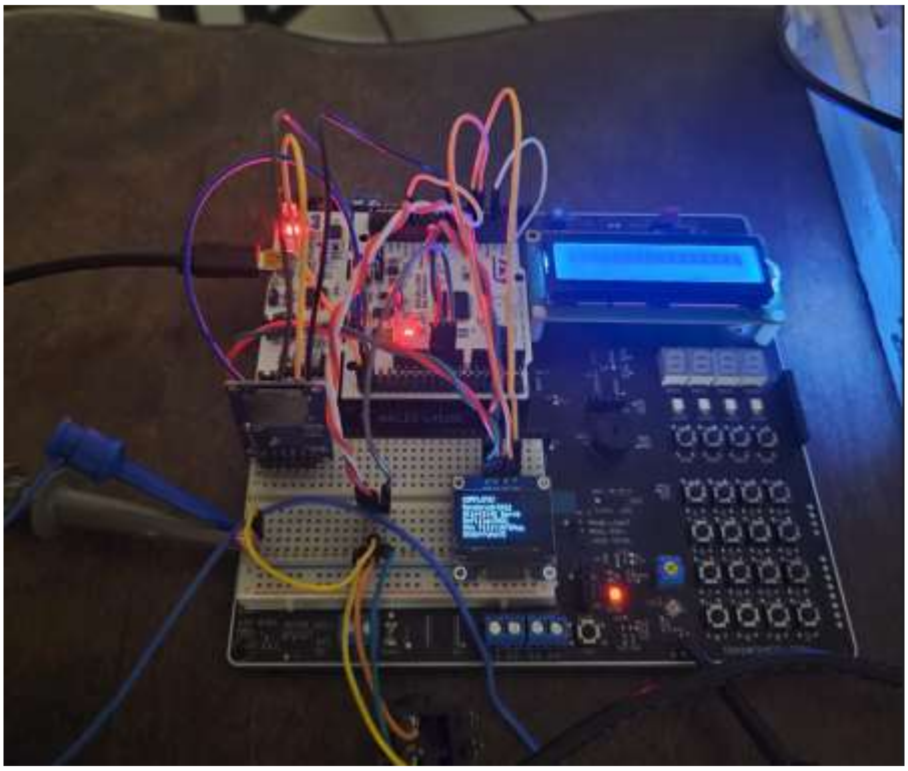
Bad Apple Video Player

Synchronized Audio/Video Playback on STM32L476RG

Technical Design Report

David Leathers

2025



Source Code: https://github.com/MidnightCastle/bad_apple_stm32.git

Abstract

This report documents the design and implementation of a synchronized audio/video player on the STM32L476RG microcontroller, capable of playing the iconic "Bad Apple" music video. The system achieves 30 FPS video playback on a 128x64 OLED display with 32 kHz stereo audio output through dual 12-bit DACs, all without an operating system.

Key technical contributions include an audio-master synchronization engine that maintains precise A/V alignment through automatic frame skip/repeat decisions, a triple-buffered display architecture eliminating visual tearing, and a custom minimal FAT32 implementation with contiguous file optimization achieving 10-16x throughput improvement over naive approaches.

The complete 3:39 video (6,566 frames, ~35MB) plays with zero audio underruns and imperceptible synchronization drift. This project demonstrates that sophisticated real-time multimedia applications are achievable on resource-constrained embedded platforms through careful architectural design and systematic optimization.

1. Introduction

1.1 Project Overview

This project implements a synchronized audio/video player on the STM32L476RG microcontroller, playing the iconic "Bad Apple!!" music video. The system demonstrates embedded systems techniques, including real-time multimedia processing, DMA-driven peripherals, and audio-video synchronization on a resource-constrained platform without an operating system.

The system reads a custom binary media file from an SD card via SPI, renders video frames to a 128x64 SSD1306 OLED display via I2C, and outputs synchronized stereo audio through the internal 12-bit DACs. This represents a significant embedded systems engineering challenge: achieving real-time multimedia playback with precise audio-video synchronization while maintaining deterministic behavior.

Demonstration video: https://youtu.be/AAY7CooOcy8

1.2 Applications and Significance

The techniques developed in this project have broad applications across embedded multimedia systems. The audio-master synchronization approach mirrors professional multimedia systems and demonstrates that complex real-time applications can be achieved on low-cost microcontrollers. Potential applications include portable media players, digital signage systems, industrial HMI displays with audio feedback, automotive infotainment interfaces, medical device displays requiring synchronized alerts, and IoT devices with audio-visual output capabilities.

"Bad Apple" has become a benchmark for demonstrating multimedia capabilities across diverse platforms - from graphing calculators to FPGA displays - making it an ideal test case for this embedded multimedia system.

1.3 Technical Achievements

The following key technical achievements were accomplished:

1. 30 FPS video playback with 1024-byte raw framebuffer rendering directly to SSD1306 native format
2. 32 kHz stereo audio via dual 12-bit DACs with circular DMA and half-transfer interrupts
3. Audio-master synchronization engine with automatic frame skip/repeat, maintaining ±2 frame drift tolerance
4. Triple-buffered display architecture with atomic swap operations, eliminating visual tearing
5. Custom minimal FAT32 filesystem implementation optimized for sequential media streaming
6. Contiguous file detection with multi-block CMD18 reads, achieving 10-16x speedup over single-block transfers
7. DWT cycle counter instrumentation for microsecond-precision performance analysis
8. Complete playback of 3:39 video (6,566 frames) with zero audio underruns

2. System Specifications

2.1 Hardware Specifications

The following table summarizes the key system specifications:

Parameter	Specification
Microcontroller	STM32L476RG (ARM Cortex-M4F, 80 MHz, 1MB Flash, 128KB SRAM)
Display	SSD1306 OLED, 128x64 pixels, I2C interface @ 400 kHz FM+
Video Format	30 FPS, 1024-byte raw framebuffer per frame (1-bit monochrome)
Audio Format	32 kHz, 16-bit signed PCM, stereo interleaved
Audio Output	Dual 12-bit DAC (PA4 Left, PA5 Right) with DMA
Storage Interface	SD Card via SPI3 (312 kHz init, 10 MHz data transfer)
Filesystem	FAT32 (custom minimal read-only implementation)
Synchronization	Audio-master clock with ±2 frame drift tolerance (±66.67ms)
Media File	BADAPPLE.BIN custom binary format (~35MB)
Media Duration	3:39 (219 seconds, 6,566 frames)
Power Consumption	<100mW during playback
SRAM Usage	~38KB of 128KB available (30%)

2.2 System Architecture

The following diagram illustrates the system architecture, showing data flow between major components:

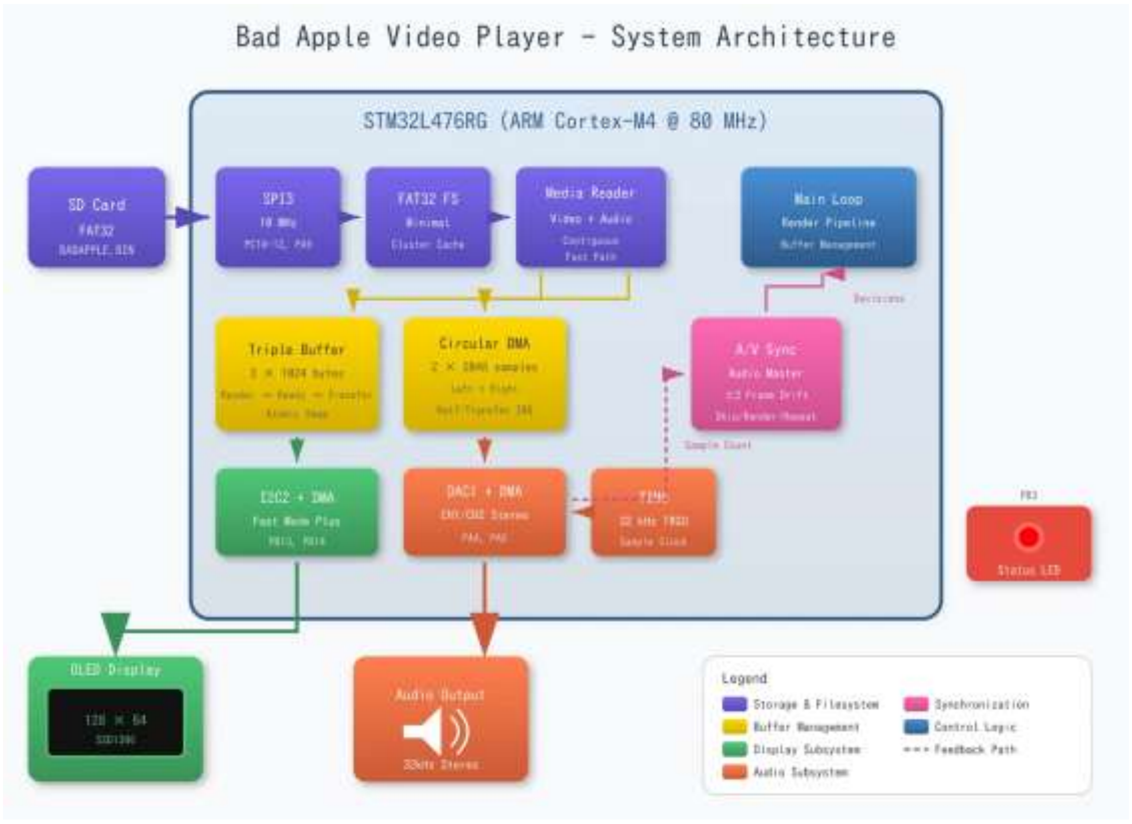


Figure 2.1: Bad Apple Video Player System Architecture

2.3 Hardware Connections

The hardware interface connections between the STM32L476RG and external peripherals:

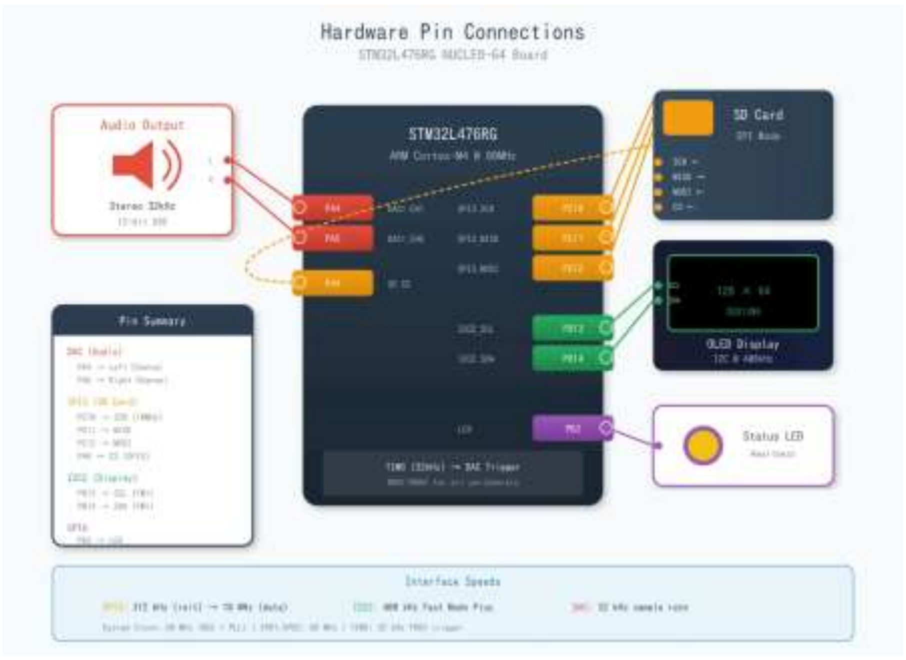


Figure 2.2: STM32L476RG Hardware Pin Connections

Interface	Type	Pins	Function
OLED Display	I2C2	PB13 (SCL), PB14 (SDA)	Video output @ 400 kHz FM+
SD Card	SPI3	PC10 (SCK), PC11 (MISO), PC12 (MOSI)	Media storage @ 10 MHz
SD Card CS	GPIO	PA9	SPI chip select (active low)
Audio Left	DAC1_CH1	PA4	Left channel analog output
Audio Right	DAC1_CH2	PA5	Right channel analog output
Status LED	GPIO	PB3	Heartbeat indicator (500ms toggle)

2.4 Operational Overview

Upon power-up, the system executes a carefully sequenced initialization: HAL library initialization, system clock configuration (MSI + PLL to 80 MHz), SSD1306 OLED initialization, SD card initialization at low speed (312 kHz) then high speed (10 MHz), FAT32 filesystem mounting, and media file header parsing.

During playback, the main loop continuously services audio buffer refill requests triggered by DMA half-transfer interrupts, queries the A/V sync module for frame decisions, and initiates display DMA transfers for completed frames. The audio-master architecture ensures video adapts to maintain synchronization with the continuous audio stream.

3. Software Architecture

3.1 Module Organization

The software follows a layered architecture with clear separation of concerns:

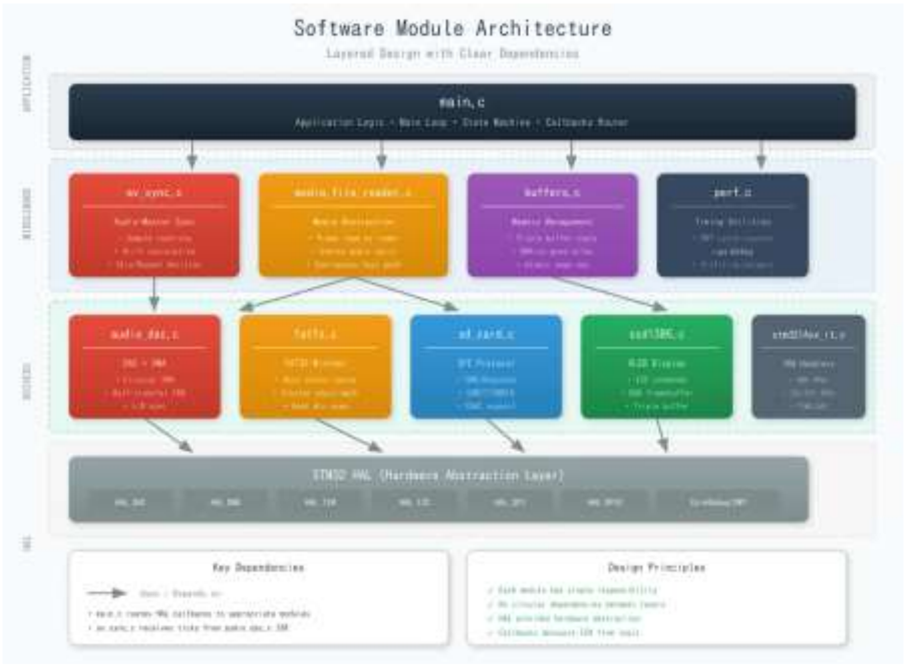


Figure 3.1: Software Module Architecture (Layered Design)

Module	Layer	Responsibility
main.c	Application	System initialization, peripheral configuration, main playback loop
av_sync.c	Middleware	Audio-video synchronization logic, sample counting, drift calculation
media_file_reader.c	Middleware	Media file parsing, frame/audio extraction, contiguous optimization
buffers.c	Middleware	Static buffer allocation, triple-buffer state machine, atomic swaps
audio_dac.c	Driver	Stereo DAC driver with circular DMA, buffer management
ssd1306.c	Driver	OLED display driver, I2C communication, DMA transfers
sd_card.c	Driver	SD card SPI driver, CMD17/CMD18 block reads, SDHC support
fatfs.c	Driver	Minimal FAT32 filesystem, boot sector parsing, cluster traversal

3.2 Initialization Sequence

The initialization follows a strict dependency order:

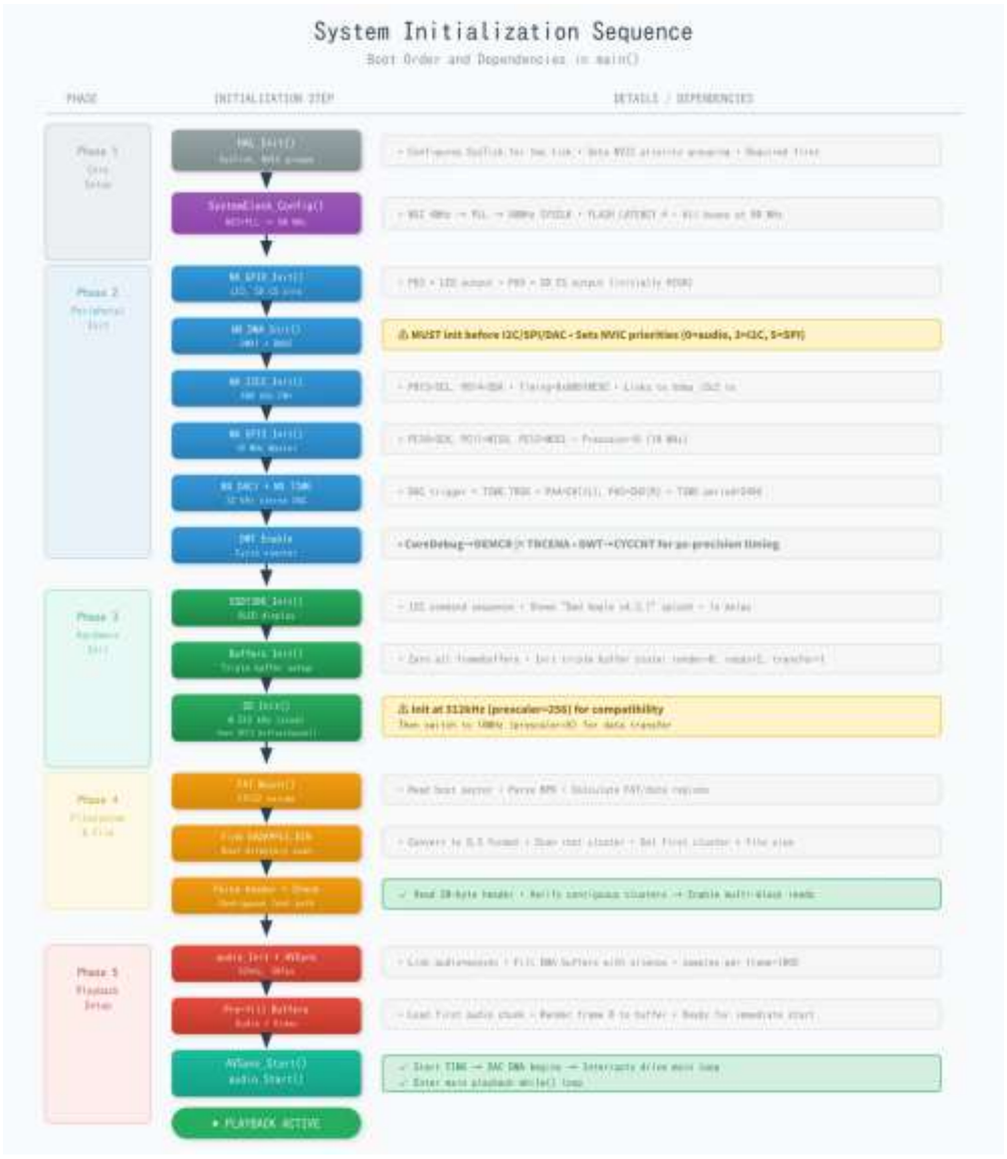


Figure 3.2: System Initialization Sequence

3.3 Clock Configuration

The system clock achieves 80 MHz using the internal MSI oscillator with PLL:

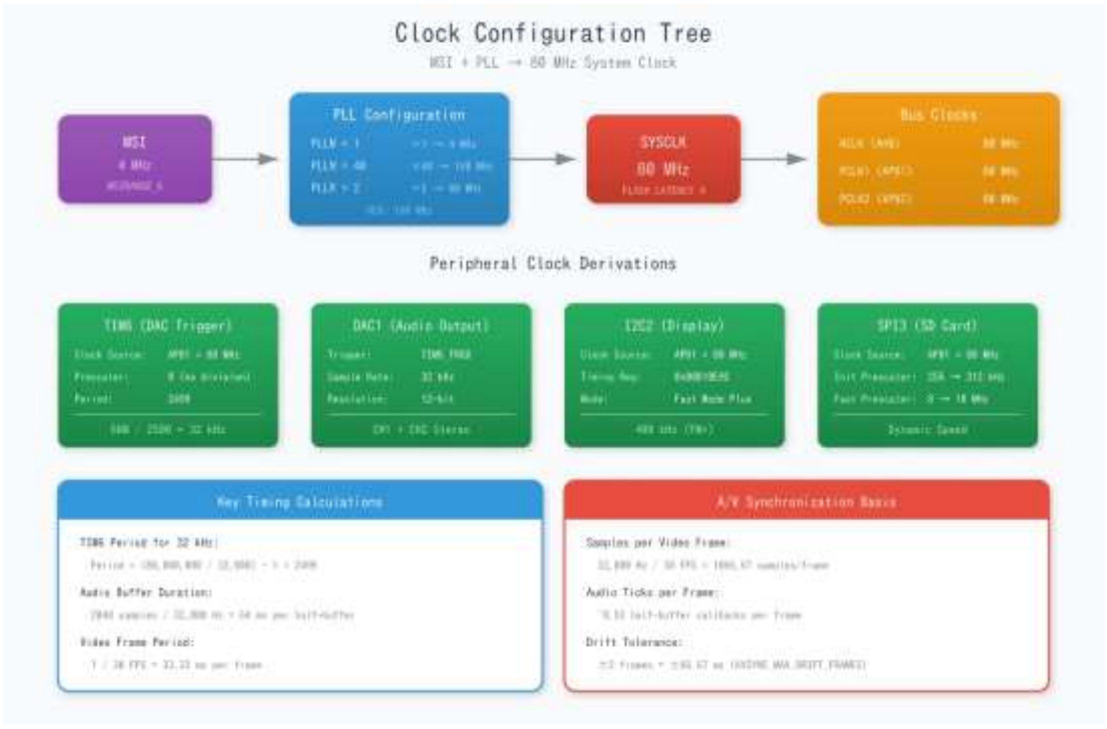


Figure 3.3: Clock Configuration Tree (MSI + PLL => 80 MHz)

The MSI oscillator at 4 MHz (MSIRANGE_6) feeds the PLL with PLLM=1, PLLN=40 (160 MHz VCO), and PLLR=2 (80 MHz SYSCLK). Flash latency is set to 4 wait states for 80 MHz at VOS1 voltage scaling.

3.4 Main Playback Loop

The main loop implements core real-time behavior with audio buffer refilling performed twice per iteration:

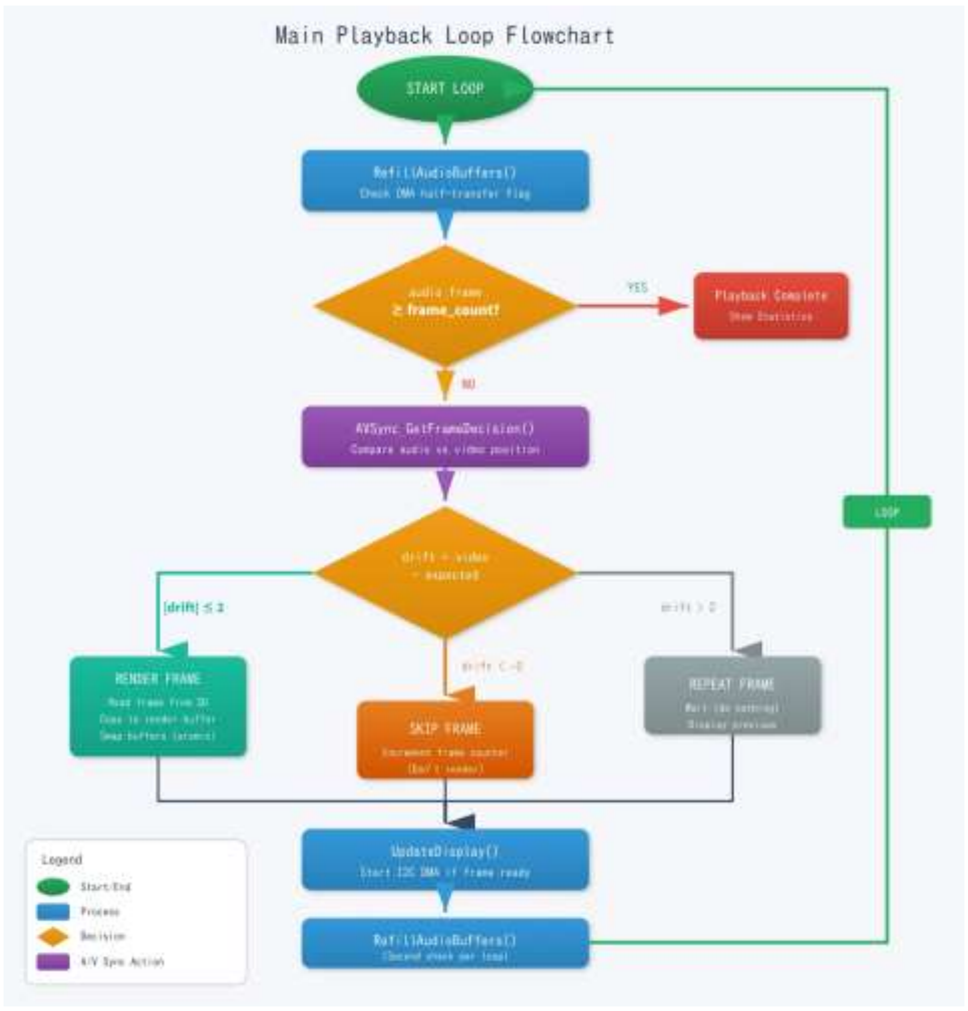


Figure 3.4: Main Playback Loop Flowchart

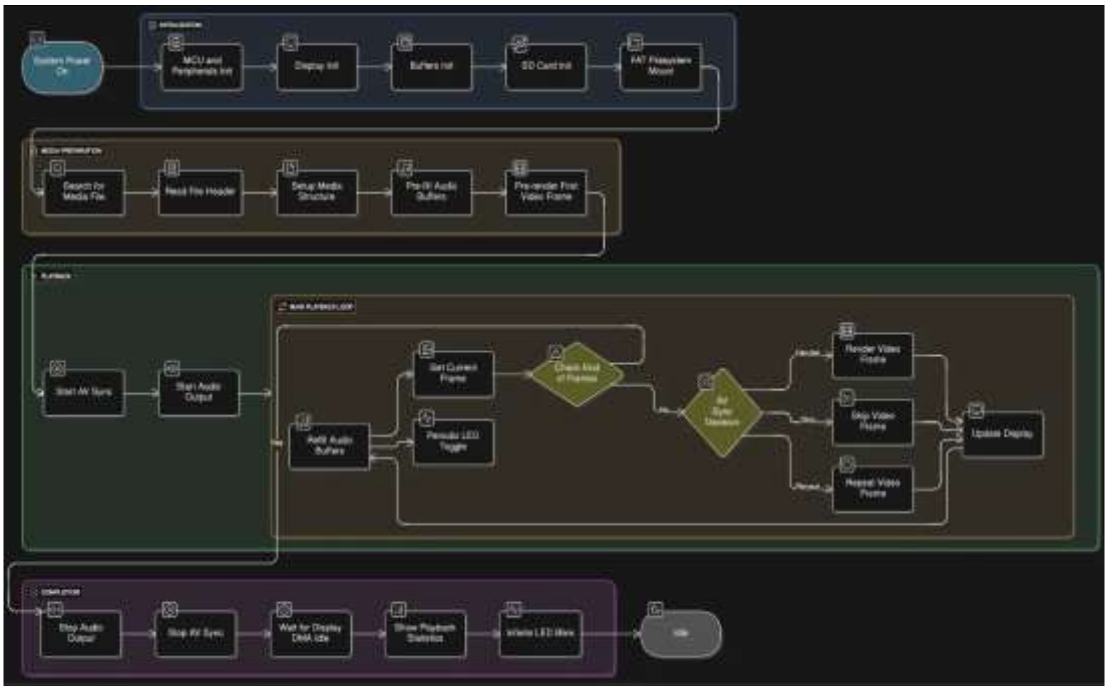


Figure 3.5: Main Playback Loop State Machine

Pseudocode:

```
WHILE playback not complete:
    RefillAudioBuffers()  // Service DMA half-transfer requests
    audio_frame = AVSync_GetCurrentFrame()
    IF audio_frame >= frame_count THEN exit loop
    decision = AVSync_GetFrameDecision()
    SWITCH decision:
        RENDER: Read frame from SD, copy to buffer, swap
        SKIP: Increment frame counter without reading
        REPEAT: Wait (display previous frame again)
    UpdateDisplay()  // Start I2C DMA if ready
    RefillAudioBuffers()  // Second check for responsiveness
```

4. Peripheral Configuration

4.1 Timer Configuration (TIM6)

TIM6 generates the audio sample clock, triggering both DAC channels at precisely 32 kHz:

Parameter	Value	Notes
Clock Source	APB1 (80 MHz)	No prescaler on APB1
Prescaler	0	No division, full 80 MHz input
Period (ARR)	2499	(80,000,000 / 32,000) - 1 = 2499
Actual Sample Rate	32.000 kHz	80 MHz / 2500 = exactly 32 kHz
Trigger Output	TIM_TRGO_UPDATE	Triggers DAC on each period completion

4.2 DAC Configuration

Both DAC channels are configured identically for stereo output (12-bit, 4096 levels, 2048 = silence):

Parameter	Value
Resolution	12-bit, right-aligned (DAC_ALIGN_12B_R)
Trigger Source	TIM6_TRGO (hardware synchronized)
DMA Mode	Circular with half/complete transfer interrupts
Buffer Size	4096 samples per channel (2048 per half)
Buffer Latency	64ms per half-buffer at 32 kHz
Pins	PA4 (CH1/Left), PA5 (CH2/Right)

4.3 DMA Channel Allocation

DMA channels are allocated with priority levels ensuring audio never starves:

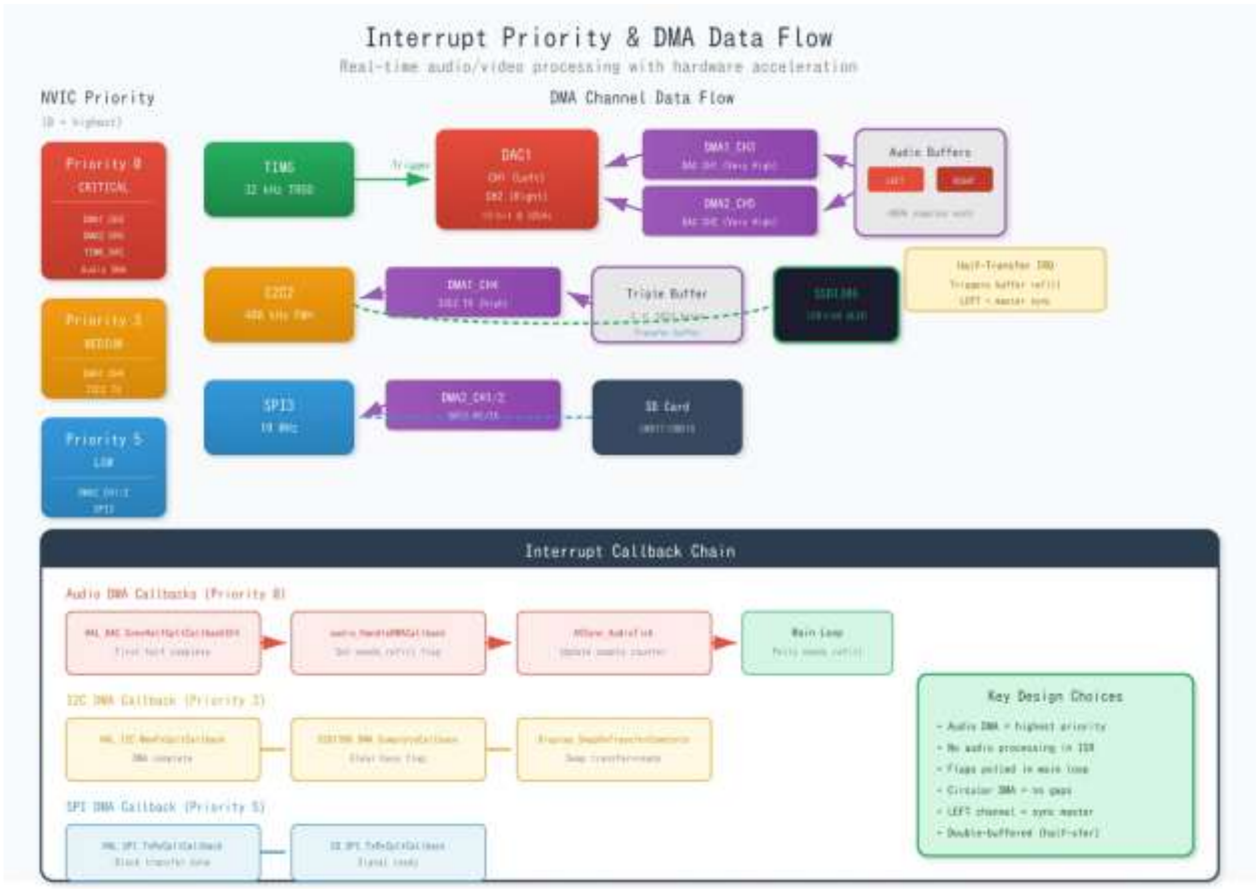


Figure 4.1: Interrupt Priority and DMA Data Flow

Channel	Peripheral	Priority	Mode	Purpose
DMA1_CH3	DAC CH1 (Left)	0 (Highest)	Circular	Left audio output, master sync
DMA2_CH5	DAC CH2 (Right)	0 (Highest)	Circular	Right audio output
DMA1_CH4	I2C2 TX	3 (Medium)	Normal	Display framebuffer transfer
DMA2_CH1	SPI3 RX	5 (Low)	Normal	SD card data receive

4.4 Memory Utilization

Static buffer allocation ensures deterministic memory usage with no heap fragmentation:

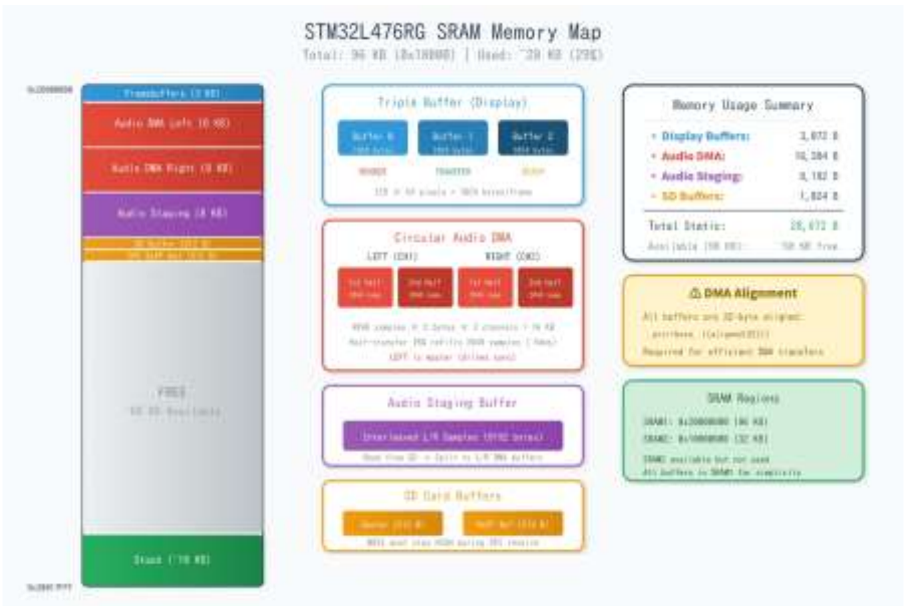


Figure 4.2: STM32L476RG SRAM Memory Map

Buffer	Size	Count	Total	Purpose
Display framebuffers	1024	3	3,072	Triple buffering for tear-free display
Audio DMA Left	4096	1	8,192	Left channel circular DMA
Audio DMA Right	4096	1	8,192	Right channel circular DMA
Audio staging	8192	1	8,192	Interleaved read buffer
SD sector buffer	512	1	512	FAT operations
Stack	~10,000	1	~10,000	Function calls, locals
Total Used			**~38,672**	30% of 128KB SRAM1

All DMA buffers are 32-byte aligned using __attribute__((aligned(32))) for efficient AHB bus transfers.

5. Key Algorithms

5.1 Audio-Video Synchronization

The A/V sync module implements an audio-master approach where audio playback is continuous and video adapts:

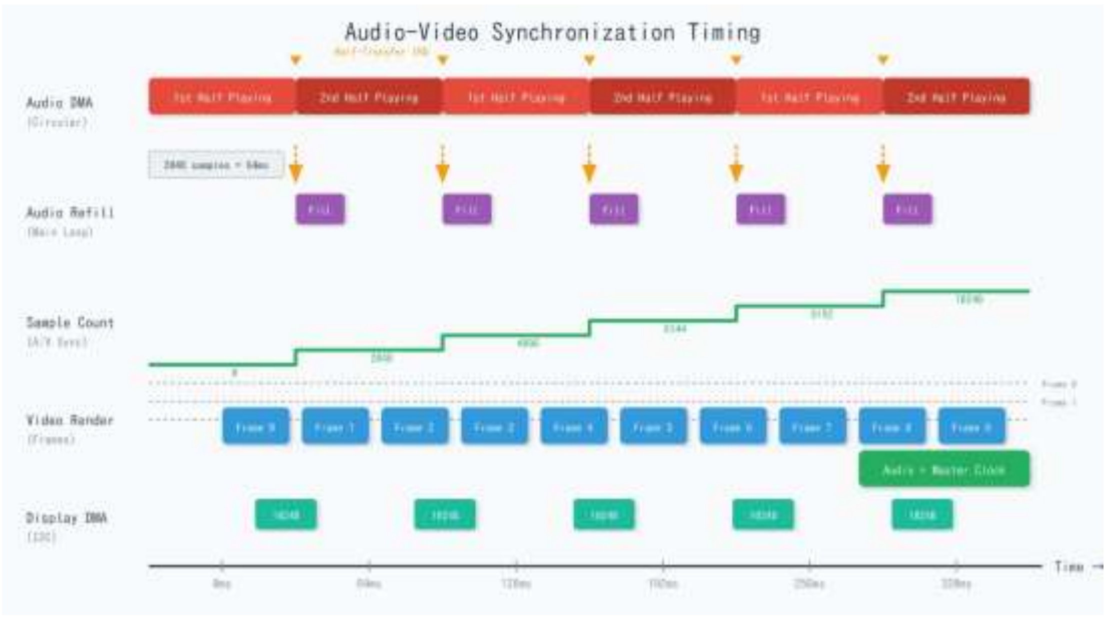


Figure 5.1: Audio-Video Synchronization Timing

Algorithm:

```
FUNCTION AVSync_GetFrameDecision(sync):
    samples = sync.audio_samples_played  // Updated in ISR
    expected_frame = samples / samples_per_frame  // 32000/30 = 1066.67
    drift = video_frames_rendered - expected_frame
    IF drift < -2 THEN RETURN SKIP_FRAME    // Video behind
    IF drift > 2 THEN RETURN REPEAT_FRAME   // Video ahead
    RETURN RENDER_FRAME                      // In sync
```

At 32 kHz and 30 FPS, each frame corresponds to ~1066.67 samples. The ±2 frame tolerance (±66.67ms) is imperceptible to human perception.

5.2 Triple Buffer State Machine

Triple buffering prevents display tearing by ensuring DMA never reads from a buffer being written:

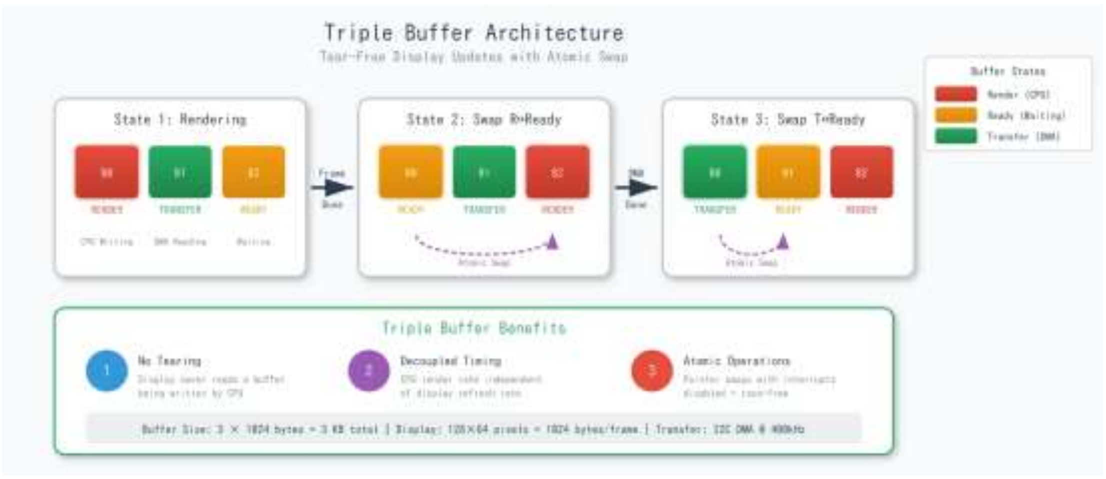


Figure 5.2: Triple Buffer Architecture with Atomic Swap

Buffer States:

- **Render buffer:** CPU actively writing new frame data
- **Ready buffer:** Contains completed frame awaiting transfer
- **Transfer buffer:** DMA actively sending to display

5.3 Contiguous File Fast Path

The contiguous file optimization bypasses FAT cluster chain traversal:

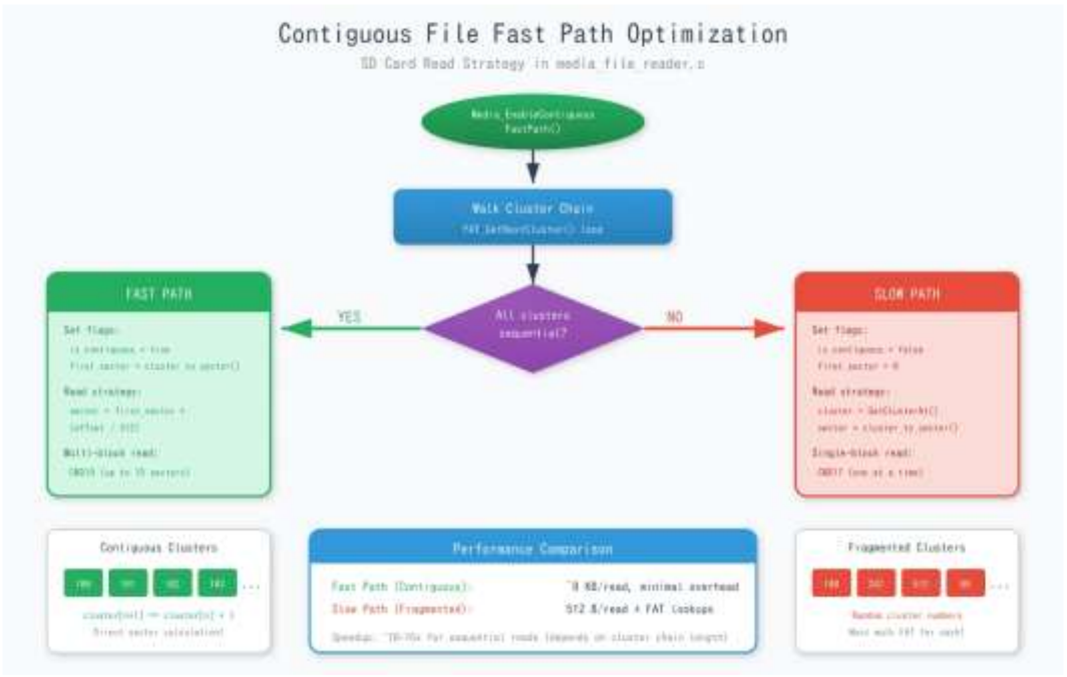


Figure 5.3: Contiguous File Fast Path Optimization

When enabled, sector addresses are calculated directly: sector = first_sector + (byte_offset / 512). Multi-block CMD18 reads transfer up to 16 sectors (8KB) per command, achieving 10-16x speedup over single-block reads.

5.4 Media File Format

The BADAPPLE.BIN file uses a custom binary format optimized for sequential streaming:

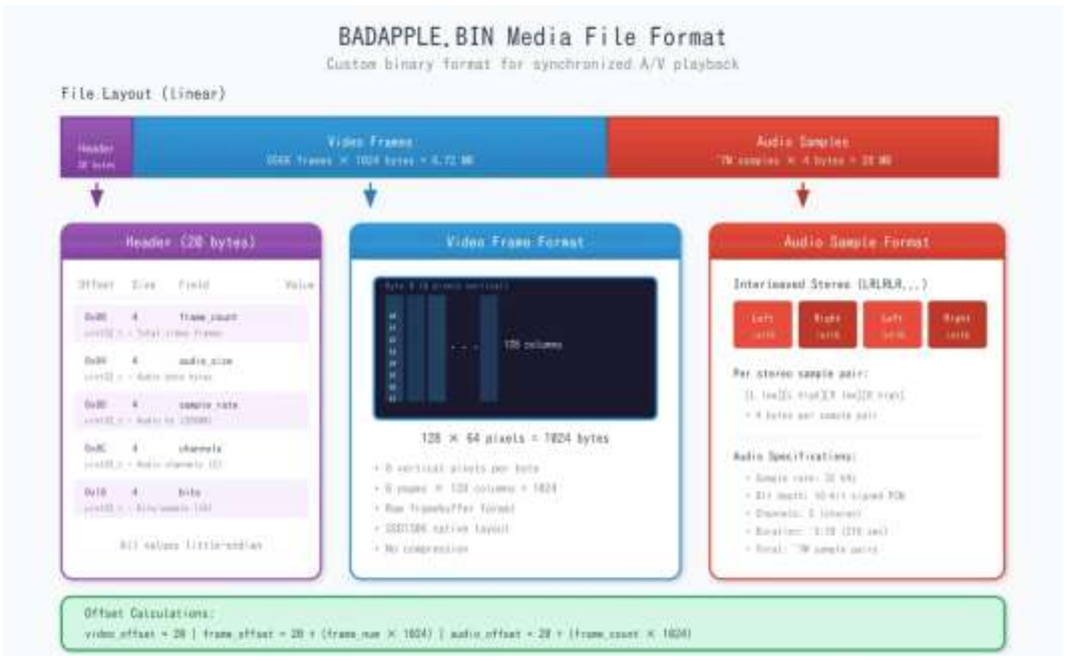
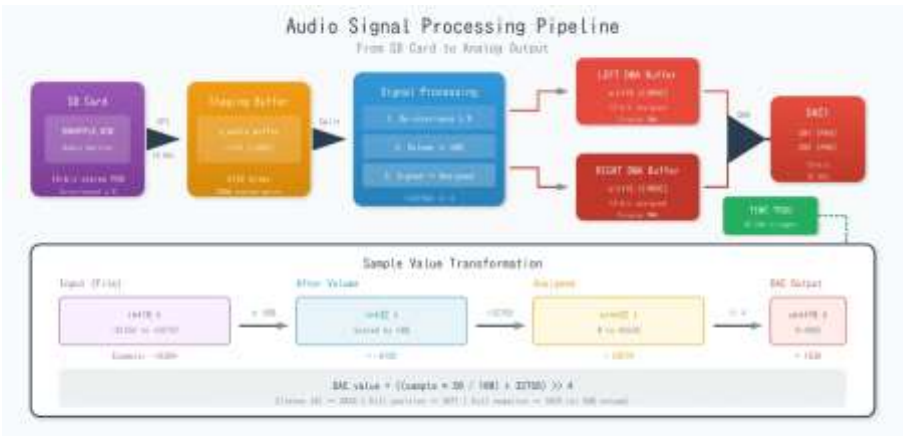


Figure 5.4: BADAPPLE.BIN Media File Format

Offset	Size	Field	Description
0x00	4 bytes	frame_count	Total video frames (6,566)
0x04	4 bytes	audio_size	Audio data size in bytes
0x08	4 bytes	sample_rate	Audio sample rate (32000)
0x0C	4 bytes	channels	Audio channels (2 for stereo)
0x10	4 bytes	bits_per_sample	Bits per sample (16)
0x14	variable	video_frames	6,566 x 1024 bytes = ~6.72 MB
after video	variable	audio_data	~7M samples x 4 bytes = ~28 MB

5.5 Audio Signal Processing

The audio pipeline converts 16-bit signed stereo to 12-bit unsigned DAC values:



Audio Signal Processing Pipeline
From SD Card to Analog Output
SD Card
Staging Buffer
Signal Processing
LEFT DMA Buffer
RIGHT DMA Buffer
DAC1
TIM7 TRGO
Sample Value Transformation
DAC value = ((sample × 50 / 100) + 32768) >> 4

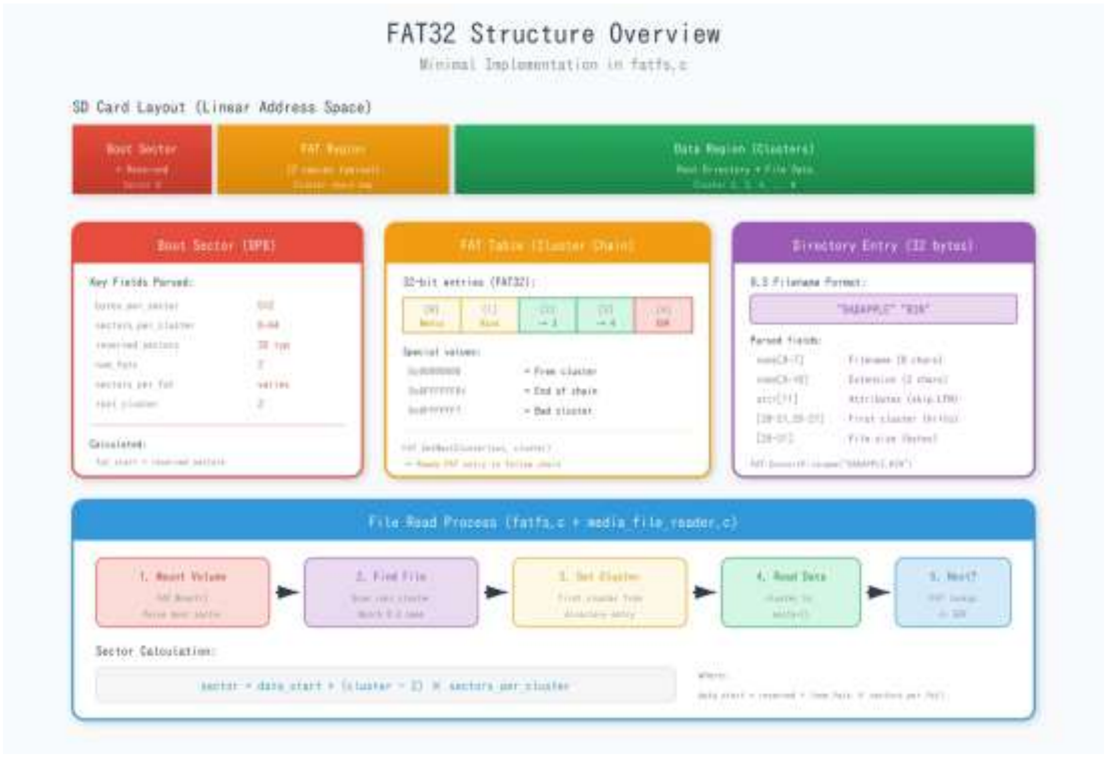
Figure 5.5: Audio Signal Processing Pipeline

Sample Transformation:

```
Input: int16_t sample (-32768 to +32767)
Step 1: scaled = sample * 50 / 100      // Volume attenuation
Step 2: unsigned = scaled + 32768       // Shift to 0-65535
Step 3: dac_value = unsigned >> 4       // Scale to 0-4095
Output: uint16_t dac_value (0 to 4095)
```

5.6 FAT32 and SD Card Implementation

The minimal FAT32 implementation supports read-only access to files in the root directory:



FAT32 Structure Overview
Minimal Implementation in fatfs.c
SD Card Layout (Linear Address Space)
Boot Sector
FAT Region
Data Region (Clusters)
Boot Sector (BPB)
FAT Table (Cluster Chain)
Directory Entry (32 bytes)
"BADAPPLE" "BIN"
File Read Process (fatfs.c + media_file_reader.c)
1. Mount Volume
2. Find File
3. Get Cluster
4. Read Data
5. Next?
Sector Calculation:
sector = data_start + (cluster - 2) × sectors_per_cluster

Figure 5.6: FAT32 Structure Overview

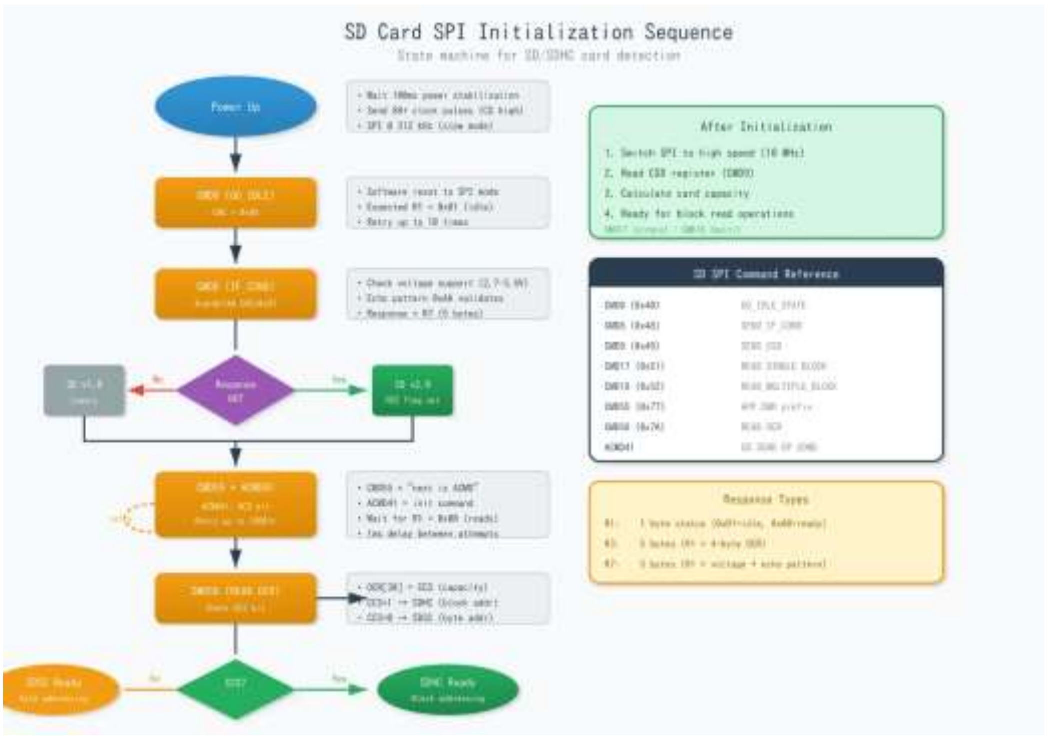

SD Card SPI Initialization Sequence
State machine for SD/SDHC card detection
Power Up
After Initialization
1. Switch SPI to high speed (10 MHz)
2. Read CSD register (CMD9)
3. Calculate card capacity
4. Ready for block read operations
SD SPI Command Reference
Response Types
SDSC Ready
SDHC Ready

Figure 5.7: SD Card SPI Initialization Sequence

5.7 Video Frame Data Path

The complete data path from SD card to OLED display is optimized for minimal CPU involvement:

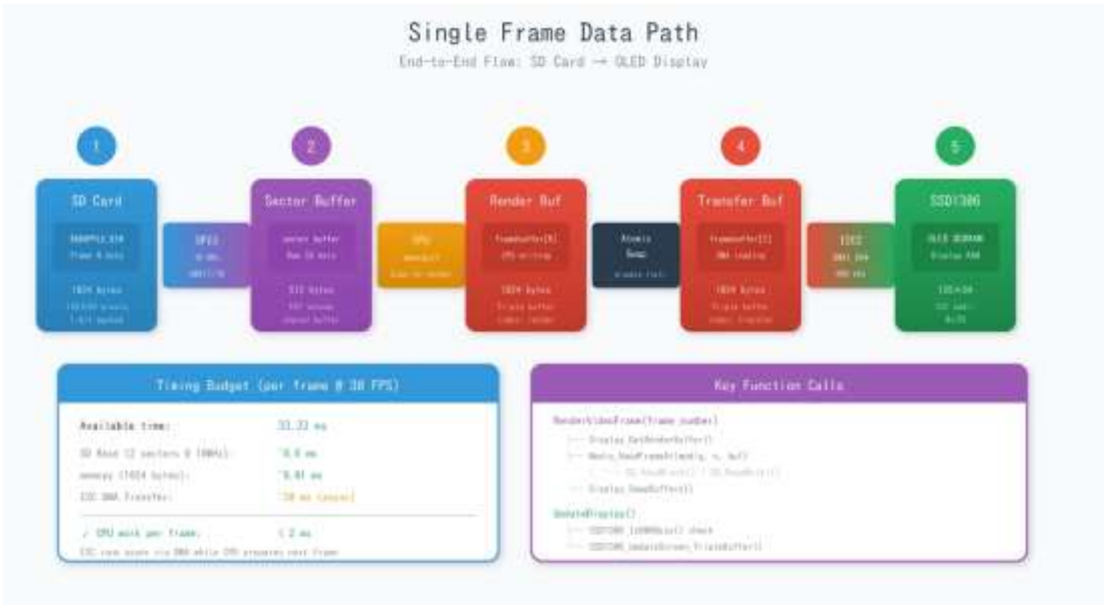


Figure 5.8: Single Frame Data Path (SD Card to OLED)

Timing budget at 30 FPS: 33.33ms available. SD read (2 sectors @ 10 MHz): ~0.8ms. memcpy: ~0.01ms. I2C DMA: ~20ms (asynchronous). Total CPU work: <2ms per frame.

6. Challenges and Solutions

6.1 Engineering Challenges

Challenge 1: Audio/Video Drift

Without synchronization, video would drift from audio due to accumulated timing errors. Solution: Audio-master clock architecture where the audio sample count serves as the authoritative timeline. Video adapts via automatic frame skip/repeat, maintaining ±2 frame tolerance.

Challenge 2: SD Card Throughput

Reading interleaved data required optimization beyond naive single-block reads. Solution: Contiguous file detection bypasses FAT traversal; multi-block CMD18 reads transfer up to 8KB per command.

Challenge 3: Display Tearing

Single-buffered display caused visible tearing. Solution: Triple-buffer architecture with atomic pointer swaps and interrupts disabled during transitions.

Challenge 4: Audio Buffer Underruns

Insufficient buffering caused glitches during SD latency spikes. Solution: 2048-sample double-buffered circular DMA with half-transfer interrupts provides 64ms buffer depth.

Challenge 5: Stereo Channel Synchronization

Independent DMA channels could drift apart. Solution: Left channel designated as "master"; right channel's buffer position synchronized in each ISR callback.

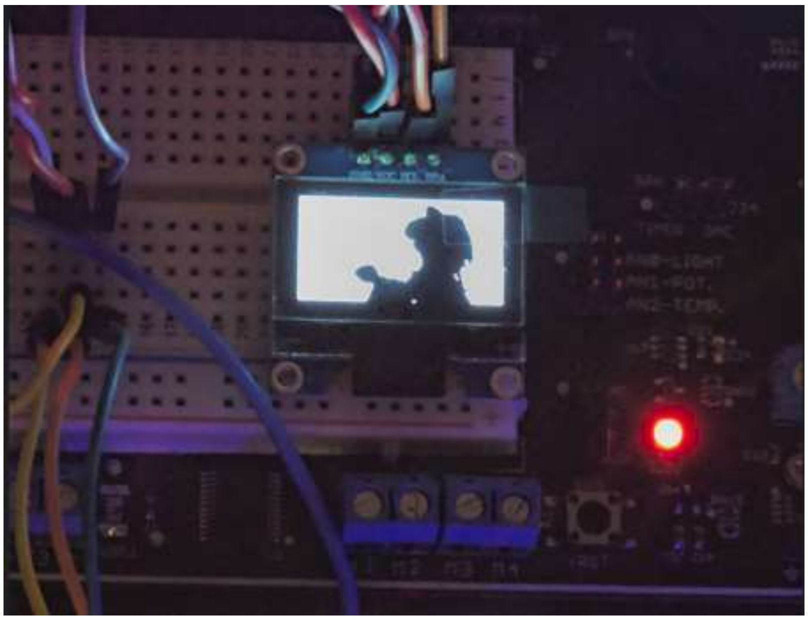


Figure 6.1: OLED display with working video

6.2 Debugging Case Studies

Development revealed several significant challenges requiring systematic debugging:

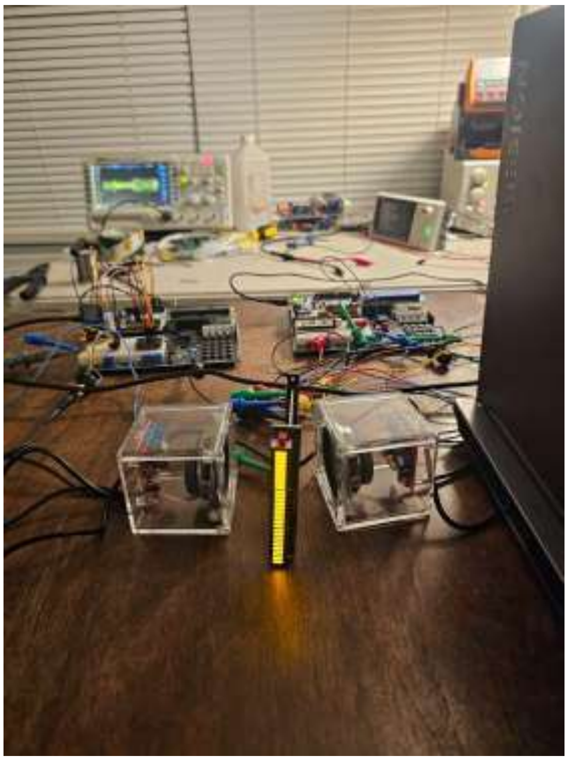
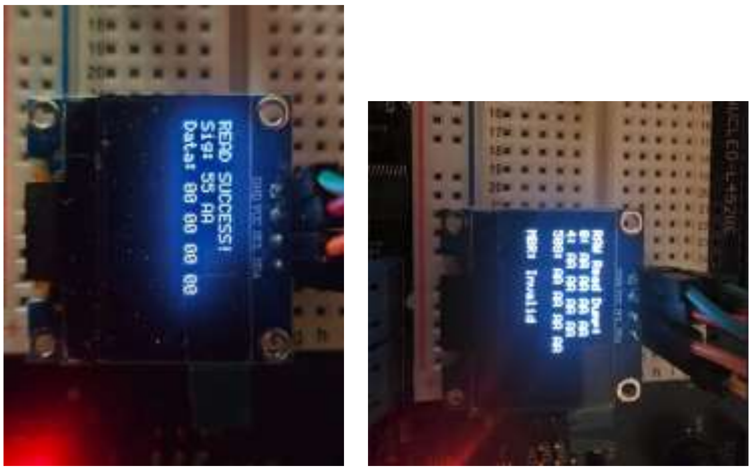


Figure 6.2: Development hardware configuration

SPI MOSI Line Protocol Bug

All sector reads returned 0xAA bytes. Root cause: HAL_SPI_Receive() transmits 0x00 on MOSI, but SD protocol requires 0xFF during reads. Solution: Use HAL_SPI_TransmitReceive() with 0xFF dummy buffer.



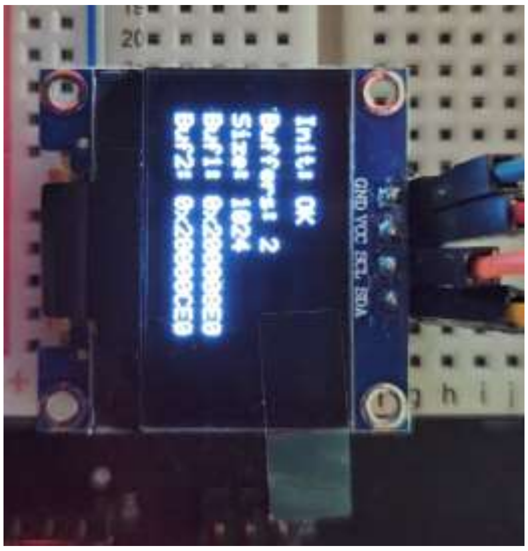
```
READ SUCCESS!
Sig: 55 AA
Data: 00 00 00 00
```

```
RAW Read Dump
0: AA AA AA AA
4: AA AA AA AA
508: AA AA AA AA
MBR Invalid
```

Figure 6.3: 0xAA pattern (left) vs valid MBR signature (right)

Framebuffer State Machine Bug

System hung after one frame ("Timeout at: 1"). Root cause: Blocking I2C transfers don't fire DMA callbacks, leaving transfer_in_progress set. Solution: Add explicit GFB_NotifyTransferComplete() calls after blocking transfers.

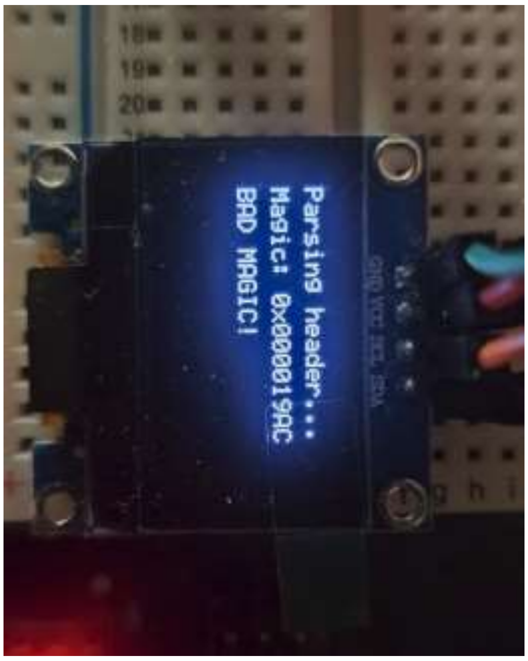


```
Init: OK
Buffers: 2
Size: 1024
Buf1: 0x200008E0
Buf2: 0x20000CE0
```

Figure 6.4: Framebuffer initialization diagnostic

Header Parsing Bug

"BAD MAGIC!" error—read 0x0000019AC instead of 0x42414441. Root cause: ARM compiler padding in C struct vs packed Python encoder. Solution: Apply __attribute__((packed)) to header structure.



```
Parsing header...
Magic: 0x000019AC
BAD MAGIC!
```

Figure 6.5: Header parsing failure diagnostic

Debugging Methodology Summary

Challenge	Debug Technique	Key Evidence
SPI MOSI Bug	Comparative testing (TxRx vs Rx-only)	0xAA pattern vs valid 55 AA signature
Framebuffer State	State machine analysis	"Timeout at: 1" with valid addresses
Header Parsing	Hex dump, sizeof() verification	Magic 0x0000019AC vs 0x42414441
Audio DMA	Extended stress testing	3,423 callbacks, 0 underruns

Note: All debugging used on-device OLED diagnostics—no UART or hardware debugger was available.

7. Testing and Results

7.1 Test Procedures

The system was validated through comprehensive multi-phase testing:

1. **Component Unit Testing:** Each driver module tested independently before integration.
2. **Audio-Only Playback:** Continuous playback verified with 0 underruns over full duration.
3. **Video-Only Playback:** Frame timing confirmed at consistent ~33ms periods.
4. **Synchronized Playback:** A/V sync validated within human perception threshold (~100ms).
5. **Stress Testing:** Multiple complete playbacks with consistent statistics.
6. **Performance Profiling:** DWT instrumentation verified <2ms worst-case audio fill time.

7.2 Playback Statistics

Metric	Typical Value	Acceptable Range
Rendered frames	~6,566 (100%)	6,500-6,566
Skipped frames	<10	<50
Repeated frames	<50	<100
Audio refills	~3,421	3,300-3,500
Max audio fill time	<2,000 us	<10,000 us
Audio underruns	**0**	0
Total playback time	219 seconds	218-220 seconds

7.3 Limitations

- **No User Interface:** Playback begins automatically; no pause/seek functionality.
- **Single File Only:** Hardcoded filename (BADAPPLE.BIN).
- **No Compression:** Raw audio/video (~35MB file size).
- **Fixed Configuration:** Sample rate, resolution, and frame rate fixed at compile time.

7.4 Lessons Learned

- **DMA Priority Matters:** Audio must have highest priority; video tolerates frame drops.
- **Buffer Sizing Trade-offs:** 64ms half-buffer balances responsiveness with reliability.
- **SD Card Init is Fragile:** Slow-speed init and retry logic essential for compatibility.
- **File Contiguity Critical:** Fresh format ensures contiguous allocation.
- **Instrumentation Essential:** DWT cycle counter identified bottlenecks.

7.5 Future Work

Potential enhancements for future development:

- Add user controls (pause/resume, seek, volume) via buttons or rotary encoder
- Implement file browser for multi-file playback support
- Add ADPCM audio compression for ~4x file size reduction
- Implement RLE or delta video compression for suitable content
- Port to color OLED/TFT display with SPI for higher bandwidth
- Port to FreeRTOS for cleaner task separation

8. Building and Reproducing

8.1 Prerequisites

Firmware Development:

- STM32CubeIDE 1.13+ (or ARM GCC 12.2+ with Makefile)
- STM32CubeMX 6.9+ (for peripheral configuration)
- ST-Link V2.1 debugger/programmer (included on Nucleo board)

Media Processing:

- Python 3.10+
- OpenCV (opencv-python >= 4.5.0)
- NumPy (>= 1.20.0)
- Pillow (>= 8.0.0)
- FFmpeg (for audio extraction)

8.2 Media File Generation

The Python tools in the tools/ directory convert source video to the custom binary format:

```
# Install dependencies
pip install opencv-python numpy Pillow

# Process video (converts to 128x64, 30 FPS, SSD1306 format)
python tools/process_video.py input.mp4 video.bin

# Extract audio (32 kHz stereo PCM)
python tools/process_audio.py input.mp4 audio.bin

# Combine into final media file
python tools/combine_files.py video.bin audio.bin BADAPPLE.BIN

# Or use the all-in-one script
python tools/process_all.py input.mp4 BADAPPLE.BIN
```

8.3 Firmware Build

```
# Clone repository
git clone https://github.com/<your-username>/bad-apple-stm32.git
cd bad-apple-stm32

# Open in STM32CubeIDE and build, or:
make all

# Flash to target
make flash
# Or use ST-Link Utility / STM32CubeProgrammer
```

8.4 SD Card Preparation

1. Format SD card as FAT32 (ensure fresh format for contiguous allocation)
2. Copy BADAPPLE.BIN to root directory as the first file
3. Insert SD card and power on—playback begins automatically

9. Source Code Reference

Complete source code is available: https://github.com/MidnightCastle/bad_apple_stm32.git

File	Lines	Description
main.c	482	System initialization, peripheral config, main playback loop
audio_dac.c/h	220	Stereo DAC driver with circular DMA
av_sync.c/h	140	Audio-video synchronization engine
buffers.c/h	136	Triple-buffer state machine
ssd1306.c/h	387	OLED display driver with DMA
sd_card.c/h	402	SD card SPI driver
fatfs.c/h	244	Minimal FAT32 filesystem
media_file_reader.c/h	334	Media file parsing
perf.c/h	51	DWT cycle counter utilities

Total: ~3,300 lines of application code (excluding HAL library and CMSIS headers).

9.1 Python Media Tools

File	Description
process_video.py	Converts video to SSD1306 vertical page format (128x64, 30 FPS)
process_audio.py	Extracts audio via FFmpeg (32 kHz stereo PCM)
combine_files.py	Combines video/audio into binary format with 20-byte header
process_all.py	Pipeline orchestrator for one-command processing
analyze_file.py	Validation tool with frame/audio sampling

10. Bill of Materials

Component	Part Number	Qty	Notes
Microcontroller Board	NUCLEO-L476RG	1	STM32L476RG Nucleo-64
OLED Display	SSD1306 128x64	1	I2C, 0.96" diagonal
SD Card Module	Generic SPI breakout	1	3.3V compatible (~$3)
MicroSD Card	Any Class 10	1	FAT32, <= 32GB
Audio Jack (optional)	3.5mm stereo	1	For headphone output
Jumper Wires	Male-Female	~10	Breadboard connections

Total estimated cost: ~$25-35 USD (assuming Nucleo board available; otherwise add ~$15).

11. References

1. STMicroelectronics. *RM0351: STM32L47xxx Reference Manual*. Rev 9, 2021.
2. STMicroelectronics. *DS10198: STM32L476xx Datasheet*. Rev 7, 2020.
3. Solomon Systech. *SSD1306 Advance Information*. Rev 1.1, April 2008.
4. SD Association. *SD Specifications Part 1: Physical Layer Simplified Specification*. Version 8.00, 2020.
5. Microsoft. *FAT32 File System Specification*. Version 1.03, December 2000.
6. ARM Limited. *ARM Cortex-M4 Technical Reference Manual*. Revision r0p1, 2015.
7. Alstroemeria Records / Masayoshi Minoshima. "Bad Apple!! feat. nomico" (original composition by ZUN, Touhou Project). 2008.

Online Resources & Citations:

- STM32CubeL4 HAL Library Documentation: st.com/stm32cubel4
- ZUN, "Bad Apple!!," in *Lotus Land Story*. Team Shanghai Alice, 1998.