



# JPEGs-2-AVI

Design Document

Rev 1.0

7<sup>th</sup> Sept. 2004

**TransChip Confidential**

**This document contains proprietary information, and except with the written permission of TransChip Inc., such information shall not be published or disclosed to others or used for any purpose. The document shall not be duplicated in whole or in part.**

**Revision Log**

Rev	Description
V1.0	Creation

**Done By**  
Lior W.

**Date**  
7<sup>th</sup> Sept 04

## Table of Contents

1.	OVERVIEW.....	4
1.1	Definitions/ Terms.....	4
1.2	Referenced Documents.....	4
2.	FUNCTIONS.....	5
3.	PSEUDOCODE.....	7

## 1. Overview

JPEGs-2-AVI is a set of API functions that enable the creation of an AVI file in MJPEG format from a series of JPEG files. For example, a set of 9 captured JPEGs can be packed together into an AVI (movie) file that can be played on a Windows based PC (assuming the MJPEG codec is installed on the PC machine)..

### 1.1 Definitions/ Terms

MJPEG	Motion JPEG
AVI*	Audio Video file format

\* AVIs are Microsoft's approach to storing movies. It uses the RIFF format to store information in a tree structure. Both audio and video data are encoded using the Windows' installable codecs. Please note that currently TransChip doesn't support audio capture

### 1.2 Referenced Documents

TC574x Programmer's Reference of 7<sup>th</sup> Sept.2004.

## 2. Functions

**Calculates the required memory needed to create an AVI file in MJPEG format from a list of still JPEG files.**

```

■ int TCjpgsToAviGetMemSize(TCjpgsToAviFilesListStruct
                           unsigned short
                           unsigned long
                           unsigned long
                           *ptcJpgsToAviFilesListStr,
                           usNumOfJpegs,
                           *pulAVIfileSize,
                           *pulTotalJpegsCodeSize);

typedef struct
{
    TCjpegHeaderInfoStruct    tcJpegHeaderInfoStr;
    unsigned char             *pJpegFile;
    unsigned long             ulJpegFileSize;
    unsigned long             ulFileOffset;           // These 2 parameters are filled by
    unsigned long             ulJpegFrameSize;        // jpgsToAvi API functions...
} TCjpgsToAviFilesListStruct;

```

**ptcJpgsToAviFilesListStr** is an array of **TCjpgsToAviFilesListStruct** that holds all the JPEG files information (e.g. file size, code size, width, height, quantization tables offsets etc...). All this information is extracted by a call to the function **TCdjpgParseJpegHeader**.

**usNumOfJpegs** is the number of files (Array length)

**pulAVIfileSize** is a pointer to uint32 that the function returns the required AVI buffer size.

**pulTotalJpegsCodeSize** is a Pointer to uint32 that the function returns the total JPEGS net code size (not including the headers...)

**Creates the AVI file header**

```

int TCjpgsToAviCreateFile(TCjpgsToAviFileCreateInfoStruct *ptcJpgsToAviFileCreateInfo);

```

```

typedef struct
{
    unsigned short    usNumOfJpegs;
    unsigned short    usFrameRate;
    unsigned long     ulTotalJpegCodeSize;
    TCjpgsToAviFilesListStruct    *ptcJpgsToAviFilesListStr;
    void              *pAviBuffer;
    unsigned long     ulAviBufferSize;
} TCjpgsToAviFileCreateInfoStruct;

```

**ptcJpgsToAviFileCreateInfo** is a pointer to a **TCjpgsToAviFileCreateInfoStruct** that describes all information needed to create the AVI file header. (e.g. frame rate, width, height, num of frames, etc.) It also includes the buffer on which the AVI file will be built. The size of the buffer was given by the previous function (**TCjpgsToAviGetMemSize**).

This function is called N times (where N is the number of JPEG files to be converted into an AVI movie)

■ **int TCjpgsToAviAddFile(TCjpgsToAviFileCreateInfoStruct      \*ptcJpgsToAviFileCreateInfo,  
                         TCjpgsToAviFilesListStruct            \*ptcJpgsToAviFilesListStr,  
                         unsigned short                                usFileIdx);**

**ptcJpgsToAviFileCreateInfo** points to a **TCjpgsToAviFileCreateInfoStruct** that describes all information needed to add a JPEG file to the AVI. **ptcJpgsToAviFilesListStr** points to the files list that holds specific info on each JPEG file (e.g. quantization tables used, code size, etc.). **usFileIdx** is the file index added to the AVI. (0-N).

This function closes the AVI file (i.e. writes the footer)

■ **int TCjpgsToAviCloseFile(TCjpgsToAviFileCreateInfoStruct \*ptcJpgsToAviFileCreateInfo,  
                         TCjpgsToAviFilesListStruct      \*ptcJpgsToAviFilesListStr,  
                         unsigned short                                usNumOfJpegFiles);**

**ptcJpgsToAviFileCreateInfo** points to a **TCjpgsToAviFileCreateInfoStruct** that describes all information needed to add a JPEG file to the AVI. **ptcJpgsToAviFilesListStr** points to the files list that holds specific info on each JPEG file (e.g. quantization tables used, code size, etc.). It is used to extract the offsets of each file.

### 3. Pseudocode

1.	The number of JPEG files to be packed into AVI file is known. Whether the user chooses N files or if a snap-shot sequence has been started. For this pseudo code example let's assume 9 files.
2.	Allocate (either dynamically or statically) an array of type <b>TCjpgsToAviFilesListStruct</b> . Number of elements is the number of files: For example: <b>TCjpgsToAviFilesListStruct tcJpgsToAviFilesInfoStr[9];</b>
3.	Allocate a variable of type <b>TCjpgsToAviFileCreateInfoStruct</b> . For example: <b>TCjpgsToAviFileCreateInfoStruct tcJpgsToAviFileCreateInfoStr</b>
4.	After each JPEG capture, fill the tcJpgsToAviFilesInfoStr[i] with the following information: <ul style="list-style-type: none"> <li>• pJpegFile</li> <li>• ulJpegFileSize</li> <li>• tcJpegHeaderInfoStr</li> </ul> Call TCdjpgParseJpegHeader to extract information to tcJpegHeaderInfoStr. For example: <b>TCdjpgParseJpegHeader(pJpegFile, ulJpegFileSize, &amp;(tcJpgsToAviFilesInfoStr[i].tcJpegHeaderInfoStr));</b>
5.	When the snap-shot process is over we have a JPEG files database that describes all information we need to create an AVI file.
6.	Call <b>TCjpgsToAviGetMemSize(tcJpgsToAviFilesInfoStr,9, &amp;ulAVIfileSize,&amp;ulTotalJpegsCodeSize);</b>
7.	Allocate AVI_Buffer (either dynamically or statically) with the size returned by the previous function call. (ulAVIfileSize).
8.	Fill <b>tcJpgsToAviFileCreateInfoStr</b> with the relevant information needed for the following functions. For example: <pre> tcJpgsToAviFileCreateInfoStr.usFrameRate = 5; tcJpgsToAviFileCreateInfoStr.ptcJpgsToAviFilesListStr = tcJpgsToAviFilesInfoStr; tcJpgsToAviFileCreateInfoStr.usNumOfJpegs = 9; tcJpgsToAviFileCreateInfoStr.ulTotalJpegCodeSize = ulTotalJpegsCodeSize; tcJpgsToAviFileCreateInfoStr.pAviBuffer = AVI_Buffer; tcJpgsToAviFileCreateInfoStr.ulAviBufferSize = ulAVIfileSize; </pre>
9.	Call <b>TCjpgsToAviCreateFile(&amp;tcJpgsToAviFileCreateInfoStr);</b>
10.	Call <b>TCjpgsToAviAddFile</b> 9 times to add the 9 JPEG files into the AVI file: <pre> for(i=0;i&lt;9;i++) {     // The reason for this implementation is to allow the user to load each file     // separately (if for example it is saved on flash or file-system).     // For example:     // tcJpgsToAviFilesInfoStr[i].pJpegFile = readNextFile(i);     TCjpgsToAviAddFile(&amp;tcJpgsToAviFileCreateInfoStr,                       tcJpgsToAviFilesInfoStr,i);// Add another jpeg to file } </pre>

- |     |   |
|-----|---|
| 11. | Call <b>TCjpgsToAviCloseFile</b> to close the AVI file with the appropriate footer:<br>TCjpgsToAviCloseFile(&tcJpgsToAviFileCreateInfoStr,<br>tcJpgsToAviFilesInfoStr,9); |
|-----|---|