Image Processing Capability Evaluation 4 - AVI Video Playback [Copy Link]

This post was last edited by ilovefengshulin on 2016-12-22 21:27

Image Processing Capability Evaluation 4 - AVI Video Playback

Hardware platform: STM32F769IDISCOVERY

Software development platform: Keil 5.1

Tool: SD card, capacity 8G, Class10 (10MB/S)

AVI file analysis:

AVI is a file that interleaves audio and video, which means that video and audio can be interwoven together for synchronous playback. There are many encoders for AVI videos, such as MPEG4, XVID, H.264, MJPEG, etc. The encoder for the AVI file I use here is MJPEG, so that each video frame file is a JPEG picture, which can be decoded using a hardware decoder. There are also many formats for audio encoders for AVI files, such as: MP3, AC3, PCM, AAC, etc. Here I use the unencoded PCM data format, so that the read data can be directly sent to the audio processing module without complex conversion, saving time and making the video playback smoother.

AVI files are based on the RIFF file structure. The most common data units are the chunk and the LIST. The structure of the entire RIFF file is as follows:

RIFF field + data size + AVI field

----LIST field + data size + hdrl field

--------avih field + data size + content

--------LIST field + data size + strl field

------------strh field + data size + content

------------strf field + data size + content

------------strd field + data size + content (optional)

--------LIST field + data size + strl field

------------strh field + data size + content

------------strf field + data size + content

------------strd field + data size + content (optional)

----LIST field + data size + INFO field + video encoder related information

----JUNK field + data size + useless data

----LIST field + data size + movi field + audio and video data

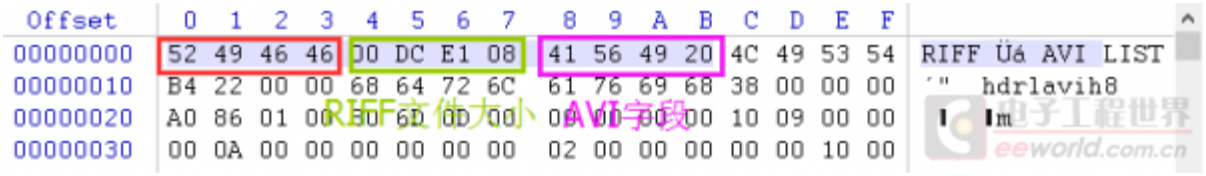
----idx1 field + data size + each frame of audio and video data size and offset address in the file (optional)

As can be seen from the above figure, an AVI RIFF file consists of the following: RIFF file header, hdrl list (AVI file data format), INFO list, movi list (AVI audio and video sequence data), and optional index block.

The AVI file we use for decoding this time uses MJPEG encoding for video and PCM

encoding for audio. We will now focus on analyzing this type of file.

RIFF header: The first 12 bytes at the beginning of the file, as shown below:



The first 4 bytes are RIFF, indicating that this is a RIFF structure file. The middle 4 bytes are the RIFF structure data size, note that the first 8 bytes are not included (**plus the first 8 bytes equals the AVI file size**), and the last 4 bytes indicate that this is an AVI file.

hdrl list: a nested series of blocks and sublists, an avih block, and one or more strl sublists.

File offset address 0x0c--0x0f: LISTcharacters.

File offset address 0x10--0x13: hdrllist data size (excluding the first 8 bytes).

File offset address 0x14--0x17: hdrlcharacters.

avih block: records the global information of the AVI file and is a sub-block of the hdrl list. File offset start address: 0x18. The data structure is as follows:

File offset address 0x18--0x1B: avihcharacter.

File offset address 0x1c--0x1f: avihword block size does not include the first 8 bytes).

File offset address 0x20--0x23: time required to display each frame (in us)

File offset address 0x24--0x27: maximum data transfer rate.

File offset address 0x28--0x2B: data filling granularity.

File offset address 0x2C--0x2F: Global mark of AVIfile, such as whether it contains index block, etc.

File offset address 0x30--0x33: total number of video frames.

File offset address 0x34--0x37: specifies the initial frame number for interactive format (non-interactive format should be specified as 0)

File offset address 0x38--0x3b: the number of streams contained in this file

File offset address 0x3C--0x3F: *It is recommended to read the cache size of this file*

File offset address 0x40--0x43: video image width (in pixels)

File offset address 0x44--0x47: video image height (in pixels)

File offset address 0x48--0x57: reserved

strl list: contains at least one strh block and one strf block. There are as many strl lists as there are streams in the file. A normal AVI file should contain video stream and audio stream files, but some AVI files have only one stream file, either video stream or audio stream. The order in which the two stream lists appear is not fixed. The first one can be the video stream or the audio stream, but the structure of the two stream data is the same.

File offset address 0x58--0x5b: LISTcharacters.
File offset address 0x5c--0x5f: strllist data size (excluding the first 8 bytes).
File offset address 0x60--0x63: strlcharacters.

strh block: used to describe the header information of the stream, and is a sub-block of the strl list. The data structure is shown in the following figure:
Whether it is an audio stream or a video stream, the playback duration = dwLength / dwRate / dwScale. The maximum value is the total playback duration of the AVI video file.

```
unsigned char fcc[4];                    //必须为strh
volatile unsigned int cb;                //strh块的大小（不包括strh字段和数据大小字段）
unsigned char fccType[4];                //流的类型（auds：音频流，vids：视频流，mids：MIDI流，txts：
unsigned char fccHandler[4];             //指定编码器的类型
volatile unsigned int dwFlags;           //是否允许这个流输出，调色板是否变化
volatile unsigned short int wPriority;   //流的优先级（当有多个相同类型的流时，优先级最高的为默认流）
volatile unsigned short int wLanguage;   //语言
volatile unsigned int dwInitialFrames;   //交互式指定初始帧数
volatile unsigned int dwScale;           //这个流使用的时间尺度
volatile unsigned int dwRate;            //dwRate/dwScale--视频：每秒帧数  音频：采样率
volatile unsigned int dwStart;           //流的开始时间
volatile unsigned int dwLength;          //流的长度
volatile unsigned int dwSuggestedBufferSize;  //读取这个流数据建议使用的缓存大小
volatile unsigned int dwQuality;         //流数据的质量指标(0~10,000)
volatile unsigned int dwSampleSize;      //音频采样的大小
struct
{
  volatile unsigned short int left;
  volatile unsigned short int top;
  volatile unsigned short int right;
  volatile unsigned short int bottom;
}_rcFrame;                               //指定这个流（视频流或者文字流）在视频主窗口显示的位置
```

strf block: This block is used to describe the specific information of the stream and is a sub-block of the strl list. For video streams, it is used to describe bitmap information. For videos encoded with MJPEG, this information is not important. For audio streams, it describes audio-related information, including the number of channels and sampling rate. For audio streams, the data structure is as follows:

```
volatile unsigned short int wFormatTag;       //音频格式
volatile unsigned short int nChannels;        //音频声道数
volatile unsigned int nSamplesPerSec;         //音频采样率
volatile unsigned int nAvgBytesPerSec;        //WAVE声音中每秒的数据量
volatile unsigned short int nBlockAlign;      //数据块的对齐标志
volatile unsigned short int biSize;           //此结构大小
```

INFO list: describes the program information for encoding the AVI file, which contains an ISFT chunk. For us, this information is not important.

JUNK block: Mainly used for internal data alignment. It contains useless information and can be skipped directly.

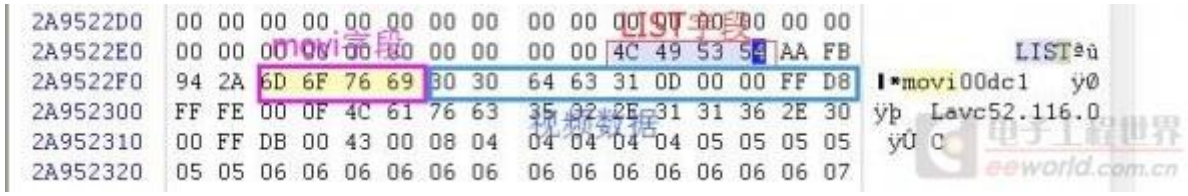
movi list: stores audio and video data blocks, which are stored in an interleaved manner in this list. The structure of the audio data block is: 01wb + the size of the audio data block + audio data. The structure of the video data block is: 00db (00dc) + the size of the video data block + video data. Among them, 01wb represents audio data, 00db represents uncompressed video data, and 00dc represents compressed video data.

Idx1 index block : describes the index block information of audio and video data. This content is optional. The data structure is: idx1+index block data size+index block data. There are two formats of index block data, one is video data index: 00dc+4

bytes describing whether it is an important frame (important frame: 0x00000010, irrelevant frame: 0x00000000)+4 bytes offset address (relative to the offset address of the movi list)+video data size; the other is audio data index: 01wb+4 bytes describing whether it is an important frame (important frame: 0x00000010, irrelevant frame: 0x00000000)+4 bytes offset address (relative to the offset address of the movi list)+audio data size.

The index block can be used to easily locate video data and audio data. Now open an AVI file and find the index block location as shown in the figure below:

You can see that the first bit of the index block is the video index block information, and its offset address in the movi list is
4 bytes. Next, find the movi list, as shown below:



As can be seen from the figure, the first data block at the beginning of the movi list is the video data, which corresponds exactly to the index block. The address of the start of the video data is: 0x26f6, and the address of the movi field in the movi list is: 0x26f2. The difference between the two addresses is 4 bytes, which corresponds to the offset address of the video index block. But this is not the starting position of the real data, and it needs to be moved 8 bytes. Finally, to summarize: the offset address of the real data = the starting address of the movi field + the offset address in the video index block + 8. The same is true for the audio index block.

Video decoding:

For us, the AVI file parameters we need to know are as follows:

1. The size of the AVI file.
2. The width and height of the video screen.
3. Video frame rate and audio sampling rate
4. Total number of video frames and offset address of each frame data
5. Total number of audio frames and offset address of each frame data
6. Number of audio channels
7. Total playback time of the file

The above parameters can be directly read or calculated from the AVI file. With the above parameters, we can decode the video.

I use SD card (class10) to store AVI files. In order to make the AVI video play more smoothly, we need to test the reading data of SD card. Here are the test results:

Data size	time
16K	2ms
32K	4ms
64K	6ms
128K	10ms

The shorter the read time, the faster the AVI video file can be decoded.

The video decoding process is as follows:

1. Initialize system clock related peripherals and configure the system clock to 200MHZ.
2. Initial SDRAM related peripherals.
3. Initialize LCD screen related peripherals.
4. Initialize SD card related peripherals, configure the clock to 25MHZ, and install the FATFS file system.
5. Initialize audio related peripherals.
5. Initialize JPEG hardware decoder related peripherals.
6. Create two video frame input buffers in SDRAM, each with a size of 256KB, to store the video frame data of the AVI file. Create a video frame index area with a size of 512KB and a structure of: video frame data offset address (4 bytes) + video frame data (4 bytes) to store all video frame information. Create an audio frame index area with a size of 512KB and a structure of: audio frame data offset address (4 bytes) to store all audio frame information.
7. Open the AVI file, enable the FATFS quick search function, obtain the relevant decoding parameters, and classify the index block information into the audio frame index area and video index area to facilitate the positioning of audio data and video data.
8. Before starting decoding, fill the first frame and the second frame of video data into the video frame input buffer respectively. At the same time, fill the first frame and the second frame of video data into the audio input buffer respectively.
9. Start decoding, transfer the content of video frame input buffer 1 to the JPEG hardware decoder, and wait for decoding to complete. During the waiting process, determine whether the video frame input buffer is empty. If it is empty, immediately fill the next frame of video data into the video frame input buffer. Transfer the content of audio frame input buffer 1 to the audio decoder, and wait for decoding to complete. During the waiting process, determine whether the video frame input buffer is

empty. If it is empty, immediately fill the next frame of video data into the audio frame input buffer.

10. Audio and video decoding are performed synchronously. After the video decoding is completed, the image is displayed.

11. Wait for the video frame interval to arrive, and process the next frame of data. Repeat this cycle, and you can play the AVI video. For specific program writing, you can refer to my program.

The following is a decoded 720X480 video with a frame rate of 12:

The following is a 480X270 video decoded at 20 frames per second:

The following is the official DEMO, 800X480 video, the nominal frame rate of the video is 25, and the maximum measured frame rate is 20 (limited by the SD card read and write speed)

If you want to get faster decoding speed, it is recommended to use a USB flash drive to store video files. For high-quality videos, decoding is more difficult, sometimes the CPU will have hardware errors and die, and there may be a phenomenon that the audio and video are out of sync, so it is not recommended to play high-quality videos. For medium-quality videos, the playback is quite smooth. . Here I would like to explain that only video files using MJPEG video encoder and PCM audio encoder are supported. If you don't have this type of AVI file, you can convert it through the Liwo AVI video converter. I personally feel that it is quite easy to use, but there is no 800X480 format in it, which is a pity.