# ddd-capture-toolkit

## Overview

1. Main Purpose

The ddd-capture-toolkit is a specialised archival tool designed to assist VHS preservation enthusiasts in achieving frame-accurate audio/video synchronisation using Domesday Duplicator and Clockgen Lite audio capture hardware. It automates the process of capturing, aligning, and processing A/V data to produce synchronised archival formats.

2. Key Libraries and External Modules

• Python Packages:

• opencv-python: For video processing.

• pillow: For image processing.

• numpy: Crucial for data handling and math operations.

• System Tools:

• ffmpeg: For video creation and processing.

• sox: For audio generation and processing.

3. Main File or Starting Point of Execution

The main entry point for the application is the ddd\_main\_menu.py file, which provides an extensive menu-driven interface to access various functionalities of the project. The ddd\_clockgen\_sync.py file includes core functions for capturing and synchronising video and audio.

## Table of Contents - DdD Capture Toolkit

Main Entry Points

• ddd\_main\_menu.py - The primary application entry point providing a comprehensive menu system for all toolkit functionality.

Classes

#### Core Classes

• SharedTimecodeRobust (tools/timecode-generator/shared\_timecode\_robust.py) - Base class providing robust FSK timecode encoding and decoding with enhanced error correction for VHS quality signals.

• VHSTimecodeGenerator (tools/timecode-generator/vhs\_timecode\_generator.py) - Generates test patterns with frame-accurate timecode for VHS calibration using robust FSK encoding.

• VHSTimecodeBase (tools/timecode-generator/vhs\_timecode\_base.py) - Base class containing shared frame generation logic for VHS timecode systems.

• VHSPatternGenerator (tools/timecode-generator/vhs\_pattern\_generator.py) - Creates repeating 4-step test patterns specifically designed for VHS recording and validation.

• VHSTimecodeAnalyzer (tools/timecode-generator/vhs\_timecode\_analyzer.py) - Analyzes captured VHS timecode patterns to determine precise audio/video synchronisation offsets.

Configuration & Setup Functions

#### Config Management (config.py)

• get\_project\_root() - Returns the absolute path of the project root directory.

• load\_config() - Loads configuration settings from config.json with fallback to defaults.

• save\_config() - Saves configuration settings to config.json file.

• get\_capture\_directory() - Returns the configured capture directory as an absolute path, creating it if necessary.

• set\_capture\_directory() - Updates the capture directory setting in configuration.

• check\_disk\_space() - Verifies sufficient free disk space in the specified directory.

• get\_config\_summary() - Returns a formatted summary of current configuration settings.

#### Dependency Checking (check\_dependencies.py)

• check\_python\_version() - Verifies Python version meets minimum requirements (≥3.7).

• check\_python\_packages() - Validates that required Python packages are installed and compatible.

• check\_system\_commands() - Tests availability of external tools like sox, ffmpeg, and vhs-decode.

• check\_platform\_specific() - Verifies platform-specific tools and capabilities are available.

Core Capture & Sync Functions

#### Main Capture System (ddd\_clockgen\_sync.py)

• shared\_capture\_process() - Coordinates parallel video and audio capture with precise timing synchronisation.

• get\_temp\_folder() - Returns the project temp directory for calibration files.

• get\_capture\_folder() - Returns the configured user capture directory.

• get\_sox\_command() - Generates platform-specific SOX commands for audio recording.

• get\_alignment\_filename() - Creates timestamped filenames for alignment captures.

• get\_alignment\_duration() - Prompts user for calibration capture duration with recommendations.

• start\_capture\_and\_record() - Main capture workflow coordinating video and audio recording.

• perform\_av\_alignment() - Executes automated audio/video alignment calibration process.

• run\_vhs\_decode() - Processes RF captures using vhs-decode with PAL settings.

• run\_tbc\_video\_export() - Exports TBC files to FFV1 video format for verification.

• analyze\_alignment\_with\_tbc() - Analyses alignment between audio captures and TBC timing data.

Menu System Functions (ddd\_main\_menu.py)

#### Interface Functions

• clear\_screen() - Clears the terminal display.

• display\_header() - Shows the project title and branding.

• display\_main\_menu() - Presents the primary menu options.

#### Video Creation Functions

• create\_sync\_test\_videos() - Menu for creating various sync test video formats.

• create\_calibration\_videos() - Generates 1-hour test videos with 1-second ON/OFF patterns.

• create\_vhs\_pattern\_generator() - Creates VHS patterns with 4-step cycles for validation.

• create\_belle\_nuit\_chart\_single() - Generates static test chart videos for hardware testing.

• create\_dvd\_isos() - Converts MP4 sync test videos to DVD-Video ISO format.

#### VHS Processing Functions

• display\_vhs\_decode\_menu() - Shows VHS decode workflow options.

• manual\_vhs\_decode() - Runs vhs-decode manually on selected RF files.

• manual\_tbc\_export() - Exports TBC files to video format manually.

• manual\_audio\_alignment() - Performs manual audio alignment with TBC timing data.

#### Calibration Functions

• display\_av\_calibration\_menu() - Shows audio/video calibration options.

• precision\_timecode\_capture() - Executes automated precision timecode-based calibration.

• capture\_vhs\_validation\_tape() - Validates existing VHS captures using enhanced detection.

• validate\_mp4\_timecode() - Tests MP4 timecode files using validation methods.

• manual\_calibration\_entry() - Allows manual entry of calibration delay values.

• validate\_calibration\_results() - Tests current calibration by measuring resulting offsets.

#### Workflow Functions

• capture\_new\_video() - Initiates new video capture workflow.

• run\_av\_alignment() - Performs audio/video alignment calibration.

• vhs\_audio\_alignment() - Wrapper for VHS audio alignment tools.

Test Pattern Analysis (analyze\_test\_pattern.py)

• analyze\_test\_pattern\_timing() - Analyses synchronised audio and video using multi-cycle detection with statistical measurements.

• find\_tone\_start\_time() - Detects the start time of 1kHz tone in audio files using amplitude threshold detection.

• find\_all\_audio\_tone\_starts() - Locates all audio tone cycle starts for comprehensive timing analysis.

• find\_all\_video\_pattern\_starts() - Identifies all video pattern cycle starts for correlation with audio.

Video Creation Tools

#### Sync Test Creation (tools/create\_sync\_test.py)

• create\_sync\_test\_video() - Creates precisely timed sync test videos with 1-second ON/OFF patterns for both PAL and NTSC formats.

• main() - Generates both PAL and NTSC versions of 1-hour sync test videos.

#### Belle Nuit Charts (tools/create\_belle\_nuit\_charts.py)

• create\_static\_chart() - Generates static test chart videos for equipment testing.

• create\_belle\_nuit\_video() - Creates long-duration test videos with continuous patterns.

• main() - Command-line interface for creating Belle Nuit test charts.

#### ISO Creation (tools/create\_iso\_from\_mp4.py)

• scan\_mp4\_files() - Scans directories for MP4 files suitable for DVD conversion.

• create\_dvd\_structure() - Creates proper VIDEO\_TS directory structure for DVD-Video format.

• create\_iso\_image() - Generates ISO disc images from DVD-Video structures.

• main() - Interactive interface for MP4 to DVD-Video ISO conversion.

Audio Processing & Alignment

#### VHS Audio Alignment (tools/audio-sync/vhs\_audio\_align.py)

• check\_dependencies() - Verifies availability of sox and mono runtime.

• find\_align\_tool() - Locates VhsDecodeAutoAudioAlign.exe tool.

• align\_audio() - Aligns VHS audio using the decode pipeline with TBC timing data.

• main() - Command-line interface for VHS audio alignment.

#### Audio Analysis (tools/simple\_audio\_analyzer.py)

• load\_audio\_with\_sox() - Loads audio files using SOX with format detection.

• detect\_test\_pattern\_pulses() - Identifies 1kHz tone pulses in audio streams.

• calculate\_pattern\_timing() - Measures timing intervals between detected pulses.

• analyze\_timing\_consistency() - Evaluates consistency of pulse timing for quality assessment.

• generate\_timing\_report() - Creates detailed reports of timing analysis results.

• main() - Comprehensive audio pattern analysis with statistical reporting.

Validation & Testing Tools

#### MP4 Timecode Validation (tools/validate\_mp4\_timecode.py)

• CycleAwareMp4Validator - Class for validating MP4 timecode with cycle structure awareness.

• main() - Command-line MP4 timecode validation with cycle detection.

#### Frequency Testing (tools/test\_frequency\_generation.py)

• generate\_test\_frequencies() - Creates test tones at specific frequencies for validation.

• test\_frequency\_detection() - Validates frequency detection algorithms.

• run\_frequency\_tests() - Comprehensive frequency generation and detection testing.

#### FSK Validation (tools/validate\_fsk\_timing.py)

• load\_and\_validate\_audio() - Loads audio files and validates FSK encoding.

• analyze\_fsk\_boundaries() - Examines FSK frequency boundaries for proper separation.

• validate\_timecode\_integrity() - Verifies timecode data integrity in FSK streams.

Debugging & Analysis Tools

#### Debug Scripts

• debug\_timing.py - main() - Analyses timing discrepancies in test patterns.

• debug\_spectrum.py - Provides spectral analysis for frequency validation.

• debug\_fsk\_frequencies.py - Debugs FSK frequency encoding and detection.

• debug\_bit\_detection.py - Analyses individual bit detection in FSK streams.

• debug\_checksum.py - Validates checksum calculations in timecode data.

#### Analysis Tools

• tools/sync\_test\_summary.py - Provides comprehensive summaries of sync test file status and metadata.

• tools/audio\_alignment\_analyzer.py - Advanced analysis of audio alignment quality and statistics.

Utility Functions

• analyze\_expected\_bits.py - main() - Analyses expected bit patterns in timecode streams.

• test\_robust\_timecode.py - main() - Tests robustness of timecode encoding under various conditions.

## Class information

### SharedTimecodeRobust Class Analysis

1. Main Purpose

The SharedTimecodeRobust class represents a robust FSK (Frequency Shift Keying) timecode encoding and decoding system optimised for VHS capture validation and MP4 timecode verification. It serves as the foundational system for generating and analysing frame-accurate timecode patterns that can survive VHS recording/playback degradation while maintaining microsecond-level precision.

2. Attributes (Data it Stores)

#### Format & Timing Attributes

• format\_type - Video format ("PAL" or "NTSC")

• fps - Frame rate (25.0 for PAL, 29.97 for NTSC)

• width, height - Video dimensions (720x576 for PAL, 720x480 for NTSC)

#### Audio Encoding Parameters

• sample\_rate - Audio sample rate (48000 Hz)

• audio\_channels - Audio channel count (1 for MONO to eliminate stereo confusion)

• freq\_0 - Frequency for binary '0' (800 Hz)

• freq\_1 - Frequency for binary '1' (1600 Hz, exactly double freq\_0)

• freq\_0\_range, freq\_1\_range - Detection frequency ranges with guard bands

• samples\_per\_frame - Number of audio samples per video frame

• bits\_per\_frame - Total bits per frame (32: 24-bit frame + 8-bit checksum)

• samples\_per\_bit - Audio samples allocated per bit

#### Visual Rendering Attributes

• font\_scale, font\_thickness - Typography parameters for visual timecode

• text\_color, bg\_color - Color scheme (white text on black background)

• corner\_color\_primary, corner\_color\_secondary - Corner marker colors (red and blue)

3. Methods (Actions it Can Perform)

#### Initialisation & Configuration

• \_\_init\_\_() - Initialises the robust timecode system with format-specific parameters and optimal FSK frequencies for VHS durability

#### Video Frame Generation

• generate\_frame\_image() - Creates video frames with visual timecode display, frame numbers, and format information

• \_add\_sync\_patterns() - Adds machine-readable binary strips and colored corner markers to video frames for automated detection

#### Audio Timecode Generation

• generate\_robust\_fsk\_audio() - Generates FSK-encoded audio for a single frame using enhanced checksum and robust frequency separation

• \_generate\_robust\_tone() - Creates clean sine wave tones with gentle envelopes to reduce transients and improve VHS recording quality

• \_calculate\_robust\_checksum() - Computes enhanced checksum using XOR with rotation for better error detection than simple checksums

#### FSK Decoding System

• decode\_fsk\_audio() - Main decoder with dual modes: strict (MP4) using deterministic frame boundaries, and tolerant (VHS) using sliding window analysis

• \_decode\_deterministic\_frames() - Strict frame-accurate decoder for MP4 validation with exact frame boundaries

• \_decode\_tolerant\_frames() - Flexible decoder for VHS with sliding window and multi-method bit analysis to handle timing variations

• \_decode\_exact\_boundaries\_robust() - Decodes at exact frame boundaries using robust bit analysis methods

• \_decode\_sliding\_windows() - Sliding window decoder to catch frames at non-standard positions due to VHS mechanical variations

#### Robust Bit Analysis

• \_decode\_frame\_robust() - Multi-method frame decoder using confidence weighting and checksum validation

• \_analyze\_bit\_robust() - Combines FFT, zero-crossing, and autocorrelation methods for robust bit detection with weighted voting

• \_analyze\_bit\_fft() - FFT-based frequency analysis with guard band detection for precise frequency identification

• \_analyze\_bit\_zero\_crossings() - Zero-crossing rate analysis for frequency estimation, robust against amplitude variations

• \_analyze\_bit\_autocorrelation() - Autocorrelation-based period detection for frequency analysis independent of phase

#### Pattern Detection & Analysis

• detect\_timecode\_window\_video() - Detects the first complete timecode window in video files by analysing pattern transitions

• detect\_timecode\_window\_audio() - Identifies timecode windows in audio files by detecting tone/silence/FSK patterns

• \_classify\_frame\_type() - Classifies video frames as 'black', 'pattern', or 'timecode' using adaptive thresholds

• \_classify\_audio\_type() - Classifies audio segments as 'silence', 'tone', or 'timecode' using energy and frequency analysis

• \_analyze\_pattern\_transitions() - Analyses frame state transitions to find complete timecode cycles

• \_analyze\_audio\_pattern\_transitions() - Analyses audio pattern transitions to locate timecode segments

#### Correlation & Synchronisation

• correlate\_timecodes() - Correlates video and audio timecodes using sequential matching to calculate precise timing offsets with statistical analysis

#### Shared Utilities

• load\_audio\_data() - Loads audio files using SOX with automatic mono/stereo detection and format conversion

• read\_binary\_strip() - Reads binary timecode from video frame top strips using adaptive thresholding for VHS compatibility

• read\_binary\_strip\_with\_corners() - Enhanced binary strip reading using corner markers for precise alignment correction

• detect\_corner\_markers() - Detects colored corner markers in video frames for geometric correction and validation

#### Validation & Quality Control

• \_validate\_frame\_id\_range() - Validates that decoded frame IDs are within reasonable ranges to reject false positives

• \_validate\_signal\_strength() - Ensures audio signals have appropriate strength and dynamics for reliable FSK decoding

• \_filter\_overlapping\_detections() - Removes duplicate detections from sliding window analysis to prevent double-counting

• \_select\_best\_detection() - Chooses the highest confidence detection from overlapping candidates

#### Utility Functions

• frame\_to\_timecode() - Converts frame numbers to human-readable HH:MM:SS:FF timecode strings

• generate\_metadata() - Creates comprehensive metadata documenting encoding parameters, robustness features, and usage instructions

Key Technical Features

This class implements several advanced techniques for VHS-optimised timecode:

1. Wide Frequency Separation - Uses 800Hz/1600Hz (2:1 ratio) instead of traditional narrow separation for better VHS tolerance

2. Guard Band Protection - Non-overlapping detection ranges with 400Hz separation prevent frequency bleeding

3. Multi-Method Validation - Combines FFT, zero-crossing, and autocorrelation analysis with weighted voting

4. Dual-Mode Operation - Strict mode for MP4 validation, tolerant mode for VHS capture analysis

5. Enhanced Error Correction - Sophisticated checksum system with rotation to detect corruption

6. MONO Audio Encoding - Eliminates stereo channel phase issues common in VHS systems

The class serves as the foundation for both timecode generation (VHSTimecodeGenerator) and analysis (VHSTimecodeAnalyzer), ensuring consistent encoding/decoding throughout the VHS archival workflow.

### VHSTimecodeGenerator Class Analysis

1. Main Purpose

The VHSTimecodeGenerator class represents a tool for generating VHS test patterns with frame-accurate timecodes for precise audio/video alignment. It supports PAL and NTSC formats, using robust FSK encoding to embed timecodes as both video and audio, enabling microsecond-accurate synchronisation after capture.

2. Attributes (Data it Stores)

#### Format & Timing Attributes

• format\_type - Video format ("PAL" or "NTSC")

• fps - Frame rate (25.0 for PAL, 29.97 for NTSC)

• width, height - Video dimensions (720x576 for PAL, 720x480 for NTSC)

#### Audio Encoding Parameters

• audio\_chunk\_seconds - Duration of audio chunk processing to maintain memory efficiency (10 seconds)

• audio\_chunk\_frames - Number of frames corresponding to audio\_chunk\_seconds

3. Methods (Actions it Can Perform)

#### Frame Generation

• generate\_frame\_image() - Generates a video frame with a large, centered timecode display, frame number, and format information

• \_add\_sync\_patterns() - Adds binary strips and corner markers for machine-readable sync

#### Audio Timecode Generation

• generate\_audio\_timecode() - Generates FSK-encoded audio representing timecode for a frame, outputting audio in both channels for compatibility

• generate\_robust\_fsk\_audio() - Utilised from the parent class, creates MONO FSK-encoded audio for timecode

#### Timecode Conversion

• frame\_to\_timecode() - Converts elapsed frame counts into timecode formatted strings (HH:MM:SS:FF)

#### Test Video Production

• generate\_test\_video() - Orchestrates the creation of a full-length test video, including generating visuals, audio, and combining them into the final file

• \_generate\_video\_stream() - Utilises OpenCV to write timecoded frames to a video file

• \_generate\_audio\_stream\_efficient() - Processes audio in small chunks to manage memory usage, writing FSK-encoded timecode audio in raw format before converting to WAV

• \_combine\_av\_streams() - Uses FFmpeg to merge the video and audio streams into a unified output file

#### Metadata Handling

• \_generate\_metadata() - Creates a JSON metadata file documenting timecode generation details

Comprehensive Workflow

• generate\_test\_video() is the main method. It integrates visual and audio processing functions and ensures proper output encoding, all while tracking progress and handling errors seamlessly.

Overall Role

The VHSTimecodeGenerator acts as a robust tool for creating precise VHS alignment tests, ensuring optimised playback quality and compatibility across VHS formats. It leverages its parent class, SharedTimecodeRobust, to manage sophisticated FSK techniques and enhances video validation capabilities through its test pattern generation.

### Here's the summary of the VHSTimecodeBase class in the vhs\_timecode\_base.py file:

Main Purpose

The VHSTimecodeBase class provides the foundational logic for generating visual and audio timecode for VHS, supporting frame generation and timecode encoding.

Attributes

• format\_type: Video format, "PAL" (25 fps) or "NTSC" (29.97 fps).

• width, height: Dimensions of the video.

• fps: Frames per second, based on format type.

• sample\_rate: Audio sample rate (48000 Hz).

• audio\_channels: Number of audio channels (2 for stereo).

• base\_frequency: Base tone frequency for audio (1000 Hz).

• bit\_duration: Duration of each bit in seconds (0.001 seconds per bit).

• sync\_frequency: Sync tone frequency (2000 Hz).

• font\_scale, font\_thickness, text\_color, bg\_color: Visual settings for text rendering.

• corner\_color\_primary, corner\_color\_secondary: Colors for corner markers.

Methods

1. \_\_init\_\_: Initializes the generator with format type, width, and height.

2. generate\_frame\_image: Creates a video frame with visual timecode and metadata.

3. \_add\_sync\_patterns: Adds machine-readable sync patterns and corner markers to frames for reliable detection.

4. generate\_audio\_timecode: Generates FSK audio timecode and sync pulse for a given frame.

5. \_generate\_fsk\_timecode: Encodes frame number into FSK-encoded audio.

6. \_generate\_sync\_pulse: Creates a sync pulse for the right audio channel.

7. frame\_to\_timecode: Converts a frame number to a human-readable timecode string.

8. generate\_metadata: Produces a metadata dictionary with encoding information.

9. save\_metadata: Saves metadata to a JSON file.

### Here's the summary of the VHSPatternGenerator class in the vhs\_pattern\_generator.py file:

Main Purpose

The VHSPatternGenerator class creates a repeating 4-step validation pattern specifically designed for VHS recording and sync validation. It generates continuous cycles of test charts, black screens, and timecode frames to help validate VHS recording quality and timing accuracy.

Attributes

• format\_type: Video format, "PAL" (25 fps) or "NTSC" (29.97 fps).

• fps: Frame rate based on the format type.

• width, height: Video dimensions (720x576 for PAL, 720x480 for NTSC).

• test\_chart\_path: Path to the appropriate test chart image file.

• sample\_rate: Audio sample rate (48000 Hz).

• audio\_channels: Number of audio channels (1 for mono).

• test\_chart\_duration: Duration of test chart step (3.0 seconds).

• black\_screen\_1\_duration: Duration of first black screen (1.0 second).

• timecode\_duration: Duration of timecode step (30.0 seconds).

• black\_screen\_2\_duration: Duration of second black screen (1.0 second).

• total\_cycle\_duration: Total duration of one complete cycle (35.0 seconds).

• timecode\_system: Instance of SharedTimecodeRobust for advanced timecode generation.

Methods

1. \_\_init\_\_: Initializes the pattern generator with format type, dimensions, and timing parameters.

2. load\_test\_chart: Loads and resizes the appropriate test chart image for the video format.

3. create\_black\_frame: Creates a black video frame.

4. generate\_pattern\_video: Generates the complete pattern video with specified number of cycles.

5. generate\_pattern\_audio: Creates frame-perfect audio with 1kHz tones for test charts and FSK audio for timecode sections.

6. generate\_tone: Generates a clean sine wave tone at specified frequency and duration.

7. save\_audio\_wav: Saves audio data as a WAV file using either scipy or sox.

8. combine\_video\_audio: Combines video frames and audio using FFmpeg to create the final MP4.

9. create\_simple\_timecode\_frame: Creates a basic timecode frame (fallback when robust system is unavailable).

10. frame\_to\_timecode\_string: Converts frame numbers to human-readable timecode strings (HH:MM:SS:FF).

11. generate\_metadata: Creates comprehensive metadata describing the pattern structure and encoding parameters.

The class creates a structured 4-step cycle:

1. Test chart + 1kHz tone (3 seconds)

2. Black screen + silence (1 second)

3. Timecode frames + FSK audio (30 seconds)

4. Black screen + silence (1 second)

This pattern is designed for VHS validation workflows where you need to test recording quality, sync accuracy, and timing precision after VHS capture.

### VHSTimecodeAnalyzer Class Analysis

1. Main Purpose

The VHSTimecodeAnalyzer class represents a precision synchronisation analyzer for VHS capture validation. It processes captured VHS video and audio files to determine microsecond-accurate audio/video synchronisation offsets by analysing both visual timecode patterns and FSK-encoded audio timecode data.

2. Attributes (Data it Stores)

#### File References

• video\_file - Path to the captured video file to be analysed

• audio\_file - Path to the captured audio file to be analysed

• metadata\_file - Optional path to metadata from timecode generation

#### Analysis Results Storage

• video\_timecodes - List of tuples storing (frame\_number, detected\_frame\_id, confidence) for each successfully decoded video frame

• audio\_timecodes - List of tuples storing (sample\_offset, decoded\_frame\_id, confidence) for each successfully decoded audio timecode

• sync\_pulses - List of tuples storing (sample\_offset, confidence) for detected synchronisation pulses

#### Inherited Attributes (from SharedTimecodeRobust)

• format\_type - Video format ("PAL" or "NTSC")

• fps - Frame rate (25.0 for PAL, 29.97 for NTSC)

• width, height - Video dimensions (720x576 for PAL, 720x480 for NTSC)

• sample\_rate - Audio sample rate (48000 Hz)

• freq\_0, freq\_1 - FSK frequencies for binary encoding (800Hz for '0', 1600Hz for '1')

3. Methods (Actions it Can Perform)

#### Initialisation & Configuration

• \_\_init\_\_() - Initialises the analyzer with video/audio files and optional metadata, inheriting robust FSK capabilities from parent class

• \_load\_metadata() - Loads and applies encoding parameters from metadata files to ensure analysis matches generation settings

#### Main Analysis Workflow

• analyze\_alignment() - Main orchestrator method that performs complete audio/video alignment analysis and returns comprehensive results with timing offsets and statistics

#### Video Analysis Methods

• \_analyze\_video\_timecode() - Processes entire video stream frame-by-frame to extract timecode information using multiple detection methods

• \_preprocess\_vhs\_frame() - Preprocesses VHS frames by deinterlacing and resizing to improve timecode detection accuracy

• \_detect\_frame\_timecode() - Extracts timecode from individual video frames using multiple fallback methods with confidence scoring

#### Video Timecode Detection Methods

• \_read\_binary\_strip() - Reads binary timecode data from the top strip of video frames using adaptive thresholding to handle VHS signal degradation

• \_read\_visual\_timecode() - Placeholder for OCR-based timecode reading from main display (currently returns None)

• \_read\_corner\_patterns() - Attempts to decode timecode from corner marker intensity patterns as a fallback method

#### Audio Analysis Methods

• \_analyze\_audio\_timecode() - Processes audio stream to extract FSK-encoded timecode using robust decoding methods optimised for VHS tape quality

• \_load\_audio\_data() - Loads audio data using SOX for format conversion and channel analysis

• \_decode\_fsk\_timecode() - Decodes FSK-encoded timecode from audio channel using sliding window analysis

• \_decode\_fsk\_segment() - Decodes individual FSK segments to extract frame IDs with checksum validation

• \_analyze\_bit\_frequency() - Analyses audio segments to determine if they represent binary '0' or '1' using frequency analysis

• \_detect\_sync\_pulses() - Detects synchronisation pulses in audio channel for frame boundary identification

• \_is\_sync\_pulse() - Determines if an audio segment contains a valid synchronisation pulse based on frequency and energy

#### Correlation & Results

• \_correlate\_timecodes() - Core analysis method that correlates video and audio timecodes to calculate precise timing offsets with statistical analysis and confidence weighting

#### Command-Line Interface

• main() - Standalone entry point providing command-line interface for VHS timecode analysis with dependency checking and comprehensive result reporting

Key Capabilities

The class is designed to handle the challenges of VHS capture analysis, including:

• Signal degradation tolerance through adaptive thresholding

• Multiple detection methods with fallback options

• Deinterlacing for VHS interlaced video

• Robust FSK decoding optimised for tape quality variations

• Statistical correlation with confidence weighting

• Microsecond-accurate measurements for precision synchronization

This makes it essential for validating and calibrating VHS archival workflows using the DdD Capture Toolkit.