TMS320C55x Image/Video Processing Library Programmer's Reference

Preliminary

Literature Number SPRU037C
January 2004

PRINTED WITH SOY INK™

TEXAS INSTRUMENTS

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Preface

Read This First

About This Manual

Welcome to the TMS320C55x™ image/video Library, or IMGLIB for short. The IMGLIB is a collection of 31 high-level optimized DSP functions for the TMS320C55x device. This source code library includes C-callable functions (ANSI-C language compatible) for general-purpose imaging functions that include compression, video processing, machine vision, and medical imaging type applications.

This document contains a reference for the IMGLIB functions and is organized as follows:

- Overview - an introduction to the TI C55x IMGLIB
- Installation - information on how to install and rebuild IMGLIB
- IMGLIB Functions - a description of the routines in the library and how they are organized
- IMGLIB Function Tables - a list of functions grouped by categories
- IMGLIB Reference - a detailed description of each IMGLIB function
- Information about performance, warranty, and support

How to Use This Manual

The information in this document describes the contents of the TMS320C55x IMGLIB in several different ways.

Chapter 1 provides a brief introduction to the TI C55x IMGLIB, shows the organization of the routines contained in the library, and lists the features and benefits of the IMGLIB.

Chapter 2 provides information on how to install, use, and rebuild the TI C55x IMGLIB.

Chapter 3 provides a brief description of each IMGLIB function.

Chapter 4 provides information about each IMGLIB function in table format for easy reference. The information shown for each function includes the syntax, a brief description, and a page reference for obtaining more detailed information.

Chapter 5 provides a list of the routines within the IMGLIB organized into functional categories. The functions within each category include arguments, descriptions, algorithms, benchmarks, and special requirements.

Appendix A describes performance considerations related to the C55x IMGLIB and provides information about warranty issues, software updates, and customer support.

Notational Conventions

This document uses the following conventions:

Program listings, program examples, and interactive displays are shown in a `special typeface`.

In syntax descriptions, the function or macro appears in a **bold typeface** and the parameters appear in plainface within parentheses. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are within parentheses describe the type of information that should be entered.

Macro names are written in uppercase text; function names are written in lowercase.

The TMS320C55x is also referred to in this reference guide as the C55x.

Related Documentation From Texas Instruments

The following books describe the TMS320C55x devices and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number. Many of these documents can be found on the Internet at http://www.ti.com.

TMS320C55x Technical Overview (SPRU393). This overview is an introduction to the TMS320C55x™ digital signal processor (DSP). The TMS320C55x is the latest generation of fixed-point DSPs in the TMS320C5000™ DSP platform. Like the previous generations, this processor is optimized for high performance and low-power operation. This book describes the CPU architecture, low-power enhancements, and embedded emulation features of the TMS320C55x.

TMS320C55x DSP CPU Reference Guide (literature number SPRU371) describes the architecture, registers, and operation of the CPU for the TMS320C55x™ digital signal processors (DSPs).

TMS320C55x DSP Algebraic Instruction Set Reference Guide (literature number SPRU375) describes the TMS320C55x™ DSP algebraic instructions individually. Also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the mnemonic instruction set.

TMS320C55x DSP Mnemonic Instruction Set Reference Guide (literature number SPRU374) describes the TMS320C55x™ DSP mnemonic instructions individually. Also includes a summary of the instruction set, a list of the instruction opcodes, and a cross-reference to the algebraic instruction set.

TMS320C55x Programmer's Guide (literature number SPRU376) describes ways to optimize C and assembly code for the TMS320C55x™ DSPs and explains how to write code that uses special features and instructions of the DSP.

TMS320C55x Assembly Language Tools User's Guide (literature number SPRU280) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for TMS320C55x™ devices.

TMS320C55x Optimizing C Compiler User's Guide (literature number SPRU281) describes the TMS320C55x™ C Compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for TMS320C55x devices.

TMS320C55x Hardware Extensions for Image/Video Application Programmer's Reference (literature number SPRU098) describes the built-in video hardware extensions for TMS320C55x™ DSPs and explains how to implement them for video application.

Trademarks

TMS320C55x is a trademark of Texas Instruments.

Contents

Figures

Tables

Chapter 1

Introduction

This chapter introduces the TMS320C55x Image/Video Library (IMGLIB) and describes its features and benefits.

Topic **Page**

1.1 Introduction

The TI C55x IMGLIB is an optimized image/video processing functions library for C programmers using TMS320C55x devices. It includes many C-callable, assembly-optimized, general-purpose image/video processing routines. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines, you can achieve execution speeds considerably faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use DSP functions, TI IMGLIB can significantly shorten your image/video processing application development time.

1.2 Features and Benefits

The TI C55x IMGLIB contains commonly used image/video processing routines. Source code is provided that allows you to modify functions to match your specific needs.

IMGLIB features include:

- Optimized assembly code routines
- C-callable routines fully compatible with the TI C55x compiler
- Support large memory model
- Benchmarks (cycles and code size)

1.3 Software Routines

The rich set of software routines included in the IMGLIB are organized into three different functional categories as follows:

- Compression and Decompression
- Image Analysis
- Picture Filtering/Format Conversion

Chapter 2

Installing and Using IMGLIB

This chapter provides information on how to install and rebuild IMGLIB.

Topic **Page**

2.1 IMGLIB Contents

The TI IMGLIB software consists of several parts:

imglib.h, a header file for C programmers

image_sample.h, a header file that contains an image sample

wavelet.h, a header file for wavelet functions

55ximagex.lib, an object library to support large memory model

55ximage.lib, an object library to support small memory model

55ximage.src, a source library to allow function optimization by the end user

Example programs and linker command files used under the *examples* subdirectory.

After following the instructions in section 2.2, *How to Install IMGLIB*, the IMGLIB directory structure and content you will find is:

IMAGELIB	Directory containing these C55x IMGLIB files:
55ximage.lib	Library file to use for small memory model
55ximagex.lib	library file to use for large memory model
blt55x.bat	Generate 55ximage.lib for small memory model based on 55ximage.src
blt55xx.bat	Generate 55ximagex.lib for large memory model based on 55ximage.src
readme.txt	Release notes
examples	Directory containing examples covering all routines included in the library
include	Directory containing the following include files:
imagelib.h	Include file with data types and function prototypes
wavelet.h	Include file with five groups of wavelet filter banks
imagesample	An image sample (128x128 Goldenhill)
55x_src	Directory containing assembly source files for functions

Warning

For the large memory model (55ximagex.lib), all the functions are supported. For the small memory model (55ximage.lib), the IMG_jpeg_vlc, IMG_jpeg_vld, IMG_sw_dct_8x8, and IMG_sw_idct_8x8 are not supported.

2.2 How to Install IMGLIB

You should read the README.txt file for specific details of the release.

IMGLIB is distributed in two forms:

- as part of Code Composer Studio in the c:\ti\c5500\imglib directory, or
- as a zip file (external web download at www.ti.com)

The zip file automatically restores the IMGLIB individual components in the same directory where you unzip the file. See section 2.1, *IMGLIB Content*, for a list of the directories and files contained in the ZIP file.

The IMGLIB root directory contains the library archive and the source archive. In order for the linker to find the library, you can update the C_DIR environment variable,for example, by adding the following line in autoexec.bat file:

```
SET C_DIR=<install_dir>/lib;<install_dir>/include;%C_DIR%
```

or under Unix/csh:

```
setenv C_DIR ”<install_dir>/lib;<install_dir>/
include; $C_DIR”
```

or under Unix/Bourne Shell:

```
C_DIR=”<install_dir>/lib;<install_dir>/include;$C_DIR”
   ; export C_DIR
```

You can also relocate the library file by copying the C55x IMGLIB object library file, 55ximage.lib, or 55ximagex.lib to your C5500 run-time-support library folder.

For example, if your TI C5500 tools are located in *c:\ti\c5500\cgtools\bin*, and your C run-time-support libraries (rts55x.lib etc) are located in *c:\ti\c5500\cgtools\lib*, copy 55ximage.lib to the second folder. Relocating the library file allows the C55x compiler/linker to find C55x imaging libraries.

2.3 How to Rebuild IMGLIB

For a full rebuild of 55ximage.lib, execute the blt55x.bat file. For a full rebuild of 55ximagex.lib, execute the blt55xx.bat file.

A partial rebuild of IMGLIB enables you to modify a specific IMGLIB function. As an example of a partial rebuild of 55ximagex.lib, follow these steps:

1) Extract the source for the selected function from the source archive. For example to alter swdctal.asm, you would enter:

   ```
   ar55 x 55ximage.src swdctal.asm
   ```

2) Assemble your new source file. For example:

   ```
   cl55 -mg -ml swdctal.asm
   ```

3) Replace the original object file in the 55ximagex.lib object library with the newly formed object file. For example:

   ```
   ar55 r 55ximagex.lib swdctal.obj
   ```

2.4 Calling an IMGLIB Function From C

In addition to correctly installing the IMGLIB software, you must follow these steps to include an IMGLIB function in your code:

Include the *imagelib.h* header file

Link your code with the IMGLIB object code library, *55ximage.lib* or *55ximagex.lib*

Use a correct linker command file describing the memory configuration available in your C55x board

For example, the following code contains a call to the histogram routine in IMGLIB:

```
#include <studio.h>
#include <stdlib.h>
#include <imagelib.h>
#include “imagesample.h”

#define MAX_PIXEL_VALUE 256
#define WIDTH 128
#define HEIGHT 128

void main()
{
    int     i
    int     size
    int     *input, *output

    size = WIDTH*HEIGHT

    input = &goldhill[0][0];
    output = (int *)malloc((size_t)(MAX_PIXEL_VALUE*sizeof(int)));

    // Initialize the histogram bins
    for(i=0; i<MAX_PIXEL_VALUE; i++)
        output[i] = 0;

    histogram(input,output, size);
}
```

In this example, the histogram IMGLIB function is used to compute the histogram of a sample image. The example code, histogram.c, is located in the *\example\histogram* subdirectory. To compile and link this code with 55ximage.lib, enter the following command:

```
cl55 -pk -g -o3 -i histogram.c -z -v0 ld3.cmd 55ximage.lib -m
hgram.map -o hgram.out
```

The examples presented in this document have been tested using the Texas Instruments C55x emulator.

Refer to the *TMS320C55x Optimizing C Compiler User’s Guide* for information about the compiler tools.

2.5 Calling an IMGLIB Function from Assembly Language Source Code

The TMS320C55x IMGLIB functions were written to be used from C. Calling the functions from assembly language source code is possible as long as the calling function conforms to the Texas Instruments C55x C compiler calling conventions. Refer to the *TMS320C55x Optimizing C Compiler User's Guide* if a more in-depth explanation is required.

Realize that the TI IMGLIB is not an optimal solution for assembly-only programmers. Even though IMGLIB functions can be invoked from an assembly program, the result may not be optimal due to unnecessary C-calling overhead.

2.6 Where to Find Sample Code

You can find examples on how to use every function in the C55x IMGLIB in the *examples* subdirectory. The examples that cover the functions in C55ximage.lib are as follows:

Example	Function(s)
swdctidct	IMG_sw_fdct_8x8
	IMG_sw_idct_8x8
dctidct	IMG_fdct_8x8
	IMG_idct_8x8
Quantize	IMG_jpeg_make_recip_tbl
	IMG_jpeg_quantize
	IMG_dequantize_8x8
jpeg_vlc	IMG_jpeg_vlc
jpeg_vld	IMG_jpeg_vld
motion_estimation	IMG_mad_16x16_4step
mad_8x8	IMG_mad_8x8
mad_16x16	IMG_mad_16x16
sad_8x8	IMG_sad_8x8
sad_16x16	IMG_sad_16x16
pixel_interpolation	IMG_pix_Inter_16x16
1D_wavelet	IMG_wave_decom_one_dim
	IMG_wave_recon_one_dim
	IMG_wavep_decom_one_dim
	IMG_wavep_recon_one_dim
2D_wavelet	IMG_wave_decom_two_dim
	IMG_wave_recon_two_dim
	IMG_wavep_decom_two_dim
	IMG_wavep_recon_two_dim
Boundary	IMG_boundary
Histogram	IMG_histogram
Perimeter	IMG_perimeter

Example	Function(s)
Threshold	IMG_threshold
conv_3x3	IMG_conv_3x3
corr_3x3	IMG_corr_3x3
Color_conversion	IMG_ycbcr_rgb565
IMG_Scale_by_2	IMG_scale_by_2

Chapter 3

IMGLIB Function Descriptions

This chapter provides a brief description of each IMGLIB function listed in three categories. It also gives representative examples of their areas of applicability.

Topic **Page**

3.1 IMGLIB Functions Overview

The C55x IMGLIB provides a collection of C-callable high performance routines that can serve as key enablers for a wide range of image/video processing applications. These functions are representative of the high performance capabilities of the C55x DSP. Some of the functions provided and their areas of applicability are listed below. The areas of applicability are only provided as representative examples. Users of this software will surely conceive many more creative uses.

3.2 Compression/Decompression

The following *Forward and Inverse DCT* (Discrete Cosine Transform) functions, with and without built-in hardware extension, are provided:

IMG_fdct_8x8
- IMG_sw_fdct_8x8
- IMG_idct_8x8
- IMG_sw_idct_8x8

The above four functions have applicability in a wide range of compression standards such as JPEG Encode/Decode, MPEG Video Encode/Decode, H.26x Encode/Decode. These compression standards are used in diverse end-applications such as:

- JPEG is used in printing, photography, security systems, etc.
- MPEG video standards are\ used in digital TV, DVD players, Set-Top Boxes, Video-on-Demand systems, Video Disc applications, Multimedia/Streaming Media applications, etc.
- H.26x standards are used in Video Telephony and some Streaming Media applications.

IMG_mad_8x8

IMG_mad_16x16

IMG_sad_8x8

IMG_sad_16x16

IMG_mad_16×16_4step

IMG_pix_inter_16×16

The first five functions enable high performance motion estimation algorithm usage in applications such as MPEG Video Encode, or H26x Encoder. The IMG_pix_inter_16×16 function enables high performance pixel interpolation algorithm usage by fractal-pixel motion estimation in video encoding. Video encoding is useful in video-on-demand systems, streaming media systems, video telephony, and other video systems. Motion estimation is typically one of the most computation-intensive operation in video encoding systems. The high performance enabled by the functions provided can enable significant improvements in such systems.

IMG_jpeg_make_recip_tbl

IMG_jpeg_quantize

IMG_dequantize_8x8

Quantization/Dequantization is an integral step in many image/video compression systems, including those based on widely used variations of DCT based compression such as JPEG, MPEG, and H.26x. The routines IMG_jpeg_quantize and IMG_dequantize_8x8 can be used in such systems to perform the quantization and dequantization steps, respectively.

- ❑ IMG_jpeg_vlc
- ❑ IMG_jpeg_vld

The JPEG variable length coding and decoding functions provide an integrated and efficient solution for performing variable length coding/decoding and running length decoding/decoding for the 8x8 macro block, following the ITU-T (CCITT) T.81 standard.

IMG_wave_decom_one_dim

- ❑ IMG_wave_recon_one_dim
- ❑ IMG_wavep_decom_one_dim
- ❑ IMG_wavep_recon_one_dim
- ❑ IMG_wave_decom_two_dim
- ❑ IMG_wave_recon_two_dim
- ❑ IMG_wavep_decom_two_dim
- ❑ IMG_wavep_recon_two_dim

Wavelet processing is finding increasing use in emerging standards such as JPEG2000 and MPEG-4, where it is typically used to provide highly efficient still picture compression. Various proprietary image compression systems are also wavelets based. Included in this release are the above eight utilities for computing 1-D/2-D Wavelet/Wavelet-Package decomposition/reconstruction transforms. Together, they can be used to compute 2-D wavelet transforms for image data. The routines are flexible enough, within documented constraints, to be able to accommodate a wide range of specific wavelets and image dimensions.

3.3 Image Analysis

The following functions are applicable to image analysis standards:

boundary

A *boundary* computation function, boundary, is provided. The boundary function, along with perimeter, is a commonly used structural operator in machine vision applications.

histogram

The routine histogram generates an image *histogram*. An image is basically a count of the intensity levels (or some other statistic) in an image. For example, for a gray scale image with 8-bit pixel intensity values, the histogram will consist of 256 bins corresponding to the 256 possible pixel intensities. Each bin will contain a count of the number of pixels in the image that have that particular intensity value. Histogram processing (such as histogram equalization or modification) are used in areas such as machine vision systems and Image/Video Content generation systems.

perimeter

A *perimeter* computation function, perimeter, is provided. The perimeter function, along with boundary, is a commonly used structural operator in machine vision applications.

threshold

Different forms of *Image Thresholding* operations are used for various reasons in image/video processing systems. For example, one form of thresholding may be used to convert gray-scale image data to binary image data for input to binary morphological processing. Another form of thresholding may be used to clamp image data levels into a desired range, and yet another form of thresholding may be used to zero out low level perturbations in image data due to sensor noise. This latter form of thresholding is addressed in the routine `threshold`.

IMG_ycbcr422_rgb565

Color space conversion from YCbCr to RGB enables display of digital video data generated, for instance, by an MPEG or JPEG decoder system on RGB displays.

3.4 Picture Filtering/Format Conversions

This section provides a description of the functions that are applicable to picture filtering and format conversions.

IMG_conv_3x3

The convolution function is used to apply generic image filters with a 3x3 filter mask, such as image smoothing, sharpening, etc.

IMG_corr_3x3

Correlation functions are provided to enable image matching. Image matching is useful in applications such as machine vision, medical imaging, and security/defense.

IMG_scale_by_2

This function upscales the image by the factor of 2, using built-in hardware extensions. This efficient routine makes it possible to match a small size video stream with a large display size.

Chapter 4

IMGLIB Function Tables

This chapter provides tables containing all IMGLIB functions, a brief description of each, and a page reference for more detailed information.

Topic **Page**

4.1 IMGLIB Function Tables

The routines included in the image library are organized into three functional categories and listed below in alphabetical order.

Table 4-1. Compression/Decompression

Function	Description	Page
void IMG_dequantize_8x8(short *quantize_tbl, short *deq_data);	Matrix dequantization	5-12
void IMG_fdct_8x8 (short *fdct_data, short *inter_buffer);	2-D forward discrete cosine transform (DCT) for 8*8 image block using built-in hardware extensions	5-6
void IMG_idct_8x8 (short *idct_data, short *inter_buffer);	2-D inverse discrete cosine transform for 8*8 DCT coefficients using built-in hardware extensions	5-8
void IMG_jpeg_make_recip_tbl(short *quantize_tbl);	Computation of the reciprocal table of the quantization terms.	5-10
void IMG_jpeg_quantize(short *quantize_input, short *zigzag, short *recip_tbl, int *quantize_output);	Matrix quantization	5-11
void IMG_jpeg_vlc(int *input_data, int *output_stream, int type);	JPEG variable length coding following ITU-T (CCITT) T.81 standard	5-13
void IMG_jpeg_vld(int *input_stream, int *lastdc, int *output_data, int type, vldvar_t *hufvar, huff_t *infor);	JPEG_variable length coding following ITU-T (CCITT) T.81 standard	5-14
void IMG_mad_8x8(unsigned short *ref_data, unsigned short *src_data, int pitc, int sx, int sy, unsighed int* match)	8x8 minimum absolute difference	5-28
void IMG_mad_16x16(unsigned short *ref_data, unsigned short *src_data, int pitc, int sx, int sy, unsigned int* match)	16x16 minimum absolute difference	5-30
void IMG_mad_16x16_4step(short *src_data, short * search_window, unsigned int *match)	Motion estimation by 4-step searching	5-32
void IMG_pix_inter_16x16(short *reference_window, short *pixel_inter_block, int offset, short *align_variable);	Half-pixel interpolation	5-35
unsigned sad_8x8(unsigned short *srcImg, unsigned short *refImg, int pitch)	Sum of absolute differences on a single 8x8 block	5-42
unsigned sad_16x16(unsigned short *srcImg, unsigned short *refImg, int pitch)	Sum of absolute differences on a single 16x16 block	5-44

Function	Description	Page
void IMG_sw_fdct_8x8 (short *fdct_data, short *inter_buffer);	2-D forward discrete cosine transform for 8*8 image block without built-in hardware extensions	5-2
void IMG_sw_idct_8x8 (short *idct_data, short *inter_buffer);	2-D inverse discrete cosine transform for 8*8 DCT coefficients without built-in hardware extensions	5-4
void IMG_wave_decom_one_dim(short *in_data, short *wksp, int *wavename, int length, int level);	1-D discrete wavelet transform	5-16
void IMG_wave_decom_two_dim(short **image, short * wksp, int width, int height, int *wavename, int level);	2-D discrete wavelet transform	5-20
void IMG_wave_recon_one_dim(short *in_data, short *wksp, int *wavename, int length, int level);	1-D inverse discrete wavelet transform	5-17
void IMG_wave_recon_two_dim(short **image, short * wksp, int width, int height, int *wavename, int level);	2-D inverse discrete wavelet transform	5-22
void IMG_wavep_decom_one_dim(short *in_data, short *wksp, int *wavename, int length, int level);	1-D discrete wavelet package transform	5-18
void IMG_wavep_decom_two_dim(short **image, short * wksp, int width, int height, int *wavename, int level);	2-D discrete wavelet package transform	5-24
void IMG_wavep_recon_one_dim(short *in_data, short *wksp, int *wavename, int length, int level);	1-D inverse discrete wavelet package transform	5-19
void IMG_wavep_recon_two_dim(short **image, short * wksp, int width, int height, int *wavename, int level);	2-D inverse discrete wavelet package transform	5-26

Table 4-2. Image Analysis

Function	Description	Page
void IMG_boundary(short * in_data, int rows, int cols, int *out_coord, int *out_gray);	Boundary structural operator	5-45
void IMG_histogram(short * in_data, short *out_data, int size);	Histogram computation	5-46
void IMG_perimeter(short * in_data, int cols, short *out_data);	Perimeter structural operator	5-47
void threshold(short * in_data, short *out_data, short cols, short rows, short threshold_value)	Image thresholding	5-49

Table 4-3. Picture Filtering/Format Conversions

Function	Description	Page
void IMG_conv_3x3(unsigned char *input_data, unsigned char *output_data, unsigned char *mask, int column, int shift);	3x3 convolution	5-50
void IMG_corr_3x3(unsigned char *input_data, unsigned char *output_data, unsigned char *mask, int row, int column, int shift, int round_val)	3x3 correlation with rounding	5-51
void IMG_scale_by_2(int *input_image, int *output_image, int row, int column)	Implements image scaling by pixel interpolation using built-in hardware extensions	5-52
void IMG_ycbcr422_rgb565(short coeff[], short *y_data, short *cb_data, short* cr_data, short *rgb_data, num_pixels)	Planarized yCbCr 4:2:2/4:2:0 to RGB 5:6:5 color space conversion	5-56

Chapter 5

IMGLIB Reference

This chapter provides a list of the routines within the IMGLIB and organized into functional categories. The functions within each category are listed in alphabetical order and include arguments, descriptions, algorithms, benchmarks, and special requirements.

Topic **Page**

5.1 Compression/Decompression

IMG_sw_fdct_8x8 — *2-D Forward Discrete Cosine Transform for an 8x8 Image Block*

Syntax

void **IMG_sw_fdct_8x8**(short *fdct_data, short *inter_buffer);

Arguments

Inputs:

fdct_data	Points to a short format array [0..63] containing an 8x8 macro-block row by row. Data format is Q16.0.
inter_buffer	Points to a short format array [0..71] used as a temporary buffer that contains intermediate results in the transform

Outputs:

fdct_data	Points to a short format array [0..63] containing results of 2-D DCT for the macro-block. Data format is Q16.0.

Description

The routine IMG_sw_fdct_8x8 implements the 2-D forward discrete cosine transform (FDCT) for an 8x8 image block. Input and output data format is singed Q16.0.

Algorithm

The forward discrete cosine transform is described by the following equation:

$$I(u,v) = \frac{a(u)s(v)}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} i(x,y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

where

$$z = 0 \Rightarrow \alpha(z) = \frac{1}{\sqrt{2}}$$
$$z \neq 0 \Rightarrow \alpha(z) = 1$$

Special Requirements

Input array fdct_data and inter_buffer must be aligned on a 32-bit boundary.

For optimal cycle performance, fdct_data and inter_buffer must be located in different DARAM banks. *r_coefs* can be located in the same DARAM bank as fdct_data. r_coeffs cannot be located in the same DARAM bank with inter_buffer.

Implementation Notes

The jpeidct.tab file contains ten DCT coefficients. These coefficients are pointed to by label r_coefs and are used by this routine. For maximum performance, r_coefs is located at DARAM.

Small Memory Model

Benchmark 1078 cycles without return

Code Size 188 words

Data Size 16 words (6 words in stack, 10 DCT coefficient words in DARAM)

Large Memory Model

Benchmark 1082 cycles without return

Code Size 192 words

Data Size 19 words (6 words in stack, 3 words in sysstack, 10 DCT coefficient words in DARAM)

Example See the examples/swDCTIDCT subdirectory.

IMG_sw_idct_8x8 — *2-D Inverse Forward Discrete Cosine Transform for an 8x8 Image Block*

Syntax

void **IMG_sw_idct_8**x**8**(short *idct_data, short *inter_buffer);

Arguments

Inputs:

idct_data	Points to a short format array [0..63] containing an 8x8 macro-block row by row. Data format is Q13.3.
inter_buffer	Points to a short format array [0..71] used as a temporary buffer that contains intermediate results in the transform

Outputs:

idct_data	Points to a short format array [0..63] containing results of 2-D DCT for the macro-block. Data format is Q16.0.

Description

The IMG_sw_idct_8×8 routine implements the 2-D Inverse Discrete Cosine Transform (IDCT) for an 8x8 input block. Input terms are expected to be signed Q13.3 values, producing signed Q16.0 results.

Algorithm

The Inverse Discrete Cosine Transform is described by the following equations:

$$i(x,y) = \frac{\alpha(u)\alpha(v)}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} I(u,v) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

where

$$z = 0 \Rightarrow \alpha(z) = \frac{1}{\sqrt{2}}$$

$$z \neq 0 \Rightarrow \alpha(z) = 1$$

Special Requirements

For maximum cycle performance, input array idct_data, intermediate buffer inter_buffer, and r_coefs must be in different DARAM banks.

Implementation Notes

Local constant, *r_coefs*, contains the ten IDCT coefficients that are from m4idct.tab. Data format is 16.0.

The m4idct.tab file contains ten IDCT coefficients. These coefficients are pointed to by the r_coefs label and used for this routine. For maximum performance, r_coefs should be located in DARAM.

Small Memory Model

Benchmark	672 cycles without return
Code Size	177 words
Data Size	21 words (11 words in stack, 10 IDCT coefficient words in DARAM)

Large Memory Model

Benchmark	676 cycles without return
Code Size	179 words
Data Size	23 words (11 words in stack, 2 words in sysstack, 10 IDCT coefficient words in DARAM)

Example See the examples/swDCTIDCT subdirectory.

IMG_fdct_8x8 — *2-D Forward Discrete Cosine Transform for an 8x8 Image Using Built-In Hardware Extensions*

Syntax void **IMG_fdct_8x8**(short *fdct_data, short *inter_buffer);

Arguments

Inputs:

fdct_data	Points to a short format array [0..63] containing an 8x8 macro-block row by row. Data format is Q16.0.
inter_buffer	Points to a short format array [0..71] used as a temporary buffer that contains intermediate results in the transform

Outputs:

fdct_data	Points to a short format array [0..63] containing the results of 2-D DCT for the macro-block. Data format is Q16.0.

Description The routine IMG_fdct_8x8 implements the 2-D Forward Discrete Cosine Transform (FDCT) using built-in hardware extensions for an 8x8 image block. Input terms are expected to be signed Q16.0 values, producing signed Q16.0 results.

Algorithm The Forward Discrete Cosine Transform is described by the following equations:

$$I(u,v) = \frac{\alpha(u)\alpha(v)}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} i(x,y)\cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$

where

$$z = 0 \Rightarrow \alpha(z) = \frac{1}{\sqrt{2}}$$
$$z \neq 0 \Rightarrow \alpha(z) = 1$$

i(x,y) : gray level of the pixel
I(u,v) : discrete cosine coefficient

Special Requirements

Input array fdct_data and inter_buffer must be aligned on a 32-bit boundary.

For maximum performance, fdct_data and inter_buffer must be located in different DARAM banks. *r_coeffs* can be located in the same DARAM bank as fdct_data. r_coeffs cannot be located in the same DARAM bank as inter_buffer.

Small Memory Model

Benchmark	238 cycles without return
Code Size	118 words
Data Size	5 words (5 words in stack)

Large Memory Model

Benchmark	240 cycles without return
Code Size	120 words
Data Size	7 words (5 words in stack, 2 words in sysstack)

Example See the examples/hwDCTIDCT subdirectory.

IMG_idct_8x8

2-D Inverse Discrete Cosine Transform for an 8x8 Image Block Using Built-In Hardware Extensions

Syntax

void **IMG_idct_8x8**(short *idct_data, short *inter_buffer);

Arguments

Inputs:

idct_data	Points to a short format array [0..63] containing an 8x8 macro-block row by row. Data format is Q13.3.
inter_buffer	Points to a short format array [0..71] used as a temporary buffer that contains intermediate results in the transform

Outputs:

idct_data	Points to a short format array [0..63] containing the results of 2-D IDCT for the input block. Data format is Q16.0.

Description

The routine IMG_idct_8x8 implements the 2-D Inverse Discrete Cosine Transform (IDCT) using built-in hardware extensions for an 8x8 image block. Input terms are expected to be signed Q13.3 values, producing signed Q16.0 results.

Algorithm

The Inverse Discrete Cosine Transform is described by the following equations:

$$i(x,y) = \frac{\alpha(u)\alpha(v)}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} I(u,v) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

where

$$z = 0 \Rightarrow \alpha(z) = \frac{1}{\sqrt{2}}$$
$$z \neq 0 \Rightarrow \alpha(z) = 1$$

Special Requirements

For maximum cycle performance, input array idct_data and inter_buffer must be located in different DARAM blocks.

Small Memory Model

Benchmark	165 cycles without return
Code Size	112 words
Data Size	2 words (2 words in stack)

Large Memory Model

Benchmark	168 cycles without return
Code Size	120 words
Data Size	4 words (2 words in stack, 2 words in sysstack)

Example See the examples/hwDCTIDCT subdirectory.

IMG_jpeg_make_recip_tbl

Computation of the Reciprocal Table of the Quantization Terms

Syntax void **IMG_jpeg_make_recip_tbl**(short *quantize_tbl);

Arguments

Inputs:

quantize_tbl	Points to an integer format array [0..63] containing an 8x8 quantization table row by row. Data format is Q16.0.

Outputs:

quantize_tbl	Points to an integer format array [0..63] containing the reciprocal table of the quantization table. Data format is Q16.0.

Description The routine IMG_jpeg_make_recip_tbl computes the reciprocal table of the quantization table. It is an initialization of quantization. The reciprocal quantization table used in IMG_jpeg_quantize reduces the computation cost by avoiding division operations. Input terms are expected to be signed Q16.0 values, producing signed Q16.0 results.

Small Memory Model

Benchmark 1413 cycles without return

Code Size 16 words

Data Size 1 word (1 word in stack)

Example See the examples/IMG_jpeg_quantization subdirectory.

IMG_jpeg_quantize — *Matrix Quantization*

Syntax

void **IMG_jpeg_quantize**(short *quantize_input, short *zigzag,
short *recip_tbl, int *quantize_output);

Arguments

Inputs:

quantize_input	Points to an integer format array [0..63] containing an 8x8 matrix row by row. Data format is Q16.0.
zigzag	Points to an integer format array [0..63] containing an 8x8 zigzag table row by row. Data format is 16.0.
recip_tbl	Points to an integer format array containing an 8x8 reciprocal table of the quantization terms which is computed by IMG_jpeg_make_recip_tbl row by row. Data format is Q16.0.

Outputs:

quantize_output	Points to a integer format array [0..63] containing the 8x8 quantized output of the input matrix row by row. Data format is Q16.0.

Description

The IMG_jpeg_quantize routine quantizes an 8x8 input matrix by multiplying the contents with a third block of values that contains reciprocals of the quantization terms in zigzag order. This step corresponds to the quantization that is performed in 2-D DCT-based compression techniques. The output is in zigzag order (JPEG required). Input and output data format is Q16.0.

Implementation Notes

Since the quantization output is in zigzag order, the data format must be changed to get the output in normal order. Since the output order is dominant by the *zigzag* argument, the output order can be changed by replacing the *zigzag* matrix to satisfy an image processing requirement.

Small Memory Model

Benchmark 215 cycles without return

Code Size 31 words

Data Size 1 word (1 word in stack)

Example

See the examples/IMG_jpeg_quantization subdirectory.

IMG_dequantize_8x8

Matrix deQuantization

Syntax void **IMG_dequantize_8x8**(short *quantize_tbl, short *deq_data);

Arguments

Inputs:

quantize_tbl	Points to an integer format array [0..63] containing an 8x8 quantization table row by row. Data format is Q16.0.
deq_data	Points to an integer format array [0..63] containing an 8x8 quantized matrix row by row. Data format is Q16.0.

Outputs:

deq_data	Points to a integer format array [0..63] containing the 8x8 dequantized output of the input matrix row by row. Data format is Q16.0.

Description The IMG_dequantize_8x8 routine dequantizes an 8x8 input matrix by multiplying the contents with a second block of values that contains the quantization terms. This step corresponds to the dequantization that is performed in 2-D DCT-based compression techniques. The output is in normal order. Input and output data format is Q16.0.

Implementation Notes The output of IMG_jpeg_quantize needs to be changed to remove the zigzag format before the application of IMG_dequantize_8x8.

Small Memory Model

Benchmark 132 cycles without return

Code Size 10 words

Data Size 0 words

Example See the IMG_jpeg_quantize example in the examples/IMG_jpeg_quantization subdirectory. You will find a linker command file in the example.

IMG_jpeg_vlc — *JPEG Baseline Variable Length Coding*

Syntax

void **IMG_jpeg_vlc**(int *input_data, int *output_stream, int *VLC_status, int type);

Arguments

Inputs:

input_data	Points to the *zigzagged* quantized 8x8 DCT coefficient of a macro block.
type	Input_data is 8x8 luminance block or chrominance block. Luminance block: 0, Chrominance block: 1
VLC_status	The buffer to hold intermediate coding status.

Outputs:

output_stream	Huffman coded VLC data stream. The length of output is data-dependent.

Description

This routine takes an 8x8 *zigzagged* quantized DCT coefficient and returns a bitstream of a JPEG baseline Huffman coding. The routine checks ITU-T T.81 Huffman tables and performs DC and AC coefficient coding, including run-length code and variable length code. Before calling the routine, the luminance code table and chrominance code table have to be initialized by the *IMG_jpeg_initialization* routine. The JPEG encoder only needs to call this routine once.

Algorithm

This routine implements running length coding and variable length coding for the 8x8 macro block following the ITU-T (CCITT) T.81 standard baseline sequential method. Note that the DCT component from the previous macro block is needed for this routine.

Benchmark

Cycles Data-dependent

Code Size 339 words

Data Size 393 for Huffman look-up table

Example

See the examples/JPEG_VLC subdirectory.

IMG_jpeg_vld — *JPEG Baseline Variable Length Decoding of an 8x8 MB*

Syntax

void **IMG_jpeg_vld**(int *input_stream, int *lastdc, int *output_data, int type, vldvar_t *hufvar, huff_t *infor);

Arguments

Inputs:

input stream	Points to a JPEG baseline VLC coded bitstream. The length of the bitstream is not fixed.
lastdc	Specifies DC coefficient from the previous MB
type	Input_stream is JPEG VLC coded bitstream of 8x8 luminance block or chrominance block. Luminance block: 0, Chrominance block: 1
vldevart_t	Points to a structure containing Huffman tables, control tables, and their property
huff_t	Points to a context structure containing coding parameters of the MB to be decoded and the current state of the bitstream buffer

Outputs:

output_data	Points to the JPEG baseline VLD decoded quantized DC coefficients in normal order

Description

This routine takes a bitstream of a JPEG Baseline VLC coded macroblock (MB) and returns the decoded IDCT coefficients. The routine checks the code book and performs DC and AC coefficient decoding, including variable length decoder, run-length expansion, inverse zigzag ordering, and saturation and mismatch control. Before this routine is called, you must initialize the variable of VLC and set up the Huffman look-up tables. Input is a JPEG VLC-coded bitstream of an 8x8 luminance block or chrominance block.

Implementation Notes

The following structure is defined in this routine.

```
 typedef struct {
          int bit_ptr;             /* bit counter for current word (MSB-16) */
          int buf_ptr;             /* buffer counter for data buffer (*databuf)
*/
          int bits_count;          /* Reserved  Default value is 0*/
          int *databuf;    /* Reserved pointer to current data buffer, Default
value is 0 */
                              } huff_t
```

```
typedef struct {
  int UvldTabDC[2][32];         /* 32 may NOT be enough */
 int UvldCtlTabDC[2][17];
  int UvldTabAC[2][256];        /* 256 may NOT be enough */
  int UvldCtlTabAC[2][17];
  int UvldLenMaxDC[2];
  int UvldLenMaxAC[2];
} vldvar_t;
struct vlccode {
    unsigned int len;
    unsigned int word;
    unsigned int pattern;
    };
struct control {
  unsigned int th_lower;
  int shift;
  int offset;
};
```

C header file, Vld.h, should be included in the source file.

Assemble file, consth.inc, should be included .

VLD.h includes the data structure definition and two utility C routines, *InitDehufVars* and *makevldtab.* Before calling the routine, the bitstream variables have to be initialized by the *InitDehufVar* routine, and then the Huffman table is set up by the *makevldtab* routine.

Algorithm This routine implements variable-length coding and inverse scan for an 8x8 chrominance block or luminance block.

Benchmark

Cycles Data-dependent

Code Size 634 words
Huffman look-up table: 700 words

IMG_wave_decom_one_dim

One Dimensional Pyramid Wavelet Decomposition

Syntax

void **IMG_wave_decom_one_dim**(short *in_data, short *wksp, int *wavename, int length, int level)

Arguments

Inputs:

in_data	Points to input vector of size *length*. Data format is Q16.0.
wksp	Points to work space of size *length*
wavename	Points to wavelet filter coefficients
length	Specifies length of input and work space data array
level	Specifies level of decomposition

Outputs:

in_data	Stores the output of the decomposition. Data format is Q16.0.

Description

One dimensional wavelet pyramid decomposition. The wavelet filter coefficients are passed by the vector *wavename*. The length of the input vector should be divided by 2^level. The decomposition output is stored in the same vector of input. The IMG_wave_decom_one_dim function calls the *decomInplace* assembly function. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the input is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark [(Filter Length + 3)/2] × [Signal Length] × [2 - 2^(1 - level)]+87 Cycles

Code Size IMG_wave_decom_one_dim: 32 words
DecomInplace: 67 words

Data Size DecomInplace: 4 words (4 words in stack)

Example See the examples/1D_Wavelet subdirectory.

IMG_wave_recon_one_dim — *One Dimensional Pyramid Wavelet Reconstruction*

Syntax

void **IMG_wave_recon_one_dim**(short *in_data, short *wksp, int *wavename, int length, int level);

Arguments

Inputs:

in_data	Points to input vector of size *length*. Data format is Q16.0.
wksp	Points to work space of size *length*
wavename	Points to wavelet filter coefficients
length	Specifies length of input and work space data array
level	Specifies level of reconstruction

Outputs:

in_data	Stores the output of the reconstruction. Data format is Q16.0.

Description

One dimensional wavelet pyramid reconstruction. The wavelet filter coefficients are passed by the vector *wavename*. The length of the input vector should be divided by 2^level. The reconstruction output is stored in the same vector of input. The IMG_wave_recon_one_dim function calls the *reconInplace* assembly function. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the input is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark [(Filter Length + 4)/2] $\times$ [Signal Length] $\times$ [2-2^(1-level)] + 85 Cycles

Code Size IMG_wave_recon_one_dim: 39 words
reconInplace: 85 words

Data Size reconInplace: 5 words (5 words in stack)

Example

See the examples/1D_Wavelet subdirectory.

IMG_wavep_decom_one_dim — *One Dimensional Wavelet Packet Decomposition*

Syntax

void **IMG_wavep_decom_one_dim**(short *in_data, short *wksp,
int *wavename, int length, int level);

Arguments

Inputs:

in_data	Points to input vector of size *length*. Data format is Q16.0.
wksp	Points to work space of size *length*
wavename	Points to wavelet filter coefficients
length	Specifies length of input and work space data array
level	Specifies level of decomposition

Outputs:

in_data	Stores the output of the decomposition. Data format is Q16.0.

Description

One dimensional wavelet packet decomposition. The wavelet filter coefficients are passed by the vector *wavename*. The length of the input vector should be divided by 2^level. The decomposition output is stored in the same vector of input. The IMG_wavep_decom_one_dim function calls the *decomInplace* assembly function. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the input is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark [(Filter Length + 3)/2] $\times$ [Signal Length] $\times$ [level] + 90 Cycles

Code Size IMG_wavep_decom_one_dim: 57 words
DecomInplace: 67 words

Data Size DecomInplace: 4 words (4 words in stack)

Example See the examples/1D_Wavelet subdirectory.

IMG_wavep_recon_one_dim

One Dimensional Wavelet Packet Reconstruction

Syntax

void **IMG_wavep_recon_one_dim**(short *in_data, short *wksp,
int *wavename, int length, int level);

Arguments

Inputs:

in_data	Points to input array of size *length*. Data format is Q16.0.
wksp	Points to work space of size *length*
wavename	Points to wavelet filter coefficients
length	Specifies length of input and work space data vectors
level	Specifies level of reconstruction

Outputs:

in_data	Stores the output of the reconstruction. Data format is Q16.0.

Description

One dimensional wavelet packet reconstruction. The wavelet filter coefficients are passed by the vector *wavename*. The length of the input array should be divided by 2^level. The reconstruction output is stored in the same array of in_data. The IMG_wavep_recon_one_dim function calls the *reconInplace* assembly function. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the input is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark [(Filter Length + 4)/2] $\times$ [Signal Length] $\times$ [level] + 94 Cycles

Code Size IMG_wavep_recon_one_dim : 69 words
reconInplace: 85 words

Data Size reconInplace: 5 words (5 words in stack)

Example See the examples/1D_Wavelet subdirectory.

IMG_wave_decom_two_dim

Two Dimensional Pyramid Wavelet Decomposition

Syntax

void **IMG_wave_decom_two_dim**(short **image, short *wksp, int width, int height, int *wavename, int level);

Arguments

Inputs:

image	Points to image matrix of size *width* by *height*. Data format is Q16.0.
wksp	Points to work space of size max (*width*, *height*)
wavename	Points to wavelet filter coefficients
width	Specifies row size of the image
height	Specifies column size of the image
level	Specifies level of decomposition

Outputs:

image	Stores the decomposed image. Data format is Q16.0.

Description

Two dimensional wavelet pyramid decomposition. The wavelet filter coefficients are passed by the vector *wavename*. The width and height of the image should be divided by 2^level. The decomposed image is stored in the same matrix of in_data. The IMG_wave_decom_two_dim function calls three assembly functions: *decomInplace*, *col2row* and *decomCol*. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the input is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark 35 + 13 **x** level + (322 **x** height +210 **x** width) **x** [1-2^(-level)] + 4/3 **x** width **x** height **x** (filter length + 2) **x** [1-4^(-level)]

Code Size IMG_wave_decom_two_dim: 71 words
DecomInplace: 67 words
col2row: 6 words
decomCol: 56 words

Data Size DecomInplace: 5 words (5 words in stack)
col2row: 0 words
decomCol: 5 words (5 words in stack)

Example See the examples/2D_Wavelet subdirectory.

IMG_wave_recon_two_dim

Two Dimensional Pyramid Wavelet Reconstruction

Syntax

void **IMG_wave_recon_two_dim**(short **image, short *wksp, int width, int height, int *wavename, int level);

Arguments

Inputs:

image	Points to image matrix of size *width* by *height*. Data format is Q16.0.
wksp	Points to work space of size max (*width*, *height*)
wavename	Points to wavelet filter coefficients
width	Specifies row size of the image
height	Specifies column size of the image
level	Specifies level of reconstruction

Outputs:

image	Stores the reconstructed image. Data format is Q16.0.

Description

Two dimensional wavelet pyramid reconstruction. The wavelet filter coefficients are passed by the vector *wavename*. The width and height of the image should be divided by 2^level. The reconstructed image is stored in the same matrix of in_data. The IMG_wave_recon_two_dim function calls three assembly functions: *decomInplace*, *col2row* and *decomCol*. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the in_data is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark

13 **x** level + 246 **x** height **x** [1-2^(-level)] + 448 **x** width **x** [1-2^(-length)] + 4/3 **x** (filter length +4) **x** [1-4^(-level)]

Code Size

IMG_wave_recon_two_dim: 87 words
InterlaceCol: 43 words

ReconCol: 58 words
reconInplace: 85 words

Data Size InterlaceCol: 5 words
ReconCol: 5 words
reconInplace. 5 words

Example See the examples/2D_Wavelet subdirectory.

IMG_wavep_decom_two_dim

Two Dimensional Wavelet Package Decomposition

Syntax

void **IMG_wavep_decom_two_dim**(short **image, short *wksp, int width, int height, int *wavename, int level);

Arguments

Inputs:

image	Points to image matrix of size *width* by *height*. Data format is Q16.0.
wksp	Points to work space of size max (*width*, *height*)
wavename	Points to wavelet filter coefficients
width	Specifies row size of the image
height	Specifies column size of the image
level	Specifies level of decomposition

Outputs:

image	Stores the decomposed image. Data format is Q16.0.

Description

Two dimensional wavelet packet decomposition. The wavelet filter coefficients are passed by the vector wavename. The width and height of the image should be divided by 2^level. The decomposed image is stored in the same matrix of in_data. The IMG_wavep_decom_two_dim function calls three assembly functions: *decomInplace*, *col2row* and *decomCol*. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the in_data is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark

(110 **x** height + 137 **x** width) **x** (2^level - 1) + (filter length **x** 0.25 + 2.25) **x** (width **x** height) **x** level + 20 **x** (width + height) **x** level

Code Size

IMG_wavep_decom_two_dim: 101 words
DecomInplace: 67 words

col2row: 6 words
decomCol: 56 words

Data Size DecomInplace: 5 words (5 words in stack)
col2row: 0 words
decomCol: 5 words (5 words in stack)

Example See the examples/2D_Wavelet subdirectory.

IMG_wavep_recon_two_dim

Two Dimensional Wavelet Package Reconstruction

Syntax

void **IMG_wavep_recon_two_dim**(short **image, short *wksp, int width, int height, int *wavename, int level);

Arguments

Inputs:

image	Points to image matrix of size *width* by *height*. Data format is Q16.0.
wksp	Points to work space of size max (*width*, *height*)
wavename	Points to wavelet filter coefficients
width	Specifies row size of the image
height	Specifies column size of the image
level	Specifies level of reconstruction

Outputs:

image	Stores the reconstructed image. Data format is Q16.0.

Description

Two dimensional wavelet packet reconstruction. The wavelet filter coefficients are passed by the vector wavename. The width and height of the image should be divided by 2^level. The reconstructed image is stored in the same matrix of in_data. The IMG_wavep_decom_two_dim function calls three assembly functions: *decomInplace*, *col2row* and *decomCol*. Input and output data format is Q16.0.

Implementation Notes

No scaling implemented for overflow prevention.

There is no overflow prevention in the wavelet functions. However, if the in_data is in the range of [0, 255], there should be no overflow up to at least five levels of decomposition for all wavelets filters. The library has the following five families of wavelets: bior, coif, daub, rbio and sym.

Small Memory Model

Benchmark Level **x** [32 + 123 **x** height + 224 **x** weight + (filter length +4) **x** width **x** height]

Code Size IMG_wave_recon_two_dim: 87 words
InterlaceCol: 43 words
ReconCol: 58 words
reconInplace: 85 words

Data Size InterlaceCol: 5 words
ReconCol: 5 words
reconInplace: 5 words

Example See the examples/2D_Wavelet subdirectory.

IMG_mad_8x8 — *8x8 Minimum Absolute Difference*

Syntax

void **IMG_mad_8x8**(unsigned short *ref_data, unsigned short *src_data, int pitch, int sx, int sy, unsigned int* match)

Arguments

Inputs

*ref_data	Points to a pixel in a reference image which constitutes the top-left corner of the area to be searched. The dimensions of the search area are given by (sx+8)x(sy+8)
src_data[]	Points to 8x8 source image pixels. Must be word aligned.
pitch	Width of reference image
sx	Horizontal dimension of the search space
sy	Vertical dimension of the search space
match [2]	match[0]: packed best match location. The upper half-word contains the horizontal pixel position and the lower half-word contains the vertical pixel position of the best matching 8x8 block in the search area. The range of the coordinates is [0,sx-1] in the horizontal dimension and [0,sy-1] in the vertical dimension, where the location (0.0) represents the top-left corner of the search area. match[1]: minimum absolute difference value at the best match location.

Description

This routine locates the position of the top-left corner of an 8x8 pixel block in a reference image which most closely matches the 8x8 pixel block in src_data[], using the sum of absolute difference metric. The source image block, src_data[], is moved over a range that is sx pixels wide and sy pixels tall within a reference image that is pitch pixels wide. The pointer, *ref_data, points to the top-left corner of the search area within the reference image. The match location as well as the minimum absolute difference value for the match are returned in the match[2] array. The search is performed in top-to-bottom, left-to-right order, with the earliest match taking precedence in the case of ties.

Algorithm

Calculate all sads in the range of [0,sx-1]x[0,sy-1].

Get the minimum sad.

Special Requirements

Sy must be a multiple of 12.

Every two pixels are packaged into one word.

Implementation Notes

This routine is developed based on built-in motion estimation hardware extension.

Small Memory Model

Benchmarks 38.5*sx*sy+95 cycles
For sx=10, sy=24, cycles=9,558
For sx=64, sy=24, cycles=60,642

Code Size 268 words

Data Size 8 words (8 words in stack)

Large Memory Model

Benchmarks 38.5*sx*sy+95 cycles
For sx=10, sy=24, cycles=9,415
For sx=64, sy=24, cycles=59,749

Code Size 270 words

Data Size 8 words (8 words in stack)

Example See the examples/mad_8x8 subdirectory.

IMG_mad_16x16 *16x16 Minimum Absolute Difference*

Syntax

void **IMG_mad_16x16**(unsigned short *ref_data, unsigned short *src_data, int pitch, int sx, int sy, unsigned int* match)

Arguments

Inputs

*ref_data	Points to a pixel in a reference image which constitutes the top-left corner of the area to be searched. The dimensions of the search area are given by (sx+16)x(sy+16)
src_data[]	Points to 16x16 source image pixels. Must be word aligned.
pitch	Width of reference image
sx	Horizontal dimension of the search space
sy	Vertical dimension of the search space
match [2]	match[0]: packed best match location. The upper half-word contains the horizontal pixel position and the lower half-word contains the vertical pixel position of the best matching 8x8 block in the search area. The range of the coordinates is [0,sx-1] in the horizontal dimension and [0,sy-1] in the vertical dimension, where the location (0.0) represents the top-left corner of the search area. match[1]: minimum absolute difference value at the best match location.

Description

This routine locates the position of the top-left corner of a 16x16 pixel block in a reference image which most closely matches the 16x16 pixel block in src_data[], using the sum of absolute difference metric. The source image block, src_data[], is moved over a range that is sx pixels wide and sy pixels tall within a reference image that is pitch pixels wide. The pointer, *ref_data, points to the top-left corner of the search area within the reference image. The match location, as well as the minimum absolute difference value for the match, are returned in the match[2] array. The search is performed in top-to-bottom, left-to-right order, with the earliest match taking precedence in the case of ties.

Algorithm

Calculate all sads in the range of [0,sx-1]x[0,sy-1].

Get the minimum sad.

Special Requirements

Sy must be a multiple of 12.

Every two pixels are packaged into one word.

Implementation Notes

This routine is developed based on built-in motion estimation hardware extension.

Small Memory Model

Benchmarks 85.5*sx*sy+95 cycles
For sx=10, sy=12, cycles=10,378
For sx=64, sy=24, cycles=131,682

Code Size 301 words

Data Size 8 words (8 words in stack)

Large Memory Model

Benchmarks 85.5*sx*sy+95 cycles
For sx=10, sy=12, cycles=10,379
For sx=64, sy=24, cycles=131,683

Code Size 304 words

Data Size 8 words (8 words in stack)

Example See the examples/mad_16x16 subdirectory.

IMG_mad_16x16_4step

Motion Estimation by 4-Step Search Using Built-In Hardware Extensions

Syntax

void **IMG_mad_16x16_4step**(short *src_data, short *search_window, unsigned int *match);

Arguments

Inputs:

src_data	Points to a packed integer format buffer [0..1q28] that contains 16x16 source data row by row. Data format is Q16.0. Every two pixels are packed into one 16-bit integer.
search_window	Points to a packed integer format buffer [0..1152] that contains the 48x48 search-window row by row. Data format is Q16.0. Every two pixels are packed into one 16-bit integer.

Outputs:

match[2]	The location of the best match block is packed in match[0]. The upper halfword contains the horizontal pixel position, and the lower halfword contains the vertical pixel position of the best matching 16x16 block in the search window. The minimum absolute difference value at the best match location is packed in match[1].

Description

The routine IMG_mad_16x16_4step implements the motion estimation by 4-step (distance=8,4,2,1) search using built-in hardware extensions. The 4-step search is a popular fast-searching technique. Input terms are packed in 16-bit integers and doubleword aligned. Input and output data format is Q16.0.

Algorithm

The motion estimation by 4-step search is described by the following equations:

Initialization:

```
d={8,4,2,1}
```

Given pixel as the start point of searching window.

```
i=0;
for( i=0; i<4; i++)
{
```

Call the macros for d[i] and generate the nine absolute-difference table. Compute the minimum of the nine error table based on distance d[i] as shown in Figure 5-2 .

Start above process around the minimum location for the new distance.

```
d[ i+1].
}
```

The algorithm is also described in Figure 5-1.

Figure 5-1. Macro Expansion Algorithm

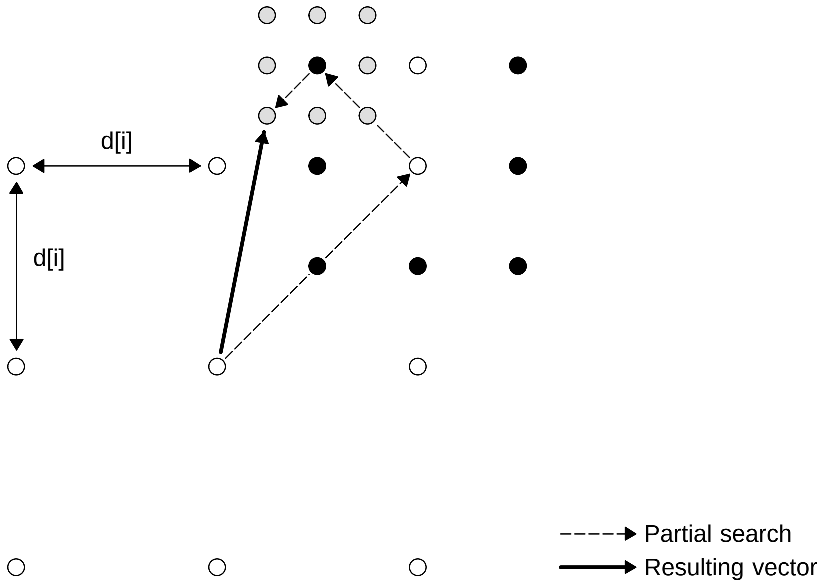
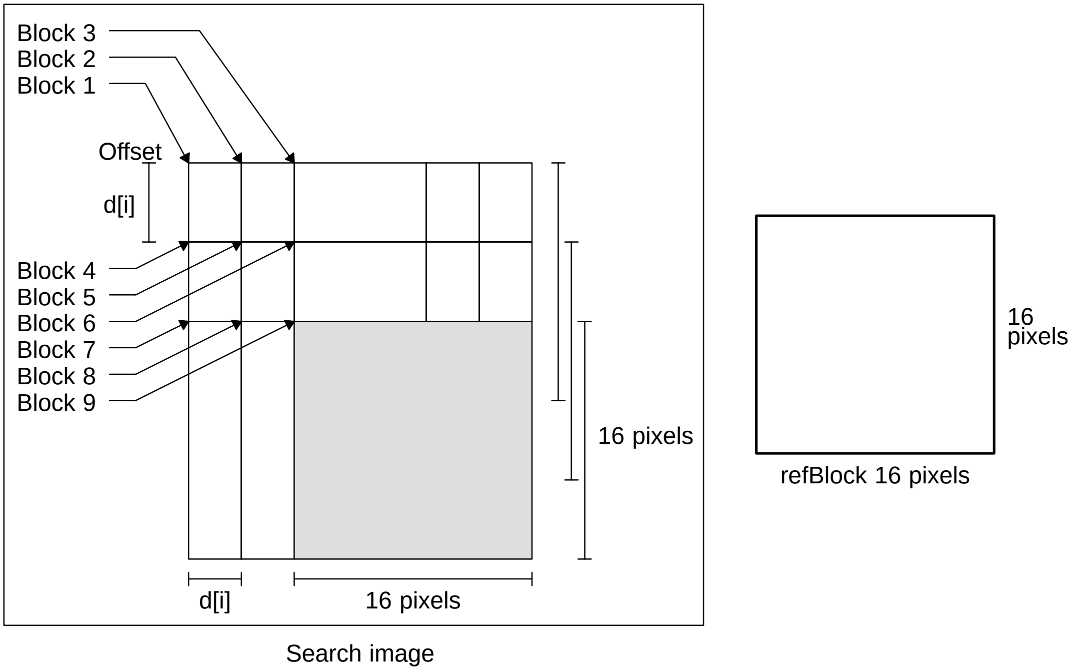


Figure 5-2 illustrates how to compute nine absolute differences.

Figure 5-2. Computing Nine Absolute Differences



Special Requirements

src_data and search_window must be aligned on a (32-bit) boundary.

src_data and search_window use word-packed memory. Every 16-bit word of src_data and search_window contains two 8-bit pixels.

src_data and search_window must be located in different DARAM bank for optimal cycle performance.

Implementation Notes The block_index section stores 12 integers needed for the location in 4-step searching.

Small Memory Model

Benchmark 2225 cycles without return

Code Size 429 words

Data Size 17 words (5 words in stack, 12 words DRAM)

Example See the examples/motion_estimation subdirectory.

IMG_pix_inter_16x16 *Pixel Interpolation for 16x16 Image Block*

Syntax

void **IMG_pix_inter_16x16**(short *reference_window, short *pixel_inter_block,
int offset, short *align_variable);

Arguments

Inputs:

reference_window	Points to a packed integer format buffer [0..1152] that contains a 48x48 image block row by row. Must be doubleword aligned. Every four pixels are packed into one 32-bit doubleword. Data format Q16.0.
offset	Specifies the top-left corner index of the 18x18 MBE (MBE=16x16 macro block + extension) in reference_window. Offset is even because of the doubleword alignment.
align_variable	Configures four alignment cases of the MBE in the reference_window

Outputs:

pixel_inter_block	Points to a packed integer format buffer [0..612] that contains the 36x34 interpolated result. Only the lower 33x33 part that corresponds to the whole 36x34 interpolated zone is usually used. Every four pixels are packed into one 32-bit doubleword.

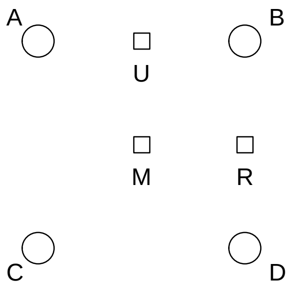
Description

The routine IMG_pix_inter_16x16 implements pixel interpolation for a 16x16 source block located in reference_window using built-in hardware extensions. This pixel interpolation routine adapts to the fractal-pixel motion estimation technique in video compression. To implement full interpolation for the 16x16 source block, the 18x18 MBE (MBE=16x16 macro block + extension) is needed. The full interpolated zone is composed of 36x34 pixels, but only the lower 33x33 part corresponding to the full interpolated zone is usually interested. The original pixels and interpolated pixels in the full interpolated zone are organized in different 16 bits to adapt to the related motion estimation technique.

Algorithm

The half-pixel interpolation is described by the following equations and diagrams.

$$U = \frac{A + B + Rnd}{2} \quad M = \frac{A + B + C + D + 1 + Rnd}{4} \quad R = \frac{B + D + Rnd}{2}$$



where A, B, C, and D are original pixels and U, M, and R are corresponding interpolated pixels. Using the above basic half-pixel interpolation to the whole MBE, you get the full interpolated zone.

Special Requirements

Reference_window must be aligned to a 32-bit boundary.

Reference_window and pixel_inter_block must be located in different DARAM banks.

Implementation Notes align_variable configures the following four alignment cases of the MBE in the reference_window.

Figure 5-3. Alignment Case 1 Align_variable=0

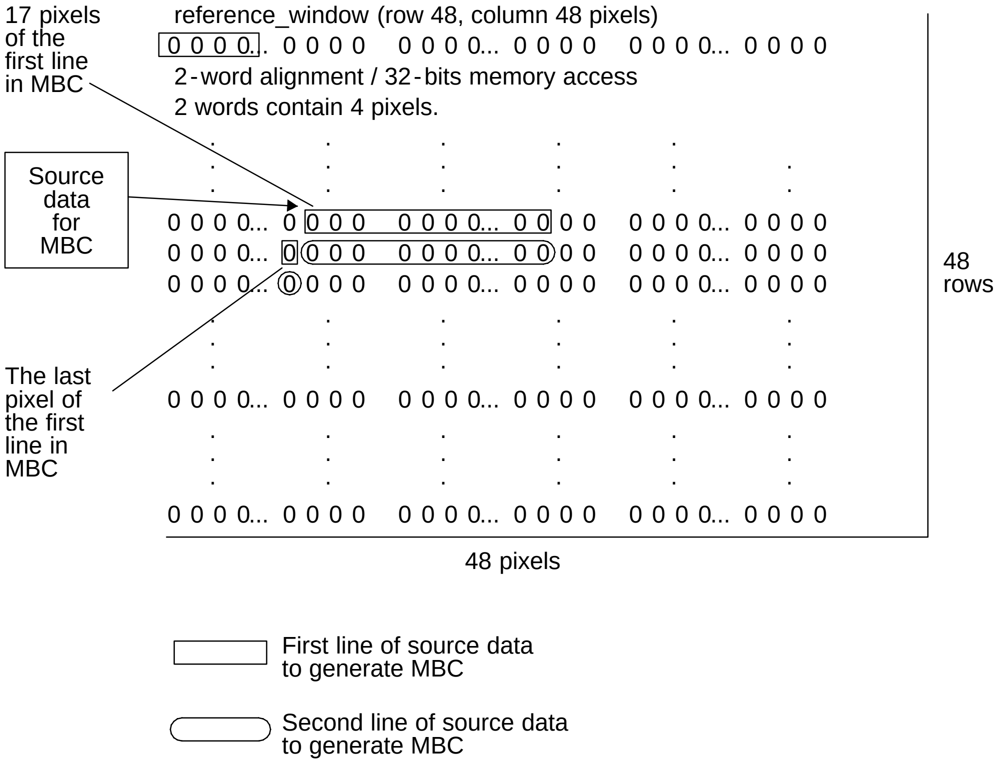


Figure 5-4. Alignment Case 2 Align_variable=1

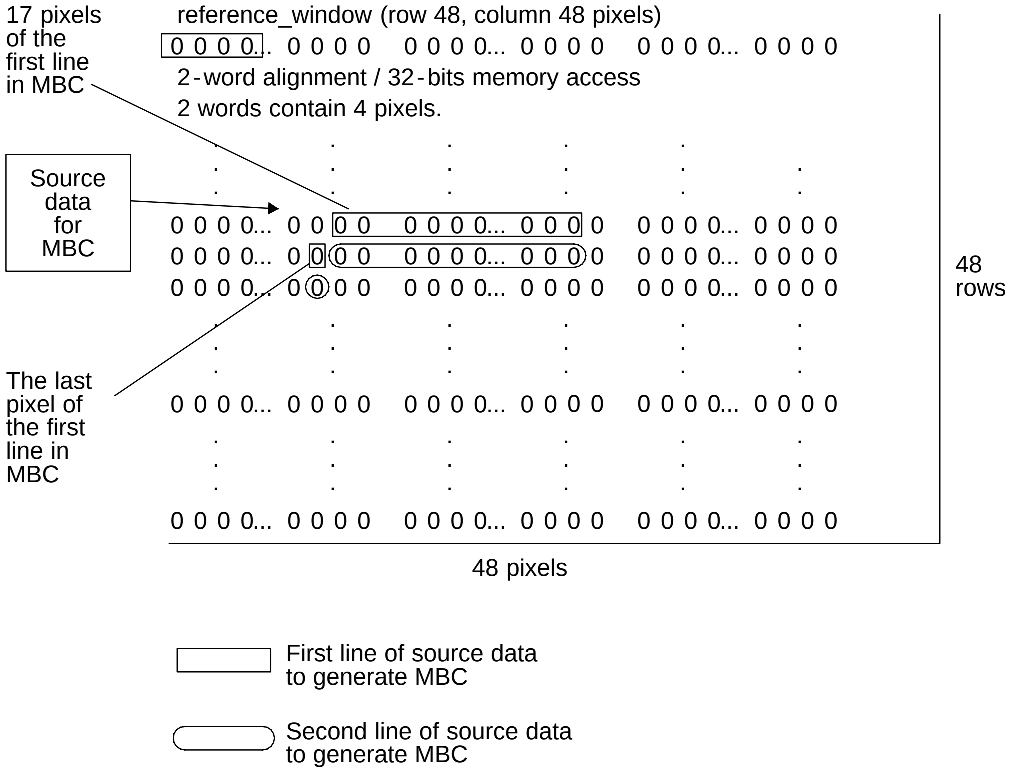


Figure 5-5. Alignment Case 3 Align_variable=2

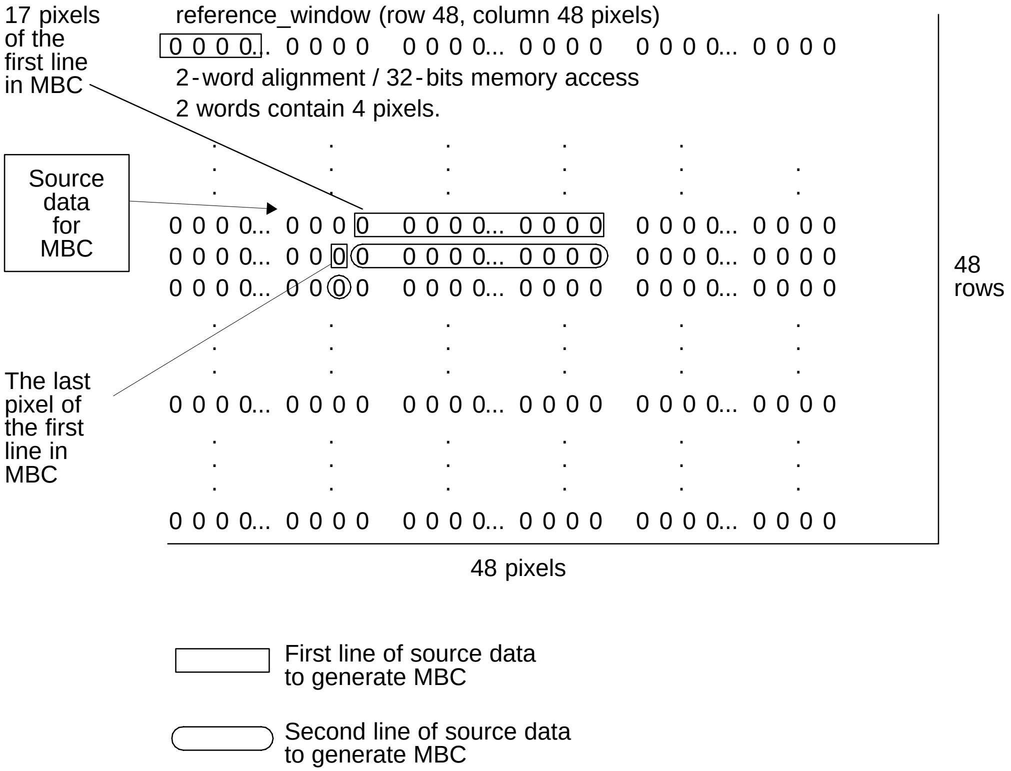
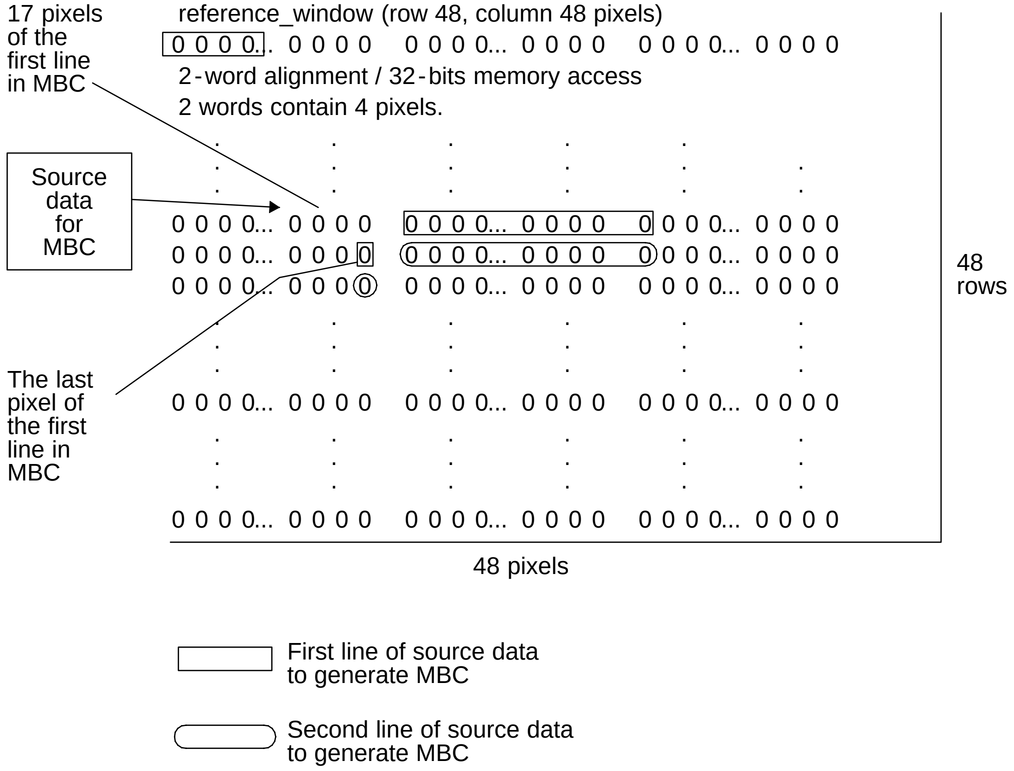
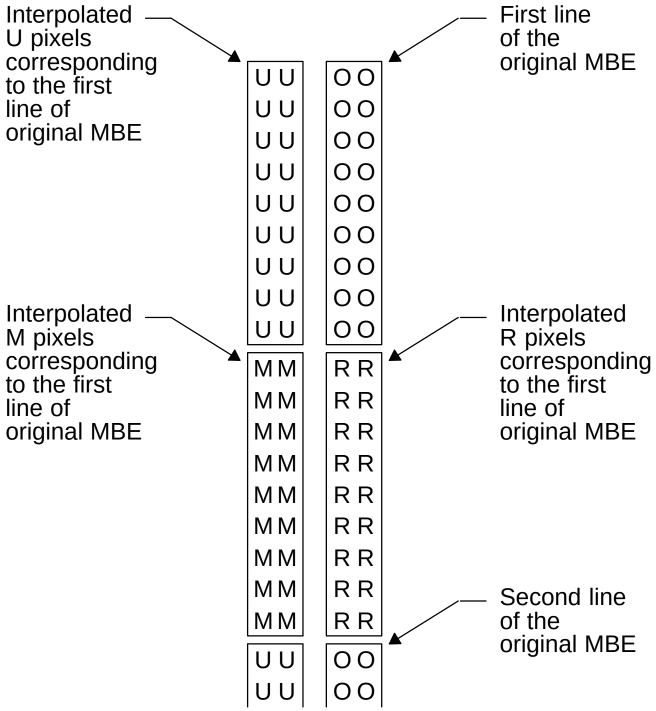


Figure 5-6. Alignment Case 4 Align_variable=3



The full interpolated zone organization is shown in Figure 5-7.

Figure 5-7. Full Interpolated Zone Organization



Small Memory Model

Benchmark The benchmark return depends on the align_variable value. The average is 530 cycles without return.

- if (align_variable=0) 571 cycles without return
- if (align_variable=1) 497 cycles without return
- if (align_variable=2) 553 cycles without return
- if (align_variable=3) 498 cycles without return

Code Size 462 words

Data Size 2 words (2 words in stack)

Example See the examples/MPEG_PI subdirectory.

IMG_sad_8x8 *Sum of Absolute Difference on Single 8x8 Block*

Syntax unsigned short **IMG_sad_8x8**(unsigned short *srcImg, unsigned short *refImg, int pitch)

Arguments **Inputs**

srcImg[]	8x8 source block. Must be double-word aligned.
refImg[]	Reference image
pitch	Width of reference image

Description This function returns the sum of the absolute differences between the source block and the 8x8 region pointed to in the reference image.

The code accepts a pointer to the 8x8 source block (srcImg), and a pointer to the upper-left corner of a target position in a reference image (refImg). The wdth of the reference image is given by the pitch argument.

Algorithm

The sad is defined by the following equation:

$$sad = \sum_{i=0}^{7}\sum_{j=0}^{7}\left|s_{ij} - r_{ij}\right|$$

where s_{ij} and r_{ij} are source block and reference block respectively.

Special Requirements

Every two pixels are packaged into one word.

Implementation Notes

This routine is developed based on built-in motion estimation hardware extension.

Small Memory Model

Benchmarks 52 cycles

Code Size 28 words

Data Size 0 words

Large Memory Model

Benchmarks 52 cycles

Code Size	28 words
Data Size	0 words

Example See the examples/sad_8x8 subdirectory.

IMG_sad_16x16 *Sum of Absolute Difference on Single 16x16 Block*

Syntax unsigned short **IMG_sad_16x16**(unsigned short *srcImg, unsigned short *refImg, int pitch)

Arguments **Inputs**

srcImg[]	16x16 source block. Must be double-word aligned.
refImg[]	Reference image
pitch	Width of reference image

Description This function returns the sum of the absolute differences between the source block and the 16x16 region pointed to in the reference image.

The code accepts a pointer to the 16x16 source block (srcImg), and a pointer to the upper-left corner of a target position in a reference image (refImg). The width of the reference image is given by the pitch argument.

Algorithm

The sad is defined by the following equation:

$$sad = \sum_{i=0}^{15}\sum_{j=0}^{15}\left|s_{ij} - r_{ij}\right|$$

where s_{ij} and r_{ij} are source block and reference block respectively.

Special Requirements Every two pixels are packaged into one word.

Implementation Notes

This routine is developed based on built-in motion estimation hardware extension.

Small Memory Model

Benchmarks 156 cycles

Code Size 28 words

Data Size 0 words

Large Memory Model

Benchmarks 156 cycles

Code Size 28 words

Data Size 0 words

Example See the examples/sad_16x16 subdirectory.

5.2 Image Analysis

IMG_boundary — *Boundary Structural Operator*

Syntax

void **IMG_boundary**(short *in_data, int rows, int cols, int *out_coord,
int *out_gray);

Arguments

Inputs:

in_data	Points to original image array
rows	Specifies the number of rows of the image
cols	Specifies the number of columns of the image

Outputs:

out_coord	Points to the boundary pixel coordinates array
out_gray	Points to the boundary pixel value array

Description

Gets the boundary of an image with a background pixel value of zero. If a boundary pixel is detected, the function outputs its coordinates into array out_coord and the pixel value into array out_gray. Input data format is Q16.0.

Small Memory Model

Benchmark $5.125 \times (cols \times rows) + 8 \times rows + 14$ cycles

Code Size 44 words

Data Size 2 words (2 words in stack)

Example See the examples/Boundary subdirectory.

IMG_histogram — *Histogram Computation*

Syntax void **IMG_histogram**(short *in_data, short *out_data, int size);

Arguments

Input:

in_data	Points to input image array
size	Specifies the size of the image

Outputs:

out_data	Specifies the histogram array

Description Histogram analysis. The in_data image value should be in the range of [0, 255]. Input and output data format is Q16.0.

Implementation Notes

Use the pixel_value as index to get the histogram value: histogram[pixel_value]

Update the histogram value, histogram[pixel_value]++

Small Memory Model

Benchmark 2.25 x size+18

Code Size 33 words

Data Size 3 words (3 words in stack)

Example See the examples/Histogram subdirectory.

IMG_perimeter — *Perimeter Structural Operator*

Syntax void **IMG_perimeter**(short *in_data, int cols, short *out_data);

Arguments

Inputs:

in_data	Points to input array. The array holds one row of an image.
cols	Specifies the length of a row

Outputs:

out_data	Points to output boundary image data

Description The routine perimeter() produces the boundary of an object in a binary image. It echoes the boundary pixels with a value of 0xFF and sets the other pixels to 0x00. Detection of the boundary of an object in a binary image is a segmentation problem and is done by examining spatial locality of the neighboring pixels. This is done by using the four connectivity algorithm:

```
               pix_up
     pix_lft   pix_cent   pix_rgt
               pix_dn
```

The output pixel at location pix_cent is echoed as a boundary pixel, if pix_cent is non-zero and any one of its four neighbors is zero. The four neighbors are as shown above and stand for the following:

pix_up: top pixel
pix_lft: left pixel
pix_rgt: right pixel
pix_dn: bottom pixel

Special Requirements

No specific alignment is expected for the input or output array.

The *cols* argument can be either even or odd.

This code expects three input lines each of width *cols* pixels and produces one output line of width (*cols* – 1) pixels.

Implementation Notes

The output pixel at location pix_cent is echoed as a boundary pixel, if pix_cent is non-zero and any one of its four neighbors is zero.

The output buffer has 2 less pixels than the input buffer. To make these two buffers have the same length, we set the first and the last value of the output buffer to be 0x00.

Small Memory Model

Benchmark $(\text{cols} - 2) \times 7 + 21$

Code Size 49 words

Data Size 3 words (3 words in stack)

Example See the examples/Perimeter subdirectory.

IMG_threshold *Image Thresholding*

Syntax void **IMG_threshold**(short *in_data, short *out_data, short col, short rows, short threshold_val);

Arguments

Inputs:

in_data	Points to original image buffer
rows	Specifies the number of rows of the image
cols	Specifies the number of columns of the image
threshold_val	Specifies the threshold value

Outputs:

output	Points to the output image buffer

Description The routine threshold() performs a thresholding operation on an input image in in_data[ ] whose dimensions are given by the arguments *cols* and *rows*. The thresholded pixels are written to the output image pointed to by out_data[ ]. The input and output are exactly the same dimensions.

Pixels that are above the threshold value are written to the output unmodified. Pixels that are less than or equal to the threshold are clamped to zero in the output image.

Small Memory Model

Benchmark cols × rows × 2.5 + 16 cycles

Code Size 28 words

Data Size 0 words

Example See the examples/thresholding subdirectory.

5.3 Picture Filtering/Format Conversions

IMG_conv_3x3 — *3x3 Convolution With Shift*

Syntax void **IMG_conv_3x3**(unsigned char *input_data, unsigned char *output_data, char *mask, int column, int shift);

Arguments

Inputs:

input_data	Points to an input image of 8-bit pixels
mask	Points to an 8-bit mask
column	Specifies the number of columns in the input image. Must be an even number
shift	Specifies the output shift number

Outputs:

output_data	Points to an output image of 8-bit pixels

Description The convolution kernel accepts three rows of column input pixels and produces one output row of column-2 pixels using the input mask of 3 by 3. The user-defined shift value is used to shift the convolution value down to the byte range. The shift amount is non-zero for low-pass filters, and zeros for high-pass and sharpening filters.

Algorithm Every output pixel results in the sum of the nine multiplications between mask and input image pixels.

Implementation Notes

Convolution loop is unrolled for optimized benchmark performance.

Dual MAC is implemented such that we calculate two convolutions each time.

Number of columns must be an even number.

Benchmark

Cycles 6x(column-2)+16

Data Size 89 words

Example See \examples\conv_3x3 subdirectory.

IMG_corr_3x3 — *3x3 Correlation With Rounding and Shifting*

Syntax void **corr_3x3**(unsigned char *input_data,unsigned char *output_data, unsigned char *mask, int row, int column, int shift, int round_val);

Arguments

Inputs:

input_data	Points to an input image of 8-bit pixels
mask	Points to an 8-bit mask
rows	Specifies the horizontal size of input image
column	Specifies the vertical size of input image. It must be even.
shift	Specifies the output shift number
round_val	Specifies the user-specified round value

Outputs:

output_data	Points to an output image of 8-bit pixels

Description The routine IMG_corr_3x3 performs a point-by-point multiplication of the 3x3 mask with the input image. The result of the nine multiplications are then added. The sum is rounded and shifted to produce an 8-bit value which is stored in an output image. The image mask to be correlated is typically part of the input image or another image. The mask is moved one column at a time, advancing the mask until the until input image is covered. The size of output image is (row-2)x(column-2).

Algorithm Every output pixel results in the sum of the nine multiplications between mask and input image pixels.

Implementation Notes

Correlation loop is unrolled for optimized benchmark performance.

Dual MAC is implemented such that we calculate two correlations each time.

Number of columns must be an even number.

Benchmark

Cycles 8.2x(row-2)x(column-2)+25

Data Size 165 words

Example See /examples/corr_3x3 subdirectory

IMG_scale_by_2 *Image Scaling (horizontally/vertically) by 2*

Syntax void **IMG_scale_by_2**(int *input_image, int *output_image, int row, int column);

Arguments

Inputs:

input_image	Points to the original image with an extension. The extension of the image are two columns attached to the original image. It is required by HWE. The image with the extension must by double-word aligned. Every four pixels are packed into one 32-bit double word.
row	Specifies the horizontal size of the input image with extension
column	Specifies the vertical size of the input image with extension

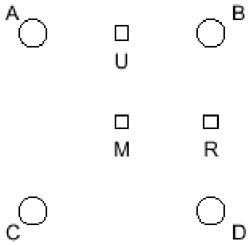
Outputs:

output_image	Points to the upscaled-by-2 image. Only the left (2xcolumn-4)columns are output corresponding to the original image. The right four columns are the symmetric padding. Every four pixels are packed into one 32-bit double word.

Description The routine IMG_scale_by_2 implements image scaling by linear pixel interpolation using built-in hardware extensions. To implement pixel interpolation for original image, we need to attach two columns to the original image as the extension. This is required by HWE. In the output of this routine, left (2xcolumn-4) columns are the scaled image corresponding the original image. The right four columns are the symmetric padding of the scaled-by-2 image.

Algorithm The pixel interpolation technique used by image scaling is described by the following equations and diagrams.

$$U = \frac{A + B + Rnd}{2} \quad M = \frac{A + B + C + D + 1 + Rnd}{4} \quad R = \frac{B + D + Rnd}{2}$$



where A, B ,C, and D are original pixels and U, M, and R are corresponding interpolated pixels in the scaled image. Using the above basic pixel interpolation technique to the whole image with extension, the scaled-by-2 image is retrieved.

Special Requirements Both original image with extension and output scaled image must be aligned to a 32-bit boundary.

Implementation Notes

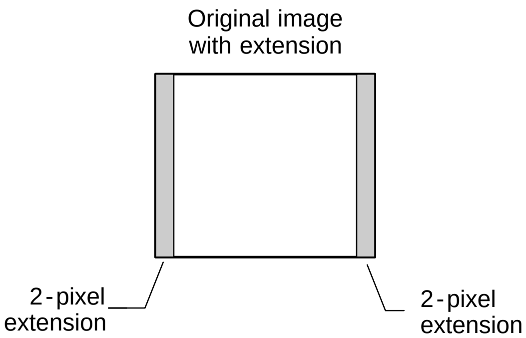


Figure 5-8. Organization of original image with extension.

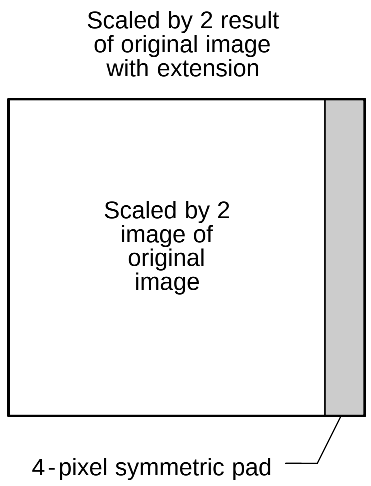


Figure 5-9. Organization of scale-by-2 output.

Example See /examples/scale_by_2 subdirectory.

Here is an example of the routine. The original image with extension is 128 x128.

Figure 5-10. Original image with extension (128x128)

The scaled-by-2 result is 256x256.

Figure 5-11. Scale-by-2 result of original image with extension.

Benchmark

Cycles 0.27x(2xrow)x(2xcolumn)+23

Code Size 216 words

Example See examples/image_scale_by2 subdirectory.

IMG_YCbCr422_rgb565

Planarized YCbCr 4:2:2 to RGB 5:6:5 Color Space Conversion

Syntax

void **IMG_ycbcr422_rgb565**(short coeff[], short *y_data, short *cb_data, short*cr_data, short *rgb_data, short num_pixels)

Arguments

Inputs:

coeff[7]	Matrix coefficients
y_data	Luminance data (Y’)
cb_data	Blue color-diff (B’-Y’)
cr_data	Red color-diff (R’-Y’)
rgb_data	RGB5:6:5 packed pixel out
num_pixels	Number of luma pixels to process

Description

This kernel performs Y’CbCr to RGB conversion. The ‘coeff[]’ array contains the color-space-conversion matrix coefficients. The ‘y_data’, ’cb_data’ and ‘cr_data’ pointers point to the separate input image planes. The ‘rgb_data’ pointer points to the output image buffer. The kernel is designed to process arbitrary amounts of 4:2:2 image data, although 4:2:0 image may be processed as well. For 4:2:2 input data, the ‘y_data’, ‘cb_data’ and ‘cr_data’ arrays may hold an arbitrary amount of image data. For 4:2:0 input data, only a single scan-line (or portion thereof) may be processed at a time. The coefficients in the coeff array must be in signed Q13 form. This code can perform various flavors Y’CbCr to RGB conversion as long as the offset on Y, Cb and Cr are -16, -128 and -128, respectively, and the coefficients match the pattern shown. The kernel implements the following matrix form, which involves 7 coefficients:

```
[ coeff[0]  0.000     coeff[1] ]      [ Y’-16  ]        [R’]
[coeff[2]   coeff[3] coeff[4] ]  *   [Cb’-128] =   [G’]
[coeff[5] coeff[6]  0.000 ]           [Cr – 128]      [B’]
```

Below are some common coefficient sets, along with the matrix equation to which they correspond. Coefficients are in Q13 notation, which gives a suitable balance between precision and range.

Y’CbCr -> RGB conversion with RGB levels that correspond to the 219-level range of Y’. Expected range are
[16..235] for Y’ and [16..240] for Cb and Cr.
Coeff[7]={0x2000, 0x2BDD, 0x2000 -0x0AC5, -0x1658, 0x2000, 0x3770};

```
[1.000      0.000   1.3707]     [Y’-16]          [R’]
```

[1.000 -0.3365 -0.6982] * [Cb-128] = [G’]
[1.000 1.7324 0.000] [Cr-128] [B’]

Y’CbCr->RGB conversion with the 219-level range of Y’ expanded to fill the full RGB dynamic range. (The matrix has been scaled by by 255/219). Expected ranges are [16..235] for Y’ and [16..240] for Cb and Cr.
Coeff[7]={0x2543, 0x3313, -0x0C8A, -0x1A04, 0x408D}
[1.1644 0.0000 1.5960] [Y’-16] [R’]
[1.1644 -0.3918 -0.8130] *[Cb-128] = [G’]
[1.1644 2.0172 0.0000] [Cr-128] [B’]

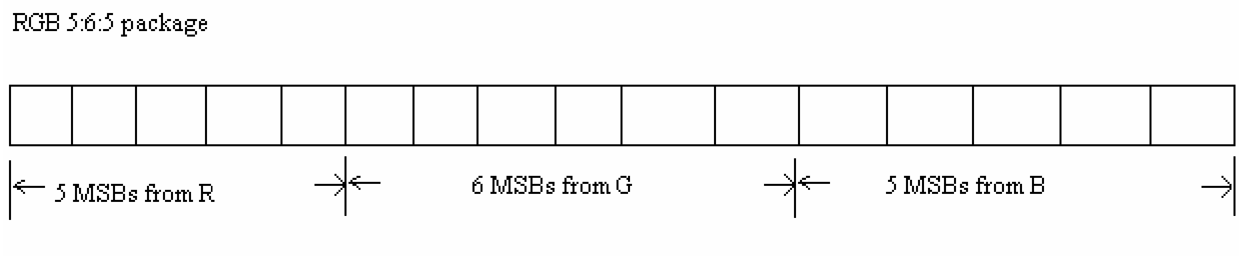
Other scalings of the color differences (B’-Y’) and (R’-Y’), (sometimes incorrectly referred as U and V), are supported, as long as the color differences are unsigned values centered around 128 rather than signed values centered around 0, as noted above. In addition to performing plain color-space conversion, color saturation can be adjusted by scaling coeff[1] through coeff[6]. Similarly, brightness can be adjusted by scaling coeff[0]. General hue adjustment cannot be performed, however, due to the two zeros hard-coded in the matrix.

Algorithm

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} c_1 & 0 & c_2 \\ c_3 & c_4 & c_5 \\ c_6 & c_7 & 0 \end{pmatrix} \begin{pmatrix} Y-16 \\ Cb-128 \\ Cr-128 \end{pmatrix}$$

where $c_1 = c_3 = c_6$ usually.

The output (R,G,B)’ is packed in the following way.



Special Requirements

Coeff cannot be allocated in the memory bank which is allocated to any of y_data, cb_data, cr_data for the best performance.

Implementation Notes

Matrix multiplication is performed as a series of Dual MAC followed by shift. In each loop, one pair of RGB 5:6:5 is calculated.

Small Memory Model

Benchmark 12xnum_pixels+47

Code Size 109 words

Data Size 3 words

Large Memory Model

Benchmark 12xnum_pixels+47

Code Size 111 words

Data Size 3 words

Example See the examples/ycbcr_rgb subdirectory.

Appendix A

Performance /Warranty and Support

This appendix describes performance considerations related to the C55x IMGLIB and provides information about warranty, software updates, and customer support issues.

Topic **Page**

A.1 Performance Considerations

Although IMGLIB can be used as a first estimation of processor performance for a specific function, you should be aware that the generic nature of IMGLIB might add extra cycles not required for customer specific usage.

Benchmark cycles presented assume best case conditions, typically assuming all code and data are placed in internal data memory. Any extra cycles due to placement of code or data in external data memory or cache-associated effects (cache-hits or misses) are not considered when computing the cycle counts.

You should also be aware that execution speed in a system is dependent on where the different sections of program and data are located in memory. You should account for such differences when trying to explain why a routine is taking more time than the reported IMGLIB benchmarks.

Table A-1 provides a listing of the routines provided in this software package as well as C55x performance data for each:

Table A-1. C55x Routines Performance Data

Function	Description	Cycles	Code Size
IMG_sw_fdct_8x8	2-D forward discrete cosine transform (DCT) for 8*8 image block without built-in hardware extensions	1078 cycles	188 words
IMG_sw_idct_8x8	2-D inverse discrete cosine transform (IDCT) for 8*8 DCT coefficients without built-in hardware extensions	672 cycles	177 words
IMG_fdct_8x8	2-D forward discrete cosine transform (DCT) for 8*8 image block using built-in hardware extensions	238 cycles	238 words
IMG_idct_8x8	2-D inverse discrete cosine transform (IDCT) for 8*8 DCT coefficients using built-in hardware extensions	165 cycles	165 words
IMG_jpeg_make_recip_tbl	Computation of the reciprocal table of the quantization terms	1413 cycles	16 words
IMG_jpeg_quantize	Matrix quantization	215 cycles	31 words
IMG_dequantize_8x8	Matrix dequantization	132 cycles	10 words
IMG__jpeg_vlc	JPEG VLC	data-dependent	501 words
IMG__jpeg_vld	JPEG VLD	data-dependent	634 words
IMG_mad_8x8	8x8 minimum absolute difference	38.5*sx*sy+95 cycles	267 words
IMG_mad_16x16	16x16 minimum absolute difference	85.5*sx*sy+95 cycles	304 words
IMG_mad_16x16_4step	Motion estimation by 4-step searching	2225 cycles	429 words
IMG_sad_8x8	Sum of absolute differences on single 8x8 block	52 cycles	28 words
IMG_sad_16x16	Sum of absolute differences on single 16x16 block	156 cycles	28 words

Table A-1. C55x Routines Performance Data

Function	Description	Cycles	Code Size
IMG_pix_inter_16x16	Pixel interpolation	530 cycles	462 words
IMG_wave_decom_one_dim	1-D discrete wavelet transform (function *decomInplace* is called)	[(Filter Length + 3)/2] × [Signal Length] *[2-2^ (1-level)]+87 Cycles	IMG_wave_decom_one_dim: 32 words DecomInplace: 67 words
IMG_wave_recon_one_dim	1-D inverse discrete Wavelet Transform (function *reconInplace* is called)	[(Filter Length + 4)/2] × [Signal Length] ×[2-2^ (1-level)] + 85 Cycles	IMG_wave_recon_one_dim: 39 words reconInplace: 85 words
IMG_wavep_decom_one_dim	1-D discrete wavelet package transform (function *decomInplace* is called)	[(Filter Length + 3)/2] × [Signal Length] × [Level] + 90 Cycles	IMG_wavep_decom_one_dim: 57 words DecomInplace: 67 words
IMG_wavep_recon_one_dim	1-D discrete inverse wavelet package transform (function *reconInplace* is called)	[(Filter Length + 4)/2] × [Signal Length] × [Level] + 94 Cycles	IMG_wavep_recon_one_dim: 69 words reconInplace: 85 word
IMG_wave_decom_two_dim	2-D discrete wavelet transform (functions *decomInplace*, *col2row* and *decomCo*l are called)	Cycles 35 + 13 **x** level + (322 **x** height +210 **x** width) **x** [1-2^(-level)] + 4/3 **x** width **x** height **x** (filter length + 2) **x** [1-4^(-level)]	IMG_wave_decom_two_dim: 71 words DecomInplace: 67 words col2row : 6 words decomCol: 56 words
IMG_wave_recon_two_dim	2-D inverse discrete wavelet transform (functions *interlaceCol, reconCol, reconInplace* are called)	Cycles 13 **x** level + 246 **x** height **x** [1-2^(-level)] + 448 **x** width **x** [1-2^(-length)] + 4/3 **x** (filter length +4) **x** [1-4^(-level)]	IMG_wave_recon_two_dim: 87 words InterlaceCol: 43 words ReconCol: 58 words reconInplace. 85 words

Table A-1. C55x Routines Performance Data

Function	Description	Cycles	Code Size
IMG_wavep_decom_two_dim	2-D discrete wavelet package transform (functions *decomInplace*, *col2row* and *decomCol* are called)	Cycles (110 **x** height + 137 **x** width) **x** (2^level -1) + (filter length **x** 0.25 + 2.25) **x** (width **x** height) **x** level + 20 **x** (width + height) **x** level	IMG_wavep_decom_two_dim: 101 words DecomInplace: 67 words col2row : 6 words decomCol: 56 words
IMG_wavep_recon_two_dim	2-D discrete inverse wavelet package transform (functions *interlaceCol, reconCol, reconInplace* are called)	Cycles Level **x** [32 + 123 **x** height + 224 **x** weight + (filter length +4) **x** width **x** height]	IMG_wavep_recon_two_dim: 87 words InterlaceCol: 43 words ReconCol: 58 words reconInplace. 85 words
IMG_boundary	Boundary structural operator	5.125 × (cols × rows) + 8 × rows + 14 cycles	44 words
IMG_histogram	Histogram computation	2.25 × size + 18	33 words
IMG_perimeter	Perimeter structural operator	(cols - 2) × 7 + 21	49 words
IMG_threshold	Image thresholding	cols × rows × 2.5 + 16 cycles	28 words
IMG_conv_3x3	3x3 convolution	6x(column-2)+16	89 words
IMG_corr_3x3	3x3 correlation with rounding	8.2x(row-2)x(column-2)+25	165 words
IMG_Scale_by_2	Image up-scale by a factor of 2	0.27x(2xrow-2)x(2column-2)+23	216 words
IMG_ycbcr422_rgb565	Planarized yCbCr 4:2:2/4:2:0 to RGB 5:6:5 color space conversion	12*num_pixels+76 cycles	111 words

A.2 Warranty

The C55x IMGLIB is distributed free of charge.

BETA RELEASE SPECIAL DISCLAIMER: This IMGLIB software release is preliminary (beta). It is intended for evaluation only. Testing and characterization has not been fully completed. Production release typically follows the beta release but there are no explicit guarantees.

A.3 IMGLIB Software Updates

C55x IMGLIB software updates may be periodically released incorporating product enhancements and fixes as they become available. You should read the README.TXT available in the root directory of every release.

A.4 IMGLIB Customer Support

If you have questions or want to report problems or suggestions regarding the C55x IMGLIB, contact Texas Instruments at dsph@ti.com.

Index

J

M