



TC574X CMOS Imager

API Programmer's Reference

V1.7
10st Nov. 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contains proprietary information, and except with the written permission of TransChip Inc., such information shall not be published or disclosed to others or used for any purpose.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	Date	Done By
1.0	Initial revision	14 th April 2004	Gal Adler
1.1	Changed I ² C address	3rd June 2004	Jack Shasha
1.2	<p>Add TCgetCoreReadyStatus, TCoutputEnable, TCdjpegGetMask, TCsetClkEdgeForPVI and TCgetPIIStatus functions</p> <p>Remove TCosdFlip and TCdjpegAnimationSettings functions</p> <p>Change parameters for the functions TCsetJpegSize, TCdjpegSettings, TCdjpegDecompress, TCmjpeg2mem and TClcdDataType</p> <p>Function TClcdSelect has new functionality (and moved from TCutil to TChstCom)</p> <p>Functions TCouputFormat and TCoutputSize moved to "Live Video output function" section from "Image Controls" section</p> <p>Changes in Custom Settings (section 3.1.2)</p> <p>Change description of TCstartCore, TCsetJpegSize, TCjpegCapture, TCmjpegQuality and TCmjpeg2mem</p> <p>Added Snapshot Capture Commands</p>	1 July 2004	YaronPaz
1.3	Differentiate API support per Chip Type (5747 / 5740) instead of 5747 only document	4 July 2004	Yaron & Amir
1.4	<p>Add functions to API:</p> <ul style="list-style-type: none"> Reset function. JPEGS to AVI conversion functions. TCprintf function. TCjpegFixOrientation TCconvertYUV422ToRGB565 JPEG post processing functions. TCsetOrientation TCdscPwrUpForRegAccess TCosdReset <p>Update TCjpegHeaderInfoStruct.</p> <p>Update HostTCgpioControl function (add reset pin).</p>	29 August 2004	Lior W.
1.5	Add floating menu API functions	28 September 2004	Lior W.
1.6	Add flash control functions.	21 October 2004	Lior W.
1.7	<p>Add functions to API:</p> <ul style="list-style-type: none"> TCjpegMinimizeHeader <p>Update TCjpegFixOrientation for 90° right rotation.</p> <p>Update TCdjpgParseJpegHeader to detect non-default Huffman tables.</p>	10 November 2004	Lior W.

Table of Contents

1.	INTRODUCTION	4
1.1	Terminology/Conventions	4
1.1.1	Terms	4
1.1.2	Abbreviations.....	4
2.	OVERVIEW.....	5
2.1	Platform Specific HW Driver Functions	7
2.2	Core Access Routines (TCapi.*).....	7
2.3	Host-Commands Interface Functions (TChstCom.*).....	8
3.	HOST COMMANDS INTERFACE.....	13
3.1	General Conventions.....	13
3.1.1	Function Return Codes.....	13
3.1.2	Custom Settings	13
3.2	Functions	14
3.2.1	Essential Functions.....	14
3.2.2	Power save Functions	15
3.2.3	Host and LCD Interface Settings Functions.....	15
3.2.4	Live Video Output Functions.....	17
3.2.5	Image Controls:	17
3.2.6	JPEG Capture Commands	18
3.2.7	Snap Shot Capture Commands.....	20
3.2.8	JPEG Decompression Commands	20
3.2.9	JPEG Postprocessing Commands	23
3.2.10	JPEG files conversion to AVI (MJPEG format)	24
3.2.11	Motion JPEG (MJPEG) capture and decompression.....	25
3.2.12	OSD commands:	27
3.2.13	Platform Dependent Custom Settings Functions	28
3.2.14	Miscellaneous Commands.....	29
4.	IMPLEMENTING PLATFORM-SPECIFIC HW DRIVER FUNCTIONS.....	34
5.	APPENDIX A: I²C SERIAL INTERFACE.....	36
5.1	Overview	36
5.2	Mode of Operation	36

1. Introduction

The TC574X Application-Programming Interface (API) is a set of functions provided for the host application to control and operate the TC574X CMOS sensor. The API is designed to be platform- and application-independent, and to provide support for the full feature set of the TC574X. The API is designed to expedite software development for the TC574X by providing the designer with high-level functions that can be easily understood and integrated into an application. This manual provides a detailed description of this API.

Written in ANSI C, the API is hardware independent and can be easily ported to different platforms. The entire interface, with hardware, has been developed with simple software mechanisms.

1.1 Terminology/Conventions

1.1.1 Terms

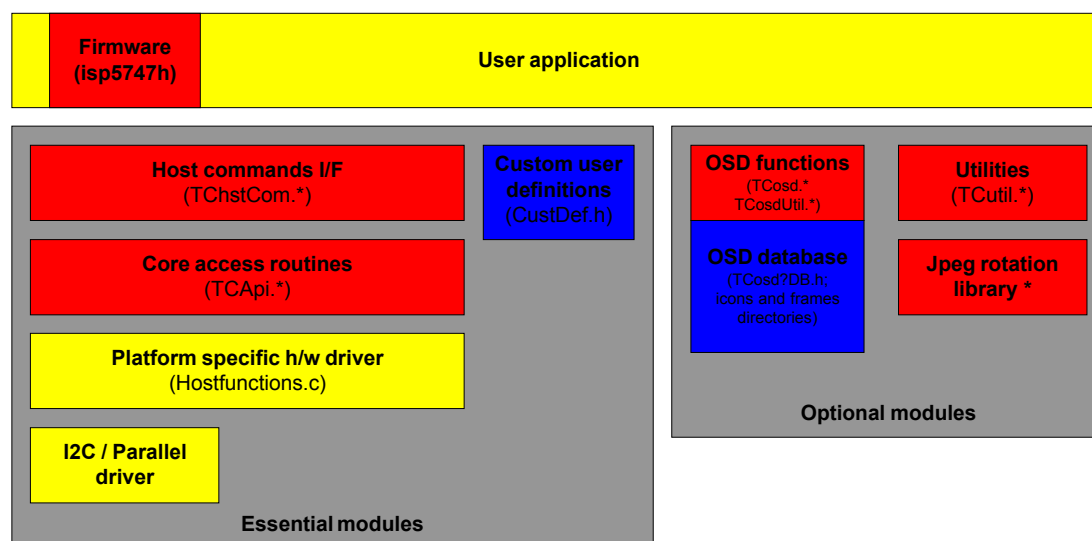
Core	TC574X internal integrated micro controller.
TC574X Firmware core software (FW)	Software stored in read-only memory (ROM) or programmable ROM (PROM) normally used for the initialization of a system.
Host Function	A function required to be implemented by the Host to enable the operation of the API.
Host Command	An API command that controls the core FW or sets its registers.
Base band processor	The target platform that controls the TC574X. It may be a Microcontroller, Microprocessor or a Base Band processor on a cellular phone

1.1.2 Abbreviations

SW	Software.
HW	Hardware.
FW	Firmware.
API	Application Programming Interface.
OSD	On Screen Display.

2. Overview

The following diagram illustrates the high-level structure of the TC574X API:



Color code legend

Read only modules (not to be modified by user)
User modules (should be implemented by the user)
Customizable modules (may be modified by the according to the required application)

Figure 1: API High-Level Structure

Layer/Function	Description	For more Information, see:
User Application Software - Top Layer	This layer consists of user application software for the operation of the camera using the Host-Commands interface functions (a high level interface providing the required camera functions, bypassing access to the lower level layers).	
Firmware - Top Layer	The user application is required to store the TC574X firmware (available as a header file or binary file), and download it to the sensor, following the power-up and reset sequence, using the applicable Host Command.	
Host Commands Interface	Provides communication with the TC574X via an intermediate layer of Core Access routines. These are hardware-independent I/O routines that provide basic camera and memory control.	Host Commands Interface
Core Access Routines	These routines should be called exclusively by the TChstCom layer, and not by the host user application. They use platform-specific hardware driver functions to communicate with the camera hardware.	Core Access Routines (TCapi.*)
Platform-specific	These functions include the I2C IO functions, camera discrete	Platform Specific HW

HW Driver Functions *	signals handling (such as power-saving pins) and real-time delay handling.	Driver Functions
I ² C / Parallel Driver	Implements the Host interface. Depending on the application, the host interface may be serial I2C, or standard 8/16 bit parallel CPU interface. When implementing the I2C Interface, If the host processor does not have dedicated hardware for the implementation of the I2C interface, the I2C may be implemented using two general-purpose IO's. When implementing the parallel interface, no special driver is required as the interface is made of regular CPU bus IO.	Appendix A: I2C Serial Interface
Custom User Definitions (CustDef.h)	The file is used to adapt API functionality to a specific application. It is used, for example, to set the application preview screen size, sensor orientation, and various other parameters.	Custom Settings
Optional Modules	Provide additional functionality that is required by some applications.	
OSD Routines	Operate the OSD (On Screen Display) features. They enable the overlay of icons and frames over the image. A sample OSD database which includes sample icons and frames is provided by TransChip. The user may also modify the OSD database and create custom icons and frames.	(See the manual: "Application Note - Updating OSD Icons and Frames Database")
JPEG Rotation Library	Enables the rotation of JPEG images 90° clockwise or counterclockwise, usually required when the TC574X camera is rotated when installed. This library is available for various platforms including PC and ARM.	Miscellaneous Commands

2.1 Platform Specific HW Driver Functions

This layer contains service functions used by **TCapi.c**. Since **TCapi.c** is designed to be hardware-independent, it is the user's responsibility to implement these low level functions:

HostTCTransferDataProc	Sends and receives data from the camera using the I ² C interface. The function should call the application specific I ² C routine. The idea is to separate TCapi from the hardware layer and make it a general purpose hardware-independent source code.
HostTCgpioControl	Controls the discrete pins that set the TC574X power save mode. It is assumed that these pins are under the control of the host processor (usually they would be implemented as GPIO pins).
HostTCdelayMSec	Generates controllable real-time delay to be used by TCapi .
HostTCresetWatchDog	This function should be implemented when the host operating system has an internal watchdog that resets the system if it is not called within a certain time. TCapi calls the internal watchdog function before operations begin that could require considerable time to complete (such as firmware uploading). If there is no such watchdog in the system, this function should remain blank.
HostTCtraceOut	When traces out are available (e.g. RS232 cable between phone to PC) then this function may be used to trace debug data out of the SDK.

2.2 Core Access Routines (TCapi.*)

This layer uses the host functions module and implements low level functions for the API layer above it (**TChstCom**). It supports the following functions:

tcRegisterAccess	Implements writing or reading from core registers. It has the ability to read or write blocks of data from the same register, exploiting the core ability to auto-increment the address.
tcWriteRegister	Writes a 16 bit word into a core register.
tcReadRegister	Reads a 16 bit word from a core register.
tcCoreMemoryAccess	Implements writing or reading from core memory. It has the ability to read or write blocks of data from the core memory.
tcWriteCoreMemory	Uses tcCoreMemoryAccess to write data to the core memory.
tcReadCoreMemory	Uses tcCoreMemoryAccess to read data from the core memory.
tcSendCommand	Sends a host command to the core.
tcGetCoreStatus	Reads the core status flags.
tcWaitForStatusFlag	Waits for a specific flag to be valid within the status flag.

Note

The host user application layer does not need to use these functions; they should be called exclusively by the **TChstCom** layer.

2.3 Host-Commands Interface Functions (TChstCom.*)

This layer implements all necessary host commands as simple ANSI-C function calls. It uses **TCapi** to control the core and allows the host application to control the camera in a simple and intuitive way by calling various C functions. For example, if the host application wants to send a “Set Brightness” command it will simply call the C function **TCbrightness** and pass it a value of **sBrightness**.

A detailed description of the functions is presented in the following chapter. The functions that are implemented in this layer are listed in the tables below.

Function prototype	Include file
Essential Functions	
int TCuploadFirmware(void* pBuffer, unsigned long ulBufferSize, unsigned long ulAddress);	TChstCom.h
int TCstartCore(unsigned short usClockInKhz, unsigned short usFastInit);	TChstCom.h
int TCgetCoreReadyStatus (unsigned short *pusCoreReady);	TChstCom.h
int TCgetDeviceStatus (unsigned short *pusCoreRunning);	TChstCom.h

Power save Functions	
int TCpowerSaveMode(TCpower eMode)	TChstCom.h
int TCdscPwrUpForRegAccess(unsigned short bPowerUp);	TCutil.h

Host and LCD Interface Settings Functions	
int TClcdSelect(unsigned short usLcdNum, unsigned short usLcdCmdMode) ;	TChstCom.h
int TClcdDataType(unsigned short usDataAccess, unsigned short usRsState) ;	TCutil.h
int TClcdControlAccess(unsigned short usCont_su, unsigned short usCont_wr) ;	TCutil.h
int TClcdWrite(unsigned short* pusData, unsigned short usWordCount) ;	TCutil.h

Live Video Output Functions	
int TCoutputFormat(TCformat eOutputFormat);	TChstCom.h
int TCoutputEnable(unsigned short bEnable, unsigned short bWaitForEndPreview);	TChstCom.h
int TCoutputSize(unsigned short usWidth, unsigned short usHeight);	TChstCom.h

Image Controls	
int TCbrightness(signed short sBrightness);	TChstCom.h
int TCcontrast(signed short sContrast);	TChstCom.h
int TCsaturation(signed short sSaturation);	TChstCom.h

int TChue(signed short sHue);	TChstCom.h
int TCgamma(unsigned short usGamma);	TChstCom.h
int TCframeRate(unsigned short usFrameRate);	TChstCom.h
int TCautoFrameRateLimitations(unsigned short usMinFR, unsigned short usMaxFR);	TChstCom.h
int TCdigitalEffect(TCeffect eEffect);	TChstCom.h
int TCsetZoom(TCzoom eZoom);	TChstCom.h

JPEG Capturing and Compression Functions

int TCjpegQuality(TCquality eQuality);	TChstCom.h
int TCsetJpegSize(unsigned short usWidth, unsigned short usHeight, unsigned short bCropToPreview);	TChstCom.h
int TCjpegCapture(TCsource eSource, unsigned short bThumbnail, unsigned short usThumbWidth, unsigned short usThumbHeight);	TChstCom.h
int TCgetJpegStatus(unsigned short * pusStatus, unsigned short bBlocking);	TChstCom.h
int TCgetJpegFileSize(unsigned short * pusFileSize);	TChstCom.h
int TCgetJpegData(unsigned short * pusBuffer, unsigned short usWordCount , TCstate eState);	TChstCom.h

Snap Shot Functions

int TCSnapCapture(unsigned short NPictures, unsigned short bThumbnail, unsigned short usThumbWidth, unsigned short usThumbHeight);	TChstCom.h
int TCgetImageStatus(unsigned short * pusStatus, unsigned short bBlocking);	TChstCom.h

JPEG Decompression Functions

int TCdjpegSettings(unsigned short usDestination, unsigned short bFit2Screen, unsigned short usWidth, unsigned short usHeight, TCorigin Origin);	TChstCom.h
int TCdjpegRequest(TCtransfer eTransferType, unsigned short *pusMaxFileSize, unsigned short bWithOSD);	TChstCom.h
int TCdjpegLoading(unsigned short * pusBuffer, unsigned short usWordCount, TCstate eState);	TChstCom.h
int TCdjpegDecompress(TCmove eMove, unsigned short bWithOSD);	TChstCom.h

int TCdjpegGetMask (TCmove eMove, unsigned char *pMask);	
int TCgetDjpegStatus(unsigned short * pusStatus, unsigned short bBlocking);	TChstCom.h
int TCgetDjpegFileSize(unsigned short * pusFileSize);	TChstCom.h
int TCgetDjpegData(unsigned short * pusBuffer, unsigned short usWordCount, TCstate eState);	TChstCom.h
int TCjpegMinimizeHeader(uint8 *pJpegFile, uint16 usFileSize, uint8 *pJpegOutputFile, uint16 *usFileOutputSize)	TCutil.h

- - also used for MJPEG loading

JPEG Post-processing Functions

int TCjpegPpStartSession(void);	TChstCom.h
int TCjpegPpModifyParam(TCJpegPPPParamID eParam, signed short sValue);	TChstCom.h
int TCjpegPpEndSession(void);	TChstCom.h

JPEG files conversion to AVI (MJPEG format)

int TCjpgsToAviGetMemSize(TCjpgsToAviFilesListStruct *ptcJpgsToAviFilesListStr, unsigned short usNumOfJpegs, unsigned long *pulAVIfileSize, unsigned long *pulTotalJpegsCodeSize)	TCutil.h
int TCjpgsToAviCreateFile(TCjpgsToAviFileCreateInfoStruct *ptcJpgsToAviFileCreateInfo);	TCutil.h
int TCjpgsToAviAddFile(TCjpgsToAviFileCreateInfoStruct *ptcJpgsToAviFileCreateInfo, TCjpgsToAviFilesListStruct *ptcJpgsToAviFilesListStr, unsigned short usFileIdx);	TCutil.h
int TCjpgsToAviCloseFile(TCjpgsToAviFileCreateInfoStruct *ptcJpgsToAviFileCreateInfo, TCjpgsToAviFilesListStruct *ptcJpgsToAviFilesListStr, unsigned short usNumOfJpegFiles);	TCutil.h

Motion JPEG Capture and Playback Functions

int TCmjpegQuality(unsigned short usQuality);	TChstCom.h
int TCmjpeg2mem(TCmjpeg2memStr *pTCmjpeg2memInfo, unsigned long *pulStartAddress);	TChstCom.h
int TCmjpegGetWriteCnt(unsigned short *pusWriteCnt);	TChstCom.h
int TCmjpegGetFlagFull(unsigned short *pusFlagFull);	TChstCom.h
int TCmjpegSetReadCnt(unsigned short usReadCnt);	TChstCom.h
int TCmjpegClearFlagFull(void);	TChstCom.h

int TCdmjpegRequest(TCdmjpegInfoStr *pdmjpegInfo,unsigned short usMaxFileSize);	TChstCom.h
int TCdmjpegDecompress(void);	TChstCom.h
int TCdmjpegEndProcess(void);	TChstCom.h

OSD Functions

int TCosdInit(void);	TCosd.h
int TCosdLoadFrame(unsigned char frame);	TCosd.h
int TCosdEnableFrame(unsigned char bEnable);	TCosd.h
int TCosdEnableIcon(unsigned char icon, unsigned char bEnable);	TCosd.h
int TCosdEnableBar(unsigned char barTag, unsigned char bEnable);	TCosd.h
int TCosdLoadCursor(unsigned char cursorTag);	TCosd.h
int TCosdEnableCursor(unsigned char bEnable);	TCosd.h
int TCosdSetCursor(unsigned short xPos, unsigned short yPos);	TCosd.h
int TCosdSetTimeStamp(TCLineNum eLine, char* timeStamp);	TCosd.h
int TCosdDisableTimeStamp(void);	TCosd.h
int TCsetOrientation(uint16 usChipOrientation);	TCosd.h
Int TCosdReset(void);	TCosd.h
int TCosdEnableFMitem(TcFloatingMenuitemStruct *pFMitem);	TCosd.h
int TCosdClearFMitem(TcFloatingMenuitemStruct *pFMitem);	TCosd.h

Optional and Platform / Chip Dependent Settings

int TCvalidVpolr(unsigned short bLow);	TChstCom.h
int TCuseChipSelect(void);	TChstCom.h
int TCsetValidHasWriteSignal(unsigned short bActiveLow , unsigned short bSplitClocks);	TChstCom.h
int TCsetClkEdgeForPVI(unsigned short bNegativeEdge);	TChstCom.h
int TCgetPIIStatus(unsigned short *bPIIActive); - supported by TC5747 only	TCutil.h
int TCpIIAdjust(unsigned long ullInputClkFreq) ; - supported by TC5747 only	TCutil.h

Miscellaneous Functions

int TCjpegGetThumbnail(unsigned char* pucJpeg, unsigned char**ppucThumbnail, unsigned short* pusWidth,unsigned short* pusHeight);	TCutil.c
int TCdjpgParseJpegHeader(unsigned char *pJpegFile,unsigned short buffSize, TCjpegHeaderInfoStruct *pJpegHeaderInfo);	TCutil.c
int TCjpegRotation(unsigned char *pJpegFile,unsigned short usFileSize, unsigned char *pRotatedJpegFile, unsigned short *usRotatedFileSize, TCrotate eDirection);	TCutil.c
int TCjpegFixOrientation(unsigned char *pJpegFile,unsigned short usFileSize, unsigned char *pFixedJpegFile, unsigned short *usFixedFileSize);	TCutil.c
void TCconvertYUV422ToRGB888(unsigned char *pYUV422buffer, unsigned char *pRGB888buffer, unsigned short usWidth, unsigned short usHeight);	TCutil.c
void TCconvertYUV422ToRGB565(unsigned char *pYUV422buffer, unsigned short *pRGB565buffer, unsigned short usWidth, unsigned short usHeight);	TCutil.c
int TCbackUpCalibrationTables(TCcalBackupStateEnum eState);	TCutil.c
int TCflashLight(uint16 usFlashType, TCFlashStatus eFlashMode);	TChstCom.h
int TCgetContFlashState(unsigned short *pbContFlashActive);	TChstCom.h
int TCconfigureFlashPulse(unsigned long ulMaxDuration, unsigned long ulMinDelay, unsigned short usMaxBursts unsigned short bEnableRedEyeReduction);	TChstCom.h
int TCconfigureFlashAutoLevels(unsigned short usContLampOnThresh, unsigned short usIlluminationAt50cm, unsigned short usPulseLampOnThresh, unsigned short usPulseLampIsDominantThresh);	TChstCom.h
int TCconfigureFlashSpecialSettings(void);	TChstCom.h
int TCsetFlashPolarity(unsigned short bActiveHigh);	TChstCom.h
int TCfwVersion(unsigned short *usFWver, unsigned short *usFWdate, unsigned short *usAPIver);	TChstCom.h

3. Host Commands Interface

This section describes the functions in **TChstCom.c**. These functions provide the complete set of controls for the operation of the TC574X.

3.1 General Conventions

3.1.1 Function Return Codes

All API functions have a return **int** value. It is recommended to assert this return value on all calls to API functions.

The applicable return codes are listed below:

Return Code	Explanation
0	Success.
776	Function failed due to register access busy error (probably another process is accessing the camera simultaneously)
777	Function failed due to memory access busy error (probably some other process is accessing the camera simultaneously).
778	Function failed due to timeout error (usually this means the camera is not responding and should be reset)
800	AVI file cannot be created from a series of JPEG files with different widths or heights.
Any value other than 0	Unspecified error (check in the function code for the error documentation).

3.1.2 Custom Settings

The API includes a custom settings header file (**custDef.h**). This file stores definitions regarding the target application screen, and TC574X mechanical orientation. This information is used by the firmware and by the OSD API. The following parameters should be defined:

CUST_MAIN/SUB_FAVORITE_FORMAT - Defines the default live video and JPEG decompression output format for the main and the sub LCD. See **TCoutputFormat** for possible values.

CUST_MAIN/SUB_MIRROR_Y/X - These four values define the required Mirror setting for the main and sub LCDs. When MirrorY is set, the camera sends to the LCD the lines starting from the bottom instead of from the top, when MirrorX is set the camera sends to the LCD the columns starting from the right, instead of from the left.

CUST_MAIN/SUB_CLK_DIV_NO_DS - Due to limitations of some LCD screens, it may be required to set the TC574X internal clock divider in order to slow down the video data rate to the rate acceptable by the LCD. Once the clock divider in the custom settings is defined for 1:1 down sampling, all the clock dividers for other down sampling modes are automatically defined. The clock divider value should be set to 0 when the LCD is fast enough. If not, a value of 1 sets the clock divider to 2 (half the output rate), a value of 2 sets the clock divider value to 4, etc.

CUST_(SUB_)LCD_WIDTH/HEIGHT - These four values describe the dimensions of the main and sub LCDs. If there is no sub LCD, the sub values should be set to 0.

CUST_MAIN/SUB_ORIENTATION – Describes the sensor orientation relative to the position of LCD. For a horizontal system **TCIF_LENS HOLDER_RIGHT** should be defined for a not-rotated camera and

TCIF_LENS HOLDER_LEFT should be defined for a camera rotated by 180 degrees. For vertical systems **TCIF_LENS HOLDER_UP** should be defined for a camera rotated 90 degrees counterclockwise. **TCIF_LENS HOLDER_DOWN** should be defined for a camera rotated by 90 degrees clockwise.

Lens data tables – the **custdef.h** includes data tables for implementation of anti-shading and color-correction. The table corresponding to the lens being used must be selected by defining only one of the following (according to the TC574X module type being used):

For 5747 either **#define TC5747XX24X** or **#define TC5747MF39A** should be used.

For 5740 either **#define TC5740MB24B** or **#define TC5740MF24G** should be used.

3.2 Functions

3.2.1 Essential Functions

- **int TCuploadFirmware(void* pBuffer, unsigned long ulBufferSize, unsigned long ulAddress);**
This function uploads the firmware from the host to the TC574X. It may be used to upload the entire firmware file at once, or, alternatively, to upload in several iterations. (It is sometimes required to divide the upload into several iterations because the whole buffer cannot be available at once, or because uploading the entire firmware at once may consume too much time.)

To upload the full firmware at once, call this function with the FW buffer pointer, the firmware file size in bytes and **ulAddress=0**.

To upload the firmware in several iterations, **ulBufferSize** may be set to 256, 512, 1024, 2048, 4096, 8192, or 16384. When uploading the firmware in iterations, the first call to **TCuploadFirmware** should be with **ulAddress =0**, while all the consecutive calls should be with **ulAddress =0xFFFFFFFF**. This signals to the function that the next buffer goes immediately after its predecessor.
- **int TCstartCore(unsigned short usClockInKhz, unsigned short usFastInit);**
In order to initialize the camera after FW uploading, call this function and pass it the clock frequency to which the camera is connected. This value is in KHz and hence if the clock input is 30 Mhz call this function with the value of 30000.
usFastInit is used to skip the TC574X calibration stage and thus provide faster init. As a default it should be always set to FALSE, unless very fast initialization is required. For more information on the calibration stage and fast initialization **TCbackUpCalibrationTables**.

Notice that since the initialization process takes few seconds, the function returns before initialization is complete. Ensure that initialization is complete before calling any other function. To check that initialization is complete call **TCgetCoreReadyStatus** until you receive a positive response.
- **int TCgetCoreReadyStatus (unsigned short *pusCoreReady);**
This function checks the TC574X core status; It checks whether the core is ready to receive commands. If it is ready, it writes TRUE to the *pusCoreReady – otherwise it writes FALSE to the *pusCoreReady.
- **int TCgetDeviceStatus (unsigned short *pusCoreRunning);**
This function checks the TC574X core status; it samples an internal core watchdog and indicates if the core is running, or, for some reason, had a critical error and stopped functioning. If the function

return code is **0** (success) and **pusCoreRunning** is returned TRUE – then the core is OK. If **pusCoreRunning** is returned FALSE – the core is not functioning and the camera should be reset and initialized. If the function return code is not **0**, it means there is some I²C interface problem with the camera.

3.2.2 Power save Functions

■ **int TCpowerSaveMode(TCpower eMode):**

To reduce power consumption, the camera can shut down digital and analog components during idle/standby mode. The function may be called with one of the following options:

- **TCIF_FULL_OPERATION**
- **TCIF_STANDBY_MODE**

When entering standby mode, all camera operations stop (such as video preview). The camera will stop responding to host commands. After returning to full operation mode, all operations resume without further commands.

■ **int TCdscPwrUpForRegAccess(unsigned short bPowerUp);**

This function is used to read or write registers when camera is in sleep mode.

3.2.3 Host and LCD Interface Settings Functions

The following function are used to control the various features and settings of the TC5747 LCD interface that is used to control the LCD display, and the parallel host interface, which is used to communicate with the host processor. These commands are also used to configure the pass-through mode, in which the host processor sends data to the LCD through TC5747.

Notice that although only TC5747 HW supports LCD interfacing, the TCldSelect function should be used also for TC5740. The above function sets general system parameters related to LCD used.

■ **int TCldSelect(unsigned short usLcdNum, unsigned short usLcdCmdMode) ;**

Important: Whenever this function is called and the firmware is active, also call **TCoutputSize** with the sizes of the appropriate LCD!! Then, if the two LCDs have different orientations from each other, but the same OSD is used, the OSD should be reloaded for flipping.

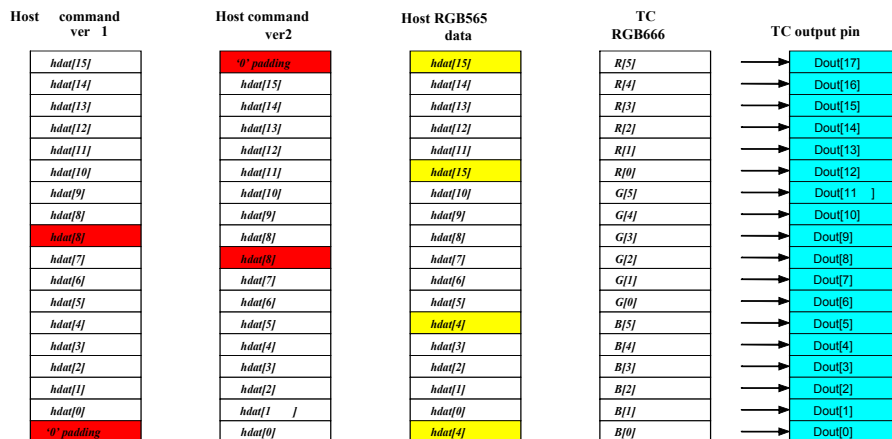
TC5747 supports dual LCD modules. This function is used to select which LCD chip select line will be driven by TC5747:

usLcdNum value	LCD CS driven
1	LCD_CS_1
2	LCD_CS_2

The above setting will affect both the video streaming from the sensor, and the pass-through interface from the host.

The second parameter – **usLcdCmdMode** - is used to determine the mapping between the 16 bit host data bus and the 18 bit LCD bus. This mapping is used when connecting a 16 bit host to an 18 bit (RGB666) display, and should be selected according to the particular display requirement.

There are two available mapping options – ver1 and ver2 - which are described in the following table, are used to send commands/data to the LCD.



Hdat[15:0] is the host interface bus, and **Dout[17:0]** is the 18 LCD bus. The two options for mapping the **hdat** bus to the **DOUT** bus are Ver.1 and Ver.2.

When connecting to an 8 or 16 bit LCD display, mapping V1 should be selected. When connecting to 9 bit LCD display, mapping V2 is recommended.

usLcdCmdMode value	Mapping mode
0	V1
1	V2

A third mapping option – RGB 565 - is also available and should only be used to convert RGB565 data from the host to RGB666 compatible output in the baseband. This third option is activated using the **TCIcdDataType** command (see below).

int TCIcdDataType(unsigned short usRGBAccess, unsigned short usRsState) ;

This function selects the default Data access mode to the LCD:

usRsState selects the default state of the register select pin (LCD_RS). The settings in this parameter only apply to accesses made by the TC5747 core to LCD. When the host controller accesses the LCD, the LCD_RS state will be controlled by the host, and this setting will be ignored.

usRGBAccess decides between standard command/data mode and RGB565 to RGB666 translation mode: The standard command/data mode is used by the host controlled to send data and commands to the controller. On applications that utilize an 18 bit (RGB666) display, it is required to translate the 16 bit RGB565 data to RGB666 data. This translation is done by TC5747 when **usRsState**=1 and should only be used for sending RGB data.

int TCIcdControlAccess(unsigned short usCont_su, unsigned short usCont_wr) ;

This function is used to control the access rate of TC5747 core to the LCD. It does not affect the direct pass-through interface from the host, or the video preview data rate.

usCont_su – controls the number of cycles the data is valid on the bus before the write signal is asserted. Valid values are 1-15.

usCont_wr - controls the duration of the write signal (in cycles). Valid values are 1-15.

- **int TCldWrite(unsigned short* pusData, unsigned short usWordCount) ;**

This function is used to control send data to the LCD using the TC5747 core to LCD access mechanism. It's main use is in applications where the pass-through interface is not used, and thus the host cannot control the LCD directly. **pusData** points to the data buffer to be sent to the LCD, and **usWordCount** indicates the number of words to be sent.

3.2.4 Live Video Output Functions

- **int TCOutputSize(unsigned short usWidth, unsigned short usHeight);**

In order to set the video preview image size, call this function with the desired image dimensions. The firmware will automatically adjust the image downscaling and cropping to fit the required size. Please consult the TC574X Data Sheet for details regarding the downscaling and cropping mechanisms, and for recommended preview image sizes. In any case, Width and Height must be even numbers.

Note

Always call this function after system initialization, power up or changing LCD.

- **int TCOutputFormat(TCformat eOutputFormat);**

This function should be used only when the output format of the LCD (as defined in **cust_def.h**) is to be changed.

Used for changing the output format of the current LCD to one of the supported formats. The supported formats (as they are defined in **TC574XApi.h**) are:

TCIF_FORMAT_DISABLE	TCIF_FORMAT_YUYV
TCIF_FORMAT_BAYER	TCIF_FORMAT_RGB565
TCIF_FORMAT_RGB666_1	TCIF_FORMAT_RGB444
TCIF_FORMAT_RGB666_2	TCIF_FORMAT_RGB888
TCIF_FORMAT_MJPEG	

- **int TCOutputEnable(unsigned short bEnable, unsigned short bWaitForEndPreview);**

Used to enable (set **bEnable** to 1) or disable (set **bEnable** to 0) the camera video output.

When disabling video out, and you need to wait (up to 200 msec) for video output to stop, set

bWaitForEndPreview to 1. If you need the function to return immediately set **bWaitForEndPreview** to 0.

3.2.5 Image Controls:

- **int TCbrightness(signed short sBrightness);**

Sets the image Target Brightness. Call this function with values between -128 and +128.

- **int TCcontrast(signed short sContrast);**

Sets the image Contrast. Call this function with values between -128 and +128.

- **int TCsaturation(signed short sSaturation);**

Sets the image Saturation. Call this function with values between -128 and +128.

- **int TChue(signed short sHue);**
Sets the image Hue. Call this function with values between -128 and +128.
- **int TCgamma(unsigned short usGamma);**
Sets the image gamma correction. Call this function with values between 0 and 255. The gamma is scaled to fit the range of 0 – 1.99. The default gamma setting is 102 (representing gamma 0.8).
- **int TCframeRate(unsigned short usFrameRate);**
Sets the video output frame rate. Valid frame rates are between 4-30 fps. Actual maximum frame rate may be limited by the system clock.
- **int TCautoFrameRateLimitations(unsigned short usMinFR, unsigned short usMaxFR);**
Sets the video output auto frame rate range. The Auto frame rate algorithm automatically selects the best frame rate within the specified range, according to the available light.
- **int TCdigitalEffect(TCeffect eEffect);**
Activates digital effects such as sepia, twilight, monochrome, and more. In order to activate such digital effects, call this function with one of the following options:

TCIF_NO_EFFECT	TCIF_MONOCHROME
TCIF_NEGATIVE_MONO	TCIF_NEGATIVE_COLOR
TCIF_SEPIA_GREEN	TCIF_SEPIA_BROWN
TCIF_SEPIA_YELLOW	TCIF_BINARY
TCIF_SOLARIZE_COLOR	TCIF_SOLARIZE_MONO
TCIF_TWILIGHT	TCIF_NIGHT

- **int TCsetZoom(TCzoom eZoom);**
In order to activate the zoom effect, call this function with one of the following optional values:
 - TCIF_NO_ZOOM
 - TCIF_X2_ZOOM

3.2.6 JPEG Capture Commands

The JPEG capture commands capture still JPEG images and download these images to the host.

- **int TCjpegQuality(TCquality eQuality);**
To set the JPEG capture quality (low medium or high), call this function with one of the following:

TCIF_JPEG_LOW_QUALITY	TCIF_JPEG_MEDIUM_QUALITY
TCIF_JPEG_HIGH_QUALITY	

Note

The Quality parameter determines the JPEG compression factor and subsequently the resulting file size.

- **int TCsetJpegSize(unsigned short usWidth, unsigned short usHeight, unsigned short bCropToPreview);**
 To set the JPEG image size, call this function with the desired image width and height of the output JPEG file. If you need to output only a section of the preview (a window in the center of the preview), **bCropToPreview** should be set to 1.
- **int TCjpegCapture(TCsource eSource, unsigned short bThumbnail, unsigned short usThumbWidth, unsigned short usThumbHeight);**
 This function performs the JPEG capture process. In order to initiate a JPEG capture, call this function with **eSource = TCIF_JPEG_SOURCE_LIVE**. **bThumbnail** decides whether thumbnail data will be added to the JPEG file. If thumbnail data is added, it can later be used to show thumbnail previews of the captured JPEG. The **usThumbWidth** and **usThumbHeight** functions define the thumbnail size. The recommended values for thumbnail sizes are 40x30 and 80x60.
- **int TCgetJpegStatus(unsigned short * pusStatus, unsigned short bBlocking):**
 The JPEG capture sequence is a time consuming process, Therefore, after sending a JPEG capture command, you must wait until the JPEG image is ready to be read out. There are two ways to do this:
 - a. **Blocking** – this function does not return until the JPEG is ready.
 - b. **Non-blocking** – the function returns immediately and the host can read the **pusStatus** in order to determine if the JPEG is ready or not. In this case the host might poll the JPEG ready flag on a regular basis (say every 20mSec) and stop when it is ready.
- **int TCgetJpegFileSize(unsigned short * pusFileSize);**
 This function returns the resulting JPEG image file size. It should be called only after the JPEG process is completed.
- **int TCgetJpegData(unsigned short * pusBuffer, unsigned short usWordCount, TCstate eState):**
 In order to read the JPEG file from the camera into the host controller, call this function with a buffer pointer and the buffer size. You may use this function to read the JPEG file in iterations. With each iteration you may read a block of data starting from 1 word (2 bytes) to the maximum JPEG file size.

Note

usWordCount represents the number of words in the buffer and *not* the number of bytes!

The first call to the function must be with **eState** assigned with **TCIF_START_TRANSFER**. Next calls should be with **TCIF_CONT_TRANSFER** and final call with **TCIF_END_TRANSFER**.

- **int TCjpegGetThumbnail(unsigned char* pucJpeg, unsigned char**ppucThumbnail, unsigned short* pusWidth, unsigned short* pusHeight);**
 When a JPEG file is captured with the Thumbnail option, the thumbnail data will contain a reduced-sized version of the captured picture. It displays a small size preview without decompressing the JPEG file.

The **TCjpegGetThumbnail** function receives a pointer to the JPEG file, and returns a pointer to the start of the thumbnail data within this file (the first pixel on the upper-left corner of the thumbnail image).

The thumbnail data is stored in YUV 4:2:2 format which is 2 bytes per pixel.

The thumbnail data size is therefore the following:

$$2 \times \text{Thumbnail_Width} \times \text{Thumbnail_Height}$$

After receiving the pointer to the thumbnail data, the data may be converted to RGB format by using either **TCconvertYUV422ToRGB888** or **TCconvertYUV422ToRGB565** function.

3.2.7 Snap Shot Capture Commands

The Snap Shot process uses snapshot capture commands and JPEG capture commands to generate a sequence of JPEG images and download these images to the host.

In order to perform the Snap Shot process, the application should call **TCSnapCapture** (see below). After calling this function, it should follow this procedure for each snapshot picture:

1. Poll **TCgetImageStatus** in iterations until it informs you that an image is ready
2. When an image is ready, you may wish to beep the user to indicate a “picture is taken”.
3. Poll **TCgetJpegStatus** in iterations until it informs you that a JPEG is ready.
4. Read the JPEG file using the functions **TCgetJpegFileSize**, **TCgetJpegData** as described in the previous section.

Repeat this procedure for the next picture.

- **int TCSnapCapture(unsigned short NPictures, unsigned short bThumbnail, unsigned short usThumbWidth, unsigned short usThumbHeight);**

This function initializes the Snap Shot process. **NPictures** defines the number of JPEG pictures that will be taken during the Snap Shot. **bThumbnail** decides whether thumbnail data will be added to each JPEG file (to show thumbnail previews of the captured JPEGs). The **usThumbWidth** and **usThumbHeight** functions define the thumbnail size. The recommended values for thumbnail sizes are 40x30 and 80x60.

- **int TCgetImageStatus (unsigned short * pusStatus, unsigned short bBlocking):**
During the Snap Shot process a time interval of at least 500 milliseconds is defined between each two pictures. There are two ways to wait for the next image:
 - a. **Blocking** – this function does not return until the image is ready.
 - b. **Non-blocking** – the function returns immediately and the host calls **TCgetImageStatus** in intervals until ***pusStatus** indicates the image is ready (when ***pusStatus** is non-zero)

3.2.8 JPEG Decompression Commands

- **int TCdjpegSettings(unsigned short usDestination, unsigned short bFit2Screen, unsigned short usWidth, unsigned short usHeight, TCorigin Origin);**
Set decompression parameters as follows:

- **usDestination** – The location to which the decompressed file will output:
(It is possible to combine several output destinations by ORing the following predefined values)
 - **TCIF_DJPEG_PVI** – decompress and sends out through PVI
 - **TCIF_DJPEG_CORE_MEMORY** – decompresses into core memory (for later re-compression with frame overlay.)
- **bFit2Screen** – When **True**– performs maximal downsampling of the input file to display on the LCD. If the downsampled image exactly fits the LCD display, no scrolling is required. If the downsampled image needs to be cropped to fit the LCD display, then the scrolling mechanism is applied (see the **Origin** parameter and the **TCdjpegDecompress** command).

When **False** – does not downsample at all. In this case, only a partial display window appears over the original file. This partial window is initialized in the location defined by the **Origin** parameter (see below) and the scrolling mechanism is applied. (see the **TCdjpegDecompress** command).
- **usWidth** and **usHeight** – are usually the same as preview size (LCD size) unless the image is to be decompressed into memory or the host processor initiated a smaller window on the LCD and wishes to display the image on a smaller window.
- **Origin** is used to set the start-point of the image window. Set **TCIF_DJPEG_ORIGIN_START** to define the start point at the top left of the image, or **TCIF_DJPEG_ORIGIN_CENTER** to define the start point in the center (center of the display window is located in the center of the original image) .

Note

If you call **TCdjpegSettings** after scrolling the preview window, the preview window is re-initialized to the **Origin**.

- **int TCdjpegRequest(TCtransfer eTransferType, unsigned short *pusMaxFileSize):**
Before loading a JPEG file into the camera (for decompression), you must request a decompression session and let the camera know about the data transfer method. The function will return the maximum allowed JPEG file size to be loaded on **pusMaxFileSize**.
eTransferType should be set to **TCIF_DJPEG_I2C_UPLOAD**.
- **int TCdjpegLoading(unsigned short * pusBuffer, unsigned short usWordCount, TCstate eState);**
This function downloads JPEG file data to TC574X.
pusBuffer is the pointer to the JPEG file buffer location in the host.
usWordCount is the buffer size to be transferred.
The function may be used to load the file in one shot, or in several iterations

To load the file in one shot use the following sequence:
TCdjpegLoading(jpg_buff_pointer, jpg_buff_size, TCIF_START_TRANSFER) ;
TCdjpegLoading(NULL, 0, TCIF_END_TRANSFER) ;

To load the JPEG in several iterations, a sequence such as the following should be used:
TCdjpegLoading(jpg_buffer[start], buffer_size, TCIF_START_TRANSFER) ;

```

TCdjpegLoading( jpg_buffer[next1], buffer_size, TCIF_CONT_TRANSFER) ;
TCdjpegLoading( jpg_buffer[next2], buffer_size, TCIF_CONT_TRANSFER) ;
//...as many iterations as necessary
TCdjpegLoading( NULL, 0, TCIF_END_TRANSFER ) ;

```

Please note that the first call to the function needs to be with **eState** assigned with **TCIF_START_TRANSFER**. Next calls with **TCIF_CONT_TRANSFER** and final call with **TCIF_END_TRANSFER**.

Note

usWordCount is the number of words (16 bit) in the buffer!

■ **int TCdjpegDecompress(TCmove eMove, unsigned short bWithOSD):**

This function is called to decompress the JPEG file loaded in the camera. The first time this function is called after **TCdjpegSettings** (with **eMove** passed as **TCIF_DJPEG_DONT_MOVE**), it defines the window to be displayed at the position defined in Origin sent in the **TCdjpegSettings** function. Subsequent calls to this function may be used to scroll the image according to the **eMove** passed, as one of the following:

- **TCIF_DJPEG_MOVE_UP** – to scroll up
- **TCIF_DJPEG_MOVE_DOWN** – to scroll down
- **TCIF_DJPEG_MOVE_LEFT** – to scroll left
- **TCIF_DJPEG_MOVE_RIGHT** – to scroll right

Use the parameter **bWithOSD** to decompress the image with frame overlay. For a clean image, this parameter should be **FALSE**.

■ **int TCdjpegGetMask (TCmove eMove, unsigned char *pMask);**

This function provides the possible directions permissible after the next **eMove** step. It is used in order to allow the application to load an appropriate OSD with the decompressed image.

The directions allowed after the next **eMove** step, are as follows:

Move Up - ***pMask** contains the bit 1 << **TCIF_DJPEG_MOVE_UP**
 Move Down - ***pMask** contains the bit 1 << **TCIF_DJPEG_MOVE_DOWN**
 Move Left - ***pMask** contains the bit 1 << **TCIF_DJPEG_MOVE_LEFT**
 Move Right - ***pMask** contains the bit 1 << **TCIF_DJPEG_MOVE_RIGHT**

■ **int TCgetDjpegStatus(unsigned short * pusStatus, unsigned short bBlocking);**

This function waits for the JPEG decompression process to end after **TCdjpegDecompress** has been called. If **bBlocking** is set, the function will only return after the JPEG process has been completed. If **bBlocking** is false, the function will return immediately and report the status through the **pusStatus** variable. If completed, the function will return the **TCIF_DJPEG_IMAGE_READY** value.

■ **int TCgetDjpegFileSize(unsigned short * pusFileSize):**

Returns the decompressed JPEG file size. Used to get the resulting image size after decompression to core memory.

- **int TCgetDjpegData(unsigned short * pusBuffer, unsigned short usWordCount, TCstate eState);**

Reads the decompressed JPEG file from the TC574X RAM; used after JPEG decompression to core memory. It can be used to read the data in one shot, or iteratively in several calls. If used in one shot, it should be called twice:

- The first time to do the actual transfer with **pusBuffer** set to the host buffer pointer, **usWordCount** set to the file size which was obtained through **TCgetDjpegFileSize** and **eState = TCIF_START_TRANSFER**.
- The second call is used only to terminate the transfer and should be called with **usWordCount=0** and **eState= TCIF_END_TRANSFER**.

If used to read the file in several iterations:

- The first iteration should be called with **TCIF_START_TRANSFER**;
- Consecutive iterations with **TCIF_CONT_TRANSFER**
- The terminating iteration with **TCIF_END_TRANSFER**.

The resulting file is always in YUV format. It may be converted later to RGB format using the **TCconvertYUV422ToRGB888** or **TCconvertYUV422ToRGB565** function.

- **int TCjpegMinimizeHeader(uint8 *pJpegFile,uint16 usFileSize, uint8 *pJpegOutputFile,uint16 *usFileOutputSize);**

Prepare a JPEG to be loaded into the camera by minimizing its header. It inserts a minimize version of a header that only holds the minimum information needed for decompression. If header file includes thumbnail or Exif header data they will be discarded as they do not have relevant information for the decompression process itself.

The function will fail if the input JPEG file is not compatible with TC574x. The reason for a non compatible file may be either sub-sampling not supported or non-default Huffman tables.

3.2.9 JPEG Postprocessing Commands

- **int TCjpegPpStartSession(void);**

This command informs the TC5747 about a new JPEG Postprocessing session.

- **int TCjpegPpModifyParam(TCJpegPPParamID eParam, signed short sValue);**

This command modifies each of the controllable parameters in the JPEG Postprocessing session. Each sending of this command can modify one parameter only.

- **eParam** – The parameter which is requested to be modified.
- **sValue** – The requested value for the selected parameter.

Parameter	Parameter description	Value range
TCIF_JPEGPP_BRIGHTNESS	Image brightness	-128 to +127
TCIF_JPEGPP_CONTRAST	Image contrast	-128 to +127
TCIF_JPEGPP_SATURATION	Image saturation	-128 to +127


```
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**).

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

ptcJpgsToAviFileCreateInfo points to a **TCjpgsToAviFileCreateInfoStruct** that describes all information needed to add 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 is called N times where N is the number of JPEG files to be converted into an AVI movie.
- **int TCjpgsToAviCloseFile(TCjpgsToAviFileCreateInfoStruct *ptcJpgsToAviFileCreateInfo, TCjpgsToAviFilesListStruct *ptcJpgsToAviFilesListStr, unsigned short usNumOfJpegFiles);**

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

ptcJpgsToAviFileCreateInfo points to a **TCjpgsToAviFileCreateInfoStruct** that describes all information needed to add the 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.

3.2.11 Motion JPEG (MJPEG) capture and decompression

The following commands are used to capture, save and decompress Motion Jpeg video files. A detailed description of MJPEG implementation and application example can be provided per demand.

The following function description is provided for reference:

- **int TCmjpegQuality(unsigned short usQuality);**

Sets the MJPEG quality factor. Accepts values from 1-99. Affects only the MJPEG that is sent to the PVI.
- **int TCmjpeg2mem(TCmjpeg2memStr *pTCmjpeg2memInfo, unsigned long *pulStartAddress);**

Used to initiate or stop a MJPEG capture to memory (streaming). It also returns the start address of the queue.

When initiating an M-JPEG capture (**bEnable** = TRUE) you need to pass a description of the queue: Number of buffers in queue and the buffer size (of 1 element). For example, if you know there are 50KB free to use in JPEG memory (see Note below) and each frame in the MJPEG is targeted to be ~6K you can initiate the queue to 10K buffer size and 5 buffers in queue. For bit rate control initiate the target frame size and the max frame size according to the requested compression ratio.

Note

The free JPEG memory to use is the same as the maximum JPEG capture limitations or the JPEG decompress limitations. It depends on the OSD usage since the OSD data base is allocated on the JPEG memory block.

- **int TCmjpegGetWriteCnt(unsigned short *pusWriteCnt);**
Used to test the queue condition and decide if there is a new frame ready. (If write **cnt** is bigger than read **cnt**.)
- **int TCmjpegGetFlagFull(unsigned short *pusFlagFull);**
Used to test the queue condition and decide if there is a queue full condition. If so it means we lost at least 1 frame and the system setting is not tuned correctly (we have real-time issues). If this condition happens frequently we need to decide either to capture with lower Quality factor (use **TCmjpegQuality**) to lower the frame rate or capture smaller images (for example, instead of QQVGA capture 128x120). All those parameters need to be tuned during system integration.
- **int TCmjpegSetReadCnt(unsigned short usReadCnt);**
Used to set the queue read count.
- **int TCmjpegClearFlagFull(void);**
Clears the flag full state in the DSC
- **int TCdmjpegRequest(TCdmjpegInfoStr *pdmjpegInfo, unsigned short usMaxFileSize);**
To initiate a MJPEG decompression process.
- **int TCdjpegLoading(unsigned short* pusBuffer, unsigned short usWordCount, TCstate eState);**
Allows JPEG file to be loaded into CORE memory for decompression. It may be used iteratively several times to load small parts of the file. In order to load a JPEG file the function has to be called at least twice. Once with **eState=START_TRANSFER** and the second time with **eState=END_TRANSFER** (so the CORE will know that the host has finished memory accesses). If the file is to be loaded in chunks then the first call would be with **START_TRANSFER**, the next few calls with **CONT_TRANSFER** and the last call with **END_TRANSFER** (with or without data).
- **int TCdmjpegDecompress(void);**
Decompress the frame that was previously loaded by **TCdjpegLoading**.
- **int TCdmjpegEndProcess(void);**
Ends a M-JPEG decompression process.

3.2.12 OSD commands:

The TC574X OSD features allow the overlay of various icons and images over the live video preview, and over the captured JPEG and M-JPEG images. The OSD feature is mainly used to display auxiliary data such as icons, time/date, etc.

The OSD functionality in a particular project is dependent upon the existence of an OSD database that is compatible with the project settings and requirements (such as preview screen size, required icons design, fonts, etc). A sample OSD database is available together with the TC574X API release. This sample database can be used as a base for customization.

A detailed description of the OSD features and application can be supplied per demand. The following function description is provided for reference:

- **int TCosdInit(void):**
Should be called once to initiate the regions of the OSD (frame, header bar and footer bar).
- **int TCosdReset(void);**
Sets all OSD regions in FW to zero. May be used to allow more memory for JPEG post processing commands. If OSD is needed again, must call TCosdInit again.
- **int TCosdLoadFrame(unsigned char frame):**
Loads a frame overlay. The 'frame' parameter is an index within the frames and icons database.
- **int TCosdEnableFrame(unsigned char bEnable):**
Once a frame is loaded, it may be enabled or disabled using this function.
- **int TCosdEnableIcon(unsigned char icon,unsigned char bEnable):**
To enable or disable an icon, use this function and pass it the icon index and a TRUE or FALSE parameter to indicate if the icon is to be enabled or disabled. Missing a function to load an icon is not an oversight. The icons are built by the SDK on a specific bar (the icons database stores the relevant information, such as the bar on which the icon is placed, its coordinates, etc.) and the bar as a whole is loaded into the camera (TCosdEnableBar loads the bar into the camera if the bar content has been changed).
- **int TCosdEnableBar(unsigned char barTag, unsigned char bEnable):**
To enable or disable a bar, use this function with the bar index and a TRUE\FALSE parameter.
- **int TCosdLoadCursor(unsigned char cursorTag):**
Loads a cursor into the camera. Receives a cursor index (from database).
- **int TCosdEnableCursor(unsigned char bEnable):**
Enables a cursor that was previously loaded. The cursor would be displayed as an overlay above all regions (live preview, frame overlay and bars (icons)).
- **int TCosdSetCursor(unsigned short xPos, unsigned short yPos):**
Sets (or moves) a cursor position on the LCD.
- **int TCosdSetTimeStamp(TClineNum eLine, char* timeStamp);**

Builds and enables the timestamp bar. The timestamp bar contains two lines. On each line the combination of the characters 0,1,2,3,4,5,6,7,8,9,A,P,M,-,: and space may be displayed. Each line is limited to 10 characters. The **eLine** parameter is called with the required line to be updated, and the **timeStamp** contains a null terminated string which is composed of the characters listed above.

■ **int TCosdDisableTimeStamp(void);**

Disables the timestamp (both lines).

■ **int TCsetOrientation(uint16 usChipOrientation);**

Usually the chip orientation is a property of the active preview LCD. For example, on the main LCD we have the horizontal chip orientation (TCIF_LENS HOLDER_RIGHT) and on the sub LCD it is 180° rotated (TCIF_LENS HOLDER_LEFT). When preview is made with OSD it is important for the FW to know the chip orientation because in our example header and footer should change their places. As long as simple actions are made, e.g. open preview on main LCD, close preview, set size to sub LCD and open preview on sub LCD, the FW can handle the OSD regions location changes. On a more complex situation, e.g. JPEG file captured from sub LCD preview and decompressed on the main LCD we need to change the chip orientation for the decompression session. If we take the above example again, we will do the following sequence:

1. Change chip orientation to TCIF_LENS HOLDER_RIGHT,
2. Update the OSD icons (e.g. “save” and “back”),
3. Call decompress function.
4. Change chip orientation back to TCIF_LENS HOLDER_LEFT. (to the previous preview).

■ **int TCosdEnableFMitem(TcFloatingMenuItemStruct *pFMitem);**

Places a floating menu item on the frame region.

■ **int TCosdClearFMitem(TcFloatingMenuItemStruct *pFMitem);**

Clears a floating menu item on the frame region.

Both floating menu functions uses the following structure:

typedef struct

```
{
    CoordStruct    pos;
    CoordStruct    size;
    const uint8    *pFloatingMenuItem;
    uint16         usItemSize;
} TcFloatingMenuItemStruct;
```

For more details see “Floating menu Reference Design.pdf”.

3.2.13 Platform Dependent Custom Settings Functions

The following functions are used to customize the TC574X hardware interface to a specific platform hardware configuration. Typically, these functions will be called following the camera firmware uploading and core initialization, to set the specific platform configuration options.

■ **int TCvalidVpolr(unsigned short bLow):**

Sets the **Vsync** output polarity with a Boolean value. 1 sets **Vsync** to active low and 0 sets it to active high.

■ **int TCuseChipSelect(void):**

Enables the camera CS input pin as the data bus enabling signal. After this function is called, the CS input will control whether the camera data bus will be active or in high-impedance mode. If this function is not called, the status of the bus will be controlled by software, using the **TCoutputFormat** function.

■ **int TCsetValidHasWriteSignal(unsigned short bActiveLow , unsigned short bSplitClocks):**

When using the camera for direct LCD streaming (camera connected to, and streaming the data directly into the LCD) the camera must be configured to have a write signal for each data it outputs (**~WR**). Calling this function enables the **ValidH** pin of the camera to be a write signal (AKA qualified clock). **bActiveLow** enables the host to determine the pin logic behavior. Since the camera data bus is usually 8 bytes, the **WR** signal is for each byte, so two **WR** signals will be output for each pixel (assuming RGB565). If a 16-bit interface is needed, the **WR** signal may be split into **WRhigh** and **WRlow**. In this case, call this function with **bSplitClock** equals TRUE. **MSB** clock goes out first through **VCLKOUT**, **LSB** clock through **VALIDH** (used as WR signal to the 16-bit **LCM**), assuming the **MSB** was latched with an 8-bit latch.

■ **int TCsetClkEdgeForPVI(unsigned short bNegativeEdge);**

Determines the clock edge on which the data is changed for output to the LCD via the PVI. Input 1 stands for negative edge and input 0 stands for positive edge.

3.2.14 Miscellaneous Commands

■ **int TCjpegRotation(unsigned char *pJpegFile,unsigned short usFileSize,
unsigned char *pRotatedJpegFile,unsigned short
*usRotatedFileSize, TCrotate eDirection);**

Rotates a downloaded JPEG file 90° left (ccw, counter clockwise) or right (cw, clockwise). Its main use is on platforms where the TC574X sensor is rotated 90° when installed, and therefore the captured image must be rotated before sending it for display on a PC or other platform.

Note

This function runs on the host controller, after the file has been downloaded from TC574X to the host. Also, in order to optimize performance, this function is included as a platform dependent library function. To use this function you have to include the **Jtlib.lib** library file in the host project.

Function usage is as follows:

- ***pJpegFile** points to the (unrotated) input file.
- **usFileSize** is the input file size.
- ***pRotatedJpegFile** points to an allocated buffer for the output file
- ***usRotatedFileSize** is the allocated buffer size. The allocated buffer size should be at least the input file size + 4KB. The function modifies this value to the exact rotated file size after rotation completed.

- **eDirection** is the required rotation direction. It may be set as **TCIF_ROTATION_RIGHT** (clockwise) or **TCIF_ROTATION_LEFT** (counter-clockwise). After the function is executed, the rotated file will reside in the output buffer.
- Depending on the host performance and the input file size, this function may require a few seconds to complete.

- **int TCjpegFixOrientation(unsigned char *pJpegFile,
 unsigned short usFileSize,
 unsigned char *pFixedJpegFile,
 unsigned short *usFixedFileSize);**

This function detects the file orientation and sets it as a straight JPEG. It is intended to replace the former TCjpegRotation function by providing a full solution to all camera orientations. The current implementation is partial and only supports 180° rotation of an image that was captured from sub-LCD that used mirror X for preview and 90° rotation right. Also note that this function depends on HW rotation that is only available on TC5747.

Function usage is as follows:

- ***pJpegFile** points to the input file.
- **usFileSize** is the input file size.
- ***pFixedJpegFile** points to an allocated buffer for the output file
- ***usFixedFileSize** is the allocated buffer size. The allocated buffer size should be at least the input file size + 4KB. The function modifies this value to the exact rotated file size after rotation is completed.

- **void TCconvertYUV422ToRGB888(unsigned char *pYUV422buffer,
 unsigned char *pRGB888buffer,
 unsigned short usWidth,
 unsigned short usHeight);**

When a JPEG capture is made with **bThumbnail=TRUE**, the thumbnail is saved in YUV422 format. If it is required to display the thumbnail on a RGB device, this function may be used to convert the thumbnail to RGB888 format. RGB888 may subsequently be converted to other RGB formats (such as RGB 666 or RGB 565). **pYUV422buffer** points to the input image, and **pThumbRGB888** points to the output thumbnail image. This function is also used when decompressing a JPEG file into memory (e.g. for a wallpaper function). The decompression to memory format is YUV422.

- **void TCconvertYUV422ToRGB565(unsigned char *pYUV422buffer,
 unsigned short *pRGB565buffer,
 unsigned short usWidth,
 unsigned short usHeight);**

When a JPEG capture is made with **bThumbnail=TRUE**, the thumbnail is saved in YUV422 format. If it is required to display the thumbnail on an RGB device, this function may be used to convert the thumbnail to RGB565 format. The function gives an optimal implementation for converting

YUV422 into RGB565 and replaces the need to call YUV422->RGB888->RGB565.

pYUV422buffer points to the input image, and **pRGB565buffer** points to the output thumbnail image. This function is also used when decompressing a JPEG file into memory (e.g. for a wallpaper function). The decompression to memory format is YUV422.

- **int TCdjpgParseJpegHeader(unsigned char *pJpegFile, unsigned short buffSize, TCjpegHeaderInfoStruct *pJpegHeaderInfo);**

Provides information on the JPEG file. The function is passed a pointer to the JPEG file. The entire JPEG header should reside in memory for the function to work.

The function returns a **TCjpegHeaderInfoStruct** which holds the following information about the file:

jpegWidth	JPEG width
jpegHeight	JPEG height
lumQToffset	offset of the QT table
chromQToffset	offset of the chromQ table
samplingFactor	sub-sampling factor
headerSize	JPEG header size in bytes
bTransChipFile	TRUE if the file source is TransChip camera
bTCCompatible	TRUE if the JPEG can be decompressed by Camera HW.
usRSTmarkers	Restart markers interval (0 if no RST used)
*pChipOrientation	Pointer to chip orientation inside header. The reason it is a pointer is to allow the fix orientation function to clear this value after fixing the image orientation (0 means a straight image).
bMjpegFile	TRUE if it is a MJPEG frame.

- **int TCbackUpCalibrationTables(TCcalBackupStateEnum eState);**

Calibration tables are a set of parameters created by the TC574X firmware during the initialization phase (when the **TCstartCore** command is called). These tables store sensor specific calibration data to reduce image noise. Since the process of creating these tables may be time consuming (up to 4 seconds) it is sometimes necessary to skip the calibration process. The **TCbackUpCalibrationTables** function allows the user to skip the calibration state by saving the sensor calibration data in the host. Therefore, after the first power-up and initialization with calibration (calibration cannot be skipped on the first initialization), it is possible to back up the calibration tables by calling the following:

- **TCstartCore(usClockInKhz, FALSE);**//Start core with calibration

- **Do**

Wait();

TCgetCoreReadyStatus(&CoreReady);

While (CoreReady == 0);

- **TCbackUpCalibrationTables (TC_BKUP_INIT);**
- **TCbackUpCalibrationTables (TC_BKUP_READ);**//save calibration table

Later, if the sensor was reset, it is possible to start the core and reload the calibration tables quickly by the following sequence:

- **TCstartCore(usClockInKhz, TRUE);**//Start core without calibration

- Do

Wait();

TCgetCoreReadyStatus(&CoreReady);

While (CoreReady == 0);

TCbackUpCalibrationTables (TC_BKUP_WRITE); is called within the function TCstartCore (if called with fast init) and should not be called from outside.

■ **int TCflashLight(unsigned short usFlashType, TCFlashStatus eFlashMode);**

Controls the TC574X flash light output pin. usFlashType parameter accept values of:

- TCIF_FLASH_CONTINUOUS – Used for continuous flash mode. This mode is usually used for preview (AKA movie mode on a flash driver device such as MAX1583¹)
- TCIF_FLASH_PULSE – Used for pulse or burst flash mode. This mode is usually used in capture mode (AKA Strobe mode on a flash driver device such as MAX1583). This mode has a very strict current time limitations. According to the flash driver specification and the LED used, the application needs to pass to the DSC FW the maximum pulse duration allowed in this mode as well as the duty-cycle. For example the LED LT5K63-AB-3W-T01-1 has a limitation of 1/10 duty cycle and 0.1mSec pulse width (peak current of 300mA). (Currently not supported by FW)
- TCIF_FLASH_MULTI_USE – Used to enable both modes at the same time. For example if both modes are set to automatic mode then in low light conditions the continuous LED mode would be used (while preview) and during capture there will be a burst pulse on the LED (strobe) that will be timed by the DSC and the duration can be controlled by an API function TCconfigureFlashPulse. (Currently not supported by FW)

eFlashMode gets the following values:

- TCIF_FLASH_DISABLE – Turns the LED off.
- TCIF_FLASH_ENABLE – Turns the LED on.
- TCIF_FLASH_AUTO – Enable the auto flash mode; Lets the DSC decide when to turn the LED on or off.

■ **int TCgetContFlashState(unsigned short *pbContFlashActive);**

When TCIF_FLASH_MULTI_USE is needed, the base-band (application) needs to poll the DSC say every 0.5-1 second. If it sees that the continuous flash is active it should turn it on, otherwise turn it off. The DSC will control the burst mode (strobe) only while capture if it recognizes a low light condition. The burst would be accurately timed within the DSC FW. (Currently not supported)

¹ See: <http://pdfserv.maxim-ic.com/en/ds/MAX1583.pdf>

- **int TCconfigureFlashPulse(unsigned long ulMaxDuration,
 unsigned long ulMinDelay,
 unsigned short usMaxBursts
 unsigned short bEnableRedEyeReduction);**
ulMaxDuration – maximum pulse duration in μSec units (10^{-6}).
ulMinDelay – minimum delay between the last time the LED was on until the next time it is allowed to turn it on again.
usMaxBursts – maximum number of bursts. This value needs to be greater than 2 for a red-eye reduction mode.
bEnableRedEyeReduction – Enable or disables the red eye reduction feature. If enabled make sure usMaxBursts value is greater than 2.
For example the LED LT5K63-AB-3W-T01-1 has ulMaxDuration=100 and ulMinDelay=900 (This will have a 1/10 duty cycle).
(Currently not supported by FW)
- **int TCconfigureFlashAutoLevels(unsigned short usContLampOnThresh,
 unsigned short usIlluminationAt50cm,
 unsigned short usPulseLampOnThresh,
 unsigned short usPulseLampsDominantThresh);**
(Currently not supported by FW)
- **int TCconfigureFlashSpecialSettings(void);**
(Currently not supported by FW)
- **int TCsetFlashPolarity(uint16 bActiveHigh);**
Sets the polarity of the signal that turns the LED flash on.
- **int TCfwVersion(unsigned short *usFWver, unsigned short *usFWdate, s
 unsigned short *usAPIver);**
Gets the firmware version number, date and API version number.

4. Implementing Platform-specific HW Driver Functions

In order to enable the operation of the TransChip API, the host should implement a set of platform-specific service functions. These functions include the following:

- I²C/Parallel host interface implementation.
- Power saving GPIO pins control
- Delay function
- Watchdog control function

The function prototypes are defined in **TCApi.h**. The following is a detailed explanation of the required implementation of each function:

- **int HostTCTransferDataProc(void* pBuffer, unsigned short usBufferSize, TC_ACCESS_TYPE tcAccessType) ;**

This function should implement the parallel or I²C Interface protocol:

- **pBuffer** points to a memory buffer containing data to write/read.
- **usBufferSize** holds the number of bytes in the buffer.
- **tcAccessType** is the type of access, which may be Read or Write.

Sample I²C functions for various platforms are available from TransChip.

When implementing this function with the parallel interface, it is important to distinguish between host to TC574X access (this function) - which is qualified by the **HCS_N** pin, and host to LCD access (not related to this function), which is qualified by the **HLCD_CS_N** input pin. Refer to the TC574X datasheet for more details.

- **int HostTCgpioControl(TC_GPIO_PIN tcPin, unsigned char bSet);**

This function implements control over the discrete pins that control the camera Power Save mode. There are 4 pins that need to be controlled:

- a. **PS1** pin
- b. **PS2** pin
- c. **CLK_EN** pin
- d. **RESET** pin

tcPin indicates the pin over which control is requested, and **bSet** is a Boolean that indicates the required state of the pin, when TRUE indicated logic '1' and FALSE indicated logic 'LOW'. The function should set the required pin to the required state.

- **void HostTCdelayMSec(unsigned short imSec);**

This function is called to implement real-time delay. **imSec** indicates the delay in milliseconds. The delay should be implemented with accuracy better than 10%.

- **void HostTCresetWatchDog (void);**

This function should be implemented in case the host operating system has an internal watchdog that will reset the system if it is not called in a timely manner. **TCApi** function will call this function before operations that may require considerable time to complete (such as firmware uploading). If there is no such watchdog in the system, the function should remain blank.

5. Appendix A: I²C Serial Interface

5.1 Overview

The I²C interface is a two-wire bi-directional serial bus (16 bit data, 16 bit address). The TC574X can operate as a slave device only.

Both wires (**SCLK** and **SDIN**) are connected to a positive supply via a pull-up resistor, and when the bus is free, both lines are high. The output stage of the device must have an open-drain or open collector type IO cell so that a wired-AND function can be performed between all devices connected to the bus.

Each device is recognized by a unique 7-bit address.

The address allocated to TC5747 is **0x47**. When performing write operations, add 0 to the LSB to get **0x8E**. For a Read operation add **1** to the LSB to get **0x8F**.

The table below summarizes:

Address	I2C Write	I2C Read
0x47	0x8E	0x8F

The address allocated to TC5740 is either **0x56** or **0x57** (configurable Pin at Reset). When performing write operations, add 0 to the LSB to get **0xAC** or **0xAE** respectively. For a Read operation add **1** to the LSB to get **0xAD** or **0xAF** respectively.

The table below summarizes:

Address	I2C Write	I2C Read
0x56	0xAC	0xAD
0x57	0xAE	0xAF

Notes

Some operating systems automatically add the 1 or 0 bit to the 7 bit address.

5.2 Mode of Operation

The bus master, typically the host DSP, initiates an access to the TC574X device. The bus master activates a START condition, and passes the address of the requested device along with the type of access (read or write bit – the MSB or the start byte). The requested device acknowledges (ACK). The host is free to perform transactions until the host issues a STOP condition. The bus is considered free after the STOP condition.

The data on the **SDIN** pin must be stable during the high period of the clock (**SCL**) as shown in the figure below. Only the host may change data while SCL is high. A high-to-low transition marks a START condition, and a low-to-high a STOP condition.

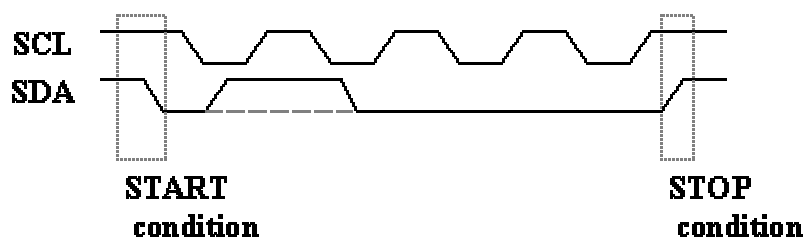


Figure 2: I2C interface START and STOP Conditions

The master device activates a START condition, and sends the first byte of data that contains the 7-bit address together with a direction bit (R/W#, 1 for read, 0 for write). The addressed device acknowledges by pulling down the SDA line.

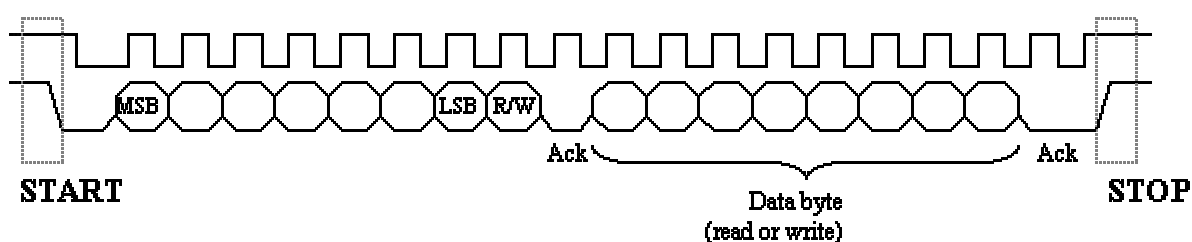


Figure 3: I2C Protocol

The TC574X expects the first two bytes after the address byte to be the register address of the first register that is to be read or written. **The most significant byte of the address is sent first.**

When writing registers to the TC574X, the words that follow are data (the MSB of the word is sent first). The TC574X performs auto increment until a STOP condition is detected. Auto increment is not performed when the initial address is in the address space allocated to the On-Chip-Memories (0x0F00-0x0FFF). This address space is reserved for tables, where multiple bytes are written to a single address.