCK is implemented on PB0, PB1-PB3 contains the device number and is decoded with a 74HC138 decoder to provide 7 device selects ss1-ss7.

MOSI uses PB6 and MISO PB7.

SPI is level converted to 3.3V for compatibility with most SPI memory and IO devices.

Aves Serial Bus (ASB)

The Aves serial bus uses the VIA shift register to implement a half duplex serial protocol, that is substantially faster than Commodore's propitiatory Serial bus (frequently incorrectly referred to as IEC). The VIA's CAn lines are used as handshake lines. PA6 is used to determine if the bus is transmitting or receiving

PS/2 Keyboard Interface

The PS/2 Keyboard interface is based on Ben Eater's PS/2 keyboard interface design, which uses 74HC595 shift registers. Unlike Ben's implementation, data is read directly from the register's parallel output, freeing up 8 additional I/O lines on the VIA. The interrupt logic remains unchanged, using the VIA's CA1 input to detect incoming data.

I/O expansion port

The I/O expansion port is presented on a 34 pin right angle male header, it comprises the 8 x data lines, 12 x address lines, 4 x select lines, RnW, Phi2, RESETb, IRQb, power and gnd.

Table 1. Expansion Port Pinout

Pin No	Description
1,2	Ground
3-10	8 bit, Address Bus, A0-A7

Pin No	Description
11	RnW signal 1=read, 0=write
12	Phi2 clock, represents phase 2 of the system clock, which is used to synchronise all CPU operations
13-20	8 bit Data bus D0-D7
21-23	IOb1-IOb3, active low IO select signals
24	IRQb, Interrupt request, active low
25	RESb, Reset signal, active low
26	RDY, signals that the CPU can continue
27	BE, bus enable signal enables cpu address and data bus tristate buffers
28, 29	Not Used
31,32	+5V Supply
33,34	Ground

References

- [1] Western Design Center, "W65C02S 8-bit Microprocessor", Publication 651xx-14 Rev. 8.0
- [2] Western Design Center, "W65C816S 16-bit Microprocessor", Publication 655xx-16 Rev. 10.0
- [3] Western Design Center, "W65C22S Versatile Interface Adapter", Publication 652xx-14 Rev. 4.0
- [4] NEC Electronics, "V25 16-bit Single Chip CMOS Microcomputer", Document ID: S11988EJ3V0UM00
- [5] Alliance Memory, "AS6C1008 128K x 8 Low Power CMOS SRAM", Rev. 1.0
- [6] Ben Eater, "Building a 6502 computer", https://eater.net/6502

NOTE

Current versions of these datasheets may be obtained from:

- Western Design Center documents: Available from manufacturer website or preserved copies in Aves repository
- NEC/Renesas documents: Available from Renesas historical documentation archive
- Memory datasheets: Available from current manufacturers of compatible devices

The specific versions used in developing Aves are preserved in the project repository under `/doc/datasheets/` to ensure reproducible builds and consistent reference.

- [7] Motorola, "MC68HC000 HCMOS Microprocessor", ADI1024R1
- [8] Zilog, "Z16C01/02 CMOS CPU with MMU", DC2144-01
- [9] NEC Electronics, "V35 16-bit Single-Chip CMOS Microcomputer User’s Manual", Document ID: S11989EJ3V0UM00

NOTE

The CPU timing specifications can be found in: * W65C02S/W65C816S - References [1] and [2] * V25/V35 - References [4] and [9] * MC68HC000 - Reference [7] * Z16C01 - Reference [8]

Chapter 5. Aves 8-bit Models

Chapter 6. Aves Peripherals

Chapter 7. Aves 16-bit Models

Chapter 8. Foundational Influences

Garth Wilson’s work has influenced multiple aspects of Aves:

Physical Layer Solutions

- VIA shift register timing fixes
- I2C implementation principles
- SPI interface design patterns

Interrupt Handling

- His high-level Forth interrupt approach influencing Kingfisher
- Clean separation of hardware and software concerns
- Efficient interrupt processing

Part II: CPU and Memory Systems

Chapter 9. Processor Architecture

Primary CPU Options

Processor Options

Aves offers a choice of 8 and 16-bit CPUs within a common architecture. The platform’s I/O operations are handled by an 8-bit CPU to maximise Kernel code reuse and simplify multi-architecture support. All CPUs are CMOS variants for reduced power consumption.

The 6502’s minimal register set and zero-page usage as pseudo registers provides excellent interrupt latency. When paired with support chips like the 6522 Versatile Interface Adapter (VIA), this makes it an ideal choice for I/O handling.

In dual processor configurations, a R65C02 running at 4MHz serves as the I/O co-processor, working alongside primary CPUs such as the V25/V35, Z16C00, or MC68HC000. Communication between processors occurs through dual port memory, creating a loosely coupled architecture that simplifies inter-CPU communication and allows each processor to run at its optimal clock speed.

Table 2. Supported CPU Types

CPU	Description
R65C02	The Rockwell R65C02 is no longer in production, but is widely available as new old stock. It is used in several Aves models and is always clocked at 4MHz. The R65C02 is pin compatible with the original MOS 6502, which was fabricated in NMOS. One of many advantages the 65C02 has over its NMOS sibling is an enhanced instruction set
W65C02	The Western Design Center W65C02 is currently in production. WDC were the creators of the CPU core used in the R65C02, and the W65C02 is later version of this core. The same instruction set is used but the CPU it can be run at 14MHz, is fully static and can has tri-state address and data bus lines. A new WAI instruction stops the CPU clock and reduces power to a minimum, it will restart when an interrupt occurs
W65816	The Western design center W65816 is a 16-bit variant of the W65C02 with an 8-bit emulation mode as well as 24 bit segmented address bus. The W65C816 has a maximum clock speed of 14MHz
NEC V25 and V35	The NEC (now Renesas) V25 and V35 are CMOS variants of the Intel 8088 and 8086 respectively, with integrated UARTs, Timers, GPIO and DMA. The V25 has an 8-bit data bus and the V35 has a 16-bit bus both have a 20 bit address bus. which uses the same segmented strategy used by Intel

CPU	Description
MC68HC000	The Motorola MC68HC000 highly successful and well regarded CPU, was used to power the Commodore Amiga and Atari ST computers. Its architecture and performance were a breakthrough, and this CPU is adopted to pay homage to such an innovation. The MC68HC000 had a 32 bit addressing capability with 24 bit address bus width. It has a maximum clock speed of 10MHz, and has a dual mode CPU bus timing that supports older 6800 and 6500 8-bit peripherals, as well as an updated asynchronous native bus.
Z16C00	The Z16C01 is a CMOS version of the Zilog Z8001 CPU, and is included as a curiosity and acknowledgment of the Commodore C900 Zilog Z8001 based "UNIX machine" prototype. The Z8001 was another 16-bit CPU that was developed at around the same time as the Intel 8086 and Motorola 68000. The Z8001 used a 24 bit segmented architecture, although only 23 bits where available on the address bus. The Z16C01 has a maximum clock frequency of 10MHz

R65C02 Implementation

- Technical Specifications
- Clock Generation
- Reset and Interrupt Handling
- Memory Interface

W65816 Implementation

- Technical Specifications
- Operating Modes (8/16-bit)
- Memory Management
- Interrupt Vectors

Dual Processor Systems

Secondary Processor Options

- V25/V35 Configuration
- Z8K Integration
- MC68K Implementation

Processor Control

- CPU Selection Logic
- Mode Switching
- Interrupt Handling
- Shared Resources

Inter-processor Communication

- Mailbox System
- Synchronization Methods
- Shared Memory Access
- Message Passing

Chapter 10. Memory Architecture

Address Space Organization

- Memory Map Overview
- Bank Layout
- Reserved Regions
- I/O Memory Mapping

Memory Management

Bank Switching

- Control Registers
- Bank Selection Logic
- Page Mapping
- Shadow RAM

Memory Protection

- Write Protection
- Read Protection
- System Reserved Areas
- Configuration Registers

Memory Types

RAM Configuration

- Static RAM
- Dynamic RAM Support
- Refresh Requirements
- Wait States

ROM/EPROM

- Boot ROM
- Character ROM
- Flash Memory
- Programming Interface

Memory Timing

Access Timing

- Read Cycles
- Write Cycles
- Refresh Cycles
- Bus Arbitration

DMA Operations

- DMA Controllers
- Transfer Modes
- Priority Levels
- Channel Assignment

Chapter 11. Implementation Details

Hardware Interface

Address Decoding

- Logic Implementation
- Chip Select Generation
- Memory Type Detection
- Error Handling

Bus Control

- Bus Arbitration
- Wait State Generation
- Clock Synchronization
- Reset Logic

Configuration

System Setup

- Jumper Settings
- DIP Switch Configuration
- Memory Size Selection
- CPU Mode Selection

Diagnostic Features

- Memory Testing
- CPU Verification
- Error Reporting
- Status Indicators

Chapter 12. CRTC VMAC

- Clock8 = 8MHz clock from divider
- Phi2 = 4MHz CPU clock is 4MHz from divider
- Chck = 2MHz Chracter clock
- Ldchb = Pulse to load character into character latch
- Ldsrb = Pulse to load shift register on next clock rising edge
- ldcolb = Loads colour latch

Control Signal Timing

```
011001100110011001  ← Clock8
011110000111100001  ← Phi2
011111111000000001  ← chck

011111111111111001  ← ldcolb
111111100111111111  ← ldchb


           ___     ___     ___     ___
Clock8    |   |___|   |___|   |___|   |___|
           _______         _______
Phi2      |       |_______|       |_______|
           _______________
CHCK      |               |_______________|
           ___________________________
LDCOLB                                |___|
           ___________     ________________
LDCHB                 |___|
```

sh/ld Timing

```
00000000111111110   ← char clock
10000000011111111   ← sh/ld
            0_  1_  2_  3_  4_  5_  6_  7_  0_
Char Clock  | |_| |_| |_| |_| |_| |_| |_| |_|_|
            __________________________     ____
SH/LD                                 |___|
                                        ▯
                                    Load occurs
                                   (rising edge captures data
                                    while SH/LD still low)
```

sh/ld Generation

```
  A______________      ___
    |   ___      |____|   |
    |__|   |          | + |____B
    |  | + |__________|   |
    |__|   |          |___|
       |___|

A = Character Clock (2MHz)
B = Shift Load pulse
```

Red Bit ----[510]----+ | [1k] | Red I Bit ---[510]--+ | [1k] | -------- | Green Bit ---[510]--+ | | | [1k] | | | Green I Bit -[510]--+ | | | Ladders terminate here [1k] | | | | v ----------------[75]----> | | Blue Bit ----[510]-- | [150] | | | [1k] | GND | | Blue I Bit --[510]--+ | | | [1k] | | | --------

WinCUPL Equations

```
/* low going pulse to load character into Latch on rising edge */
ldchb = !(!clk8 & !phi2 & chck)

/* low pulse loads first colour latch */
ldcol1b = !(!clk8 & !phi2 & !chck)

/* low pulse loads first colour latch */
ldcol2b = !(!clk8 & !phi2 &  chck)
```

Chapter 13. Aves Video Controller Timing Analysis

Display Modes

- 40 Column Mode (320x300)
  - 16MHz dotclock
  - 2MHz character clock
  - 1K character + 1K colour memory
- 80 Column Mode (640x300)
  - 32MHz dotclock
  - 2MHz character clock
  - 2K character + 2K colour memory

Clock Selection

- External hardware jumper selects dotclock frequency
  - 16MHz for 40 column mode
  - 32MHz for 80 column mode
- Feeds directly to GAL dotclock input (pin 8)
- Character clock remains constant at 2MHz

Internal GAL Timing

Dotclock is used for:

- Cursor inversion logic
- Latch enable pulse generation
- Shift register load timing

Timing Analysis

- 10nS GAL propagation delay is acceptable because:
  - Latch enables occur during stable periods
  - Cursor inversion happens while data is stable
  - Not in critical path for pixel timing
  - All signals synchronized to system clocks

Monitor Output

- SVGA compatible timing
- Horizontal: 40/80 characters (320/640 pixels)
- Vertical: 25 characters (300 pixels)
- Character cell: 8 x 12 pixels

Timing Relationships

The shift/load timing is controlled by the dot clock:

```
00000000111111110   ← dot clock
10000000011111111   ← sh/ld

Dot Clock   □□□□□□□□□|_________|□□□□□□□□
                     □         □
SH/LD       □□□□□□□□□□□|__________|□□□□□
                               □
                          Load occurs
```

The character and colour loading signals (ldchb and ldcolb) are generated from Clock8, Phi2 and Chck:

```
ldchb = !(!clk8 & !phi2 & chck)
ldcolb = !(!clk8 & !phi2 & !chck)
```

These signals coordinate the loading of character and colour data during the sh/ld pulse window shown above.

Chapter 14. Video Controller Timing Analysis

Display Modes

- 40 Column Mode (320x300)
  - 16MHz dotclock
  - 2MHz character clock
  - 1K character + 1K colour memory
- 80 Column Mode (640x300)
  - 32MHz dotclock
  - 2MHz character clock
  - 2K character + 2K colour memory

Clock Selection

- External hardware jumper selects dotclock frequency
  - 16MHz for 40 column mode
  - 32MHz for 80 column mode
- Feeds directly to GAL dotclock input (pin 8)
- Character clock remains constant at 2MHz

Internal GAL Timing

Dotclock is used for:

- Cursor inversion logic
- Latch enable pulse generation
- Shift register load timing

Timing Analysis

- 10nS GAL propagation delay is acceptable because:
  - Latch enables occur during stable periods
  - Cursor inversion happens while data is stable
  - Not in critical path for pixel timing
  - All signals synchronized to system clocks

Monitor Output

- SVGA compatible timing
- Horizontal: 40/80 characters (320/640 pixels)
- Vertical: 25 characters (300 pixels)
- Character cell: 8 x 12 pixels

Timing Relationships

Shift/Load Timing

```
000000000111111110   ← dot clock
100000000011111111   ← sh/ld

Dot Clock   □□□□□□□□□|_________|□□□□□□□□
                     □         □
SH/LD       □□□□□□□□□□□|__________|□□□□□
                               □
                          Load occurs
```

The character and colour loading signals (ldchb and ldcolb) are generated from Clock8, Phi2 and Chck:

```
ldchb = !(!clk8 & !phi2 & chck)
ldcolb = !(!clk8 & !phi2 & !chck)
```

These signals coordinate the loading of character and colour data during the sh/ld pulse window shown above.

Part III: ASB Protocol Architecture

Chapter 15. Foundational Influences

Garth Wilson’s work has influenced multiple aspects of Aves:

Physical Layer Solutions

- VIA shift register timing fixes
- I2C implementation principles
- SPI interface design patterns

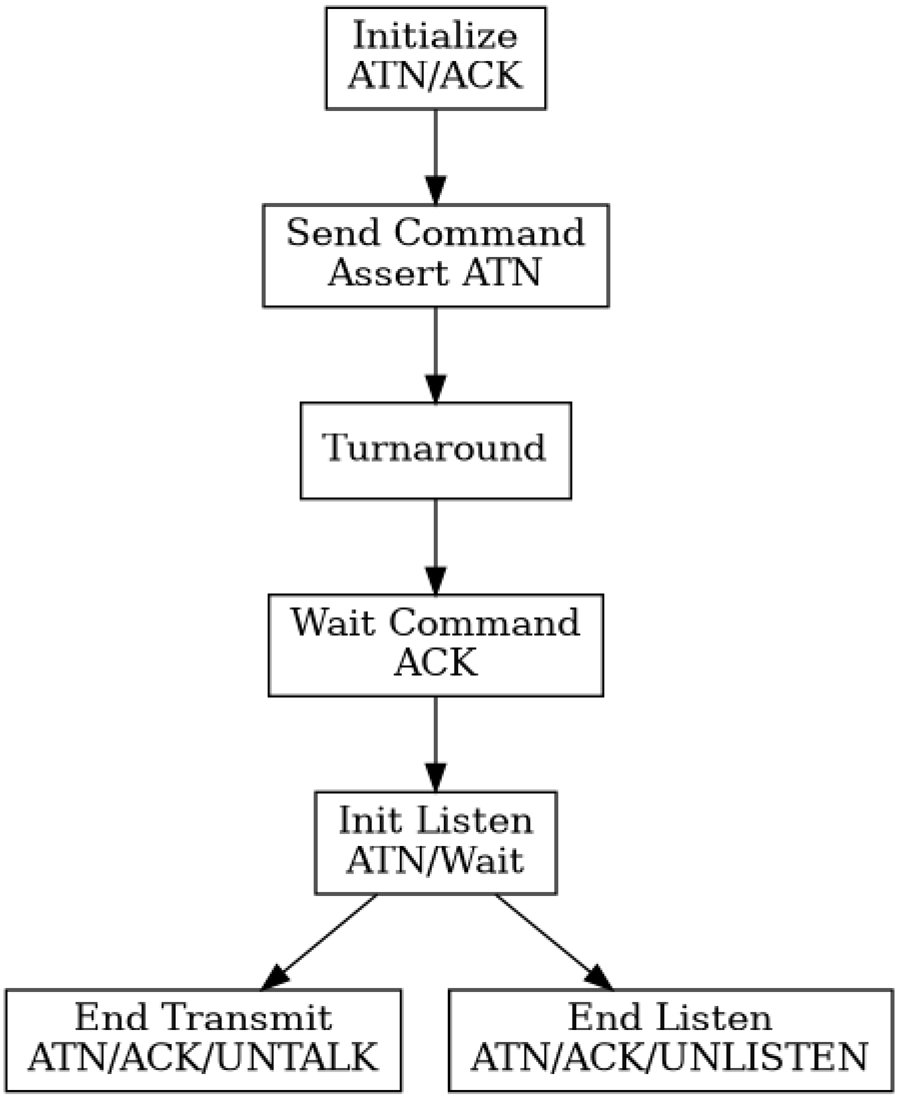
Interrupt Handling

- His high-level Forth interrupt approach influencing Kingfisher
- Clean separation of hardware and software concerns
- Efficient interrupt processing

Chapter 16. Bus Implementation

Flap Transport Protocol

The transport protocol ensures reliable end-to-end data exchange between devices on the ASB bus. It manages the handshaking sequence, flow control, and connection states required for dependable communication between controller and responder devices.



```
digraph transport_protocol {
    rankdir=TB;
    node [shape=box];

    init [label="Initialize\nATN/ACK"];
    cmd [label="Send Command\nAssert ATN"];
    turn [label="Turnaround"];
    wait [label="Wait Command\nACK"];
    listen [label="Init Listen\nATN/Wait"];

    eoi_tx [label="End Transmit\nATN/ACK/UNTALK"];
    eoi_rx [label="End Listen\nATN/ACK/UNLISTEN"];
    init -> cmd;
    cmd -> turn;
    turn -> wait;
    wait -> listen;

    listen -> eoi_tx;
    listen -> eoi_rx;
}
```

Initialize Transaction

1. Send ATN pulse
2. Wait for ACK Pulse

Command/Data Transmission

1. Assert ATN (low)
2. Command/Data

Turnaround

1. Wait for command acknowledge
2. Initialize listen
3. Send ATN pulse
4. Wait for data

End Transmission (EOI)

1. Pulse ATN
2. Wait for ACK
3. Send UNTALK command

End Listen (EOI)

1. Pulse ATN
2. Wait for ACK
3. Send UNLISTEN command

Protocol Features

- ATN pulse signaling for synchronization
- Handshake acknowledgment required
- Clear state transitions

- Explicit end-of-transmission handling
- Separate paths for transmit/listen completion

Protocol Overview

Synchronization

- Every transaction begins with ATN/ACK handshake
- All devices synchronize at start of each transaction
- No edge cases possible - always returns to known state

State Management

- Only two primary states:
    - IDLE (default state)
    - ACTIVE (during transaction)
- All error conditions return to IDLE
- New transaction always starts with synchronization

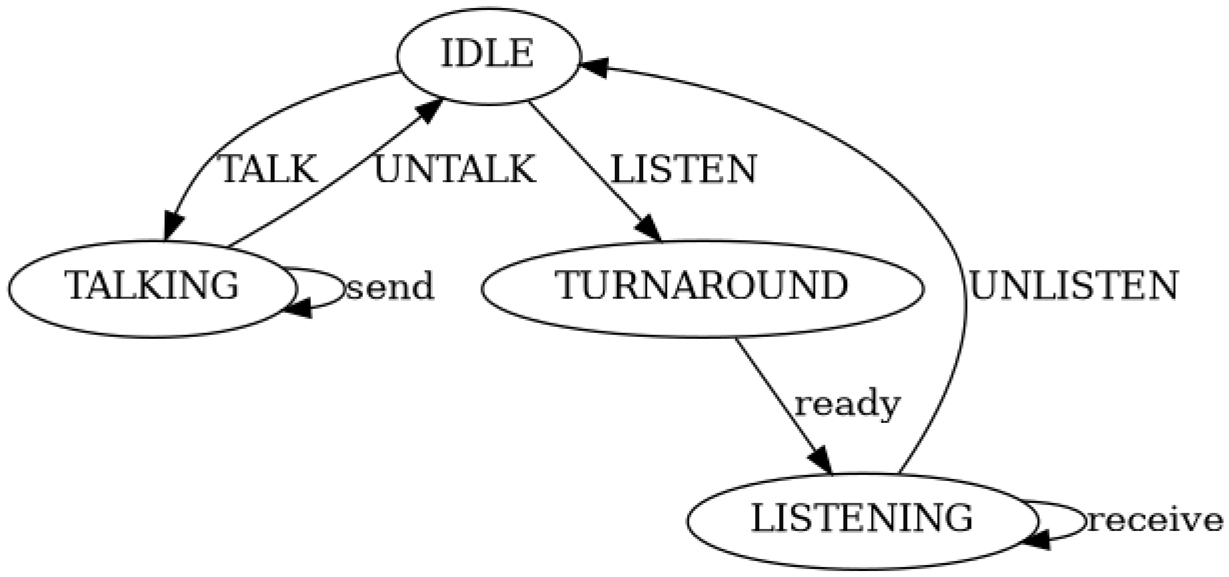
Design Benefits

- Self-synchronizing protocol
- No ambiguous states possible
- Clean recovery from all error conditions
- Simple, deterministic behavior

Device States

Controller Device States

The controller implements a four-state machine model that manages bus operations and data flow. This design ensures orderly transitions between sending commands, transmitting data, and receiving responses.



State Transitions

```
digraph controller_states {
    idle [label="IDLE"]
    talking [label="TALKING"]
    turnaround [label="TURNAROUND"]
    listening [label="LISTENING"]

    idle -> talking [label="TALK"]
    idle -> turnaround [label="LISTEN"]

    talking -> talking [label="send"]
    talking -> idle [label="UNTALK"]

    turnaround -> listening [label="ready"]

    listening -> listening [label="receive"]
    listening -> idle [label="UNLISTEN"]
}
```

State Descriptions

- IDLE
  - Default bus state
  - Ready to initiate commands
  - No active transfers
- TALKING
  - Controller is sending data
  - Maintains state for multiple sends
  - Returns to IDLE via UNTALK
- TURNAROUND

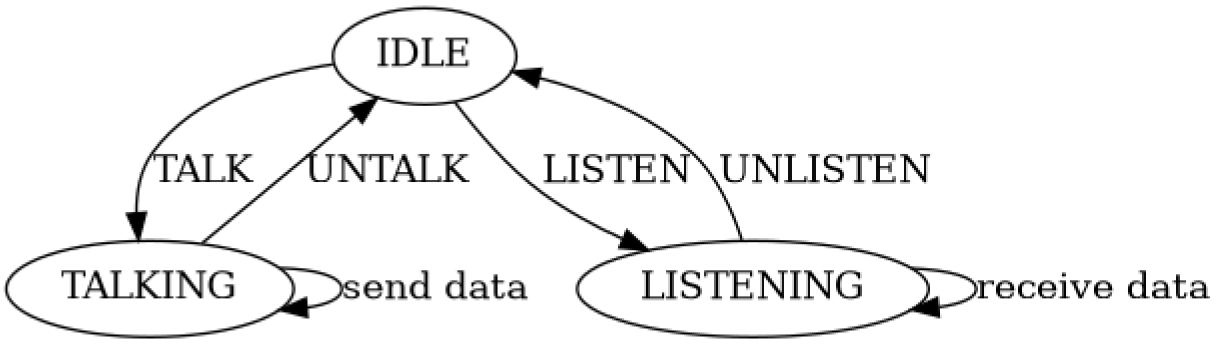
    - Transitional state between IDLE and LISTENING
    - Preparing bus for receive operation
    - Transitions to LISTENING when ready
- LISTENING
    - Controller receiving data
    - Can receive multiple data bytes
    - Returns to IDLE via UNLISTEN

Transition Rules

- All transfers start from IDLE
- TURNAROUND required before LISTENING
- Self-loops on TALKING/LISTENING for data transfer
- Clean return to IDLE via UNTALK/UNLISTEN

Responder Device States

The responder device follows a simple but robust state machine model that ensures reliable communication on the ASB bus.



State Transitions

```
digraph responder_states {
    idle [label="IDLE"]
    talking [label="TALKING"]
    listening [label="LISTENING"]

    idle -> talking [label="TALK"]
    idle -> listening [label="LISTEN"]

    talking -> talking [label="send data"]
    talking -> idle [label="UNTALK"]

    listening -> listening [label="receive data"]
    listening -> idle [label="UNLISTEN"]
}
```

State Descriptions

IDLE State

- Default power-on state
- Device is offline and not participating in bus transactions
- Monitors bus for ATN signal
- Must complete ATN/ACK handshake before responding to any commands
- Only transitions from IDLE after:
  1. Detecting ATN signal
  2. Completing ATN/ACK handshake
  3. Receiving command with matching address

TALKING State

- Device is online and transmitting data
- Entered from IDLE after successful handshake and address match
- Maintains control of bus until transmission complete
- Returns to IDLE after completion

LISTENING State

- Device is online and receiving data
- Entered from IDLE after successful handshake and address match
- Monitors incoming data until transaction complete
- Returns to IDLE after completion

Reset Sequence

Operation

- Sets all devices to IDLE state
- No handshaking required
- No acknowledgment needed

Device Response

- Immediate return to IDLE
- Clear any pending transactions

- Ready for new ATN/ACK sequence

That’s all there is to it - simplicity is a feature here. The reset provides a clean slate without any complex negotiation or state management.

Chapter 17. Physical Layer Specification

Physical Characteristics

- Half duplex communication using 65C22 VIA
- System clock (Phi2) requirement: 4MHz minimum
- Maximum data rate: 1Mbps

Hardware Implementation

- Pulsed handshake using edge-sensitive CA1/CA2 I/O
- External tristate buffers for I/O direction control
- Clock synchronization via Phi2-clocked latch

Network Topology

- Single controller architecture
- Supports up to 15 responder devices
- Each device uniquely addressable

Data Transfer

- Byte-by-byte transmission
- Each byte requires handshake acknowledgment
- Self-pacing through ACK mechanism
- No fixed timing requirements between bytes

Reliability Features

- Edge-triggered handshaking
- Hardware flow control via ACK
- Automatic speed matching to receiver capabilities
- Robust clock synchronization

Chapter 18. FLAP Data-Link Protocol

IEEE488 Command Structure

- LISTEN (0x20 + device) - Assigns device as data receiver
- TALK (0x40 + device) - Assigns device as data transmitter
- UNLISTEN (0x3F) - Releases all devices from listen mode
- UNTALK (0x5F) - Releases current talker

Connection Management

Establishing Connection

1. Controller sends LISTEN command to target device(s)
2. Controller sends TALK command to source device
3. Data transfer can begin

Terminating Connection

1. Controller sends UNLISTEN to release listeners
2. Controller sends UNTALK to release talker
3. Bus returns to idle state

Device Addressing

- Device addresses: 0-14 (4 bits)
- Address 15 reserved
- Commands: Upper 2 bits define command type
- Lower 4 bits contain device address

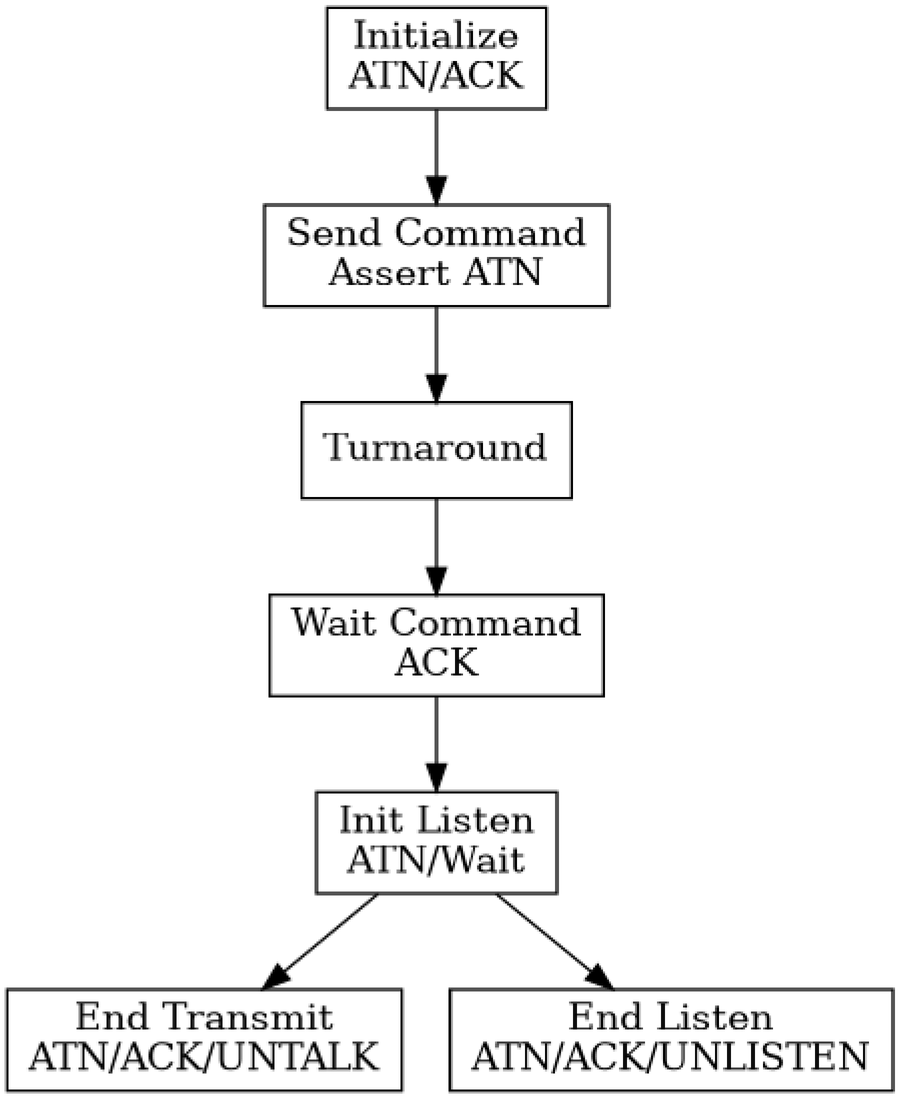
Protocol Features

- Clear command structure
- Deterministic bus control
- Multiple listener support
- Single talker at a time

Flap Transport Protocol

The transport protocol ensures reliable end-to-end data exchange between devices on the ASB bus.

It manages the handshaking sequence, flow control, and connection states required for dependable communication between controller and responder devices.



```
digraph transport_protocol {
    rankdir=TB;
    node [shape=box];

    init [label="Initialize\nATN/ACK"];
    cmd [label="Send Command\nAssert ATN"];
    turn [label="Turnaround"];
    wait [label="Wait Command\nACK"];
    listen [label="Init Listen\nATN/Wait"];

    eoi_tx [label="End Transmit\nATN/ACK/UNTALK"];
    eoi_rx [label="End Listen\nATN/ACK/UNLISTEN"];

    init -> cmd;
    cmd -> turn;
    turn -> wait;
    wait -> listen;
    listen -> eoi_tx;
    listen -> eoi_rx;
}
```

Initialize Transaction

1. Send ATN pulse
2. Wait for ACK Pulse

Command/Data Transmission

1. Assert ATN (low)
2. Command/Data

Turnaround

1. Wait for command acknowledge
2. Initialize listen
3. Send ATN pulse
4. Wait for data

End Transmission (EOI)

1. Pulse ATN
2. Wait for ACK
3. Send UNTALK command

End Listen (EOI)

1. Pulse ATN
2. Wait for ACK
3. Send UNLISTEN command

Protocol Features

- ATN pulse signaling for synchronization
- Handshake acknowledgment required
- Clear state transitions
- Explicit end-of-transmission handling
- Separate paths for transmit/listen completion

Frame Format

Structure

- Command byte (1 byte)
- Length byte (1 byte, 0-255)
- Data payload (length bytes)
- CRC-16 (2 bytes, present if length > 0)
  - Covers all preceding bytes (command, length, and payload)
  - Always placed at end of frame
  - Omitted for zero-length frames

Frame Types

Command Frame (length = 0)

- Command byte
- Length = 0
- No payload
- No CRC

Data Frame (length > 0)

- Command byte
- Length byte (1-255)
- Data payload (length bytes)
- CRC-16 covering all preceding bytes

CRC-16 Specification

- Polynomial: 0x8408 (reversed 0x1021)
- Initial value: 0xFFFF
- Final XOR: 0xFFFF
- Right-shifting implementation
- Calculated over all frame bytes before CRC

Error Management and Recovery

Error Types

- Initialization Timeout

    - No devices responding error
    - Occurs during initial ATN/ACK sequence
    - Indicates no active devices on bus
- Device Command Timeout
    - Device <n> not responding error
    - Occurs during LISTEN, UNLISTEN, TALK, or UNTALK commands
    - Indicates specific device failure or absence
- Transaction Timeout
    - General timeout error
    - Occurs during data transfer or turnaround
    - Indicates communication failure during active transaction

Error Recovery Process

1. Error condition detected (>100μs timeout)
2. Both controller and responder:
    - Raise appropriate error type
    - Abort current transaction
    - Return to IDLE state

Implementation Benefits

- Error type indicates failure point
- Specific device identification when relevant
- Consistent recovery mechanism for all errors
- Enables targeted retry strategies

State Recovery

- All devices return to IDLE regardless of error type
- Higher layers can implement appropriate retry logic based on error type
- New transactions can begin immediately after timeout

Protocol Performance Comparison

Commodore Serial (IEC)

- Clock rate: ~1 bit/ms typical
- Byte transfer: ~8-10ms per byte
- Effective transfer rate: ~100-125 bytes/second

- Limited by software bit-banging and long settling times

ASB Protocol

- Timeout boundary: 100μs per handshake
- Minimum theoretical throughput: ~5KB/second
- Practical transfer rates:
  - ~2-3KB/second typical
  - Up to 10KB/second possible with optimized code
- Hardware-assisted handshaking via VIA
- 20-30x faster than Commodore serial typical case

Key Differences

- ASB uses hardware handshaking vs IEC software timing
- ASB transfers full bytes vs IEC bit-by-bit transfer
- ASB timeout is worst-case vs IEC being typical case
- ASB self-paces to device capabilities vs IEC fixed timing
- Both protocols support multiple devices but ASB allows higher device count (15 vs 8)

Bus Turnaround

Timing Characteristics

- Self-pacing through handshake acknowledgment
- No fixed timing requirement
- 100μs timeout as worst-case boundary
- Actual speed determined by device capabilities

Turnaround Sequence

1. Controller initiates turnaround
2. Waits for device acknowledgment
3. Device signals ready state
4. Transfer proceeds at negotiated pace

Implementation Benefits

- Natural speed matching between devices
- No artificial delays required
- Robust operation across different device speeds

- Consistent with overall protocol philosophy
- Same timeout boundary as other operations (100µs)

Performance Considerations

- Turnaround overhead minimized through self-pacing
- No need for fixed delay loops
- System automatically finds optimal timing
- Reliable operation without performance penalty