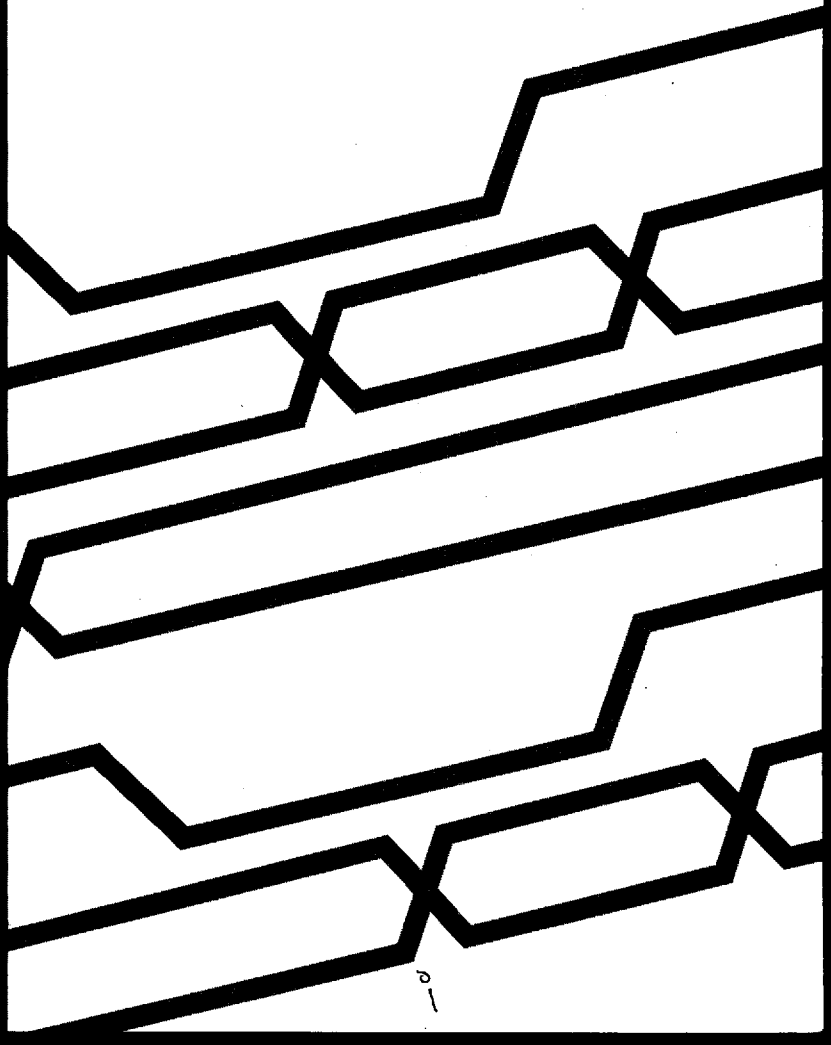
LOGIC

L64702
JPEG Coprocessor
Technical Manual

July 1993



This document is preliminary. As such, it contains data derived from functional simulations and performance estimates. LSI Logic has not verified either the functional descriptions, or the electrical and mechanical specifications using production parts.

First Edition
Document Number MD71-000102-99 A
This document applies to revision B of the L64702 and to all subsequent versions unless otherwise indicated in a subsequent edition or an update to this edition of the document.

Publications are stocked at the address given below. Requests should be addressed to:

LSI Logic Corporation
Literature Distribution, M/S D-102
1551 McCarthy Boulevard
Milpitas, CA 95035
Fax: 408.433.8989

Preface

This book is the primary reference and user's manual for the L64702 JPEG Coprocessor. It contains a complete functional description for the L64702 and includes complete physical and electrical specifications for the L64702.

Audience

This book assumes that you have some familiarity with digital signal processing, microprocessors, and related support devices. The people who benefit from this book are:

- Engineers and managers who are evaluating the L64702 JPEG Coprocessor for possible use in a system.
- Engineers who are designing the L64702 into a system.

Organization

This book has the following chapters:

- Chapter 1, **Introduction**, defines the general characteristics and capabilities of the L64702.
- Chapter 2, **Interface Description**, describes the characteristics of the L64702 signals that are used to interface with an external CPU, DMA controller, or video memory.
- Chapter 3, **Internal Registers and Data Tables**, describes how data is represented inside the L64702. This chapter also provides a summary of the L64702 registers and tables.
- Chapter 4, **System Interface**, describes the CPU and DMA transfer modes supported by the L64702.
- Chapter 5, **JPEG Processing Unit**, discusses the role of the L64702 JPEG Coprocessor in image compression and decompression.
- Chapter 6, **Compressed Image Data Structure**, describes the JPEG file structure and organization for both compressed and decompressed images.
- Chapter 7, **Video Memory Organization**, describes how the image data is represented in external video memory and describes raster-to-block conversion, image sub-sampling, and color memory organization.

- Chapter 8, **Pixel Data Processing Overview**, presents a summary of pre- and post-processing the L64702 performs on image data. It describes the functions of the Pixel Reformatting Buffer, Color Space Converter, MCU Buffer, and explains level shifting and range limiting.
- Chapter 9, **Video Memory Interface Port**, describes the interface between the L64702 and an external video memory bus. Included are descriptions of the video bus arbitration protocol, video bus request criteria, and video bus interface timing details.
- Chapter 10, **System Interface Port**, discusses the interface timing details of CPU and DMA transfers to and from various registers and tables within the L64702.
- Chapter 11, **System Software Control**, describes how the system software interacts with the L64702 in order to perform L64702 initialization as well as image compression and decompression.
- Chapter 12, **Specifications**, describes the electrical and mechanical characteristics of the L64702.
- Appendix A, **The JVieW™ (JPEG Video-in-a-Window) Multimedia Kit**, describes the JVieW evaluation board, a single-board multimedia design that allows full-motion video to be displayed on a PC.
- Appendix B, **Huffman Table Software Listing**, describes the source code that generates the Huffman Tables stored in the L64702 RAM.
- Appendix C, **Customer Feedback**, includes a form that you may use to fax us your comments about this document.

Related Publications

JPEG Chipset Technical Manual, Order No. I1400.A

JVieW (JPEG Video in a Window) Evaluation Kit User's Guide, Document No. MD75-000102-99 A

Conventions Used in this Manual

The first time a word or phrase is defined in this manual, it is *italicized.*

The following signal naming conventions are used throughout this manual:

- A level-significant signal that is true or valid when the signal is LOW always has an overbar ($\overline{\quad}$) over its name.
- An edge-significant signal that initiates actions on a HIGH-to-LOW transition always has an overbar ($\overline{\quad}$) over its name.

The word *assert* means to drive a signal true or active. The word *deassert* means to drive a signal false or inactive.

Hexadecimal numbers are indicated by the prefix "0x" before the number—for example, 0x32CF. Binary numbers are indicated by a subscripted "2" following the number—for example, $0011.0010.1100.1111_2$.

Contents

Figures

Tables

Equations

Chapter 1
Introduction

This document describes the system interface and the architecture specification of the L64702 JPEG Coprocessor, and contains the following sections:

- Section 1.1, Processor and System Overview
- Section 1.2, Features
- Section 1.3, System Description
- Section 1.4, JPEG Image Compression
- Section 1.5, Standards Compliance
- Section 1.6, Terms and Concepts
- Section 1.7, Applications

1.1 Processor and System Overview

The L64702 is a single-chip JPEG coprocessor designed for microcomputer graphics and video applications including personal computers, engineering workstations, true-color display cards and monitors, and color laser printers. The L64702's advanced software and hardware interfaces make applications and system-level programming efficient and straightforward. Its performance and high integration make the L64702 a cost-effective component while improving substantially the performance over any system that uses general-purpose DSP devices for JPEG image compression or decompression tasks.

The L64702 is designed to support several image formats dedicated to specific needs within the microcomputer environment. The high-level system approach uses image compression to reduce storage capacity needs and eliminate data transfer bottlenecks over the system bus. Using the JPEG standard for file compression can achieve these benefits while preserving excellent reproduction results.

The L64702 is a half-duplex device, meaning that it can support either image compression or image decompression at one time. The L64702 complies with the latest JPEG baseline recommendation standard (JPEG DIS 10918). Byte stuffing, byte stripping, and restart marker codes are supported. In addition, the device provides user-programmable DCT coefficients, as well as color-space and format conversion capabilities, making the device a flexible and simple solution for image compression and decompression tasks.

Figure 1.1 is a general block diagram of the L64702, and Figure 1.2 is a more detailed block diagram.

Figure 1.1
L64702 General
Block Diagram

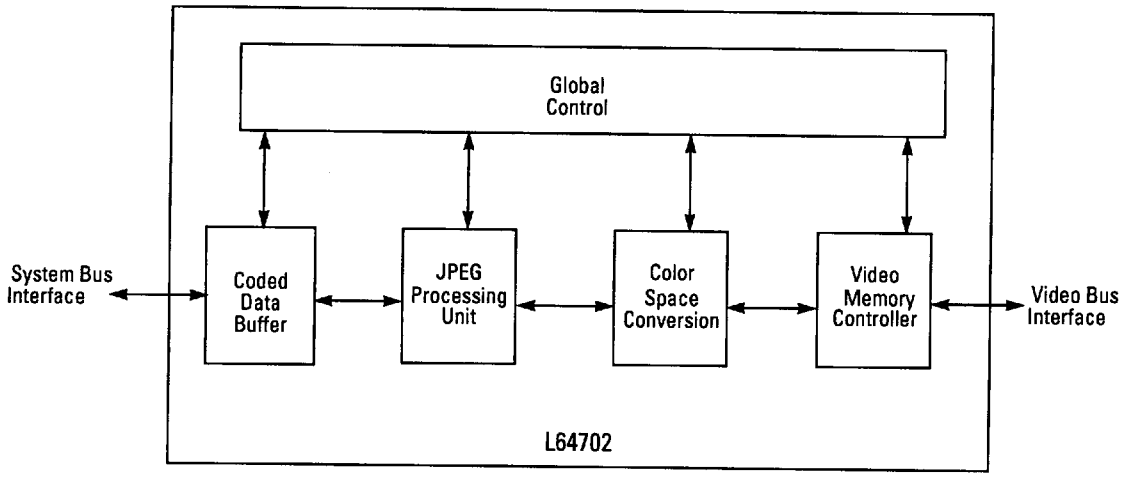
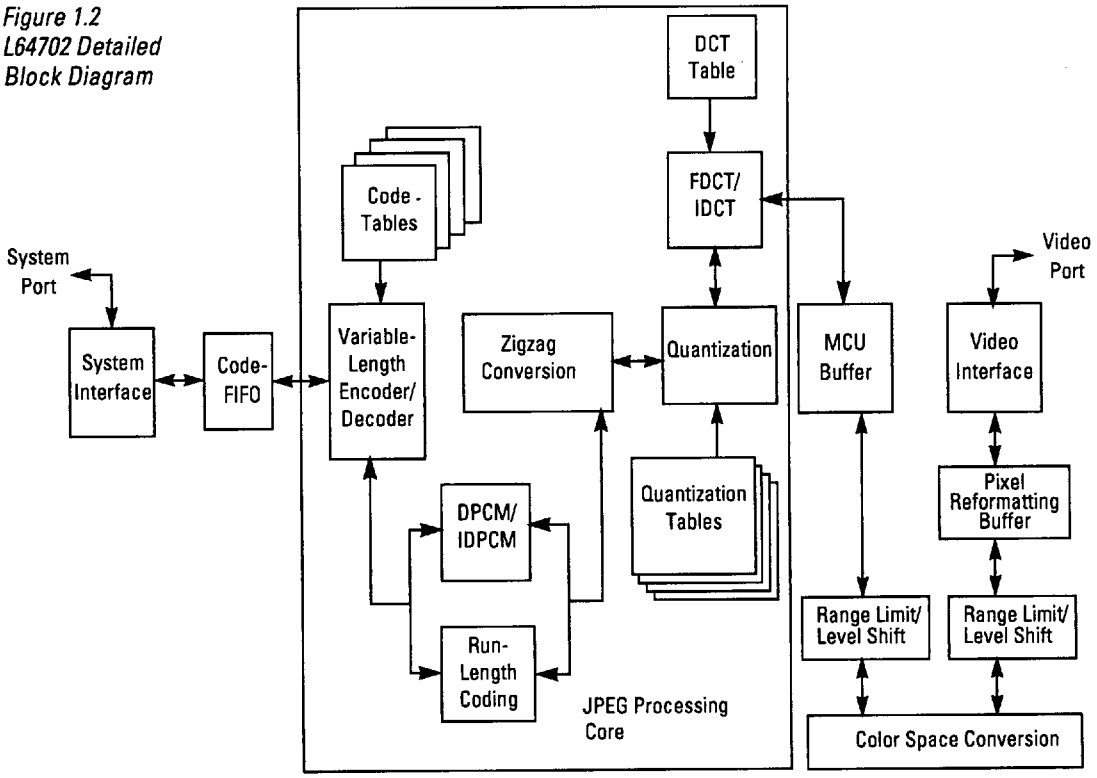


Figure 1.2
L64702 Detailed
Block Diagram



The main blocks in the L64702 that comprise the JPEG processing core are:

- 8 x 8 FDCT/IDCT
- Quantization
- DPCM/IDPCM, Run Length Coding, Zigzag Conversion
- Variable Length Encoder/Decoder

The JPEG processing core communicates with two independent interfaces, the video port and the system port. The JPEG processing unit adjusts the processing rate to the bus activity on both sides and temporarily suspends processing when needed with no risk of data stream breakdown. This capability is extremely important for a device that operates in a CPU environment where data and tasks are changing randomly. The System and Video Interfaces allow simple, straight-forward connection of the L64702 to the system bus on one side and video memory on the other side.

On the video port side, the L64702 buffers image data in one of two Minimum Code Unit (MCU) buffers. Each MCU buffer is 256 bytes deep and operates in a ping-pong fashion. While one MCU buffer is transferring data to or from the JPEG processing core, the other MCU buffer is transferring image data to or from external video memory. At the end of each transfer, the buffers are swapped so that there is an efficient and continuous image data transfer between the L64702 and external video memory.

On the video port, the chip supports colorspace conversion of three components with nine down-loadable coefficients, which allow any user-defined color-space conversion (RGB to YCrCb, RGB to YUV, etc.). When the L64702 is programmed to handle four components (CYMK), it bypasses the color-space conversion block and generates the proper addressing scheme to convert the raster-ordered image structure in the video memory to block-ordered data required by the L64702 JPEG processing unit. A set of three registers specify the subsampling scheme for each one of the color components.

The L64702 contains integrated DRAM/VRAM controllers with programmable and advanced windowing and addressing mechanisms. Within the programmable active window, the L64702 generates the addresses for the raster-to-block or block-to-raster conversion for a variety of image data structures and subsampling schemes. The L64702 accesses the video memory in master fashion using its own memory controller unit.

The L64702 contains a complete set of control signals that allow direct interface to DRAM/VRAM. The L64702 can be programmed to periodically generate memory refresh cycles. It can also perform periodic screen refresh or image capture functions utilizing the memory-to-shift register or shift register-to-memory transfer cycle. The split read transfer mode is also supported along with regular memory read and write cycles.

The system port is dedicated to compressed data or configuration data transfers. The L64702 operates as a slave peripheral from this port. Memory-mapped data transfers are controlled by an external CPU. DMA data transfers are controlled by an external DMA controller. The system interface port contains all necessary signals to perform handshaking with an external CPU or DMA controller.

The FIFO organization is 32 words by 16 bits. The Code-FIFO buffers data between the JPEG processing unit and the external 16-bit system bus. The

buffering rate can go as high as 40 Mbyte/second. The FIFO also allows burst transfers of compressed data.

Memory-mapped configuration tables and status registers can be accessed by an external CPU via the SRS[1:0] pins in conjunction with an automatically incrementing address pointer. An external DMA controller can access the FIFO or the configuration tables.

The L64702 provides a bypass mode, which allows indirect CPU access of video memory through the L64702 device. This feature allows for easy testing and manipulation of image data in the video memory without any extra hardware needed to provide an interface between the system bus and the video bus. In addition, it allows a low-cost CPU or microcontroller, which may have limited addressing ability, to access up to 64 Mbytes.

1.2 Features

This section summarizes the key features of the L64702:

- Implements the proposed baseline JPEG DIS 10918 standard for image compression.
- Supports the detection of restart (RSTn), End Of Image (EOI), and Illegal Marker (IMC) marker codes, byte stripping (during decompression), and byte stuffing (during compression).
- Supports the specification of a compressed file active window structure, allowing small window scrolling in a large compressed file.
- Contains four downloadable quantization tables.
- Contains downloadable Huffman code tables (two DC and two AC tables).
- Supports downloading of all 64 DCT or IDCT coefficients.
- Provides a bypass mode that is capable of transferring original image data to or from memory at a rate of 16.5 Mbytes/second.
- Provides a flexible color-space transformation capability with a downloadable, 3x3 matrix to allow any linear transformation of three-element vectors.
- Contains a dual-port architecture consisting of a video memory bus interface and an asynchronous system bus interface.
- Includes a multiplexed address and data bus. Supports 24-bit addressing capability allowing direct access up to 16M, 32-bit words.

- Contains an on-chip video memory controller that integrates a programmable active window mechanism and raster-to-block or block-to-raster address mechanism.
- Processes data transfer rates of up to 8.25 Mbytes/second during compression or decompression with a 33-MHz external clock. Some of the L64702 image data processing rates supported include:
  - 4.125 Mpixels/second for 2:1:1 color (RGB_x)
  - 2.75 Mpixels/second for 1:1:1 color (RGB_x)
  - 8.25 Mpixels/sec for component sequential (CMYK) in any subsampling scheme
- Supports three-component color (for example, RGB) as well as up to four sequential components (for example, CMYK).
- Supports the following color image structures:
  - Multiple Sources, RR.....R, GG.....G, BB.....B
  - Unpacked Color, RGBxRGBx...RGBx
  - Packed Color, RGBRGBRG....BRGB
  - The components sequential image data structure (up to four components)
- Supports the following image structure subsampling schemes:
  - 1:0:0 is used for color to gray scale conversion
  - 1:1:1 has no subsampling
  - 2:1:1 subsamples components 2 and 3 by factor of two
  - Components sequential (four components) at any sampling scheme allowed by the JPEG baseline specification. Both horizontal and vertical subsampling schemes are supported.
- Supports a handshake mechanism for sharing the video bus with other bus master processors.
- Provides direct support for VRAM and DRAM, including all control and refresh signals for five operating schemes:
  - read cycle
  - write cycle
  - $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh cycle

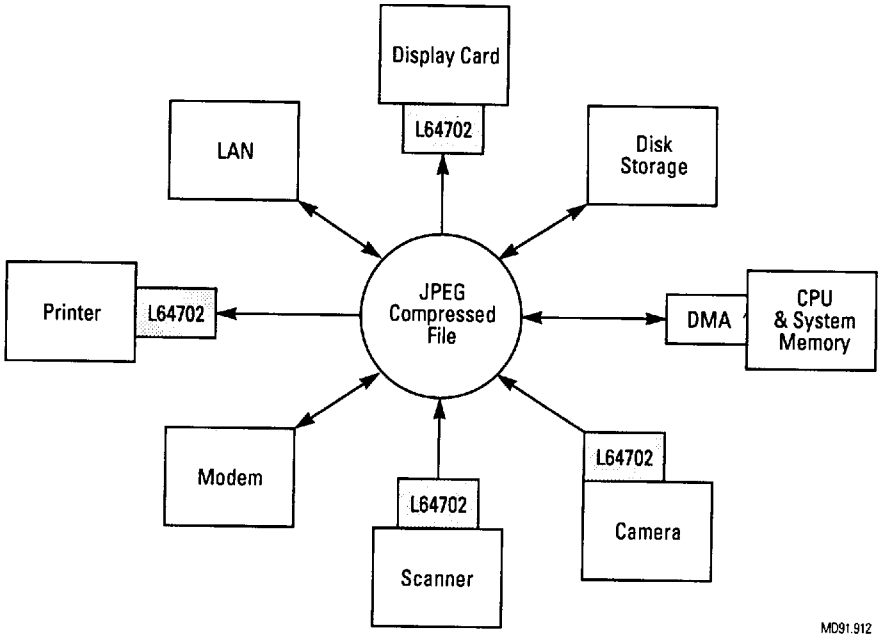
  - memory-to-shift register cycle (Split Read and Normal Read Transfers supported)
  - shift register-to-memory cycle (Pseudo Write and Alternate Write Transfers supported)

- Provides control signals for flexible multiplexing of row and column addresses and for easy implementation of video banks and interleaved memory structures.
- Supports an asynchronous system interface, which gives the L64702 the capability to operate either as a slave I/O device to an external CPU or as a peripheral device communicating with an external DMA controller.
- Supports 16-bit system bus configurations.
- Includes two external address pins with an internal auto-increment address pointer that implements the memory mapping of all accessible internal resources.
- Includes a programmable threshold for the FIFO, which allows burst code data transfers between the FIFO and system memory.
- Places no restrictions on the data rate from both ports. The JPEG pipeline processing unit can temporarily suspend operation at any time during the process without compromising data integrity.
- Packaged in a 100-pin Plastic Quad Flat Pack (PQFP) package.
- Fabricated using a 0.7 µm, CMOS process with maximum power dissipation of 1.5 watts at 33 MHz.

1.3 System Description

Figure 1.3 shows the high-level system approach behind the L64702 device. The chip is designed to serve as either a stand-alone or coprocessor device in each one of the microcomputer peripheral nodes, capable of compressing or decompressing images in various formats.

Figure 1.3 File Interchange in a Microcomputer System



This section describes the operation of the L64702 in three configurations: coprocessor, stand-alone, and single-bus coprocessor. Coprocessor configurations are used in applications that require both high-speed image compression/decompression and high-speed graphics processing. Stand-alone configurations are intended for systems in which the main system CPU performs the graphics or image processing. Single-bus coprocessor configurations are useful for systems that include printers.

Coprocessor Configuration

Figure 1.4 shows a basic coprocessor system, which includes the major blocks of a display card design that uses the L64702 JPEG coprocessor and an existing graphic coprocessor. This system configuration applies to high-performance display card applications requiring both high-speed image compression or decompression and high-speed graphics processing. The L64702 improves the processing time when image compression or decompression is needed.

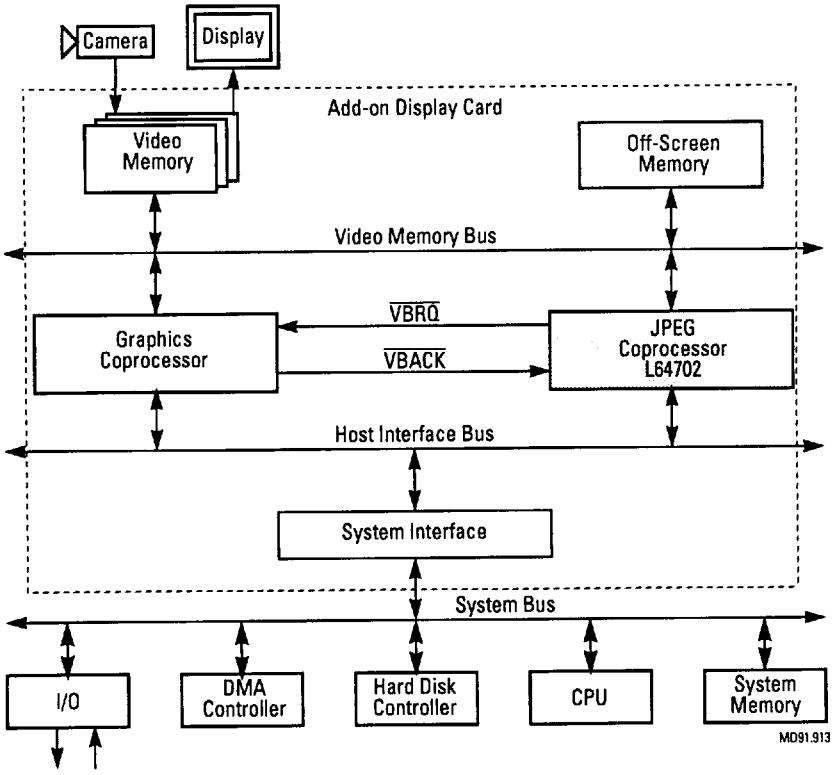


Figure 1.4 Coprocessor System Configuration

The configuration shown in Figure 1.4 is similar to the floating-point coprocessor method used in many microprocessor systems. The system software driver, which monitors both the graphics and JPEG coprocessors, controls the decision to start image compression or decompression, and uses the L64702 to accelerate the operation.

The L64702 interacts with the graphics coprocessor when the L64702 needs the video bus. Both coprocessors can operate concurrently and use the same video bus. The L64702 asserts the $\overline{\text{VBRQ}}$ signal to request video bus mastership. The graphics coprocessor asserts $\overline{\text{VBACK}}$ to acknowledge the request. The L64702 maintains control of the video bus for as long as $\overline{\text{VBRQ}}$ and $\overline{\text{VBACK}}$ are asserted.

To temporarily gain control of the video bus, the graphics coprocessor can deassert $\overline{\text{VBACK}}$, in which case the L64702 relinquishes the bus immediately. Worst case delay is one video bus cycle (8 CLKs). The L64702 resumes control of the video bus when the graphics coprocessor reasserts $\overline{\text{VBACK}}$.

The camera and display monitor connect to the video memory serial port. The L64702 is capable of performing both image capture and screen refresh cycles through its video port interface.

The system interface block connects the graphics coprocessor and the L64702 to the system bus. The L64702 system port transfers compressed or configuration data between the L64702 and either the system memory or the system disk. Either a CPU slave access or a DMA service can accomplish this data transfer.

The graphics coprocessor uses the off-screen memory as scratchpad memory for windows applications, video editing, or special effects.

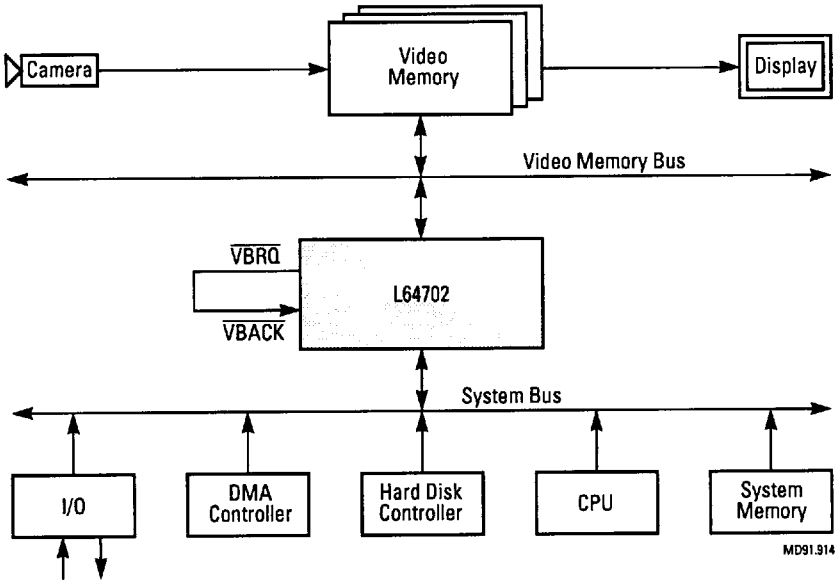
Stand-Alone Configuration

The stand-alone configuration allows the video bus to be completely devoted to the L64702. The $\overline{\text{VBACK}}$ input connects to the $\overline{\text{VBRQ}}$ output. Therefore video bus access is granted to the L64702 upon every $\overline{\text{VBRQ}}$ request.

Figure 1.5 shows a compression/decompression implementation with the L64702 resident on the CPU system bus and used as a coprocessor or hardware accelerator for JPEG image compression and decompression tasks. The CPU can indirectly access the video memory through the L64702. Indirect video memory access allows a CPU with limited addressing capability to address up to 64 Mbytes of DRAM/VRAM.

Image manipulation using the indirect method as portrayed in Figure 1.5 is limited in performance when compared to the graphics coprocessor configuration shown in Figure 1.4. However, for low-cost applications, the indirect method is quite acceptable.

Figure 1.5
Stand-Alone System



Single-Bus Coprocessor Configuration

Figure 1.6 shows the single-bus coprocessor configuration. This configuration is most likely to be implemented in printer systems. The CPU must access the print buffer memory to perform the task of print page making. The L64702 system port and video port both connect to the system bus.

In this configuration, the system bus has three possible master devices: the CPU, the DMA Controller, and the L64702. The CPU, by definition, is the primary master and has the highest priority. The DMA Controller and the L64702 are secondary masters that share the bus in a round-robin arbitration scheme. The advantage of this scheme is that the L64702 system port utilizes an external DMA Controller for compressed image data transfers or device configurations, while the video port acts as a master port sharing the system bus with other bus masters. During image compression or decompression the system bus is shared between DMA transfers of compressed data (performed by the external DMA Controller) and image data transfers (performed by the L64702).

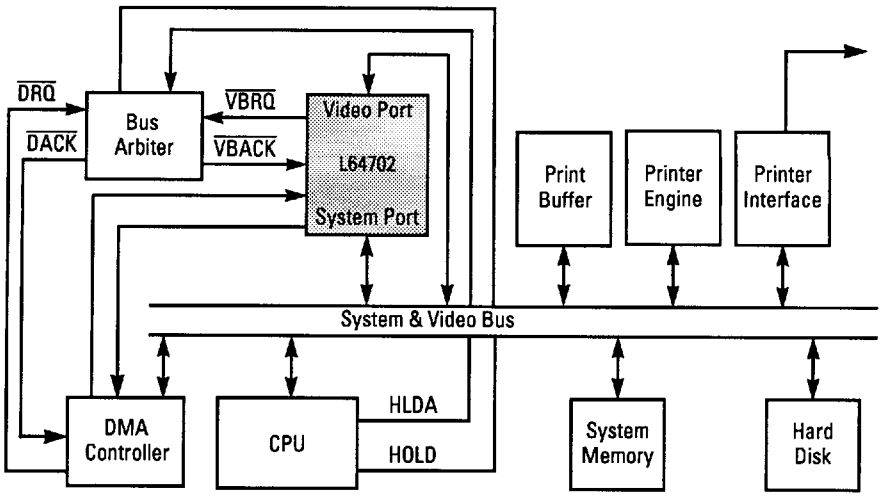


Figure 1.6 Single-Bus Coprocessor Configuration

1.4 JPEG Image Compression

When trying to implement multimedia capability in a low cost PC environment, the system designer faces two major problems: limited PC bus bandwidth and storage capacity.

An analog video signal must first be digitized so the computer can process the image data. A typical digitized image contains a huge amount of information. For instance, a 640 x 480 pixel image at 16-bit bits per pixel resolution requires almost 300 Mbytes of storage for only 15 seconds of digital video (at 30 images per second). This size is more than the size of a typical hard disk in a PC system.

Image compression solves this data storage problem. The Joint Photographers Expert Group (JPEG) has created an international standard for image compression. The JPEG standard was originally invented for still image applications, but has proved to be very useful in video applications as well. Using the JPEG algorithm, an image can be compressed 24 to 1 and still retain very good display quality. The image quality can be improved, however, but the higher the image quality, the greater the amount of data storage and bus bandwidth required. At a 24-to-1 compression ratio a user can store more than six minutes of realtime video on a 300 Mbyte hard disk.

The hard disk recording and playback speed may limit the multimedia PC's ability to record and display in realtime. In PCs, the sustained hard disk read or write rate can vary from 150 to 500 Kbytes per second. More expensive hardware, like SCSI disk drives, provide up to 1 Mbyte per second. CDROMs typically have a transfer rate of 300 Kbytes per second.

At a 500 Kbyte per second access rate the video data must be compressed by a factor of almost 40 to 1. This very high compression ratio delivers poor quality image reproduction, so it has become quite common to use quarter screen (320 x 240) size images. A 320 x 240 image requires one fourth the data of a 640 x 480 image, so only a 10 to 1 compression ratio is needed to achieve the data transfer rate of 500 Kbytes per second required for realtime digitization and playback from disk. At an access time of 300 Kbytes per second, a CDROM requires a 15-to-1 compression ratio.

LSI Logic's L64702 JPEG Coprocessor provides programmable compression ratios with the ability to process up to 8.25 Mbytes of pixel data per second.

JPEG also allows compatibility among the various image standards found in desktop multimedia systems containing cameras, copiers, and scanners. Although each produces or displays images in different ways and in conformance with different standards, JPEG allows for intercommunication among all of them.

1.5 Standards Compliance

The L64702 conforms to these international standards:

- CCIR Recommendation 601 (Mod F), *Encoding Parameters of Digital Television for Studios*
- JPEG DIS 10918, *Digital Compression and Coding of Continuous-tone Still Images*

The compliance of the L64702 with each of these standards is discussed below.

CCIR 601 (Mod F)

The CCIR Recommendation 601 presents a proposed standard for color space conversions. CCIR 601 specifies the conversion between R, G, B and Y, Cr, Cb; and other aspects of color conversions. The L64702 internal Color Converter complies with CCIR 601 (Mod F).

JPEG DIS 10918

The Joint Photographic Experts Group (JPEG) is proposing an international standard for the compression and decompression of still photographic images. LSI Logic designed the L64702 internal DCT Processor and JPEG Coder to comply with JPEG DIS 10918.

In the terminology of the JPEG standard, the L64702 supports baseline sequential DCT-based Huffman coding for eight-bit source images. The L64702 also supports interleaved data containing up to four components.

1.6 Terms and Concepts

This section explains some important terms and concepts used in the rest of this guide. The subsections are:

- Minimum Coded Unit (MCU)
- MCU Boundaries
- Display Active Image Area
- Compressed Image Active Window
- Color Conversion
- Subsampling
- Image Reconstruction
- Raster-to-Block Conversion
- DCT Functions
- Quantization Function
- Zigzag Ordering
- DPCM Coding
- Run-Length Coding
- Variable-Length Coding (Huffman Coding)
- Encoding Example

Minimum Coded Unit (MCU)

The JPEG DIS 10918 standard specifies the concept of a *minimum coded unit (MCU)*. An MCU is a sequence of data units (8 x 8 blocks) defined by the sampling factors of the image components. The L64702 supports a wide variety of sampling factors. The sampling factor determines the size of the MCU.

When the L64702 operates in the color mode and color space conversion is used, three color components are supported (for example, RGB or YCrCb). The color space conversion is completely programmable, allowing any number of color conversion possibilities. Three sampling factors are used to define the MCUs. The sampling factors are 1:0:0, 1:1:1, or

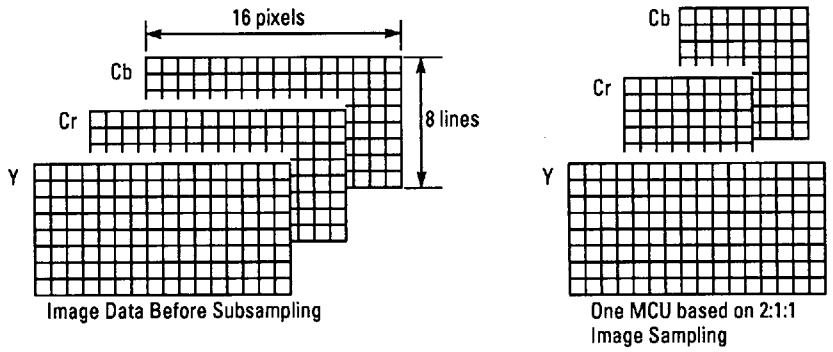
2:1:1. A sampling factor of 1:0:0 indicates that an MCU is constructed from one 8 x 8 block of the first color component only. A sampling factor of 1:1:1 indicates that an MCU is constructed from one 8 x 8 block from each color component.

When an image consists of the YCrCb color components, for example, and 2:1:1 sampling is implemented, each MCU contains two 8 x 8 data blocks of the *luminance* (intensity) component, Y, and one 8 x 8 data block each of the two *chrominance* (color) components, Cr and Cb.

With 2:1:1 sampling, the MCU is constructed from an image area of 16 pixels wide by eight lines high. Every pixel of the Y component is included in the sample, but only every other pixel of the Cr and Cb components is included. The resulting MCU consists of four 8 x 8 blocks—two Y blocks, one Cr block and one Cb block.

Figure 1.7 shows how an MCU is constructed with 2:1:1 subsampling.

Figure 1.7 Construction of an MCU from Three Color Components

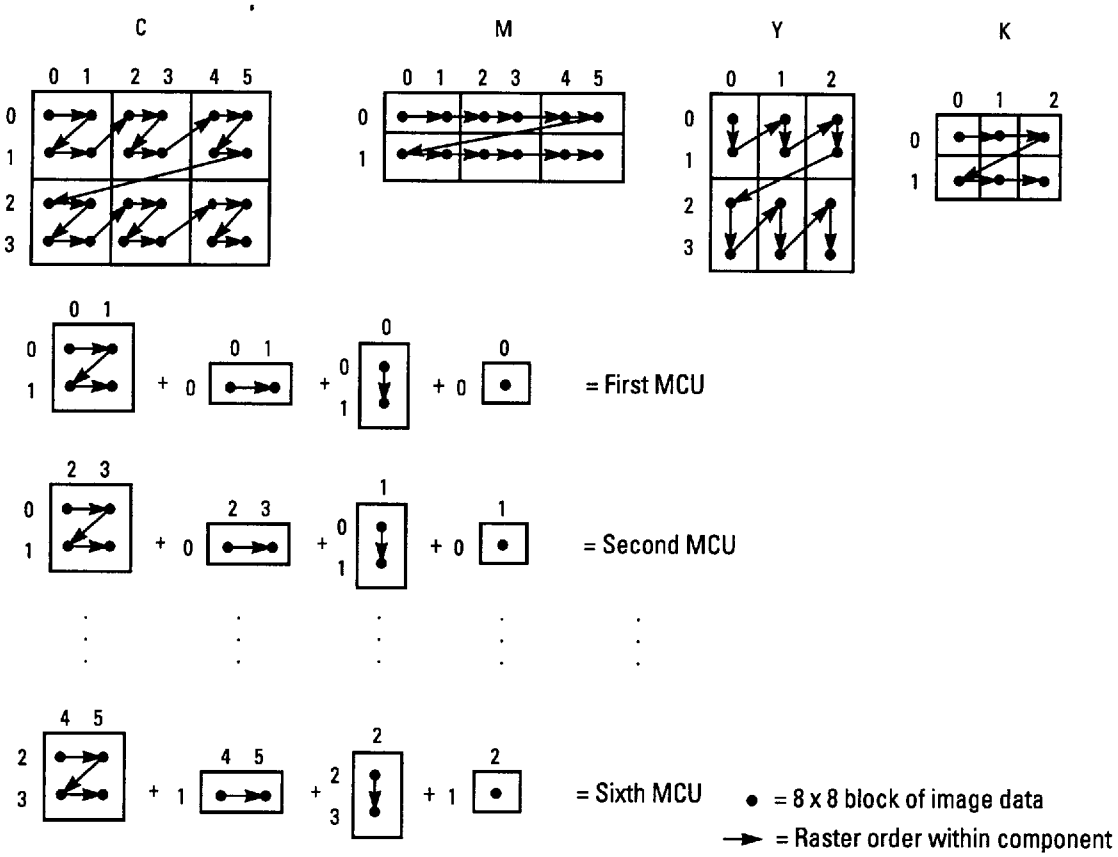


When the L64702 operates in the components sequential mode, it does not perform color space conversion, and supports up to four color components. The L64702 is compatible with any sampling factor specified by the baseline JPEG standard.

Figure 1.8 illustrates how an MCU is constructed from a four-color image. The sampling ratio is 4:2:2:1, which means that the MCU is constructed from four 8 x 8 blocks of the C component, two blocks of the M component, two blocks of Y and one block of K. Registers in the L64702 allow the construction of an MCU to be completely described, in terms of how many blocks are assembled from each color component, and the vertical and horizontal ordering in which the assembly takes place. Each MCU

contains a total of 9 8 x 8 blocks (4 + 2 + 2 + 1). Each dot in the figure represents an 8 x 8 block of image data.

Figure 1.8
Construction of an MCU from Four Color Components



The MCU is constructed from an image area 16 pixels wide by 16 lines high (four 8 x 8 blocks). Every pixel of the C component is included in the sample, but only every other pixel of the M and Y components is included, and every fourth pixel of the K component.

The components of an image are subsampled *prior* to image compression, and the L64702 assumes that the components are stored in the memory in four separate regions.

The first MCU is constructed of 8 x 8 blocks taken first from the top left-most region of the C color component (four blocks), followed by two blocks from the same region of the M component, then from Y, and finally from K. The second MCU is constructed following the same ordering for data taken from the next region to the right for the four components. This same procedure is followed for constructing all six MCUs.

MCU Boundaries

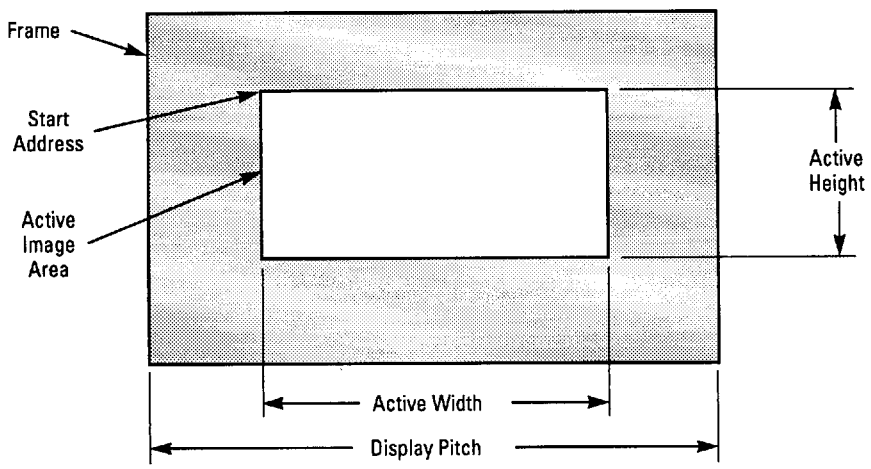
If the image size is such that the edge of the image does not fall on an MCU boundary, the L64702 fills the remainder of the MCU with zeroes to make sure that all compressed data consists of complete MCUs.

Display Active Image Area

In many applications, only a portion of the source image stored in video memory is processed. This portion of the image is known as the *active image area*. The active image area is defined in terms of parameters written by the CPU to the L64702 internal registers. These parameters are: *start address, active height, active width,* and *display pitch.*

Figure 1.9 shows the definition of these parameters relative to the complete image. The start address defines the location in video memory where the active image begins. The active height, active width, and display pitch are measured in MCUs and provide the L64702 with information on the size of the active window relative to the rest of the display area. The shaded portion of the frame is outside of the active image area and is ignored by the L64702. Data in the shaded region is not accessed by the L64702 during image compression or decompression.

Figure 1.9
Active Image Area

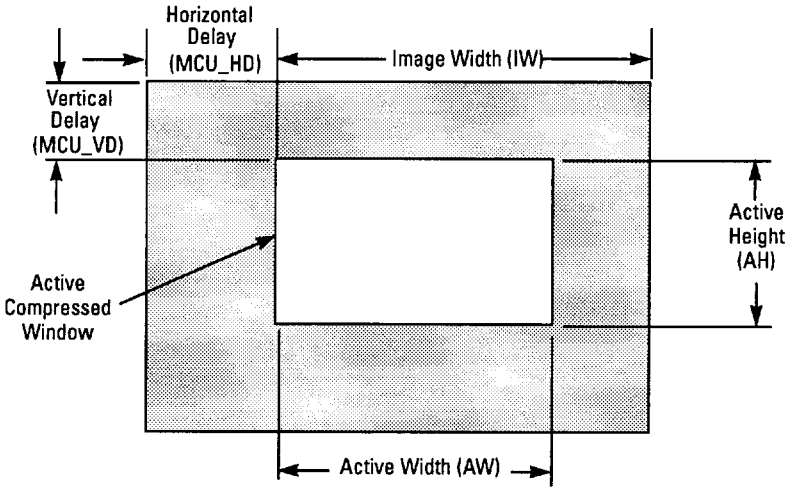


Compressed Image Active Window

A special feature of the L64702 allows an active window to be defined within a compressed file. The compressed file represents the entire image. However, a portion of the image can be defined as active so that when the L64702 operates in the decompression direction, it displays only the active portion. An active window display capability is useful if you wish to preview or scroll and display only a portion of a very large image that will not fit on the display screen. Figure 1.10 shows the active window in a compressed file.

Two additional parameters, *horizontal delay* (MCU_HD) and *vertical delay* (MCH_VD), are provided in addition to those shown in the Active Image area of Figure 1.9. These parameters tell the L64702 how many MCUs to skip in the compressed file before starting to decompress.

Figure 1.10 Compressed Image Active Window



Color Conversion

The L64702 is capable of flexible *color space conversion*. The RGB-to-YCrCb and YCrCb-to-RGB color conversions are defined in CCIR 601. Any set of 3 x 3 matrix coefficients may be downloaded into the L64702, providing an extremely flexible color space conversion system. The L64702 supports any linear transformation of a 3 x 1 vector by multiplying it by the downloaded 3 x 3 matrix.

As an example, the first set of equations below shows how the L64702 operates on RGB image data to transform RGB data to YCrCb data. The RGB-to-YCrCb color conversion is shown as a series of linear equations

and also as a matrix equation. The second set of equations shows how the same procedure is used for the YCbCr-to-RGB color conversion. RGB and Y values are in the range 16-235 inclusive; Cr and Cb values are in the range 16-240 inclusive. The actual coefficients that are used in the matrix range from 1.998 to -2.000.

With this flexible arrangement, the L64702 supports other conversions, such as RGB-to-Y and Y-to-RGB. By setting up the 3 x 3 conversion matrix as a unit matrix, the color space conversion may be altogether bypassed.

Equation 1.1
RGB to YCrCb Conversion

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.5 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Equation 1.2
YCrCb to RGB Conversion

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 1.402 & 0 \\ 1.0 & -0.7143 & -0.3437 \\ 1.0 & 0.0 & 1.772 \end{bmatrix} \times \begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix}$$

Subsampling

When the Color Space Converter is used in the encoding direction, the L64702 can *subsample*. Subsampling is a process in which data is sampled periodically, for example, every other pixel. See the subsection entitled "Minimum Coded Unit (MCU)" on page 1-14, along with Figure 1.7 for an explanation of 2:1:1 subsampling performed by the L64702.

The image subsampling operation reduces the number of pixels of one color component relative to another. The subsampling process is a way of accomplishing up-front video image compression besides the regular compression scheme involving quantization. The basic principle underlying subsampling is that there is some redundancy in color components. This redundancy can be eliminated by subsampling in such a way that the image quality is not impacted. The best known subsampling scheme is the 4:2:2 YCrCb format. This scheme samples the Y component in the color subcarrier four times for every two times that the Cr and Cb components are each sampled. Twice as many Y pixels are produced as either Cr or Cb pixels.

In a JPEG system, the Y, Cr, and Cb components that result from subsampling are interleaved and transferred as 8 x 8 blocks. The 4:2:2 format is translated to 2:1:1 JPEG subsampling, which means that there are two 8 x 8 blocks of Y followed by one 8 x 8 block of Cr and one 8 x 8 block of

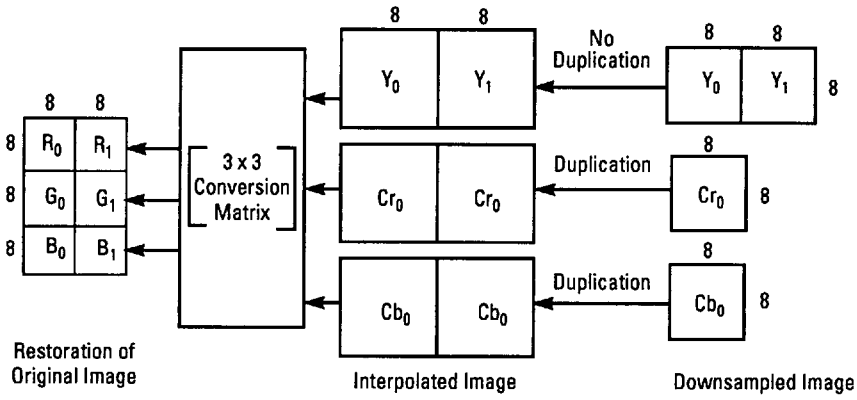
Cb. These four blocks are also called an MCU. To construct this particular MCU, an area of 16 pixels by 8 lines is needed.

Refer to Section 7.4, "Image Subsampling," for more information.

Image Reconstruction

When the Color Space Converter is used in the decoding direction to decode 2:1:1 coded images, the chrominance components (Cr and Cb) are replicated. Each block of 8 x 8 pixels for a given chrominance component is expanded into a 16 x 8 block. The additional 64 values, which were earlier eliminated by downsampling during encoding, are interpolated based on the current chrominance values. In fact, the additional 64 values are duplicates of the current chrominance values. Since none of the luminance pixels (Y) were eliminated by downsampling, there is no need to interpolate missing pixels for that part of the image. Figure 1.11 illustrates the process of image reconstruction.

Figure 1.11 Image Reconstruction from YCrCb to RGB



Raster-to-Block Conversion

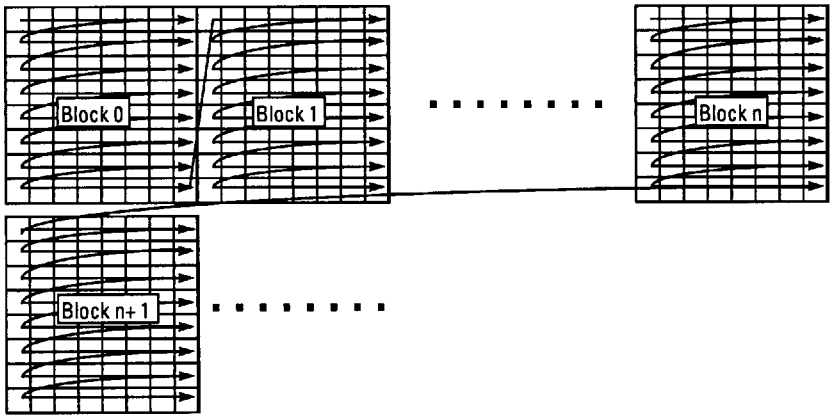
The L64702 converts the full-image raster data into data blocks of eight pixels by eight pixels, the format used by the DCT Processor and the JPEG Coder.

In the encoding direction, the L64702 reads the first eight pixels of the video memory in the horizontal direction, then reads the next eight pixels of the next display line, and so on, until an image area eight pixels by eight pixels is read. The eight pixels of each line are read in *raster order*; that is, from left to right. The resulting array is an 8 x 8 block, thus fulfilling the *raster-to-block* conversion requirement. The next raster-to-block conversion starts back on the first display line, on the ninth pixel from the

beginning. Thus the L64702 reads through the video memory and prepares a sequence of 8 x 8 blocks in the order described. Figure 1.12 illustrates raster-to-block conversion.

In the decoding direction, the L64702 reads 8 x 8 blocks and performs a *block-to-raster* conversion. The block-to-raster conversion takes pixels from each 8 x 8 block and writes an 8 x 8 area in video memory (eight pixels wide by eight lines high).

Figure 1.12 Raster-to-Block Conversion



DCT Functions

The DCT Processor performs the *forward* (FDCT) and *inverse discrete cosine transforms* (IDCT). The purpose of FDCT and IDCT is to transform image data between the time domain and the frequency domain.

In video compression, the FDCT function is implemented first, followed by quantization. In video decompression, inverse quantization occurs first, followed by the IDCT function. IDCT performance meets the CCITT requirements. The DCT/IDCT coefficients may be downloaded into the L64702. The equations for these transforms are shown below. The symbols used in the equations are defined following the equations.

Equation 1.3 Forward DCT

$$F_{v,u} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f_{y,x} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \quad \text{(FDCT)}$$

Equation 1.4 Inverse DCT

$$f_{y,x} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v F_{v,u} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \quad \text{(IDCT)}$$

where:

u = horizontal frequency index
v = vertical frequency index
x = horizontal position index
y = vertical position index

$$C_u, C_v = \frac{1}{\sqrt{2}} \text{ for } u, v = 0 \quad C_u, C_v = 1 \text{ otherwise}$$

The L64702 calculates the DCT or IDCT coefficients in two separate matrix operations as follows:

$$[\bar{Y}] = [A] \bullet [X]$$

$$[Y] = [A] \bullet [\bar{Y}]^T$$

where:

[A] is the 8 x 8 DCT or IDCT matrix.
[X] is the 8 x 8 input data matrix to the FDCT/IDCT L64702 block.
[$\bar{Y}$] is the result of the first operation.
[Y] is the final result.

Quantization Function

Quantization categorizes data into discrete values. Quantization is a lossy process, meaning that information is lost in the quantization process.

The forward and inverse *quantization functions* performed by the JPEG Coder are shown below. The data used in the quantization is stored within the L64702 in four *quantization tables*. The symbols used are defined following the equations.

Equation 1.5 Forward Quantization

$$Q(x(i), i) = round\left(\frac{x(i)}{q(i)}\right)$$

Equation 1.6 Inverse Quantization

$$x'(i) = Q(x(i), i) \cdot q(i)$$

where:

x(i) = DCT coefficient created by the DCT Processor.

x'(i) = reconstructed DCT coefficient created by the quantizer in the L64702. Since quantization is a lossy process, x'(i) is not necessarily equal to x(i).

Q(x(i), i) = quantized DCT coefficient. Q(x(i), i) is the output of the quantizer and the input to the coding modules when the L64702 is in DCT encoding configuration. Q(x(i), i) is the output of the decoding modules and the input to the inverse quantizer when the L64702 is in DCT decoding configuration.

q(i) = quantization stepsize from the quantization table for the component being quantized.

round(x) = rounding function. round(x) rounds to the nearest integer. When x is halfway between two integers, round(x) rounds up for positive and down for negative values of x.

i = index of the DCT or reconstructed DCT coefficient. i is in the range from 0 to 63 inclusive.

Figure 1.13 shows the forward and inverse quantization functions. Empty circles indicate that the endpoint of the line segment is not included.

Figure 1.13 Quantization Functions

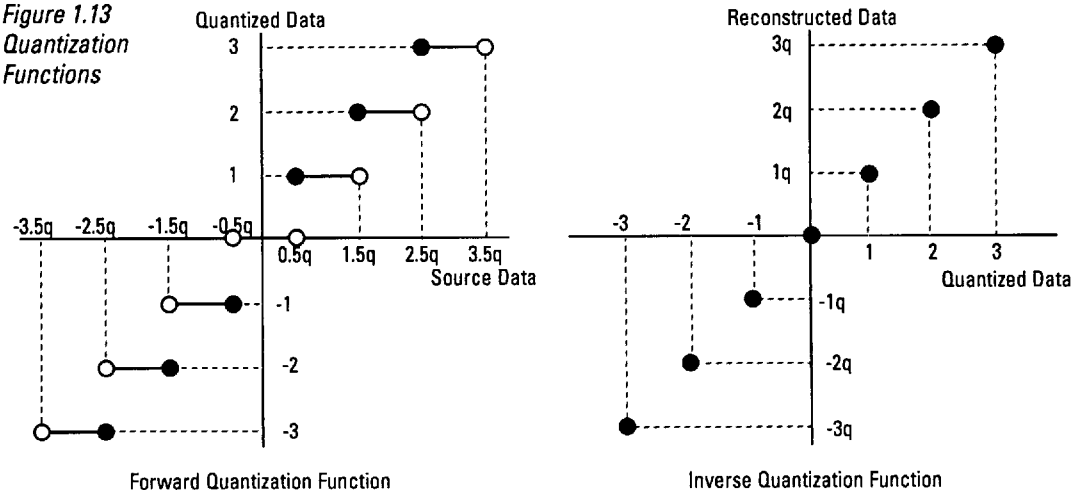


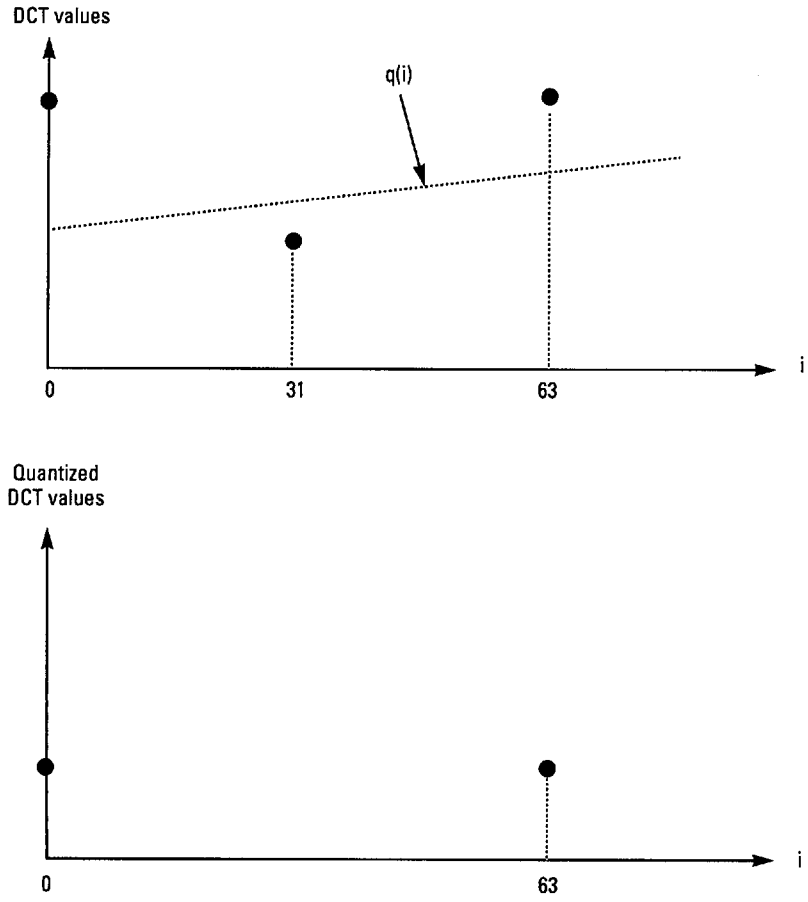
Table 1.1 shows examples of a forward and inverse quantization of several input values. The table assumes that source data is first quantized, then reconstructed using the same quantization stepsize q. Note that in general, the reconstructed value differs from the source value.

Table 1.1 Quantization Examples

Source Data	*Quantized Data*	*Reconstructed Data*
1.2q	1	1q
-0.4q	0	0
3.0q	3	3q

Figure 1.14 shows an example of forward quantization on an input signal with three non-zero values. The input function and the quantization function are shown on the upper graph, and the resulting quantized function is shown on the lower graph.

Figure 1.14 Quantization Example



Zigzag Ordering

The DCT processor encodes DCT coefficients and outputs the data in either raster order or, more commonly, *zigzag order*. Zigzag ordering orders the coefficients in increasing order of frequencies, which creates longer runs of zero values and increases coding efficiency in the JPEG Coder.

Figure 1.15 shows zigzag ordering. The figure shows the rearrangement of the block from raster order. Coefficient 0 is the DC coefficient and coefficients 1 through 63 are AC coefficients, with the lower frequencies having the lower-numbered coefficients.

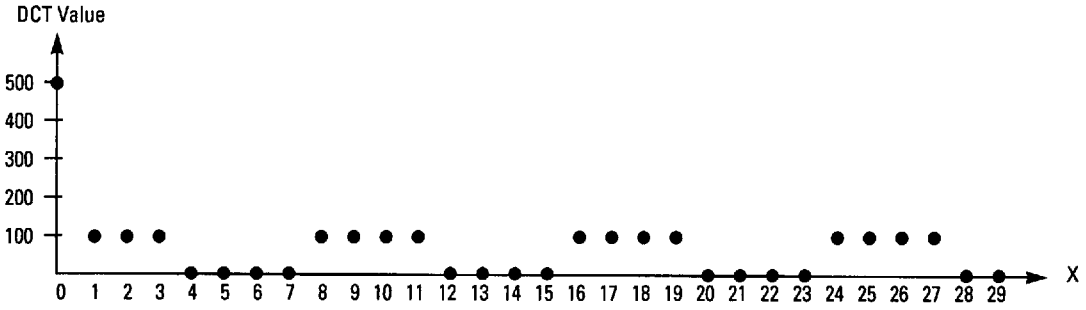
Figure 1.15
Zigzag Ordering

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

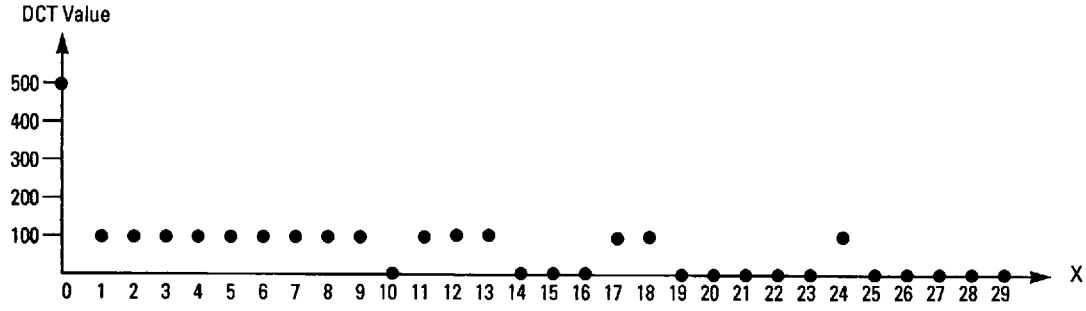
Figure 1.16 shows an example of the benefits of zigzag ordering. The data block shows a typical distribution of DCT coefficients. The graphs below the data block show the runs for raster and zigzag ordering. Notice that zigzag ordering concentrates the non-zero coefficients and in general produces longer zero-length runs.

Figure 1.16
Raster and Zigzag
Encoding Example

0	500	100	100	100	0	0	0	0
8	100	100	100	100	0	0	0	0
16	100	100	100	100	0	0	0	0
24	100	100	100	100	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0



Raster Order



Zigzag Order

DPCM Coding

Each DC coefficient is coded using the DC coefficient value for the previous block of the same component as a predictor. This coding scheme is called *differential pulse-code modulation (DPCM)*. The difference between the current DC coefficient and the previous value is coded. The previous value of the DC coefficient is stored in one of four internal DC predictor registers (one for each component). When the L64702 is reset, the internal DC predictor values are set to zero. The DPCM coding in the L64702 conforms to the proposed JPEG DIS 10918 standard for encoding DC coefficients in sequential DCT-based processes.

The equation for the DPCM is shown below. $\Delta(n)$ is the predictor that is coded at the start point of the coding operation $DC_{-1} = 0$ so $\Delta(0) = DC_0$.

Equation 1.7
DPCM Coding

$$\Delta(n) = DC_n - DC_{n-1}$$

Run-Length Coding

The AC coefficients are first coded by a *run-length coding* algorithm. In run-length coding, each AC coefficient is coded as an event consisting of a run of zero coefficients and a non-zero value. (DC coefficients always have a zero run length, since the DC coefficient is the first coefficient in the block.) The run-length coding in the JPEG Coder conforms to the proposed JPEG standard[1] for encoding AC coefficients in sequential DCT-based processes.

Variable-Length Coding (Huffman Coding)

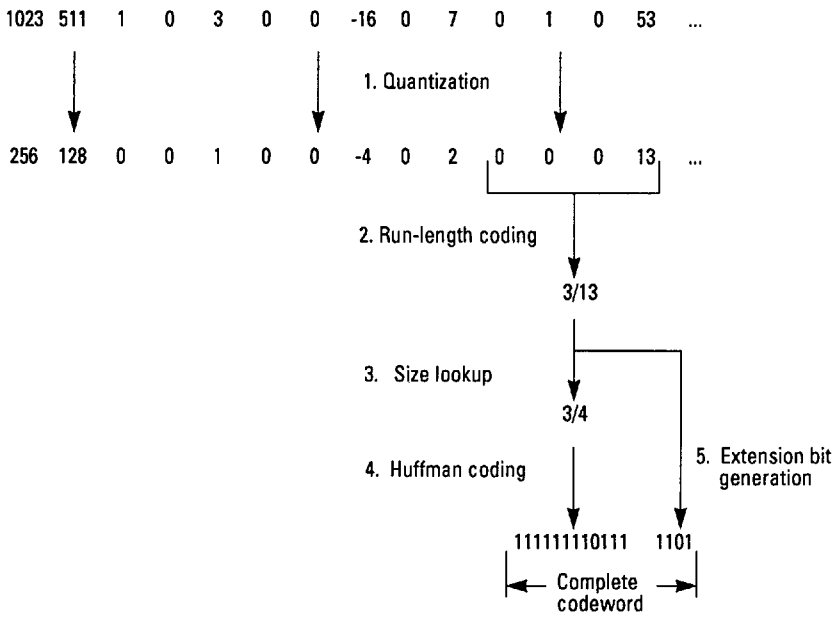
The code preprocessor and code tables generate *variable-length codewords* or *Huffman codes*. The codewords are chosen to maximize compression for the image data type to be processed. The most commonly occurring events are assigned the shortest codewords. See the proposed JPEG standard for more information about creating code tables.

1. The International Standard Draft is expected to be approved as an international standard shortly.

Encoding Example

The following example shows the JPEG Coder's operation in DCT-based encoder configuration. Figure 1.17 and the explanations following the figure show how a portion of an input data stream is encoded.

Figure 1.17 Encoding Example



Quantization

Quantization reduces the number of data values and increases the number of zeros. In this example, all values in the quantization table are assumed to be four. The quantized values are determined by the forward quantization function described earlier in the subsection entitled "Quantization Function." For example, the DCT coefficient 53 in the input stream is quantized as shown below:

Equation 1.8 DCT Coefficient Calculation

$$Q(x(i), i) = round\left(\frac{53}{4}\right) = round(13.25) = 13$$

Run-Length Coding

The run-length coder determines the number of zeros between successive non-zero coefficients. In the indicated portion of the data stream, three zeros precede the non-zero coefficient 13.

Size Lookup

The code preprocessor looks up the size corresponding to the coefficient in the size table shown in Table 1.2. In the example, the coefficient 13 corresponds to the size 4.

Table 1.2 AC Coefficient Size Table

Size	*AC Coefficient*
1	-1,1
2	-3,-2,2,3
3	-7...-4,4...7
4	-15...-8,8...15
5	-31...-16,16...31
6	-63...-32,32...63
7	-127...-64,64...127
8	-255...-128,128...255
9	-511...-256,256...511
10	-1023...-512,512...1023

Huffman Coding

The variable-length coder uses the run and size values to look up the Huffman code in the code table. The code tables are set up by the user as described in Chapter 3, "Internal Registers and Data Tables." Table 1.3 shows a portion of the code table used in this example. The entry used in the example is shown in bold.

Table 1.3 Sample AC Code Table

Run/Size	*Huffman Code*
.	.
.	.
3/2	11111000
3/3	1111111000
3/4	**111111110111**
3/5	1111111110010001
3/6	1111111110010010
.	.
.	.

Extension Bit Generation

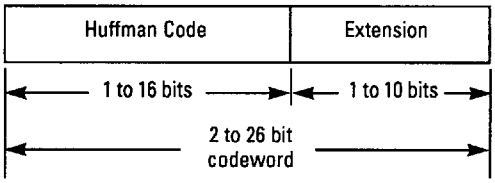
The code preprocessor determines the extension bits using the algorithm given in the following paragraphs and shown in Figure 1.18.

C is the coefficient in two's complement notation, and S is the size. If C is greater than zero, the extension bits are the S low-order bits of C. If C is less than zero, the extension bits are the S low-order bits of $(C - 1)$.

In this example, C is equal to 13 (binary code $000.0000.1101_2$) and S is equal to four. The extension bits are the four low-order bits 1101_2.

Figure 1.18 shows the general structure of the codewords generated by the L64702.

Figure 1.18
Codeword Structure



1.7 Applications

This section lists the typical applications for the L64702:

- Multimedia
- Image transmission systems, modems, color faxes, and LANs
- Image storage and retrieval systems
- Display cards
- Color and gray-scale printers, copiers, and scanners
- Color publishing and pre-press systems
- Digital cameras

Chapter 2
Interface Description

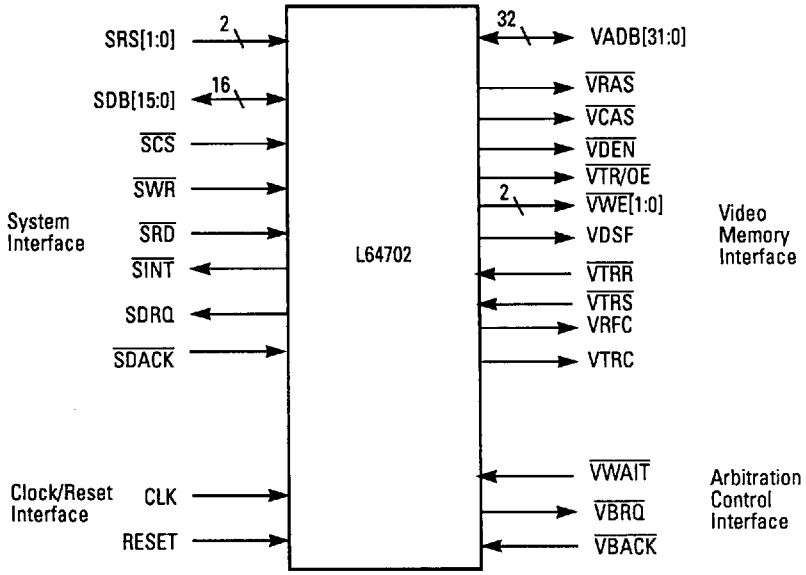
This chapter provides detailed information on the L64702 signal descriptions. These descriptions are useful for hardware designers who are interfacing the L64702 with other devices.

This chapter contains the following sections:

- Section 2.1, System Interface Signals
- Section 2.2, Clock/Reset Interface Signals
- Section 2.3, Video Memory Interface Signals
- Section 2.4, Video Bus Arbitration Signals

Figure 2.1 shows the logic symbol for the L64702.

Figure 2.1
L64702 Logic Symbol



2.1 System Interface Signals

The system interface signals allow communication between the L64702 and a system processor. To the system processor, the L64702 appears as a memory-mapped device containing a block of memory resources and control and status registers. The L64702 occupies only four address locations in the system processor memory space—the internal registers and tables are made accessible through an internal address pointer technique. Each signal in the system interface is described below.

$\overline{SCS}$ **System Chip Select** **Input**

An external CPU asserts $\overline{SCS}$ LOW to access L64702 internal memory resources. The L64702 selects the specific resource to be accessed by means of the SRS[1:0] signals. A CPU must never assert $\overline{SCS}$ at the same time that $\overline{SDACK}$ is asserted.

$\overline{SDACK}$ **System DMA Acknowledge** **Input**

The external DMA controller asserts $\overline{SDACK}$ LOW to notify the L64702 that the L64702's request for DMA service has been acknowledged, and that a DMA read or write operation will be performed. The L64702 asserts SDRQ to request the DMA operation. $\overline{SDACK}$ acts as a chip select signal during DMA transfers, in a similar manner to the operation of $\overline{SCS}$ during CPU transfers.

$\overline{SDACK}$ and $\overline{SCS}$ must never be active at the same time.

SDB[15:0] **System Data Bus** **Bidirectional**

SDB[15:0] transfer data between a selected register or memory location in the L64702 and external memory. SDB0 is the LSB and SDB15 is the MSB. SDB[15:0] acts as an input data bus during a write cycle, and as an output data bus during a read cycle. The L64702 3-states the signals of SDB[0:15] if there is no access to the device ($\overline{SCS}$ and $\overline{SDACK}$ signals are not active).

SDRQ **System DMA Request** **Output**

The L64702 asserts SDRQ HIGH to request DMA service from an external DMA Controller. The L64702 can only assert SDREQ provided the DMA Enable (DE) bit in the SMR Register is set.

The START bit in the SMR Register affects DMA operation. The status of SDRQ depends on whether the L64702 is in an initialization state or in a processing state.

During initialization, the INIT bit in the SMR Register must be set to one. The L64702 asserts SDRQ immediately after the DE bit is set regardless of the START bit, and remains active until DE is reset.

This configuration, by definition, allows access only to registers and tables from Group 2. Refer to Chapter 3, "Internal Registers and Data Tables," for more information on Group 2 registers and tables.

The L64702 is in the processing state and performs image compression and decompression when DE and START are both set to one and INIT is reset to zero. In this case, DMA operation services the FIFO only. SDRQ becomes active or inactive depending on the state of the FIFO, which is dependent on its programmed threshold or burst level, its full or empty condition, and detection of markers in the code stream. DMA becomes active or inactive for image compression or decompression based on events that affect the FIFO, as shown in the table below. Refer to Section 3.5, "Group 3 FIFO," for more information.

Image Compression		*Image Decompression*	
DMA Active	*DMA Inactive*	*DMA Active*	*DMA Inactive*
The number of words in the FIFO is greater than or equal to the burst threshold.	The burst transfer of words out of the FIFO has finished.	The number of words in the FIFO is less than the burst threshold.	The burst transfer of words into the FIFO has finished or an RI, EOI, or IMC marker has been detected.

$\overline{\text{SINT}}$ **System Interrupt Request** **Output**

The L64702 asserts $\overline{\text{SINT}}$ LOW when an internal, unmasked interrupt flag is set. $\overline{\text{SINT}}$ remains asserted as long as the internal interrupt condition persists and the interrupt flag is not masked.

$\overline{\text{SRD}}$ **System Read Internal Strobe** **Input**

An external CPU or DMA controller asserts $\overline{\text{SRD}}$ LOW to initiate a read operation. $\overline{\text{SCS}}$ must be active for CPU read operations and $\overline{\text{SDACK}}$ must be active for DMA read operations. When $\overline{\text{SRD}}$ becomes active, the L64702 transfers the contents of the selected internal memory or register resource to the system data bus, SDB[15:0]. The L64702 automatically increments the Address Pointer Register by one on the LOW-to-HIGH transition of $\overline{\text{SRD}}$ when a CPU reads from Group 2 (SRS[1:0] = 10_2).

SRS[1:0] **System Register Select** **Input**

SRS[1:0] determine which of the four L64702 internal memory resources is selected during a CPU transfer. When the L64702 is in DMA transfer mode, the L64702 ignores these signals. The table below shows the meaning of SRS[1:0].

SRS1	*SRS0*	*Register*
0	0	Group 0: Address Pointer Register
0	1	Group 1: Status Register/Mode Register
1	0	Group 2: Tables and Control Registers
1	1	Group 3: FIFO

$\overline{\text{SWR}}$ **System Write Strobe** **Input**

An external CPU asserts $\overline{\text{SWR}}$ LOW to initiate a write operation. $\overline{\text{SCS}}$ must be active for CPU write operations or $\overline{\text{SDACK}}$ must be active for DMA write operations. On the LOW-to-HIGH transition of $\overline{\text{SWR}}$, the CPU writes the contents of the system data bus, SDB[15:0], to the selected internal memory or register resources. When writing to Group 2 (SRS[1:0] = 10_2), the L64702 automatically increments the Address Pointer Register by one on the LOW-to-HIGH transition of $\overline{\text{SWR}}$.

2.2 Clock/Reset Interface Signals

These signals are the clock and reset signals for the L64702.

CLK **System Clock** **Input**

CLK is the input clock signal that controls all operations within the L64702. All internal operations occur on the LOW-to-HIGH transition of CLK.

RESET **Reset Control** **Input**

An external CPU must assert RESET HIGH for a minimum of two clock cycles to reset the L64702. During a reset operation, the L64702 terminates all internal processing and enters the idle state. The L64702 synchronizes RESET internally with the LOW-to-HIGH transition of CLK.

After a reset operation, all internal control and status registers return to their default values, but data in the internal tables and most control registers are not affected.

2.3 Video Memory Interface Signals

The L64702 uses the video bus interface signals to communicate with external video memory (DRAMs and VRAMs) and external I/O devices. The L64702 is a master device on the video port—it controls the timing, the address, and the data flow. The Video Memory Interface is designed to share the bus with other bus masters by means of special purpose arbitration signals. Each signal in the video memory interface is described as follows.

VADB[31:0] Video Address and Data Bus **Bidirectional**

VADB[31:0] form a bidirectional multiplexed address and data bus. At the beginning of a video memory cycle, 24 bits of address on VADB[25:2] and status bits on VADB[29:26] are valid with the falling edge of the $\overline{\text{VRAS}}$ signal and stay valid until the falling edge of $\overline{\text{VDEN}}$ signal. On the falling edge of $\overline{\text{VDEN}}$ signal, the L64702 either drives the entire 32-bits with valid data during a video memory write cycle or accepts valid data from the video memory during a read cycle. The L64702 drives the VADB[31:0] signals when the L64702 has control over the video bus, and $\overline{\text{VBRQ}}$ and $\overline{\text{VBACK}}$ have been asserted. The L64702 3-states VADB[31:0] when the L64702 has no control over the video bus, indicated by the state of $\overline{\text{VBACK}}$.

$\overline{\text{VCAS}}$ Video Memory Column Address Strobe **3-State Output**

The L64702 asserts $\overline{\text{VCAS}}$ LOW to latch the column address into VRAM/DRAM devices. $\overline{\text{VCAS}}$ can be connected to the $\overline{\text{CAS}}$ input of these devices. The $\overline{\text{VCAS}}$ HIGH-to-LOW transition occurs one CLK cycle after VADB[31:0] switches from address to data. The column address should be latched into a temporary buffer (74LS373) prior to this transition.

The L64702 drives $\overline{\text{VCAS}}$ when it has control over the video bus. Otherwise, the L64702 3-states $\overline{\text{VCAS}}$.

$\overline{\text{VDEN}}$ Video Bus Data Enable **3-State Output**

The L64702 asserts $\overline{\text{VDEN}}$ LOW whenever the L64702 has placed valid data on VADB[31:0]. $\overline{\text{VDEN}}$ can drive the active-LOW output-enable signal of a bidirectional bus transceiver such as the 74ALS245. These transceivers are used as buffers between the L64702 video data bus VADB[31:0] and large memory arrays.

The L64702 drives $\overline{\text{VDEN}}$ when it has control over the video bus. Otherwise, it is 3-stated.

VDSF Video Special Function **3-State Output**

The VDSF signal from the L64702 allows utilization of special features of external VRAM devices. The level of the VDSF output (HIGH or LOW) at the falling edge of VRAS determines which of the VRAM special features are used during the current memory cycle. Refer to any VRAM data book for a full explanation of the special features available.

The L64702 drives VDSF when it has control over the video bus. Otherwise, it is 3-stated.

$\overline{\text{VRAS}}$ **Video Memory Row Address Strobe** **3-State Output**

The L64702 asserts $\overline{\text{VRAS}}$ LOW when the address lines on VADB are valid. $\overline{\text{VRAS}}$ can be connected to the $\overline{\text{RAS}}$ input pin of VRAM/DRAM devices. $\overline{\text{VRAS}}$ is intended for latching row address and other control information into these devices during the HIGH-to-LOW transition. Because address and data are multiplexed on the VADB bus, $\overline{\text{VRAS}}$ can latch the column address and status in 74LS373 devices in order to maintain address and status validity through the entire memory cycle.

The L64702 drives $\overline{\text{VRAS}}$ when it has control over the video bus. Otherwise, the L64702 3-states $\overline{\text{VRAS}}$.

VRFC **Video Refresh Cycle** **Output**

When HIGH, VRFC indicates that a video memory $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh cycle is in progress. VRFC may be used as an indicator that such a refresh cycle is in progress—it stays LOW at all other times. VRFC in conjunction with VTRC indicate activity on the video memory bus.

$\overline{\text{VTR/OE}}$ **Video Data Transfer and Output Enable** **3-State Output**

The L64702 asserts $\overline{\text{VTR/OE}}$ LOW to enable the output drivers of the video memory devices. $\overline{\text{VTR/OE}}$ is connected to the $\overline{\text{TR/OE}}$ inputs of the VRAMs and DRAMs in video memory. In SRAM devices it is connected to the $\overline{\text{OE}}$ input of these memory devices.

During a read cycle, $\overline{\text{VTR/OE}}$ is HIGH when $\overline{\text{VRAS}}$ falls. The L64702 later asserts $\overline{\text{VTR/OE}}$ LOW to enable data from the video memory to be placed on VADB[31:0]. Data is latched into the L64702 on the LOW-to-HIGH transition of $\overline{\text{VTR/OE}}$. $\overline{\text{VTR/OE}}$ can also be used as the active-LOW read signal for SRAMs and other peripheral devices connected to the L64702 on the video memory interface.

$\overline{\text{VTR/OE}}$ is used for the Normal Read Transfer and Split Read Transfer operations. In these cases, $\overline{\text{VTR/OE}}$ is asserted LOW before and after the falling edge of $\overline{\text{VRAS}}$.

$\overline{\text{VTR/OE}}$ is driven when the L64702 has control over the video data bus and is 3-stated otherwise.

VTRC **Video Transfer Cycle** **Output**

When HIGH, VTRC indicates that a video memory Read Transfer cycle or a video memory Write Transfer cycle is in progress. The VTRC signal is a video controller status flag. The L64702 drives

VTRC regardless of whether the L64702 is controlling the video bus or not.

$\overline{\text{VTRR}}$ Video Transfer Request **Input**

An external video timing generator asserts $\overline{\text{VTRR}}$ LOW to initiate a video line transfer cycle. Depending on the C_D bit in the SCR Register, the $\overline{\text{VTRR}}$ signal initiates either a video memory Read Transfer or Write Transfer cycle. These cycles involve transfers of complete rows of data between the DRAM core and serial shift register of a video memory device. The L64702 supports internally the addressing scheme for video memory Read and Write Transfer cycles by means of two internal addressing registers, Transfer Start Address (TSA) and Transfer Increment (TI). For more details on these registers, refer to Chapter 3, "Internal Registers and Data Tables."

If the L64702 is in the Image Capture mode (C_D = 1), the HIGH to LOW transition of $\overline{\text{VTRR}}$ indicates that a new line is ready to be transferred from the video shift register to the memory. If the L64702 is in the Image Display mode (C_D = 0), the HIGH to LOW transition of $\overline{\text{VTRR}}$ indicates that the video shift register is ready to accept new line from the video memory.

$\overline{\text{VTRS}}$ Video Transfer Reset **Input**

An external device asserts $\overline{\text{VTRS}}$ LOW to initialize the Transfer Start Address pointer to the initial address. In terms of video timing signals, $\overline{\text{VTRS}}$ indicates the start of a new frame or new field. In most cases, $\overline{\text{VTRS}}$ is connected to the external $\overline{\text{VSYNC}}$ signal.

$\overline{\text{VWE}}$[1:0] Video Memory Write Enable **3-State Output**

The L64702 asserts $\overline{\text{VWE}}$[1:0] LOW during video memory write operations. The $\overline{\text{VWE}}$[1:0] signals are connected to the $\overline{\text{WE}}$ inputs of DRAM/VRAM or SRAM devices. A HIGH-to-LOW transition on $\overline{\text{VWE}}$ indicates the beginning of a memory write cycle.

In DRAM/VRAM devices, the HIGH-to-LOW transition of $\overline{\text{VCAS}}$ while $\overline{\text{VWE}}$[1:0] are LOW latches the data into the device.

In SRAM devices, the LOW-to-HIGH transition of $\overline{\text{VWE}}$[1:0] latches the data into the device.

$\overline{\text{VWE0}}$ is the control signal for the lower 24 data bits (VADB[23:0]), while $\overline{\text{VWE1}}$ controls the most-significant eight bits (VADB[31:24]). When the L64702 is programmed to operate in the RGBX image data format, $\overline{\text{VWE1}}$ is not active and remains HIGH. In all other image modes, the two pins are identical.

The L64702 drives $\overline{\text{VWE}}$[1:0] when it has control over the video bus. Otherwise, it is 3-stated.

2.4 Video Bus Arbitration Signals

The L64702 uses three signals, $\overline{\text{VBRQ}}$, $\overline{\text{VBACK}}$, and $\overline{\text{VWAIT}}$ to generate a handshake mechanism for sharing the video memory bus between an external master processor and the L64702. These signals are described below.

$\overline{\text{VBACK}}$ Video Bus Acknowledge **Input**

The external video bus master asserts $\overline{\text{VBACK}}$ LOW to grant the L64702 control of the video memory bus. The other video bus master can preempt the bus (force control away from the L64702) by deasserting $\overline{\text{VBACK}}$ during an L64702 video bus operation. In this case, the L64702 acknowledges the preemption as described under $\overline{\text{VBRQ}}$. $\overline{\text{VBACK}}$ is synchronous to the L64702 clock.

$\overline{\text{VBRQ}}$ Video Bus Request **Output**

The L64702 asserts $\overline{\text{VBRQ}}$ LOW to request control of the video memory bus from an external video bus master. The other master device asserts $\overline{\text{VBACK}}$ to grant control of the video bus to the L64702. Upon receiving $\overline{\text{VBACK}}$, the L64702 starts with the video memory transfer. $\overline{\text{VBRQ}}$ remains asserted while the L64702 either has data to transfer or can accept data to or from the on-chip MCU buffers. The L64702 can alternately assert and deassert $\overline{\text{VBRQ}}$ as a function of the full or empty status of the MCU buffer during compression or decompression.

When $\overline{\text{VBRQ}}$ is deasserted HIGH, $\overline{\text{VBACK}}$ must be deasserted (HIGH) before $\overline{\text{VBRQ}}$ is reasserted, indicating that the video bus release has been granted by the other master. The other video bus master can preempt the L64702 from the video bus by deasserting $\overline{\text{VBACK}}$. When this happens, the L64702 completes the current cycle, deasserts $\overline{\text{VBRQ}}$ for eight CLK cycles, and asserts $\overline{\text{VBRQ}}$ again if it still requires access to the video bus.

$\overline{\text{VBRQ}}$ is asserted for memory refresh, video memory transfer cycles, and normal read or write cycles. If the TRTH bit in the SMR Register is set, the L64702 asserts $\overline{\text{VBRQ}}$ until TRTH is reset.

$\overline{\text{VWAIT}}$ Video Bus Wait **Input**

$\overline{\text{VWAIT}}$ asserted indicates that the external video memory is not yet ready to complete the memory cycle. $\overline{\text{VWAIT}}$ also affects the video transfer and refresh cycles. $\overline{\text{VWAIT}}$ is synchronous to the L64702 clock.

Chapter 3
Internal Registers and Data Tables

This chapter discusses the L64702 internal registers and data tables. It also provides a description of the internal memory mapping and the access to these registers and tables from the system interface. This chapter is intended primarily for system programmers who are developing software drivers.

This chapter contains five sections:

- Section 3.1, L64702 Register Overview
- Section 3.2, Group 0 Address Pointer Register (APR)
- Section 3.3, Group 1 Registers
- Section 3.4, Group 2 Registers and Tables
- Section 3.5, Group 3 FIFO

3.1 L64702 Register Overview

The L64702 registers and memory resources are divided into four groups: Group 0, Group 1, Group 2, and Group 3. Group 0 contains the APR Register, which addresses the registers and tables in Group 2. Group 0 is also used for transfer-through operations, which allow a CPU direct access to video memory. Group 1 contains the status and mode registers. Group 2 contains registers and tables that configure the compression and decompression operation of the L64702. Group 3 contains the FIFO.

Most registers and table entries are 16 bits wide—some registers are 24 bits or 32 bits wide. All accesses are done in 16-bit widths. Initialize the APR Register before accessing the control registers or tables in Group 2.

Internally, the L64702 has a 16-bit architecture. During data transfers from the CPU port, the CPU should perform memory accesses on 16-bit boundaries. All registers and tables are memory-mapped to the system with 16-bit resolution. When performing 16-bit transfers, the CPU requires only one memory cycle per register access. During write operations, the L64702

disregards bits that it does not use. During read operations, the L64702 sets any unused bits to zero.

The L64702 contains some registers that are longer than 16 bits. These registers are divided into two 16-bit sections: the least-significant word (LSW) and the most-significant word (MSW). Each 32-bit section is assigned a specific address, and therefore requires two 16-bit memory cycles during programming.

The L64702 uses an auto-increment address pointer to simplify the initialization procedure and to reduce the number of memory locations occupied by the L64702 in CPU memory. The address pointer is active only when accessing Group 2 data tables and control registers (SRS[1:0] = 10_2). After each complete access to this group, the L64702 automatically increments the address pointer by one on the rising edge of either $\overline{\text{SRD}}$ or $\overline{\text{SWR}}$. After the address pointer increments, it points to the next table or register entry. During L64702 initialization (INIT = 1), the external system (CPU or DMA controller) can access only Groups 0, 1, and 2. If INIT is set HIGH, the L64702 is in the initialization phase, the START bit is ignored, and no compression or decompression processing can take place. When INIT = 0, the FIFO becomes accessible to the external system, and the Group 2 resources of the L64702 become inaccessible to the external system.

Table 3.1 shows how the START and INIT bits are used in accessing the Group 0, 1, 2, and 3 resources of the L64702. Be aware that merely setting the bits to the indicated states does not perform the corresponding operation, but is required as a precondition to performing the operation.

Table 3.1
Accessing the L64702 Internal Groups

START	*DE*	*INIT*	*TRTH*	*CPU Access Groups*	*DMA Access Groups*	*Note*
0	0	0	1	0, 1, 3	–	CPU Transfer-Through Mode, Group 2 Protected
0	1	0	1	0, 1, 3	3	DMA Transfer-Through Mode
1	0	0	0	0, 1, 3	–	Processing Operation in CPU Transfer Mode
1	1	0	0	0, 1, 3	3	Processing Operation in DMA Transfer Mode
x[1]	0	1	0	0, 1, 2	–	Initialize mode, CPU Transfer Mode
x	1	1	0	0, 1, 2	2	Initialize Mode, DMA Transfer Mode

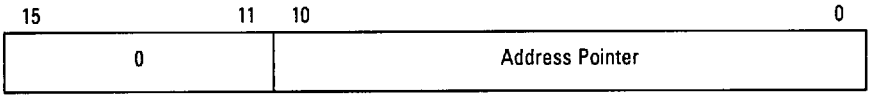
1. x = don't care

After the START bit has been set, any attempt to write to Group 2 (registers and tables) terminates L64702 processing. You must reset the device to restart operation. You can read Group 2 registers and tables during operation with no risk of data corruption only if you stop operation and go temporarily into the initialization mode.

3.2 Group 0 Address Pointer Register (APR)

The Address Pointer Register (APR), shown in Figure 3.1, is an 11-bit register, which points to the internal registers and tables from Group 2. It is accessed when SRS[1:0] = 00_2. Before accessing a register or table location from Group 2, you must pre-initialize the APR with the address of the first desired register or table entry according to the internal memory mapping (see Section 3.4, "Group 2 Registers and Tables"). The APR automatically increments by one on the LOW-to-HIGH transition of $\overline{SRD}$ or $\overline{SWR}$ when accessing the Group 2 registers (SRS[1:0] = 10_2).

Figure 3.1 Address Pointer Register



3.3 Group 1 Registers

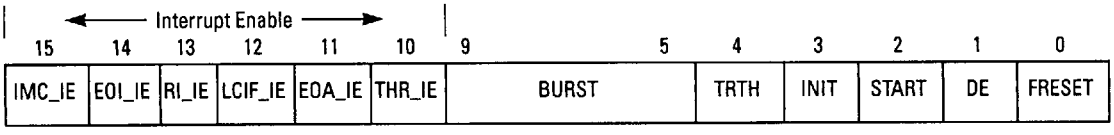
Group 1 contains two registers: the System Mode Register (SMR) and the System Status Register (STS). A CPU may access these registers by setting SRS[1:0] = 01_2. An external system can access these registers without any restrictions during processing without interrupting the L64702 internal processing unit. The following two subsections describe the two registers.

System Mode Register (SMR)

The SMR Register is a write-only register that allows external system control of the L64702. The SMR Register contains bits that start or stop the device operation, reset the device, enable or disable DMA service, set the FIFO burst level, and enable interrupts.

Figure 3.2 shows the bit organization of the SMR Register. Descriptions of the fields follow the figure. The L64702 clears all bits in the SMR Register to zero after a frame, software, or hardware reset.

Figure 3.2 System Mode Register



FRESET **Frame Reset** **0**

When the CPU sets the FRESET bit to one, the L64702 partially resets itself and generates a FRAME RESET pulse. The CPU does not need to set the bit back to a zero to complete the frame reset. The L64702 issues a single FRAME RESET pulse each time the CPU writes a one to this bit. When the FRESET bit is set, the L64702 resets the JPEG processing unit and video addressing state machine to their initial states. Video memory refresh cycles and the video transfer mode of operation set up by the configuration of the $\overline{\text{VTRR}}$, $\overline{\text{VTRS}}$, and C_D bits are not affected by setting the FRESET bit. A frame reset does not affect the contents of Group 2 tables and registers.

FRESET	*Definition*
0	No Frame Reset
1	The L64702 Issues a Frame Reset

DE **DMA Transfer Enable** **1**

When the CPU sets the DE bit HIGH, the L64702 is programmed to request external DMA controller service. DMA service request can be used for either loading configuration data, transfer-through data to the video memory or, in conjunction with the START bit, for transferring code data to or from the L64702 during compression or decompression.

DE	*Definition*
0	Disable DMA Transfer Request
1	Enable DMA Transfer Request

START **Start Operation Control Bit** **2**

When the CPU sets the START bit HIGH, the L64702 enters the operation mode and begins image compression or decompression. Initially, the CPU must clear the INIT bit LOW to allow the start of operation. During operation, an external CPU can arbitrarily reset the START bit, which pauses operation of the internal JPEG processing unit. There is an internal pipeline delay from the deassertion of START until the chip enters the pause state, which is indicated by the INIT flag set to a one in the STS Register. An external CPU can evaluate internal memory and registers by first checking the INIT flag in the STS Register. If the flag is set, setting the INIT bit in the SMR Register allows you to access the Group 2 tables and registers.

Setting the START bit after clearing the INIT bit (if set) resumes the operation from the point at which the L64702 entered the pause state. START remains set even after reaching the End Of Image (EOI)

condition for both compression and decompression. To restart with a new image, a frame reset must be performed and then the START bit can be set again.

START	*Definition*
0	Stop L64702 Operation
1	Start L64702 Operation

INIT **Initialize Mode** **3**

When the CPU sets the INIT bit HIGH the L64702 enters the Initialize mode. In this mode, the external system rather than the internal JPEG unit has access to the Group 2 tables and registers. If the CPU sets INIT during JPEG processing, the processing operation may be disrupted. To avoid processing disruption, the INITFLG flag in the STS Register indicates when the L64702 is ready to be initialized. The INITFLG flag in the STS Register should be set before you set the INIT bit in the SMR Register.

INIT	*Definition*
0	Group 2 Elements Cannot be Accessed
1	Group 2 Elements Can be Accessed

TRTH **Transfer Through** **4**

When the CPU sets the TRTH bit HIGH, the system L64702 video memory controller can be used by the system to access video memory through the L64702, and transfer data between the system bus and the video bus by means of read or write transfer cycles. While this operation is in progress, the video bus is still used for video memory refresh cycles and the video memory transfer operations set up by $\overline{\text{VTRR}}$ and VTRC. The JPEG processing unit of the L64702 is completely bypassed during the transfer-through operation.

If the transfer-through operation is initiated while JPEG processing is underway, the bit stream will be corrupted because the JPEG process and transfer-through both used the FIFO.

The TRTH and INIT bits should not both be set HIGH at the same time.

TRTH	*Definition*
0	Normal Read/Write Operation
1	Transfer Through Operation

BURST **Burst Level** **[9:5]**

These five bits specify the FIFO burst level, or the number of words to be transferred in a burst transfer interval. The bits are meaningful only when the L64702 is in the image compression, decompression, or the transfer-through mode. The specified burst level affects the handshake mechanism of compressed data transfer between the system memory and the FIFO. In general, it affects the functioning of the System DMA Request (SDRQ) pin, used in DMA transfers, and the functioning of the THR interrupt flag, used in the CPU transfer mode.

BURST	*Definition*
00001	Burst Level 1
00010	Burst Level 2
.	.
.	.
11111	Burst Level 31
00000	Burst Level 32

THR_IE **FIFO Burst Level Interrupt Enable** **10**

The CPU may set THR_IE to enable an interrupt when the FIFO burst level is reached. During compression (ENC = 1), if the number of words in the FIFO is greater than or equal to the specified burst level (see BURST field) the L64702 sets the THR bit in the STS. During decompression (ENC = 0), if the number of words in the FIFO is less than or equal to the specified burst level, the L64702 sets the THR bit.

THR_IE	*Definition*
0	Disable Burst Level Interrupt
1	Enable Interrupt Upon Burst Level

EOA_IE **End Of Active Window Interrupt Enable** **11**

The CPU may set EOA_IE to enable an interrupt when an end-of-active window condition has been reached. This condition occurs when the last pixel in the specified active window has been accessed. If this condition occurs, the L64702 then sets the EOA bit in the STS.

EOA_IE	*Definition*
0	Disable End-of-Active Window Interrupt
1	Enable Interrupt Upon End-of-Active Window Condition

LCIF_IE **Last Code In FIFO Interrupt Enable** **12**

The CPU may set LCIF_IE to enable an interrupt when the JPEG processing unit writes the last codeword into the FIFO during compression (ENC = 1), regardless of whether or not the burst level has been

reached. If this condition occurs, the L64702 then sets the LCIF bit in the STS.

LCIF_IE	*Definition*
0	Disable Last Code in FIFO Interrupt
1	Enable Interrupt on Last Code in FIFO

RI_IE **Restart Marker Code Interrupt Enable** **13**

The CPU may set RI_IE to enable an interrupt when the L64702 detects a RST marker in the code data stream. If this condition occurs, the L64702 decompresses all the data entered prior to the reset marker, resets the DC predictors, resumes operation, then sets the RI_IE bit in the STS. RI_IE is set during decompression only (ENC = 0).

RI_IE	*Definition*
0	Disable Restart Marker Code Interrupt
1	Enable Interrupt on Restart Marker Code

EOI_IE **End Of Image Interrupt Enable** **14**

The CPU may set EOI_IE to enable an interrupt when the L64702 detects an EOI (End-of-Image) marker in the code data stream. If this condition occurs, the L64702 sets the EOI bit in the STS and then decompresses all the data received. Meanwhile, video processing continues until the L64702 sets the EOA bit in the STS, indicating the end of the active window. The EOI condition is active during decompression only (ENC = 0).

EOI_IE	*Definition*
0	Disable End-of-Image Interrupt
1	Enable Interrupt on End-of-Image

IMC_IE **Illegal Marker Code Interrupt Enable** **15**

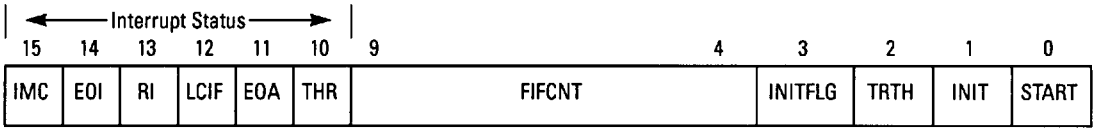
The CPU sets IMC_IE to enable the illegal marker code interrupt. When IMC_IE is set, the L64702 generates an interrupt upon detecting an illegal marker code in the code bit stream during decompression only (ENC = 0). If the L64702 detects an illegal marker code, it sets the IMC bit in the STS and terminates execution. In order to restart operation, you must reset the L64702.

IMC_IE	*Definition*
0	Disable Interrupt
1	Enable Interrupt on Illegal Marker Code

System Status Register (STS)

The STS Register, shown in Figure 3.3, provides the external system with status information on the L64702. This 16-bit register is read-only. It provides information on the number of words in the FIFO and any information that may cause generation of an internal interrupt condition. The interrupt status bits are set regardless of the enable interrupt bits in the SMR Register. The internal status is updated every L64702 CLK, and when the CPU reads the status, the current information is buffered in a special purpose STS buffer, which locks the STS value until the end of the CPU status read.

Figure 3.3
System Status Register



START **Start Operation Control Bit** **0**
This bit gives the current status of the START bit in the SMR Register.

INIT **Initialize Mode** **1**
This bit gives the current status of the INIT bit in the SMR Register.

TRTH **Transfer Through** **2**
This bit gives the current status of the TRTH bit in the SMR Register.

INITFLG **Initialize Flag** **3**
When the L64702 sets the INITFLG flag, it indicates that the internal JPEG processing unit is paused or in the idle state and the Group 2 tables and registers can be accessed. The INIT bit in the SMR Register can be set only when the INITFLG flag in the STS Register is set.

INITFLG	*Definition*
0	The L64702 is not in the INIT State
1	The L64702 is in the Idle state, and initialization can be done

FIFCNT **FIFO Word Count** **[9:4]**
The L64702 sets these six bits to the number of codewords currently in the FIFO. This information is useful when the CPU or DMA controller needs to burst out the remainder of the data not on a specified burst boundary, which may occur at the end of a compressed data

stream. The number of codewords in the FIFO is also useful for the CPU to know as it polls the FIFO.

FIFCNT	*Definition*
000000	FIFO Empty
000001	One Word in the FIFO
.	.
.	.
011111	31 Words in the FIFO
100000	32 Words in the FIFO

THR **FIFO Threshold** **10**

The L64702 sets THR if the FIFO burst level is reached. It also generates an interrupt if the THR_IE bit is set in the SMR when the FIFO burst level is reached. During compression (ENC = 1), the L64702 sets THR when the number of words in the FIFO is greater than or equal to the specified burst level (see the THRL field). During decompression (ENC = 0), the L64702 sets THR when the number of words in the FIFO is less than or equal to the specified burst level. After reset, the L64702 clears THR to zero.

THR	*Definition*
0	Burst Level not Reached
1	Burst Level Reached

EOA **End Of Active Window** **11**

The L64702 sets EOA when it detects that an end-of-active window condition has been reached. It also generates an interrupt if the EOA_IE bit is set in the SMR. The end-of-active window condition occurs when the last pixel in the specified active window has been accessed. EOA stays set until a hardware or software reset occurs. After a reset, the L64702 sets EOA to a logical zero and disables the interrupt.

EOA	*Definition*
0	No End-of-Active Window Reached
1	End-of-Active Window Reached

LCIF **Last Code In FIFO** **12**

The L64702 sets LCIF HIGH when the JPEG processing unit writes the last codeword into the FIFO. It also generates an interrupt if the LCIF_IE bit is set in the SMR. The L64702 writes to the FIFO during

compression (ENC = 1). After reset, the L64702 clears LCIF to zero and disables the interrupt.

LCIF	*Definition*
0	Last Codeword not Written in FIFO
1	Last Codeword Written in FIFO

RI **Restart Marker Code** **13**

The L64702 sets RI when it detects a RST marker code in the code data stream during decompression (ENC = 0). RST marker codes include all the codes in the range from 0xFFD0 through 0xFFD7. The L64702 also generates an interrupt if the RI_IE bit is set in the SMR. Once RI is set, the L64702 enters a Restart Interval, during which the previously received code data is decompressed. At this point, the CPU should stop sending compressed data to the FIFO. Otherwise, data corruption may occur. Once all the previously received data has been decompressed, the CPU may continue sending compressed data to the L64702 over the system interface. After reset, the L64702 sets RI to a logical zero and disables the interrupt.

RI	*Definition*
0	No RST Marker Code has been Detected
1	RST Marker Code has been Detected

EOI **End Of Image** **14**

The L64702 sets EOI when it detects an EOI (End of Image) marker in the code data stream. An EOI code has the value of 0xFFD9. The L64702 also generates an interrupt if the EOI_IE bit is set in the SMR. This bit can be set during decompression only (ENC = 0). EOI remains set until a hardware or software reset sets it to zero. During compression EOI will always remain a logical zero. After reset, the L64702 sets EOI to a logical zero and disables the interrupt.

EOI	*Definition*
0	No End-of-Image Marker Detected
1	End-of-Image Marker Detected

IMC **Illegal Marker Code** **15**

The L64702 sets IMC during decompression (ENC = 0) if it detects an illegal marker code in the code bit stream. An illegal marker code is any code that has a value of 0xFFxx where the xx byte is neither 00, D9, nor in the range D0 through D7. It also generates an interrupt if the IMC_IE bit is set in the SMR. If an illegal marker code is detected, the L64702 stops execution and enters the HALT state. A

software reset causes the L64702 to exit from the HALT state. At this point, operation must start over. Decompression cannot continue from the point where the execution was terminated. After reset, the bit is cleared to zero.

IMC	*Definition*
0	No Illegal Marker Code Detected
1	Illegal Marker Code Detected

3.4 Group 2 Registers and Tables

Group 2 contains all the registers and tables that determine L64702 operation. From the JPEG perspective, the Group 2 resources can be programmed with values for the DCT/IDCT conversion matrix, quantization factors, and the Huffman tables. From the video perspective, you can load the nine Color Space Conversion coefficients and a set of Video Controller registers.

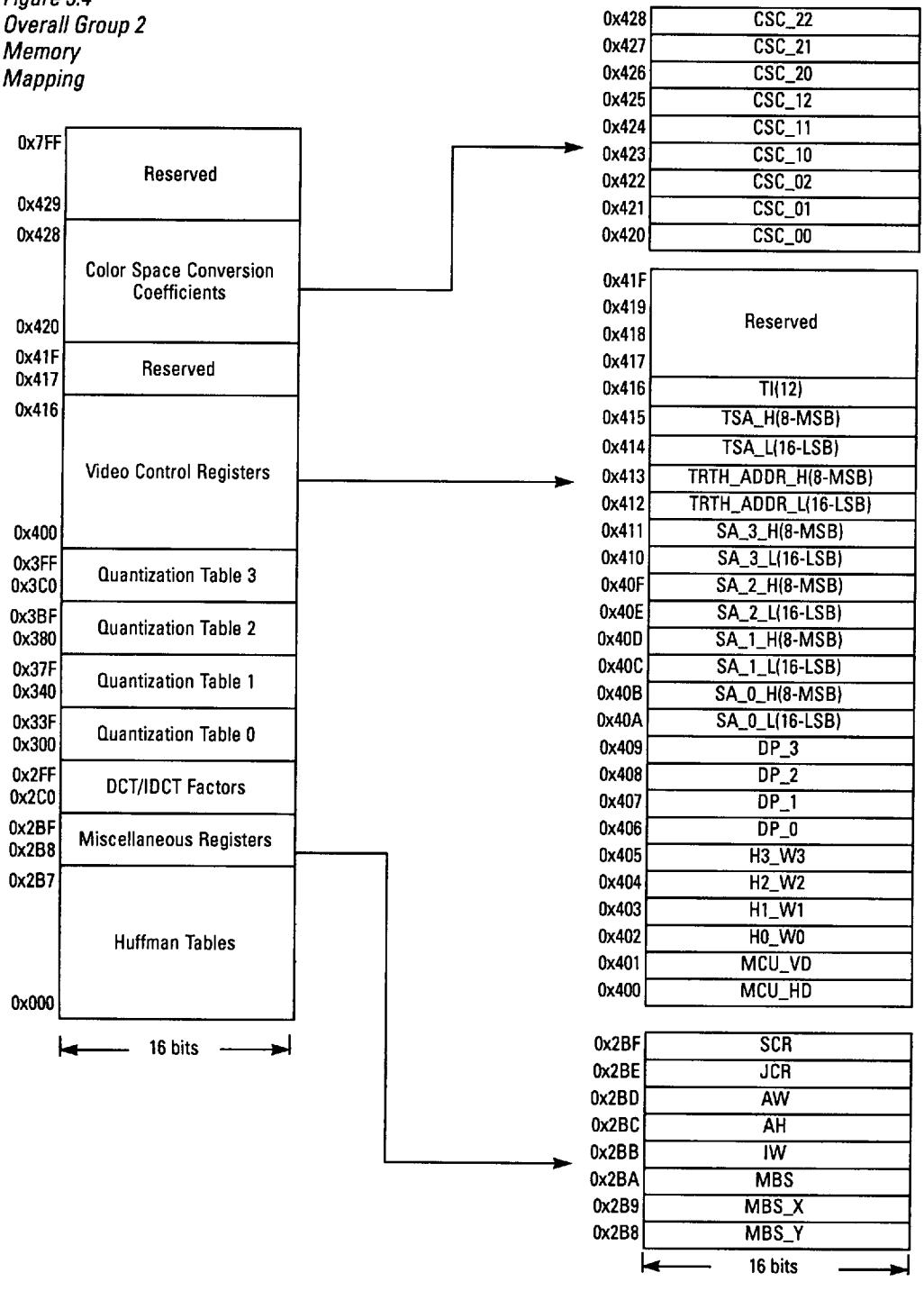
The L64702 uses Group 2 registers and tables during operation. Changing the values in these registers and tables during operation may corrupt the results of the operation in progress. The INIT bit in the SMR Register may be used to protect Group 2 resources during operation. If you want to read or write Group 2 resources, first set the INIT bit. When the L64702 has finished with the operation in progress, it sets the INITFLG flag in the STS Register to indicate that Group 2 resources are ready to be read or written.

Group 2 tables and registers are accessible to the system only if the INIT bit in the SMR Register is set—otherwise access is not allowed and the L64702 ignores any attempt to access Group 2. Because Group 2 contains tables and registers organized in a register file, you must use the APR Register to address specific locations in the group. You must initialize the APR Register to the starting address of the destination memory in Group 2 prior to the access of Group 2. The APR Register is automatically incremented by one at the end of each memory read or write cycle to Group 2.

A CPU can access Group 2 by setting the SRS[1:0] pins to 10_2. A DMA controller can access Group 2 registers and tables when the INIT and DE bits in the SMR Register are set HIGH.

Figure 3.4 shows the overall memory mapping of Group 2 registers and tables within the L64702. Avoid using the *reserved* areas within the memory map, because the L64702 may overwrite these areas during operation.

Figure 3.4
Overall Group 2
Memory
Mapping



Group 2 Register and Table Summary

Table 3.2 summarizes the Group 2 registers and tables, including register or table name, function, access type, internal address mapping location, and bit width.

Table 3.2
Group 2 Registers and Tables Summary

Register	*Function*	*Access Type*	*Int Addr Mapping*	*Width*
Huffman Tables				
HFT	Huffman Tables (0 and 1)	R/W	0x000–0x2B7	12 bits
DCT Factors Register				
DCT	DCT Factors	R/W	0x2C0–0x2FF	12 bits
Quantization Tables				
QT3	Quantization Table Number 3	R/W	0x3C0–0x3FF	8 bits
QT2	Quantization Table Number 2	R/W	0x380–0x3BF	8 bits
QT1	Quantization Table Number 1	R/W	0x340–0x37F	8 bits
QT0	Quantization Table Number 0	R/W	0x300–0x33F	8 bits
Video Control Registers				
TI	Transfer Increment Register	R/W	0x416	12 bits
TSA_H	Transfer Start Address High Register	R/W	0x415	8 bits
TSA_L	Transfer Start Address Low Register	R/W	0x414	16 bits
TRTH_ADDR_H	Transfer Through Address High Word	R/W	0x413	8 bits
TRTH_ADDR_L	Transfer Through Address Low Word	R/W	0x412	16 bits
SA_3_H	Start Address Plan 3 High Word	R/W	0x411	8 bits
SA_3_L	Start Address Plan 3 Low Word	R/W	0x410	16 bits
SA_2_H	Start Address Plan 2 High Word	R/W	0x40F	8 bits
SA_2_L	Start Address Plan 2 Low Word	R/W	0x40E	16 bits
SA_1_H	Start Address Plan 1 High Word	R/W	0x40D	8 bits
SA_1_L	Start Address Plan 1 Low Word	R/W	0x40C	16 bits
SA_0_H	Start Address Plan 0 High Word	R/W	0x40B	8 bits
SA_0_L	Start Address Plan 0 Low Word	R/W	0x40A	16 bits
DP_3	Display Pitch for Plan 3	R/W	0x409	12 bits
DP_2	Display Pitch for Plan 2	R/W	0x408	12 bits
DP_1	Display Pitch for Plan 1	R/W	0x407	12 bits
DP_0	Display Pitch for Plan 0	R/W	0x406	12 bits
H3_W3	Number of lines and pixels in last MCU 3	R/W	0x405	15 bits
H2_W2	Number of lines and pixels in last MCU 2	R/W	0x404	15 bits
H1_W1	Number of lines and pixels in last MCU 1	R/W	0x403	15 bits
H0_W0	Number of lines and pixels in last MCU 0	R/W	0x402	15 bits
MCU_VD	MCU Horizontal Delay	R/W	0x401	10 bits
MCU_HD	MCU Horizontal Delay	R/W	0x400	10 bits

(Sheet 1 of 2)

Table 3.2 (Continued)
Group 2 Registers and
Tables Summary

Register	*Function*	*Access Type*	*Int Addr Mapping*	*Width*
Miscellaneous Registers				
SCR	System Configuration Register	R/W	0x2BF	16 bits
JCR	JPEG Control Register	R/W	0x2BE	16 bits
AW	Active Window Width	R/W	0x2BD	9 bits
AH	Active Window Height	R/W	0x2BC	9 bits
IW	Image Width	R/W	0x2BB	9 bits
MBS	Macro Block Size	R/W	0x2BA	16 bits
MBS_X	Macro Block Size in the Y Direction	R/W	0x2B9	16 bits
MBS_Y	Macro Block Size in the X Direction	R/W	0x2B8	16 bits
Color Space Conversion Registers				
CSC_22	Color Space Conversion Component 22	R/W	0x428	11 bits
CSC_21	Color Space Conversion Component 21	R/W	0x427	11 bits
CSC_20	Color Space Conversion Component 20	R/W	0x426	11 bits
CSC_12	Color Space Conversion Component 12	R/W	0x425	11 bits
CSC_11	Color Space Conversion Component 11	R/W	0x424	11 bits
CSC_10	Color Space Conversion Component 10	R/W	0x423	11 bits
CSC_02	Color Space Conversion Component 02	R/W	0x422	11 bits
CSC_01	Color Space Conversion Component 01	R/W	0x421	11 bits
CSC_00	Color Space Conversion Component 00	R/W	0x420	11 bits

(Sheet 2 of 2)

Huffman Code Tables

The L64702 uses a proprietary algorithm of mapping standard Huffman tables into internal codewords that are loaded during initialization. The algorithm allows you to load two sets of Huffman code tables. Each set of tables contains one AC table and one DC table and coding/decoding.

You need to translate the regular Huffman table values to values that the L64702 understands. This translation is supported by the supplied L64702 Huffman table translator. Each value in the table is 12 bits wide.

The L64702 uses different tables, depending on whether compression or decompression is being performed. The tables must be loaded appropriately as the encoding or decoding direction changes.

Table 3.3
Huffman Tables

Register	*Function*	*Access Type*	*Int Addr Mapping*	*Width*
HFT	Huffman Tables	R/W	0x000–0x2B7	12 bits

System Configuration Register (SCR)

The System Configuration Register contains a set of configuration bits that determine the type of L64702 operation. The register contains bits that determine the operation direction (encode or decode), specify the image data structure, determine the memory refresh cycle, and determine the direction of the video memory serial port, image capture, or display.

Figure 3.5 System Configuration Register

15	14	13	12	11	10	9	8	7	6	5 4	3 1	0
SRESET	AEOI	PDWN	SP2	SP1	SP0	SC2	SC1	SC0	C_D	IDS	RR	ENC

ENC **Encode/Decode Processing Direction** **0**

The ENC bit determines the direction (compression or decompression) of L64702 image processing. When the ENC bit is set HIGH, the L64702 encodes image data. If the bit is reset LOW, the L64702 decodes image data. The START bit must be HIGH to enable image encoding or decoding.

ENC	*Definition*
0	Image Decoding (Decompression)
1	Image Encoding (Compression)

The ENC bit also sets the direction of read-through and write-through operations.

ENC	*Definition*
0	Write-through
1	Read-through

RR **Refresh Rate** **[3:1]**

These three bits determine the refresh rate period of the external video memory. When active, the internal refresh counter generates an internal request to perform a periodic $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh cycle. The specified value for RR corresponds to the following refresh rate in L64702 CLK cycles:

Refresh Rate = (RR + 1) * 64

RR	*Definition*
000	64 Cycles Refresh
001	128
010	192
011	256
100	320
101	384
110	448
111	No Refresh

IDS **Image Data Structure** **[5:4]**

These two bits determine the image data structure over the video bus and also determine the external video memory image organization.

IDS	*Definition*
00	Multisource source color (RR, ...R, GGG, ...G, BBB, ...B), consisting of three color components occupying three separate regions in the video memory.
01	Unpacked color (RGBx), made up of a three-color component structure where VADB[23:0] holds the actual data (RGB) and VADB[31:24] is ignored.
10	Packed color (RGBR, ...), consisting of a three-color component structure occupying the entire 32-bit video data bus.
11	Sequential components (CCC ... C, MMM ... M, YY ... Y, KKK ... K) mode, which turns off the color space conversion logic from one to four components occupying four separate regions in the video memory. The sequential components mode fully supports the JPEG baseline system as far as the number of components and the subsampling scheme.

C_D **Capture or Display** **6**

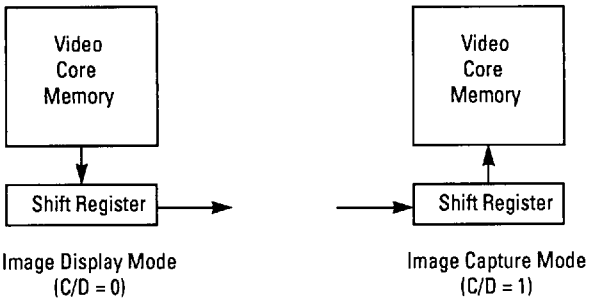
The C_D bit determines what kind of video shift register memory cycle the L64702 will perform, as well as the direction of the video memory serial port data. If set HIGH, the L64702 is in the image capture mode. In this mode, the L64702 responds to the video request transfer signal, $\overline{\text{VTRS}}$, by performing a pseudo write cycle, and to the video transfer request input, $\overline{\text{VTTR}}$, by performing alternate write cycles. Image capture accomplishes a transfer of image data from the video memory serial shift register to the video memory core. For more detailed information on pseudo write cycles and alternate write cycles, please refer to Chapter 10, "Video Memory Interface Timing."

When C_D is zero, the L64702 performs an image display operation by transferring data between the video core memory and the video memory serial shift register each time a transfer request has been detected.

C_D	*$\overline{VTRS}$*	*$\overline{VTRR}$*	*Definition*
0	Normal Read Transfer	Pseudo Split Read	Image Display
1	Pseudo Write Transfer	Alternate Write Transfer	Image Capture

Figure 3.6 shows the image capture and image display modes of the L64702 as they relate to the state of the C_D bit.

Figure 3.6
Image Capture and Display Modes



SC0 **Sign—Color Space Conversion Component 0** **7**

Sign of Color Space Conversion Component 0. This bit affects the sign of the low-byte component (bit 0 to bit 7) between the MCU buffer and the CSC unit. If set HIGH, the component is represented as a two's complement number. If cleared LOW, the component is represented as an unsigned number.

SC0	*Definition*
0	Unsigned Number
1	Two's Complement Number

SC1 **Sign—Color Space Conversion Component 1** **8**

Sign of Color Space Conversion Component 1. This bit affects the sign of the middle-byte component (bit 8 to bit 15) between the MCU buffer and the CSC unit. If set HIGH, the component is represented as a two's complement number. If cleared LOW, the component is represented as an unsigned number.

SC1	*Definition*
0	Unsigned Number
1	Two's Complement Number

SC2 **Sign—Color Space Conversion Component 2** **9**

Sign of Color Space Conversion Component 2. This bit affects the sign of the high-byte component (bit 16 to bit 23) between the MCU buffer and the CSC unit. If set HIGH, the component is represented as a two's complement number. If cleared LOW, the component is represented as an unsigned number.

SC2	*Definition*
0	Unsigned Number
1	Two's Complement Number

SP0 **Sign—Pixel Component 0** **10**

Sign of the pixel value for Component 0. This bit affects the sign of the low-byte component (bit 0 to bit 7) on the video bus, VADB. If set HIGH, the component is represented as a two's complement number. If cleared LOW, the component is represented as an unsigned number.

SP0	*Definition*
0	Unsigned Number
1	Two's Complement Number

SP1 **Sign—Pixel Component 1** **11**

Sign of the pixel value for Component 1. This bit affects the sign of the middle-byte component (bit 8 to bit 15) on the video bus, VADB. If set HIGH, the component is represented as a two's complement number. If cleared LOW, the component is represented as an unsigned number.

SP1	*Definition*
0	Unsigned Number
1	Two's Complement Number

SP2 **Sign—Pixel Component 2** **12**

Sign of the pixel value for Component 2. This bit affect the sign of the high-byte component (bit 16 to bit 23) on the video bus, VADB. If set HIGH, the component is represented as a two's complement number. If cleared LOW, the component is represented as an unsigned number.

SP2	*Definition*
0	Unsigned Number
1	Two's Complement Number

PDWN **Power Down** **13**

The PWDN bit, when set, deactivates the high density RAMs and greatly reduces the static power consumption of the L64702. This bit reduces the current by 125 mA. The internal content of the RAMS is retained. If this bit is set HIGH during the middle of an operation, data will be corrupted. PWDN must be set between images only. This bit must be LOW when the L64702 is operating.

PDWN	*Definition*
0	Full Power Consumption
1	Reduced Power Consumption

AEOI **Add End-of-Image Marker Code** **14**

When set HIGH, the AEOI bit causes the L64702 to append an 0xFFD9 marker code at the end of image compression.

AEOI	*Definition*
0	Do Not Add 0xFFD9 Marker Code
1	Add 0xFFD9 Marker Code

SRESET **Software Reset** **14**

The SRESET bit, when set HIGH, resets the L64702. Writing a zero to this bit ends the reset condition. Unlike FRESET, a HIGH on SRESET causes a total reset of the L64702 as does assertion of the RESET input.

SRESET	*Definition*
0	Do Not Reset the L64702
1	Reset the L64702

JPEG Configuration Register (JCR)

The 16-bit JPEG Configuration Register (JCR) determines which Huffman table and which quantization table will be used by each of the color components during operation. For each color component, you can select among two DC code tables, two AC code tables, and four quantization tables. The decision on when and how to use the tables for each component is determined by the sampling registers.

Figure 3.7 JPEG Configuration Register (JCR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q3		Q2		Q1		Q0		AC3	AC2	AC1	AC0	DC3	DC2	DC1	DC0

DC0 **DC Code Table Selector for Component 0** **0**
Selects between DC Table 0 and DC Table 1 for Component 0. If HIGH, this bit selects DC Table 1. If LOW, it selects DC Table 0.

DC1 **DC Code Table Selector for Component 1** **1**
Selects between DC Table 0 and DC Table 1 for Component 1. If HIGH, it selects DC Table 1. If LOW, it selects DC Table 0.

DC2 **DC Code Table Selector for Component 2** **2**
Selects between DC Table 0 and DC Table 1 for Component 2. If HIGH, it selects DC Table 1. If LOW, it selects DC Table 0.

DC3 **DC Code Table Selector for Component 3** **3**
Selects between DC Table 0 and DC Table 1 for Component 3. If HIGH, it selects DC Table 1. If LOW, it selects DC Table 0.

AC0 **AC Code Table Selector for Component 0** **4**
Selects between AC Table 0 and AC Table 1 for Component 0. If HIGH, it selects AC Table 1. If LOW, it selects AC Table 0.

AC1 **AC Code Table Selector for Component 1** **5**
Selects between AC Table 0 and AC Table 1 for Component 1. If HIGH, it selects AC Table 1. If LOW, it selects AC Table 0.

AC2 **AC Code Table Selector for Component 2** **6**
Selects between AC Table 0 and AC Table 1 for Component 2. If HIGH, it selects AC Table 1. If LOW, it selects AC Table 0.

AC3 **AC Code Table Selector for Component 3** **7**
Selects between AC Table 0 and AC Table 1 for Component 3. If HIGH, it selects AC Table 1. If LOW, it selects AC Table 0.

Q0 **Quantization Table Selectors for Component 0** **[9:8]**

This two-bit field selects the quantization table for Component 0. The value of the field corresponds to the number of the quantization table to be used: Table 0, 1, 2, or 3.

Q0	*Definition*
00	Select Quantization Table 0
01	Select Quantization Table 1
10	Select Quantization Table 2
11	Select Quantization Table 3

Q1 **Quantization Table Selectors for Component 1** **[11:10]**

This two-bit field selects the quantization table for Component 1. The value of the field corresponds to the number of the quantization table to be used: Table 0, 1, 2, or 3.

Q1	*Definition*
00	Select Quantization Table 0
01	Select Quantization Table 1
10	Select Quantization Table 2
11	Select Quantization Table 3

Q2 **Quantization Table Selectors for Component 2** **[13:12]**

This two-bit field selects the quantization table for Component 2. The value of the field corresponds to the number of the quantization table to be used: Table 0, 1, 2, or 3.

Q2	*Definition*
00	Select Quantization Table 0
01	Select Quantization Table 1
10	Select Quantization Table 2
11	Select Quantization Table 3

Q3 **Quantization Table Selectors for Component 3** **[15:14]**

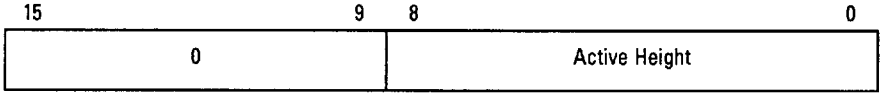
This two-bit field selects the quantization table for Component 3. The value of the field corresponds to the number of the quantization table to be used: Table 0, 1, 2, or 3.

Q3	*Definition*
00	Select Quantization Table 0
01	Select Quantization Table 1
10	Select Quantization Table 2
11	Select Quantization Table 3

Active Height Register (AH)

The nine-bit Active Height Register determines the Active Window height of a specific color component. Both the AH Register and the Height Width (Hx_Wx) Register specify the total number of MCUs in the active window area, counted in the vertical direction.

Figure 3.8 Active Height Register (AH)



Equation 3.1 calculates the active height. Regardless of which color component is selected, the AH result should be the same for all color components.

Equation 3.1 Active Height Calculation

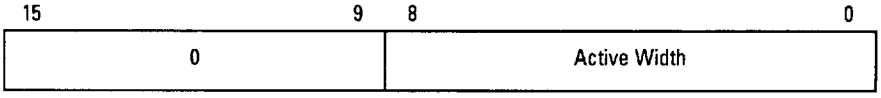
$$AH = \lceil Y/A \rceil - 1 \qquad \text{where } A = 8 \times MBS\_Y_i$$

The value in the AH Register represents the total integer number of MCUs minus one within the active window area. In other words, if the AH value is 39, the number of MCUs in the vertical direction is actually 40, which represents the number of MCUs starting at zero and ending at 39.

Refer to the subsection entitled "Active Window Height" on page 7-27 for sample calculations.

Active Width Register (AW)

The nine-bit Active Width (AW) Register determines the Active Window width. It is used in conjunction with the Height Width Register for a particular color component to specify the total number of MCUs in the active window area, counted in the horizontal direction.



Equation 3.2 specifies how the L64702 uses the AW and H0_W0 Registers (H0_W0 Registers are explained later in this chapter) to calculate the number of MCUs within the active window in the horizontal direction for color component zero. The same equations apply to all color components (0, 1, 2, or 3).

Equation 3.2 Active Width Calculation

$$AW = \lceil (X)/B \rceil - 1 \quad \text{where } B = 8 \times MBS\_X_i$$

The value in the AW Register represents the total integer number of MCUs minus one within the active window area. In other words, if the AW value is 39, the number of MCUs in the horizontal direction is actually 40, which represents the number of MCUs starting at zero and ending at 39.

Refer to the subsection entitled "Active Window Width" on page 7-26 for sample calculations.

Image Width Register (IW)

The L64702 uses the Image Width parameter during decompression. Note that during compression: 1) IW must be equal to the Active Width (AW), and 2) the Horizontal Delay (HD) and Vertical Delay (VD) must be zero.

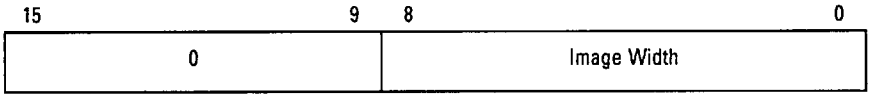


Figure 3.9 shows a two-dimensional view of a compressed image, where X is number of image pixels in the horizontal direction, and Y is the number of image lines in the vertical direction. This figure also shows that the L64702 allows the decompression of a portion of the compressed image. The space allocated to this decompressed portion of the image is called the active image area and is specified by the horizontal delay (HD), vertical delay (VD), Active Width (AW), and Active Height (AH) parameters. Note that when HD is zero, the Image Width (IW) value equals AW.

Figure 3.9
Active Image Area

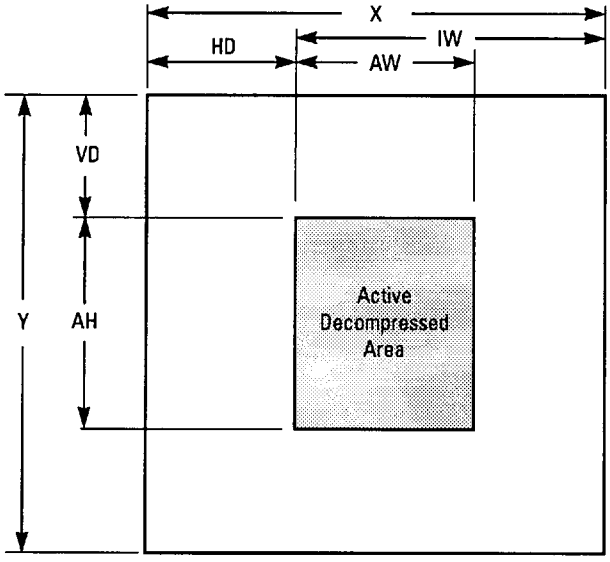


Figure 3.10 shows the meaning of display pitch (DP). Display pitch is the address gap in the video memory between two adjacent vertical pixels, and is equal to the width of the overall display.

Figure 3.10
Display Pitch

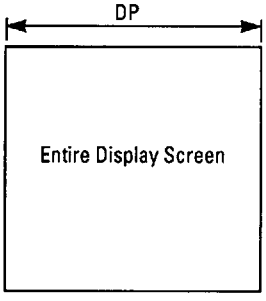
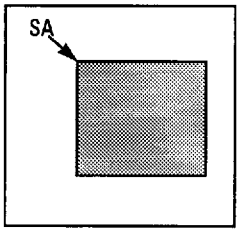


Figure 3.11 shows how the decompressed portion of an image (the active image) is located within the overall display by using the SA (starting address) parameter.

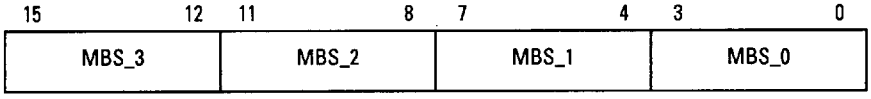
Figure 3.11
Starting Address



Macro Block Size Register (MBS)

The MBS Register specifies how many Data Units (8 x 8 blocks) comprise a Minimum Code Unit (MCU) for each one of the four color components, as defined in the JPEG standard. The values defined in this register define the sampling ratio for the color components. For example, in the RGBX mode, a 2:1:1 ratio indicates that two 8 x 8 blocks of the luminance component (Y) are processed for each single 8 x 8 block of the remaining chrominance (Cr, Cb) color components.

Figure 3.12 Macro Block Size Register (MBS)



MBS_0 **Macro Block Size for Color Component 0** **[3:0]**
This field specifies the total number of Data Units (8 x 8 blocks) that forms the macro block for Color Component 0.

MBS_0	*Definition*
0000	0 Data Units in the Macro Block
0001	1 Data Units in the Macro Block
0010	2 Data Units in the Macro Block
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_1 **Macro Block Size for Color Component 1** **[7:4]**
This field specifies the total number of Data Units (8 x 8 blocks) that forms the macro block for Color Component 1.

MBS_1	*Definition*
0000	0 Data Units in the Macro Block
0001	1 Data Units in the Macro Block
0010	2 Data Units in the Macro Block
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_2 **Macro Block Size for Color Component 2** **[11:8]**
This field specifies the total number of Data Units (8 x 8 blocks) that forms the macro block for Color Component 2.

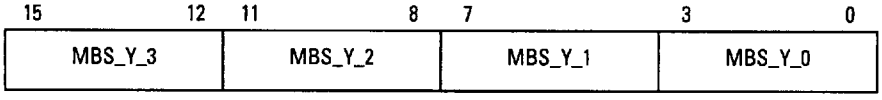
MBS_2	*Definition*
0000	0 Data Units in the Macro Block
0001	1 Data Units in the Macro Block
0010	2 Data Units in the Macro Block
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_3 **Macro Block Size for Color Component 3** **[15:12]**
This field specifies the total number of Data Units (8 x 8 blocks) that forms the macro block for Color Component 3.

MBS_3	*Definition*
0000	0 Data Units in the Macro Block
0001	1 Data Units in the Macro Block
0010	2 Data Units in the Macro Block
.	.
.	.
1010	10 Data Units in the Macro Block

Macro Block Size in the Y Direction Register (MBS_Y)

The MBS_Y Register specifies the number of Data Units (8 x 8 blocks) in the Y, or vertical, direction for each color component. The specified number of data units (8 x 8 blocks) determines the sampling ratio in the vertical direction. The number specified should be less than or equal to the number specified in the MBS Register for the particular component.



MBS_Y_0 **Macro Block Size in the Y Direction for Color Component 0** **[3:0]**
This field specifies the total number of Data Units (8 x 8 blocks) in the Y, or vertical, direction for Color Component 0. The value of this

field must be less than or equal to the value for MBS_0 specified in the MBS Register.

MBS_Y_0	*Definition*
0000	0 Data Units in the Macro Block
0001	1 Data Units in the Macro Block
0010	2 Data Units in the Macro Block
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_Y_1 Macro Block Size in the Y Direction for Color Component 1 **[7:4]**

This field specifies the total number of Data Units (8 x 8 blocks) in the Y, or vertical, direction for Color Component 1. The value of this field must be less than or equal to the value specified for MBS_1 in the MBS Register.

MBS_Y_1	*Definition*
0000	0 Data Units in the Macro Block in the Y Direction
0001	1 Data Units in the Macro Block in the Y Direction
0010	2 Data Units in the Macro Block in the Y Direction
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_Y_2 Macro Block Size in the Y Direction for Color Component 2 **[11:8]**

This field specifies the total number of Data Units (8 x 8 blocks) in the Y, or vertical, direction for Color Component 2. The value of this field must be less than or equal to the value specified for MBS_2 in the MBS Register.

MBS_Y_2	*Definition*
0000	0 Data Units in the Macro Block in the Y Direction
0001	1 Data Units in the Macro Block in the Y Direction
0010	2 Data Units in the Macro Block in the Y Direction
.	.
.	.
1010	10 Data Units in the Macro Block

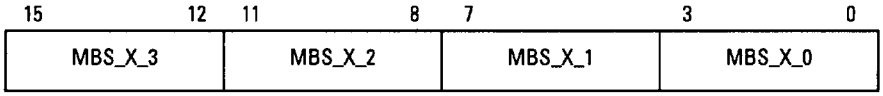
MBS_Y_3 Macro Block Size in the Y Direction for Color Component 3 **[15:12]**

This field specifies the total number of Data Units (8 x 8 blocks) in the Y, or vertical, direction for Color Component 3. The value of this field must be less than or equal to the value specified for MBS_3 in the MBS Register.

MBS_Y_3	*Definition*
0000	0 Data Units in the Macro Block in the Y Direction
0001	1 Data Units in the Macro Block in the Y Direction
0010	2 Data Units in the Macro Block in the Y Direction
.	.
.	.
1010	10 Data Units in the Macro Block

Macro Block Size in the X Direction Register (MBS_X)

The MBS_X Register specifies the number of Data Units (8 x 8 blocks) in the X direction for each color component. The specified number of data units (8 x 8 blocks) determines the sampling ratio in the horizontal direction. The number specified should be less than or equal to the number specified in the MBS Register for the particular component.



MBS_X_0 Macro Block Size in the X Direction for Color Component 0 **[3:0]**

This field specifies the total number of Data Units (8 x 8 blocks) in the X, or horizontal, direction for Color Component 0. The value of this field must be less than or equal to the value for MBS_0 specified in the MBS Register.

MBS_X_0	*Definition*
0000	0 Data Units in the Macro Block
0001	1 Data Units in the Macro Block
0010	2 Data Units in the Macro Block
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_X_1 Macro Block Size in the X Direction for Color Component 1 **[7:4]**

This field specifies the total number of Data Units (8 x 8 blocks) in the X, or horizontal, direction for Color Component 1. The value of this field must be less than or equal to the value specified for MBS_1 in the MBS Register.

MBS_X_1	*Definition*
0000	0 Data Units in the Macro Block in the X Direction
0001	1 Data Units in the Macro Block in the X Direction
0010	2 Data Units in the Macro Block in the X Direction
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_X_2 Macro Block Size in the X Direction for Color Component 2 **[11:8]**

This field specifies the total number of Data Units (8 x 8 blocks) in the X, or horizontal, direction for Color Component 2. The value of this field must be less than or equal to the value specified for MBS_2 in the MBS Register.

MBS_X_2	*Definition*
0000	0 Data Units in the Macro Block in the X Direction
0001	1 Data Units in the Macro Block in the X Direction
0010	2 Data Units in the Macro Block in the X Direction
.	.
.	.
1010	10 Data Units in the Macro Block

MBS_X_3 Macro Block Size in the X Direction for Color Component 3 **[15:12]**

This field specifies the total number of Data Units (8 x 8 blocks) in the X, or horizontal, direction for Color Component 3. The value of

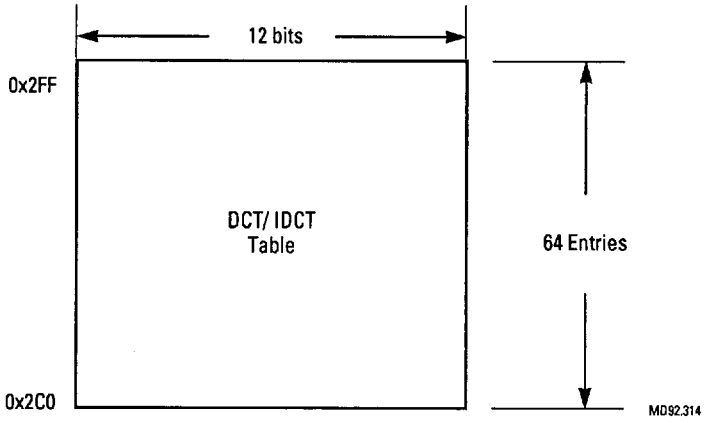
this field must be less than or equal to the value specified for MBS_3 in the MBS Register.

MBS_X_3	*Definition*
0000	0 Data Units in the Macro Block in the X Direction
0001	1 Data Units in the Macro Block in the X Direction
0010	2 Data Units in the Macro Block in the X Direction
.	.
.	.
1010	10 Data Units in the Macro Block

DCT/IDCT Factors

The L64702 contains a table with 64 entries for holding the DCT or IDCT factors. The table is downloaded in raster block order. Figure 3.13 shows the configuration of the DCT/IDCT table.

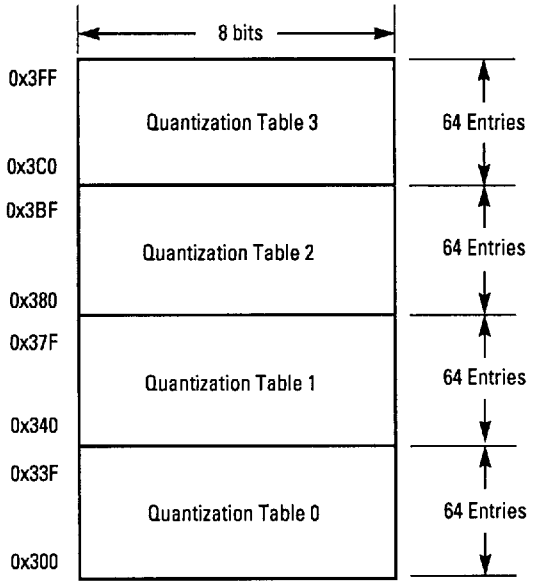
Figure 3.13
DCT/IDCT Table



Quantization Tables

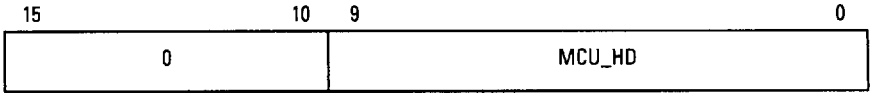
The L64702 contains four quantization tables that hold the quantization values for the DCT coefficients. The tables shown in Figure 3.14 are downloaded in raster block order. You can specify any one of the tables for each color component.

Figure 3.14
Quantization Tables



MCU Horizontal Delay Register (MCU_HD)

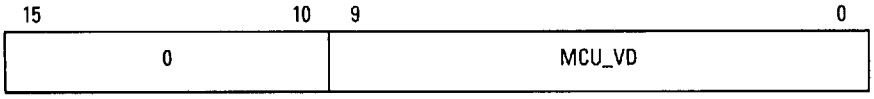
This 10-bit signed register (nine bits plus sign) determines the number of MCU blocks that the JPEG processing unit skips in the horizontal direction before writing image data into the MCU buffer. The specified number should be less than or equal to zero.



During a compression operation, this register should be cleared to zero. When scrolling a small display window in a large compressed file, this register should be programmed with the appropriate value: MCU_HD = -1 to skip one MCU, MCU_HD = -2 to skip two MCUs, etc.

MCU Vertical Delay Register (MCU_VD)

This 10-bit signed register (nine bits plus sign) determines the number of MCU blocks that the L64702 skips in the vertical direction before writing image data into the MCU buffer. The specified number should be less than or equal to zero.



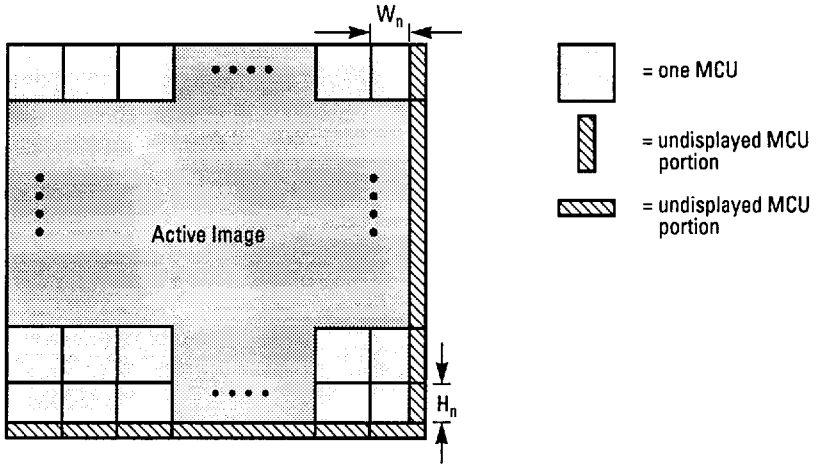
During a compression operation, this register should be cleared to zero. When scrolling a small display window in a large compressed file, this register should be programmed with the appropriate value: MCU_VD = -1 to skip one MCU, MCU_VD = -2 to skip two MCUs, etc.

Height/Width Component Registers

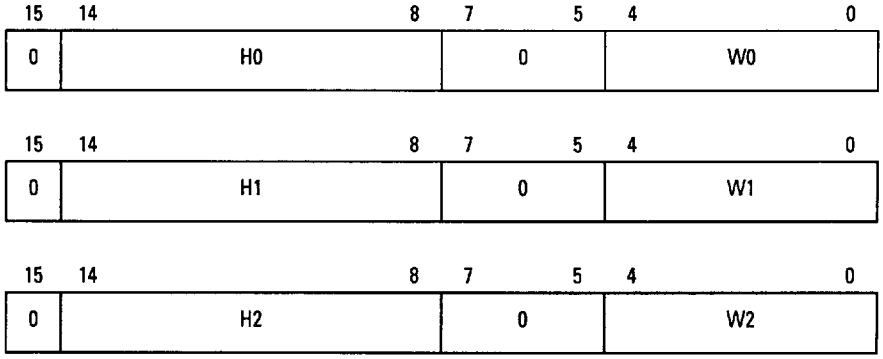
The video image is organized in terms of Minimum Code Units (MCUs). An MCU is typically a two-dimensional area of image data containing 8 x 8 blocks. The L64702 contains four Height/Width Component registers: H0_W0, H1_W1, H2_W2, and H3_W3. These registers specify both the number of pixels, W_n, in the last horizontal MCUs and the number of lines, H_n, in the last vertical MCUs of the display image.

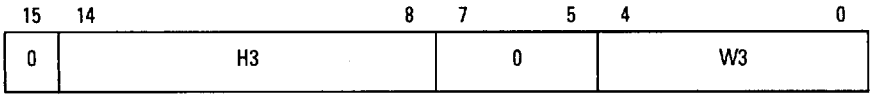
Figure 3.15 shows the location of W_n and H_n with regards to the active image and the MCUs.

Figure 3.15 W_n and H_n in the Active Image



The four Height/Width Component Registers are listed below.





Each Height/Width register is associated with a specific color component. The H0_W0 Register is used in both the color modes image data structure and the component sequential image data structure. The H1_W1, H2_W2, and H3_W3 Registers are only used in the components sequential image data structure mode.

Depending on the image structure, W_n is specified either in terms of pixels or groups of pixels. Table 3.4 shows the valid values of W_n for three of the four image structures and formats.

Table 3.4
W_n Values

		W_n Values	
Image Data Structure	***Image Format***	***1:0:0, 1:1:1***	***2:1:1***
RGBx	Unpacked	0 through 7 pixels	0 through 15 pixels
RGBR	Packed	2, 5 pixel groups	2, 5, 8, 11 pixel groups
RRRR...	Multisource	0, 1 pixel groups	0, 1, 2, 3 pixel groups

In the component sequential image format, the value of W_n depends on the sampling value MBS_X:

$$W_n = 0 \rightarrow (MBS\_X\_n \times 2) - 1 \qquad \text{for components sequential}$$

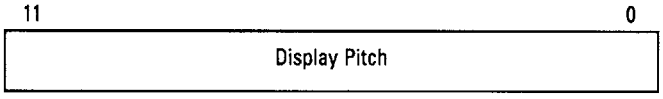
The value of H_n depends on the type of image structure as shown in Table 3.5.

Table 3.5
H_n Values

Image Format	***H_n Values***
Unpacked	0 through 7
Packed	0 through 7
Multisource	0 through 7
Component Sequential	0 through $(MBS\_Y\_n \times 8) - 1$

Display Pitch Registers

Four registers specify the display pitch factor for each color component: , DP_0, DP_1, DP_2, and DP_3, Display pitch is the address gap in the video memory between two adjacent vertical pixels. *The specified display pitch should be always greater than or equal to the active window width.*



Start Address Registers

The L64702 contains four Start Address registers: SA_0, SA_1, SA_2, and SA_3. These 24-bit registers specify the starting address of the active window in video memory for Components 0 through 3. The SA_0 Register is used in all color modes and in the component sequential mode for Component 0. The SA_1 and SA_2 Registers are used in the multisource and components sequential modes. The SA_3 Register is used in the components sequential mode only.

Figures 3.16 through 3.19 show the Start Address Registers. The LOW registers hold the 16 least-significant bits of the address and the HIGH registers hold the eight most-significant bits of the start address.

Figure 3.16 SA_0 LOW and HIGH Registers

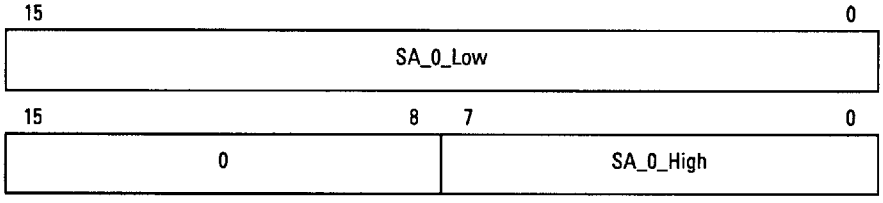


Figure 3.17 SA_1 LOW and HIGH Registers

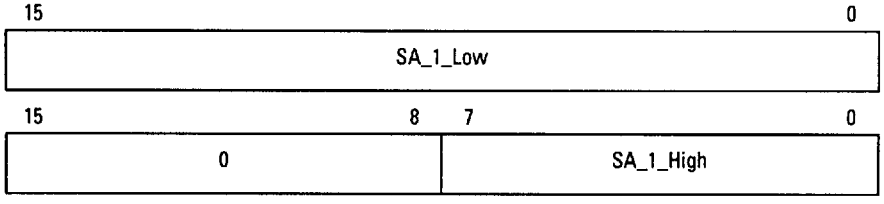


Figure 3.18 SA_2 LOW and HIGH Registers

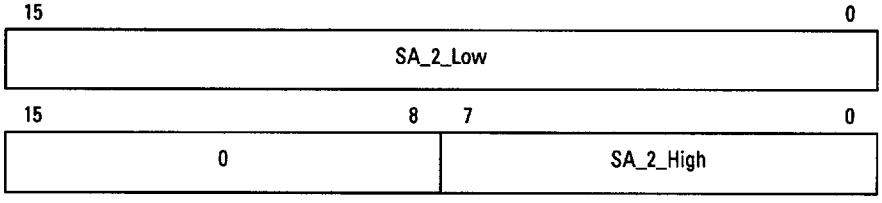
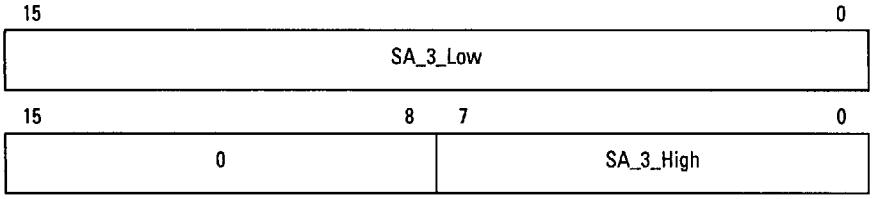


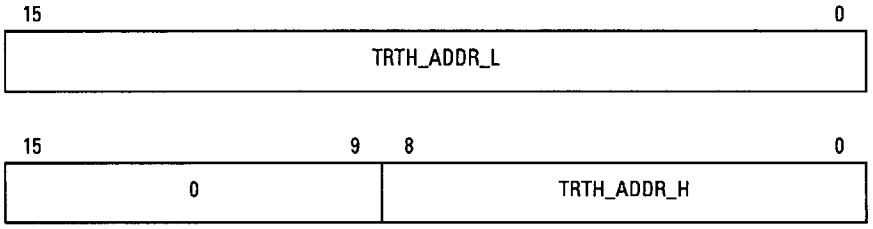
Figure 3.19 SA_3 LOW and HIGH Registers



Transfer-Through Address Register

The 24-bit Transfer-Through Address Register specifies the starting video memory address to be accessed during read-through or write-through transfers. This register is used only in the transfer-though mode. The 16 least-significant bits are in TRTH_ADDR_L. The eight most-significant bits are in TRTH_ADDR_H.

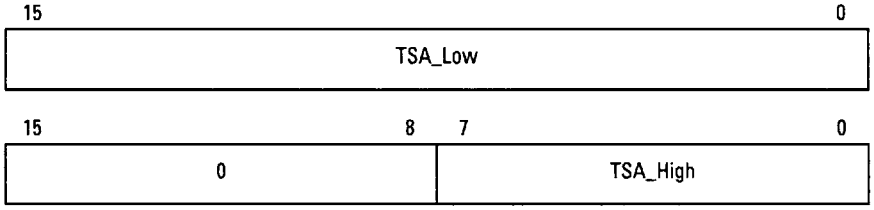
Figure 3.20 TRTH_ADDR Register



Transfer Start Address Register

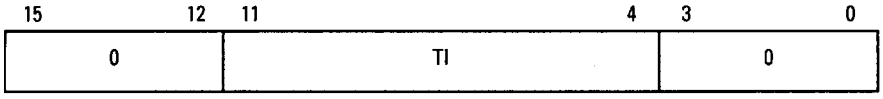
The 24-bit Transfer Start Address (TSA) Register specifies the starting address for reading or writing VRAM raster scan pixel data. TSA_Low holds the 16 least-significant bits and TSA_High holds the eight most-significant bits. This register is used when the $\overline{\text{VTRS}}$ pin is controlling the image capture or display features. At the falling edge of the $\overline{\text{VTRS}}$ signal, the contents of TSA are transferred to a temporary video transfer latch. The video transfer latch holds the updated transfer address created as a result of incrementing the TSA Register by the value contained in the Transfer Increment Register (TI).

Figure 3.21 TSA Register



Transfer Increment Register

The Transfer Increment (TI) Register specifies the incremental address space between adjacent scan lines. In regular use of the video transfer feature, this register's value should be the size of the shift register in the VRAM. On the falling edge of the $\overline{\text{VTRR}}$ signal, the L64702 adds the TI value to the address value in the temporary video transfer latch start address. Incrementing with the TI value is always done before the video transfer cycle. The lower four bits of this register must always be set to zero.



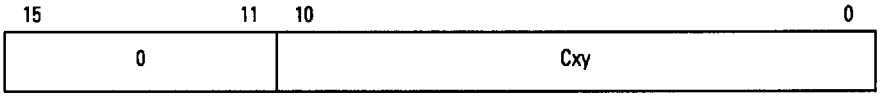
Color Space Conversion Registers (CSC)

Table 3.6 lists the nine 11-bit Color Space Conversion (CSC) Registers.

Table 3.6 Color Space Conversion Registers

Register	*Function*	*Int Addr Mapping*
CSC_22	Color Space Conversion Component 22	0x428
CSC_21	Color Space Conversion Component 21	0x427
CSC_20	Color Space Conversion Component 20	0x426
CSC_12	Color Space Conversion Component 12	0x425
CSC_11	Color Space Conversion Component 11	0x424
CSC_10	Color Space Conversion Component 10	0x423
CSC_02	Color Space Conversion Component 02	0x422
CSC_01	Color Space Conversion Component 01	0x421
CSC_00	Color Space Conversion Component 00	0x420

Each register contains a 3 x 3 matrix conversion value as shown below. The value in each register corresponds to an individual matrix coefficient value that corresponds to the name in the function column in the table.



3.5 Group 3 FIFO

Group 3 contains the FIFO. The L64702 uses the FIFO: 1) to store data just after it is compressed and before it is written into system memory, 2) just after data is retrieved from system memory and before it is decompressed for use by video memory, and 3) in read-through and write-through modes. A CPU may access the FIFO by setting SRS[1:0] = 11_2.

The L64702 may be programmed by an external CPU to allow the FIFO operation to match system requirements.

When INIT = 0, the FIFO is accessible to the external system, and the Group 2 resources of the L64702 are inaccessible to the external system.

The SMR Register contains bits that set the FIFO burst level, and enable generation of an external interrupt when the burst level is reached. The FIFO burst level indicates the number of words to be transferred in a burst transfer interval. These bits are meaningful only when the L64702 is in the image compression, image decompression, read-through, and write-through modes. The specified burst level affects the handshake mechanism of compressed data transfer between the system memory and the FIFO. In general, the burst level affects the functioning of the System DMA Request (SDRQ) pin, used in DMA transfers, and the functioning of the THR interrupt flag, used in the CPU transfer mode.

The CPU may set the THR_IE bit to enable an interrupt when the FIFO burst level is reached. During compression (ENC = 1), if the number of words in the FIFO is greater than or equal to the specified burst level, the L64702 sets the THR bit in the STS. During decompression, (ENC = 0), if the number of words in the FIFO is less than or equal to the specified burst level, the L64702 sets the THR bit in the STS. After reset, the L64702 clears THR_IE to zero to disable the interrupt.

The CPU may set LCIF_IE to enable an interrupt when the JPEG processing unit writes the last codeword into the FIFO during compression. If this condition occurs, the L64702 then sets the LCIF bit in the STS. After reset, the L64702 clears this flag to zero, disabling the interrupt.

The STS Register contains a FIFCNT field consisting of six bits that indicate the number of words in the FIFO. This information is useful when the CPU or DMA controller needs to burst out the remainder of the data not on a specified burst boundary, which may occur at the end of a compressed

data stream. The number of codewords in the FIFO is also useful for the CPU to know as it polls the FIFO.

Refer to the descriptions of the SMR and STS Registers on page 3-3 and page 3-8, respectively, for more information on the FIFO-related fields.

Accessing the FIFO

The procedure for accessing the FIFO is the same regardless of the operation: compression, decompression, read-through, or write-through. The FIFO can be accessed in either CPU or DMA transfer mode.

The following steps detail the procedure for accessing the L64702 FIFO:

1. The burst level must first be set in the SMR Register.
2. The external controller (CPU or DMA) must determine whether a transfer to the FIFO is legal. The FIFO can be read from when the number of words in it is equal to or greater than the burst level. The FIFO is ready to be written to when the number of empty slots in it is equal to or greater than the burst level.

   Before accessing the FIFO, the DMA controller should first check that the SDRQ pin is asserted.

   Before accessing the FIFO, the CPU should either read the FIFO_CNT bit in the STS Register to determine the FIFO's status or check for a one on the THR bit in the STS Register.
3. When the external controller accesses the FIFO, the exact number of transfers (equal to the burst level) must be performed before the L64702 updates the FIFO_CNT and THR status.

   For example, assume you set the burst level to 16 during decompression or write-through mode. After checking that it is legal to access the FIFO, perform three writes. At this point, if you try to read the FIFO_CNT field, the value will be zero. If you write the remaining 13 words into the FIFO and then read the FIFO_CNT, the FIFO_CNT value should be 16. Note that there is a four-cycle latency after the last write before the L64702 updates the FIFO_CNT value.

   Certain conditions cause the L64702 to update the FIFO_CNT even though the burst level has not been reached. Some of these conditions include IMC, RI, and EOI during decompression.

Video Memory Transfer-through Mode

The transfer-through feature allows an external CPU to indirectly (via the L64702) manipulate or evaluate data in the video memory. The L64702 uses the FIFO to hold temporary data coming from or written to the video memory. Because the L64702 has to take control over the video bus when transferring the data to or from the video memory, $\overline{\text{VBRQ}}$ and $\overline{\text{VBACK}}$ arbitrate the operation.

You must first write the address of the TRTH_ADDR_L Register into the APR with a Group 0 access (SRS[1:0] = 00_2). Next, use Group 2 accesses (SRS[1:0] = 10_2) to write the 32-bit video memory starting address into the TRTH_ADDR_L and TRTH_ADDR_H Register pair.

The ENC bit in the SCR determines whether the L64702 is performing a read-through or write-through operation in TRTH mode. When ENC = 1, the L64702 is setup for a read-through operation, and when ENC = 0, the L64702 is setup for a write-through operation.

Before setting the TRTH bit in the SMR Register, ensure that the FIFO is empty because the read-through and write-through operations use the FIFO. Setting TRTH to one initiates the loading of the TRTH_ADDR to a pointer register. This pointer is used for the video address. With TRTH set, the L64702 uses Group 3 as a buffer for read-through and write-through transfers.

For read-through (ENC = 1) operations, the L64702 accesses the video memory and places the data into the FIFO. Every access on the 32-bit wide video bus places two 16-bit words in the FIFO. The L64702 writes the least-significant word first. As the FIFO fills up, the system bus can access it using CPU or DMA mode as in the normal processing mode. The L64702 stops accessing the video memory when the FIFO is full or when the TRTH bit is cleared.

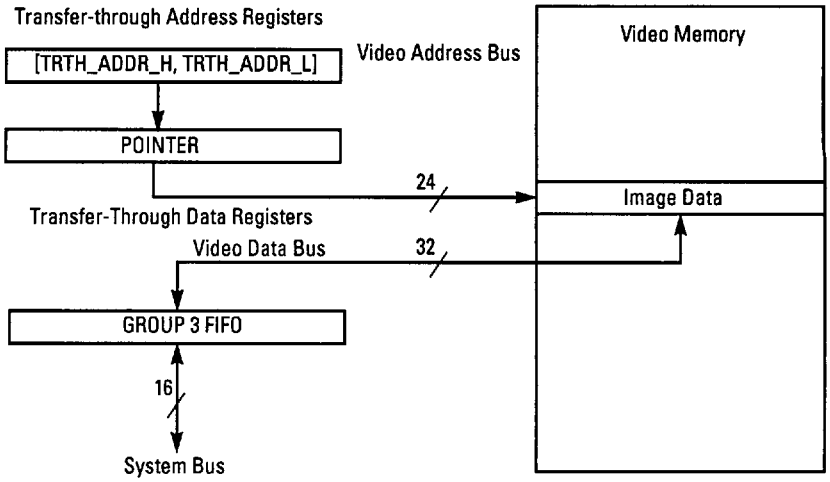


Figure 3.22 Video Memory Transfer-Through Mode

For write-through (ENC = 0) operations, the system must write into the FIFO using CPU or DMA mode transfers. The L64702 starts a video memory write cycle when there are at least two 16-bit words in the FIFO. The first 16 bits are written into the least significant word position of the VADB bus. Video memory write-throughs are inactive when the FIFO is empty or when TRTH is reset.

To reinitialize the pointer register, you must clear the TRTH bit to zero and then set it to one again.

Write-Through Access

A CPU must perform the following steps in order to accomplish a write-through of the L64702 to access video memory:

1. Write the memory map address of the TRTH_ADDR_L Register (0x412) into the APR Register.
2. Set the INIT bit in the SMR.
3. Write the 16 least-significant bits of the starting pixel address. This information is automatically sent to the TRTH_ADDR_L Register, because its address was previously loaded into the APR. The APR address now automatically increments by one to point to the TRTH_ADDR_H Register (0x413).
4. Write the eight most-significant bits of the starting pixel address. This information is automatically sent to the TRTH_ADDR_H Register.
5. Load the APR with the SCR Register address.

6. Write a zero to the ENC bit of the SCR Register to set the transfer-through direction (write-through).
7. Perform a frame reset to clear the FIFO count. This step is not necessary if the FIFO count is already zero.
8. Set HIGH the TRTH bit in the SMR Register. The L64702 now uses the FIFO for transfer through.
9. The CPU can now write data to the VRAM through the FIFO using either the CPU or DMA mode.

Read-Through Access

A CPU must perform the following steps to accomplish a read-through of the L64702 in order to access video memory:

1. Write the memory map address of the TRTH_ADDR_L Register (0x412) into the APR Register.
2. Set the INIT bit in the SMR.
3. Write the 16 least-significant bits of the starting pixel address. This information is automatically sent to the TRTH_ADDR_L Register, because its address was previously loaded into the APR. The L64702 now automatically increments the APR Register by one to point to the TRTH_ADDR_H Register (0x413).
4. Write the eight most-significant bits of the starting pixel address. This information is automatically sent to the TRTH_ADDR_H Register.
5. Load the APR with the SCR Register address.
6. Write a one to the ENC bit of the SCR Register to set the transfer-through direction (read-through).
7. Perform a frame reset to clear the FIFO count. This step is not necessary if the FIFO count is already zero. Set SRS[1:0] to 00_2 (points to Group 0, APR).
8. Set HIGH the TRTH bit in the SMR Register. The L64702 now uses the FIFO for transfer-through.
9. The CPU can now read data from the VRAM through the FIFO using either the CPU or DMA mode.

Chapter 4
System Interface

The L64702 system interface supports two different modes of data transfers: CPU and DMA. Both modes transfer data between the external system memory and the L64702 FIFO, internal memory, and registers.

This chapter contains three sections:

- Section 4.1, CPU and DMA Transfer Mode Overview
- Section 4.2, CPU Transfer Mode
- Section 4.3, DMA Transfer Mode

4.1 CPU and DMA Transfer Mode Overview

The CPU and DMA transfer modes differ in the way the data is transferred and in the handshaking method between the L64702 and the external CPU or external DMA controller. In addition, a CPU may access the Group 0 and Group 1 registers.

The types of data that are transferred are as follows:

1. 16-bit compressed data (from L64702 FIFO to system memory)
2. 16-bit compressed data (from system memory to L64702 FIFO)
3. Register and table data (from CPU to L64702 internal memory)
4. Register and table data (from L64702 internal memory to CPU)
5. Video data (read-through from video memory to CPU)
6. Video data (write-through from CPU to video memory)

The system interface is designed to communicate with an external CPU or DMA controller asynchronously (not related to the L64702 clock), using the input control signals, $\overline{\text{SRD}}$ (system read), $\overline{\text{SWR}}$ (system write), $\overline{\text{SCS}}$ (system chip select), and $\overline{\text{SDACK}}$ (system DMA acknowledge), and output control signal SDRQ (system DMA request). $\overline{\text{SINT}}$ (system interrupt), $\overline{\text{SRD}}$, and $\overline{\text{SWR}}$ are used for both CPU and DMA transfers. $\overline{\text{SCS}}$ (chip

select) is used only for CPU transfers, and $\overline{\text{SDACK}}$ (DMA acknowledge) and SDRQ (DMA request) are used only for DMA transfers. The L64702 system interface functions as a slave to the external system when operating with either an external CPU or DMA controller.

The L64702 System Mode Register (SMR) contains three control bits (START, DE, and INIT), which determine which register groups are available to the external CPU or DMA controller. As described in Chapter 3, "Internal Registers and Data Tables," the L64702 is memory mapped into four groups (0, 1, 2, or 3), which can be selected by either the SRS[1:0] pins and the SMR control bits START, DE, and INIT when in the CPU transfer mode, or by the SDRQ and $\overline{\text{SDACK}}$ pins along with the previously mentioned SMR control bits when in the DMA transfer mode.

Table 4.1 shows the various groups within the L64702 that may be accessed as determined by the SMR control bits START, DE, and INIT. The groups are as follows:

- Group 0: L64702 Address Pointer Register (APR)
- Group 1: L64702 System Mode Register (SMR) and System Status Register (STS)
- Group 2: L64702 Registers and Tables
- Group 3: L64702 FIFO

Table 4.1
Accessing the L64702 Internal Groups

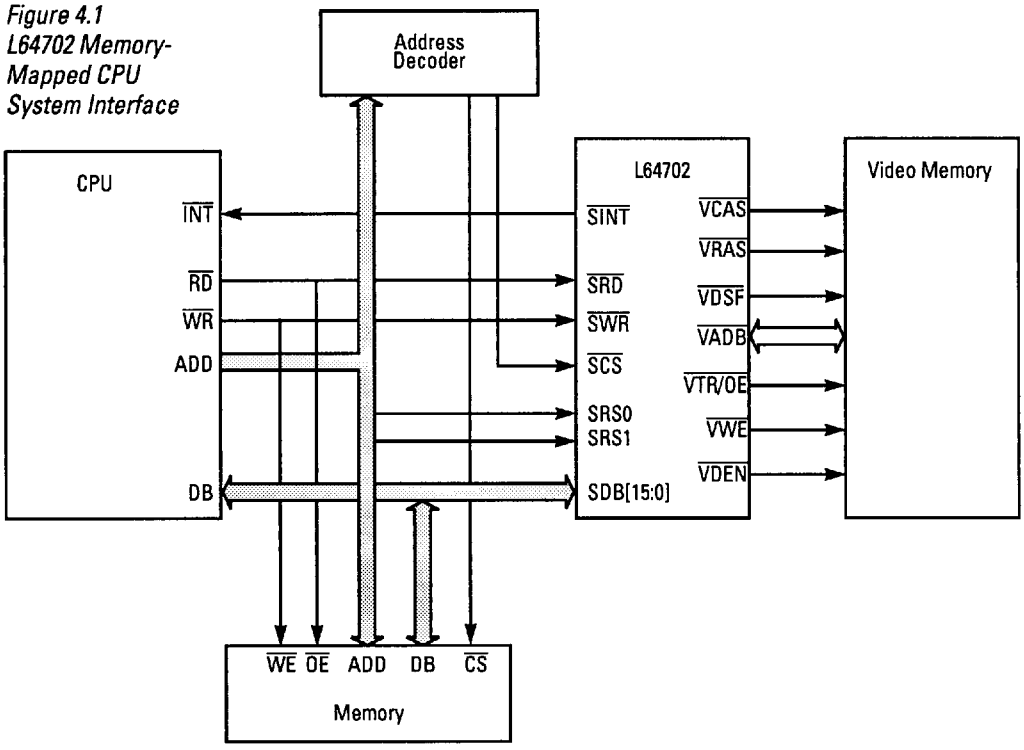
START	*DE*	*INIT*	*TRTH*	*CPU Access Groups*	*DMA Access Groups*	*Note*
0	0	0	1	0, 1, 3	–	CPU Transfer-through Mode, Group 2 Protected
0	1	0	1	0, 1, 3	3	DMA Transfer-through Mode
1	0	0	0	0, 1, 3	–	Processing Operation in CPU Transfer Mode
1	1	0	0	0, 1, 3	3	Processing Operation in DMA Transfer Mode
x[1]	0	1	0	0, 1, 2	–	Initialize Mode, CPU Transfer Mode
x	1	1	0	0, 1, 2	2	Initialize Mode, DMA Transfer Mode

1. x = don't care

4.2 CPU Transfer Mode

Figure 4.1 shows the control signals that provide the communication between the external CPU, external system memory, video memory, and the L64702. This figure also demonstrates how to configure the system so that the L64702 exists in the CPU address space.

Figure 4.1 L64702 Memory-Mapped CPU System Interface



The CPU communicates with the L64702 for three principal reasons:

1. To initialize the L64702.
2. To transfer the compressed data stream between the L64702 FIFO and external memory.
3. To access video memory through the L64702.

In CPU mode, the L64702 supports 16-bit CPU data transfers. The external CPU asserts $\overline{SCS}$ LOW to direct a CPU access (read/write) to the L64702. This signal can be driven directly from the address lines or from external system memory mapping address decoding logic. $\overline{SCS}$ should not be asserted in the DMA transfer mode. A system memory address decoder activates the $\overline{SCS}$ line at the appropriate system memory address.

The L64702 $\overline{SRD}$ and $\overline{SWR}$ signals are connected to the CPU read and write signals, respectively. The $\overline{SRD}$ and $\overline{SWR}$ signals activate either a read or a write cycle, respectively.

The register select lines, SRS[1:0], are connected to the lower CPU address bits. The external CPU uses SRS[1:0] to access internal memory and registers. The SRS pins select Groups 0 and 1 during CPU transfers only—the DMA controller cannot access these groups. A CPU or a DMA controller can access Groups 2 and 3.

Some L64702 registers are longer than 16 bits. These registers are divided into two 16-bit words—the least-significant 16 bits and the most-significant 16 bits. Each 16-bit word resides at a specific address, so that accessing a 24-bit or 32-bit register requires two 16-bit memory cycles. When the CPU writes to registers and tables of less than 16 bits, the L64702 ignores all unused bits. When the CPU reads L64702 registers or tables, the L64702 sets any unused bits to zero.

The L64702 uses an auto-increment address pointer to address the L64702 internal registers and tables. The auto-increment mechanism provides an elegant sequential initialization method. The L64702 can access Group 2 resources only when a CPU sets the INIT bit in the SMR Register. For more detailed information on the auto-increment mechanism, see Chapter 3, "Internal Registers and Data Tables."

To initialize the L64702, the CPU first sets the INIT bit in the SMR Register. Once this bit is set, the CPU may read and write the L64702 Group 2 tables and registers.

During image compression or decompression, the FIFO buffers compressed data and allows burst data transfers between the external system memory and the L64702. The FIFO becomes ready to be read from upon being filled with a number of words equal to or greater than the burst level. The FIFO can be written after being emptied to a number of words equal to or more than the burst level.

If the CPU attempts to read the FIFO while empty, or write to it while full, the integrity of the compressed data stream may be compromised. To avoid this situation, the CPU keeps track of the FIFO readiness state by properly setting up the L64702 SMR Register and monitoring the L64702 STS Register flags. Use these two registers to enable a hardware interrupt when changes in the FIFO occur. When an interrupt occurs, the L64702 asserts

the $\overline{INT}$ pin LOW. System software must handle changes in the FIFO readiness in order to avoid data corruption and to avoid compromising compressed data integrity.

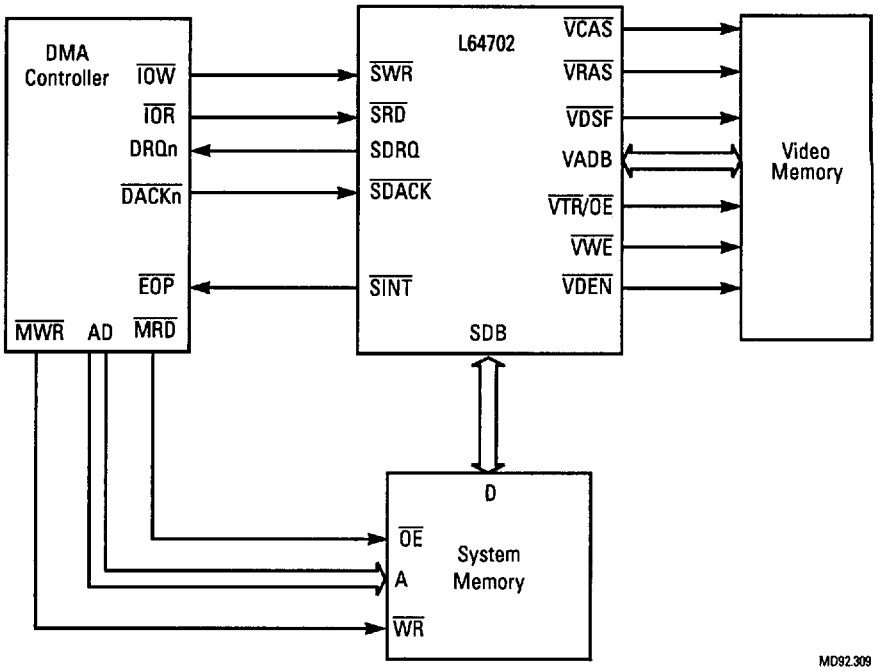
4.3 DMA Transfer Mode

The DMA transfer mode is very similar to the CPU transfer mode. The L64702 initiates DMA transfer mode when the DE bit in the SMR Register is set to one. The DMA transfer mode utilizes the capability of an external DMA controller device (such as an Intel 8237A) to directly transfer data in one memory cycle between system memory and the L64702. The DMA controller transfers data between the system memory and the L64702 by coordinating the read and write signals to both. DMA bus cycles ensure efficient utilization of the system bus during extensive data transfers between a peripheral device and the video memory.

A DMA transfer is performed when high utilization of the system bus is required. DMA transfer cycles allow blocks of data to be transferred directly between system memory and the L64702 without intervention by the CPU. The CPU need only set up the DMA controller registers and initiate the transfer. Better utilization of the system bus results because the CPU continues to perform internal program cycles while the external bus is being used for DMA transfers. The resulting parallel computation results in faster overall operation.

Figure 4.2 shows the interface between a DMA controller, the L64702, video memory, and system memory.

Figure 4.2
DMA Transfer Configuration



The DMA controller only needs four read and write control lines ($\overline{SRD}$, $\overline{SWR}$, SDRQ, and $\overline{SDACK}$) to simultaneously coordinate data transfers between the system memory and the L64702.

The L64702 acts as a slave peripheral on the system interface and uses an external DMA controller to implement DMA transfers. The L64702 allows DMA operation only with Groups 2 and 3. To initiate DMA transfer service, you must set the DE bit (HIGH) in the SMR Register. The SDRQ and $\overline{SDACK}$ pins handshake with an external DMA controller. The L64702 asserts SDRQ to request DMA transfer service. The DMA controller asserts $\overline{SDACK}$ to tell the L64702 that a DMA transfer is in progress. $\overline{SDACK}$ performs a function similar to that of the $\overline{SCS}$ pin in the CPU data transfer mode. Once the DMA service request has been initiated, any read or write cycles are valid cycles. As long as SDRQ and $\overline{SDACK}$ are asserted, the DMA operation continues.

There are two types of DMA transfers:

1. Group 2 registers and table transfers, where the DE and INIT bits in the SMR Register are set to one and the START and TRTH bits are cleared to zero.

2. Group 3 FIFO transfers, where:

Group 3 FIFO Operation	*Mode*	*INIT*	*ENC*	*START*	*TRTH*
Read	Compression	0	1	1	0
Read	Read-through	0	1	1	1
Write	Decompression	0	0	1	0
Write	Write-through	0	0	0	1

Refer to Chapter 10, "System Interface Port," for more details regarding DMA operation and timing parameters.

Initialization of Group 2 Tables and Registers Using DMA

In the initialization phase (DE = 1 and INIT = 1), the CPU initializes the L64702 tables and registers using DMA to perform Group 2 transfers. The L64702 asserts SDRQ and keeps it asserted as long as DE and INIT are set. Unlike the Group 3 DMA burst transfers, there are no restrictions on the number of words to transfer during a DMA cycle. It is important to remember that the CPU must initialize the APR Register prior to the DMA transfer just as in the CPU transfer mode, so that access is made to the appropriate memory location. Also, the APR increments by one on the rising edge of $\overline{\text{SRD}}$ or $\overline{\text{SWR}}$ to point to the next memory location.

Reading and Writing the Group 3 FIFO Using DMA

When data is transferred in a burst to or from the L64702 FIFO, the exact number of words as specified in the burst level should be transferred during each burst interval. The external DMA controller should not activate any additional DMA cycles after deactivation of the SDRQ signal. An improper DMA burst operation may corrupt the compressed data integrity.

The BURST field, located in the SMR Register, determines the number of words per transfer. The number of words in a burst can vary from 1 to 32. Refer to Section 3.5, "Group 3 FIFO," for more information.

The next two figures show the DMA request signals and protocol. The basis of the protocol is that the L64702 assumes that $\overline{\text{SDACK}}$ indicates DMA service to the L64702 I/O port, if the L64702 is in DMA transfer mode. Additionally, $\overline{\text{SDACK}}$ activity indicates that DMA service is in progress.

Note that the SDRQ signal is deasserted at the end of the burst count of words regardless of the internal FIFO condition, and it stays deasserted for a minimum of four clocks. The L64702 needs this period for internal

recovery. SDRQ is deasserted on the falling edge of $\overline{SRD}$ or $\overline{SWR}$ corresponding to the last codeword cycle in the burst transfer except during decompression when either an IMC, RI, or EOI condition occurs.

Figure 4.3 shows how SDRQ is deasserted at the last codeword transfer within the DMA cycle. In the figure, the number of burst words is 10 and SDRQ is deasserted at the beginning of $\overline{SRD}$ or $\overline{SWR}$ of the tenth word.

Figure 4.3
DMA Burst Transfer Timing

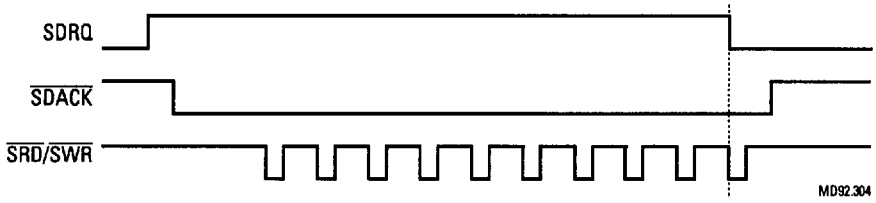
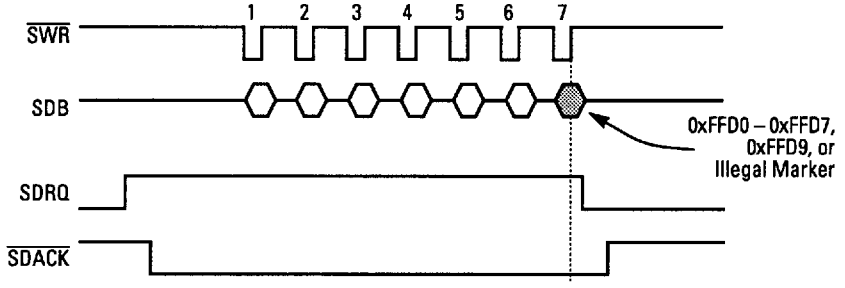


Figure 4.4 shows how SDRQ is deasserted after the occurrence of an EOI, RI, or IMC condition during decompression. In this example, the burst level is 16. Note that SDRQ is deasserted after the rising edge of $\overline{SWR}$ when SDB is an EOI, RI, or IMC condition even though the number of transfers has not reached 16.

Figure 4.4
DMA Burst Transfer Timing with EOI, RI, or IMC Condition

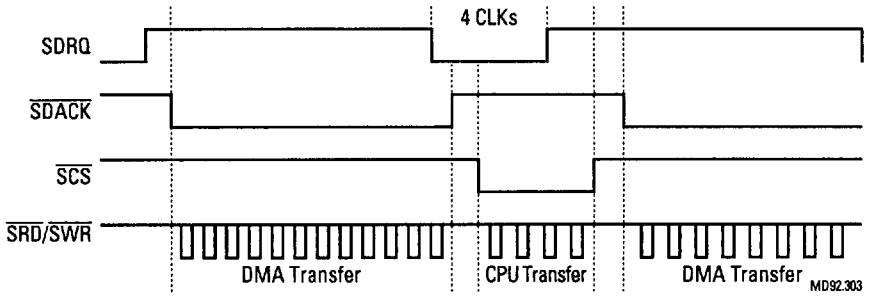


The minimum recovery time between consecutive DMA bursts is four clocks. The maximum delay time between two bursts is the recovery time (four clocks) plus the time it takes for the L64702 to process the amount of words in the FIFO. For example, during decompression assume the burst level is set to 32. After a burst transfer of 32 codewords into the FIFO, the L64702 takes four cycles to update the FIFO_CNT. At this point, SDRQ is deasserted. The L64702 reasserts SDRQ only when the THR condition is true, in other words, the FIFO is empty.

Figure 4.5 shows that CPU transfers or accesses can be done between DMA transfers, provided that $\overline{SCS}$ and $\overline{SDACK}$ are not asserted at the

same time. The CPU can access the L64702 in the pauses between DMA cycles, provided $\overline{\text{SDACK}}$ is not asserted, regardless of the state of the SDRQ signal.

Figure 4.5
DMA Transfer Protocol



Chapter 5
JPEG Processing Unit

The JPEG Processing Unit (JPU) performs image compression and decompression and resides between the L64702 system port and the L64702 video port.

This chapter contains the following sections:

- Section 5.1, JPU Overview
- Section 5.2, FDCT/IDCT Block
- Section 5.3, Quantizer/Inverse Quantizer Block
- Section 5.4, Zigzag Conversion
- Section 5.5, Differential/Inverse Differential Pulse Code Modulation
- Section 5.6, Variable-Length Coding and Decoding (VLC/VLD)

5.1 JPU Overview

The JPU communicates with the video port through the MCU buffer and communicates with the system port through the FIFO buffer. The JPU uses the system port to read or write compressed data and uses the video port to read or write uncompressed image data. The MCU buffers image data, which is organized in an 8 x 8 block order. The FIFO buffers the JPEG compressed data stream, which is packed into 16-bit words. Figure 5.1 shows the placement of the JPU in the L64702 data path and indicates the direction of the data during compression or decompression.

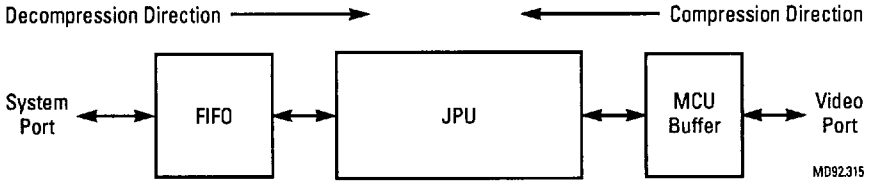


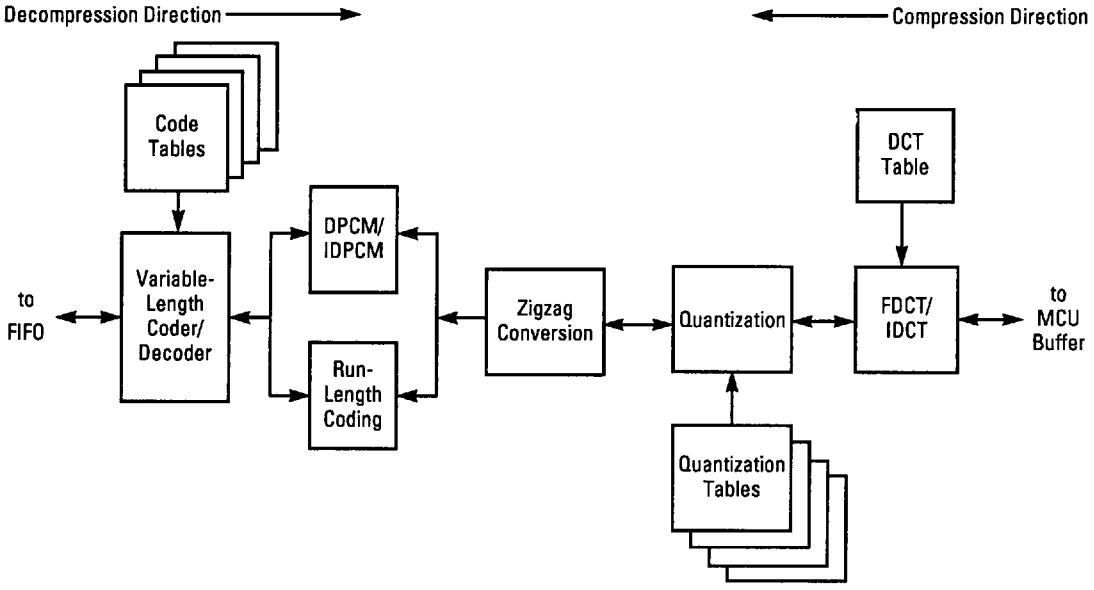
Figure 5.1 JPEG Image Processing Operations

On the video port side, the L64702 operates as a master device and controls the image data transfer rate based on the status of the MCU buffer. The

L64702 operates as a slave peripheral on the system port side. The system is responsible for controlling the data flow to or from the FIFO. The L64702 provides an external CPU or DMA controller with on-the-fly status information, which helps in determining the FIFO status.

Figure 5.2 shows the main components that comprise the JPU. The same functional blocks are used for both image compression and decompression. The L64702 is a half-duplex device, meaning it can perform in only one processing direction at a time.

Figure 5.2
JPEG Processing
Unit Block Diagram



When the START bit in the L64702 SMR Register is set, the L64702 enters the processing phase, where it performs image compression or decompression. The ENC bit in the SMR Register determines the processing direction. If ENC is set, the L64702 encodes (compresses) image data; if ENC is reset, the L64702 decodes (decompresses) image data.

During the processing phase, the JPU may momentarily suspend operation, depending on the full or empty status of the MCU or FIFO buffers. The L64702 performs the suspension automatically, without external system intervention. To force the L64702 to suspend operation, an external

CPU can reset the START bit. Once the CPU sets the START bit back to a one, the L64702 resumes operation from its last state.

While operation is suspended by the external system, a CPU can read from and write to the internal memory-mapped resources. The L64702 assumes that any write operations to Group 2 are attempts to change the configuration parameters. At this point it is likely that the internal data stream is corrupted, and the L64702 must be reset and operation must commence from the beginning.

5.2 FDCT/IDCT Block

The FDCT/IDCT (Forward Discrete Cosine Transform/Inverse Discrete Cosine Transform) block transforms image data between the time domain and the frequency domain. In video compression, the FDCT function is implemented first, followed by quantization. In video decompression, inverse quantization occurs first, followed by the IDCT function.

During compression, the image data is transferred from the MCU buffer to the JPU FDCT block and is assumed to be an 8 x 8 block of eight-bit, signed (-128 to 127) data values. The FDCT operation transforms each 8 x 8 block and yields a set of 11-bit, signed values (-1024 to 1023) known as DCT coefficients. One of these values is the DC coefficient, and the other 63 values are the AC coefficients.

Equation 5.1 and Equation 5.2 show the mathematical equations for the FDCT and IDCT transforms performed by the L64702.

Equation 5.1 Forward Discrete Cosine Transforms

$$F_{v,u} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f_{y,x} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \quad \text{(FDCT)}$$

Equation 5.2 Inverse Discrete Cosine Transforms

$$f_{y,x} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v F_{v,u} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \quad \text{(IDCT)}$$

where:

$$C_u, C_v = \frac{1}{\sqrt{2}} \text{ for } u, v = 0 \qquad C_u, C_v = 1 \text{ otherwise}$$

The L64702 contains a table of 64 DCT values, which specify the 12-bit, signed FDCT or IDCT factors to be used in implementing Equation 5.1 and Equation 5.2. Figure 5.3 shows the FDCT and IDCT tables.

Each table shown in Figure 5.3 contains 64 default factors. The tables correspond to the 8 x 8 raster ordered data as follows. The first eight entries (1 through 8) correspond to the top row of the 8 x 8 block, reading from left to right. The next eight entries in the table (9 through 16) correspond to the second row from the top of the 8 x 8 block, and so on. The FDCT or IDCT transform operates on the data byte found in a particular location in any 8 x 8 data block, depending on the processing direction, using the factor that corresponds to the location of the data byte in the 8 x 8 block.

You can load custom FDCT and IDCT factors to modify the transform according to the particular application.

The L64702 calculates the DCT or IDCT coefficients in two separate matrix operations as shown in Equation 5.3 and Equation 5.4, as follows:

Equation 5.3 First Step in Calculation

$$[\bar{Y}] = [A] \bullet [X]$$

Equation 5.4 Second Step in Calculation

$$[Y] = [A] \bullet [\bar{Y}]^T$$

where:

[A] is the 8 x 8 DCT or IDCT matrix.
[X] is the 8 x 8 input data matrix to the FDCT/IDCT L64702 block.
[$\bar{Y}$] is the result of the first operation.
[Y] is the final result.

Figure 5.3
FDCT and IDCT Factors

$$\begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} & -\cos\frac{7\pi}{16} & -\cos\frac{5\pi}{16} & -\cos\frac{3\pi}{16} & -\cos\frac{\pi}{16} \\
\cos\frac{2\pi}{16} & \cos\frac{6\pi}{16} & \cos\frac{10\pi}{16} & \cos\frac{14\pi}{16} & \cos\frac{14\pi}{16} & \cos\frac{10\pi}{16} & \cos\frac{6\pi}{16} & \cos\frac{2\pi}{16} \\
\cos\frac{3\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{21\pi}{16} & -\cos\frac{21\pi}{16} & -\cos\frac{15\pi}{16} & -\cos\frac{9\pi}{16} & -\cos\frac{3\pi}{16} \\
\cos\frac{4\pi}{16} & \cos\frac{12\pi}{16} & \cos\frac{20\pi}{16} & \cos\frac{28\pi}{16} & \cos\frac{28\pi}{16} & \cos\frac{20\pi}{16} & \cos\frac{12\pi}{16} & \cos\frac{4\pi}{16} \\
\cos\frac{5\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{25\pi}{16} & \cos\frac{35\pi}{16} & -\cos\frac{35\pi}{16} & -\cos\frac{25\pi}{16} & -\cos\frac{15\pi}{16} & -\cos\frac{5\pi}{16} \\
\cos\frac{6\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{30\pi}{16} & \cos\frac{42\pi}{16} & \cos\frac{42\pi}{16} & \cos\frac{30\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{6\pi}{16} \\
\cos\frac{7\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{49\pi}{16} & -\cos\frac{49\pi}{16} & -\cos\frac{35\pi}{16} & -\cos\frac{21\pi}{16} & -\cos\frac{7\pi}{16}
\end{bmatrix}$$

FDCT Factors

$$\begin{bmatrix}
\frac{1}{\sqrt{2}} & \cos\frac{\pi}{16} & \cos\frac{2\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{4\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{6\pi}{16} & \cos\frac{7\pi}{16} \\
\frac{1}{\sqrt{2}} & \cos\frac{3\pi}{16} & \cos\frac{6\pi}{16} & \cos\frac{9\pi}{16} & \cos\frac{12\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{18\pi}{16} & \cos\frac{21\pi}{16} \\
\frac{1}{\sqrt{2}} & \cos\frac{5\pi}{16} & \cos\frac{10\pi}{16} & \cos\frac{15\pi}{16} & \cos\frac{20\pi}{16} & \cos\frac{25\pi}{16} & \cos\frac{30\pi}{16} & \cos\frac{35\pi}{16} \\
\frac{1}{\sqrt{2}} & \cos\frac{7\pi}{16} & \cos\frac{14\pi}{16} & \cos\frac{21\pi}{16} & \cos\frac{28\pi}{16} & \cos\frac{35\pi}{16} & \cos\frac{42\pi}{16} & \cos\frac{49\pi}{16} \\
\frac{1}{\sqrt{2}} & -\cos\frac{7\pi}{16} & \cos\frac{14\pi}{16} & -\cos\frac{21\pi}{16} & \cos\frac{28\pi}{16} & -\cos\frac{35\pi}{16} & \cos\frac{42\pi}{16} & -\cos\frac{49\pi}{16} \\
\frac{1}{\sqrt{2}} & -\cos\frac{5\pi}{16} & \cos\frac{10\pi}{16} & -\cos\frac{15\pi}{16} & \cos\frac{20\pi}{16} & -\cos\frac{25\pi}{16} & \cos\frac{30\pi}{16} & -\cos\frac{35\pi}{16} \\
\frac{1}{\sqrt{2}} & -\cos\frac{3\pi}{16} & \cos\frac{6\pi}{16} & -\cos\frac{9\pi}{16} & \cos\frac{12\pi}{16} & -\cos\frac{15\pi}{16} & \cos\frac{18\pi}{16} & -\cos\frac{21\pi}{16} \\
\frac{1}{\sqrt{2}} & -\cos\frac{\pi}{16} & \cos\frac{2\pi}{16} & -\cos\frac{3\pi}{16} & \cos\frac{4\pi}{16} & -\cos\frac{5\pi}{16} & \cos\frac{6\pi}{16} & -\cos\frac{7\pi}{16}
\end{bmatrix}$$

IDCT Factors

5.3 Quantizer/ Inverse Quantizer Block

During compression, the FCDT operation is followed by the quantizer function. The quantizer block quantizes each of the 64 DCT coefficients resulting from the transform performed by the FDCT operation. During decompression, the dequantizer function is first, followed by the IDCT function. The dequantizer recreates the 64 DCT coefficients, which are then operated on by the IDCT transform.

The quantization table contains the corresponding quantization step size for each coefficient. The step size is designated by $Q_{v,u}$. A single quantization table is used for both quantization and inverse quantization.The first formula of Equation 5.5 is for quantization; the second formula for inverse quantization.

The L64702 contains four user-programmable quantization tables (QT0 through QT3). Each 8 x 8 FDCT block can be quantized with one of the four tables. The Q3 through Q0 fields in the JCR Register select the quantization tables for each block. Refer to Chapter 3, "Internal Registers and Data Tables," for more information on the quantization tables and the JCR Register.

Equation 5.5 shows the functions implemented during quantization and inverse quantization. The upper equation shows that the FDCT results, $F_{v,u}$, are divided by the corresponding quantization factor $Q_{v,u}$, and rounded to the nearest value. This operation, by definition, is lossy. The lower equation shows the inverse quantization operation. Quantized FDCT values, $FQ_{v,u}$, are multiplied by the corresponding quantization factor $Q_{v,u}$. $R_{v,u}$ is not equivalent to $F_{v,u}$ because of arithmetic loss incurred during quantization.

Equation 5.5 Quantization and Inverse Quantization

$$FQ_{v,u} = round\left(\frac{F_{v,u}}{Q_{v,u}}\right)$$

$$R_{v,u} = FQ_{v,u} \times Q_{v,u}$$

Figure 5.4 shows two quantization tables (one for luminance data, one for chrominance data) taken from the JPEG standard draft. While you can load any values into the quantization tables, these standard quantization tables are recommended to accomplish a 24:1 compression ratio for typical images.

Figure 5.4 Luminance and Chrominance Quantization Tables

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Luminance

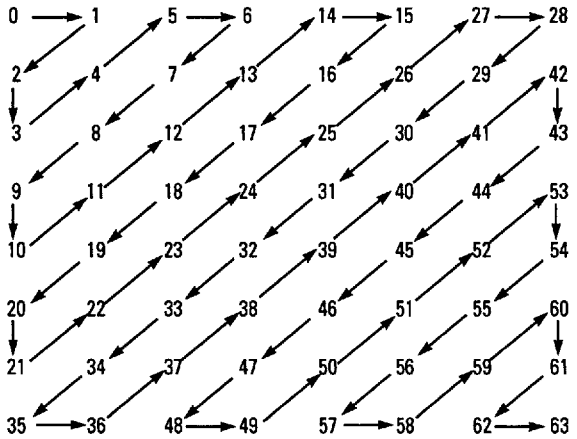
17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Chrominance MD92.317

5.4 Zigzag Conversion

The zigzag conversion block converts quantized AC coefficients from raster order to zigzag order in the compression direction, and from zigzag order to raster order in the decompression direction. Zigzag ordering places the AC coefficients in ascending frequency order, which substantially improves coding efficiency. In practice, long runs of zeroes are produced, which are coded very efficiently using the Huffman coding technique. Figure 5.5 shows the zigzag conversion table. The quantized DC coefficient (location 0 in the array) is treated separately. All the other locations are AC coefficients.

Figure 5.5 Zigzag Sequence of Quantized DCT Coefficients



5.5 Differential/ Inverse Differential Pulse Code Modulation

The Differential and Inverse Differential Pulse Code Modulation (DPCM/IDPCM) block decodes DC coefficients during decompression and codes DC coefficients during compression. Quantized DC coefficients are processed differently from the 63 AC coefficients. A differential pulse code modulation scheme is used in place of zigzag coding. The DC term of each 8 x 8 block is subtracted from the DC term of the previous block for that color component. The L64702 contains four DC predictors to hold the temporary differences for each of the color components. The DC predictors are reset at the beginning of operation or upon detection of a restart marker code during decompression.

Equation 5.6 is implemented during compression; Equation 5.7 is implemented during decompression. PR_i represents the DC predictor and DC_i represents the quantized DC term.

Equation 5.6 DC Difference

$$\Delta DC_i = DC_i - PR_{i-1}$$

$$PR_i = DC_i \qquad PR_{-1} = 0$$

Equation 5.7 DC Term Reconstruction

$$DC = \Delta DC_i + PR_{i-1}$$

$$PR_i = DC_i \qquad PR_{-1} = 0$$

5.6 Variable-Length Coding and Decoding (VLC/ VLD)

The Variable-Length Coder (VLC) generates variable codewords to be placed in the FIFO from the quantized DCT coefficients. The Variable-Length Decoder (VLD) produces quantized DCT coefficients from codewords found in the FIFO. The VLC process assigns the shortest codewords to those events that statistically have been determined to occur most frequently, so maximum compression is accomplished for the image data type to be processed. This process is known as *entropy encoding*.

The L64702 contains four internal Huffman code tables: two DC Code Tables (0 and 1), and two AC Code Tables (0 and 1). The entire code table must be written at one time in order to ensure proper operation. The number of words per table is defined in Chapter 3, "Internal Registers and Data Tables."

The code-table data format has been optimized for coding efficiency and differs from the JPEG interchange data format. LSI Logic supplies code-table formatting software that reformats JPEG-formatted code tables into L64702-compatible format.

Chapter 6
Compressed Image Data Structure

This chapter describes the data file structure for both compressed and decompressed images. The L64702 supports image compression or decompression from the scan level, as specified in the proposed JPEG standard CD 10918-1 (baseline only), with one exception—the L64702 does not process the scan header information. In general, the L64702 does not process any header information specified in the JPEG standard. The L64702 assumes that a general-purpose CPU processes the header information.

The L64702 is responsible for the computational intensive task of image encoding or decoding. The L64702 supports byte stuffing during compression and byte stripping during decompression. Insertion of the EOI marker is also supported. In addition, the Restart and End of Image (EOI) marker codes are supported during decompression.

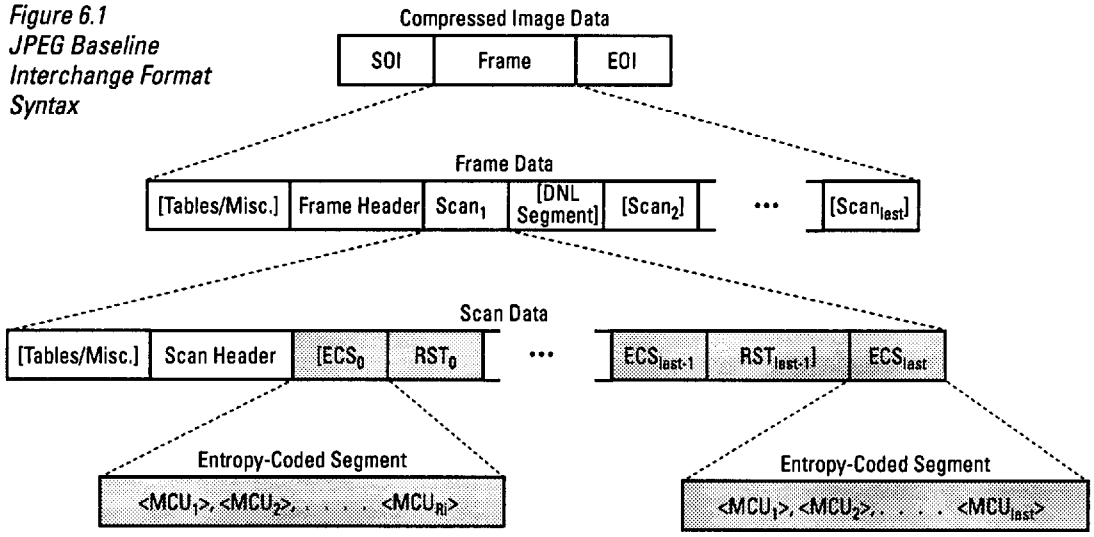
This chapter describes the following JPEG data file structures:

- Section 6.1, JPEG Interchange Format
- Section 6.2, Compressed Data Segments
- Section 6.3, Decompression Code Data Structure

6.1 JPEG Interchange Format

Figure 6.1 shows the JPEG interchange format layers. The top level defines the compressed image structure, starting with the start of image (SOI) marker, followed by the frame data, and ending with the end of image (EOI) marker.

Figure 6.1
JPEG Baseline Interchange Format Syntax



Frame Data

The second level of Figure 6.1 shows the frame data format. Frame data begins with a frame header and contains one or more scan fields. A frame header may be preceded by one or more table specification or miscellaneous marker segments. If a Define Number of Lines (DNL) marker is present, it immediately follows the first scan. Each scan contains from one to four image components. If two to four image components are contained in a scan, they are interleaved within the scan.

Scan Data

The scan field begins with a scan header and contains one or more entropy-encoded segments (ECS). Each scan header may be preceded by one or more table-specification or miscellaneous marker segments. If restart is not enabled, there is only one entropy-encoded segment (labeled *last*), and no restart markers (RST) are present. If restart is enabled, the number of entropy-coded segments is defined by the size of the image and the defined restart interval. In this case, a restart marker follows each ECS except the last one.

Entropy-Coded Segment (ECS)

Each ECS is comprised of a sequence of entropy-encoded MCUs. The VLC process mentioned in Chapter 5, "JPEG Processing Unit," assigns the shortest codewords to those events that statistically have been determined

to occur most frequently, so maximum compression is accomplished for the image data type to be processed. This process is known as *entropy encoding*. If restart is enabled and restart is defined to be R_i, each entropy-encoded segment except the last one contains R_i MCUs. The last ECS contains whatever number of MCUs completes the scan.

6.2 Compressed Data Segments

The L64702 compressed data segment containing the entropy-encoded output data from the variable-length coder is stored in the FIFO. Data are packed into 16-bit words in the FIFO. Because the code segment may end on an arbitrary bit within the 16-bit word, the L64702 fills the last 16-bit word of the code segment with ones. Any JPEG-compatible information tailored to a specific application can be appended to the code segment.

When compressing an image, the L64702 produces a JPEG compressed file that consists of pure JPEG entropy-coded words. The L64702 does not insert any restart or other markers during compression except the EOI marker, provided the EOI bit is set in the SCR Register.

The JPEG standard requires that an 0xFF byte precede the marker codes to denote their presence within the coded bit stream. A valid marker code is one that follows an 0xFF but is not equal to 0xFF or 0x00. Therefore an 0xFF that is naturally created during compression must be followed by a 0x00. This scheme allows a naturally occurring 0xFF in the coded bit stream to be distinguished from a marker code following an 0xFF. The L64702 process of inserting a 0x00 byte after a naturally occurring 0xFF byte is known as *byte stuffing*.

Figure 6.2 shows four examples of code data streams created by the L64702. The "0" and "1" represent the least-significant and most-significant bytes, respectively, of the 16-bit FIFO words. Cn represents the code data words that the L64702 created during compression. The 0x00 bytes that follow the 0xFF bytes represent those bytes that the L64702 has *byte stuffed*.

Additionally, the JPEG standard specifies that any marker may optionally be preceded by any number of fill bytes, which are bytes assigned to the value 0xFF.

The MSB of a byte (bit 7) is the first bit of the compressed data stream for any particular byte.

Figure 6.2
L64702-created
Code Data Streams

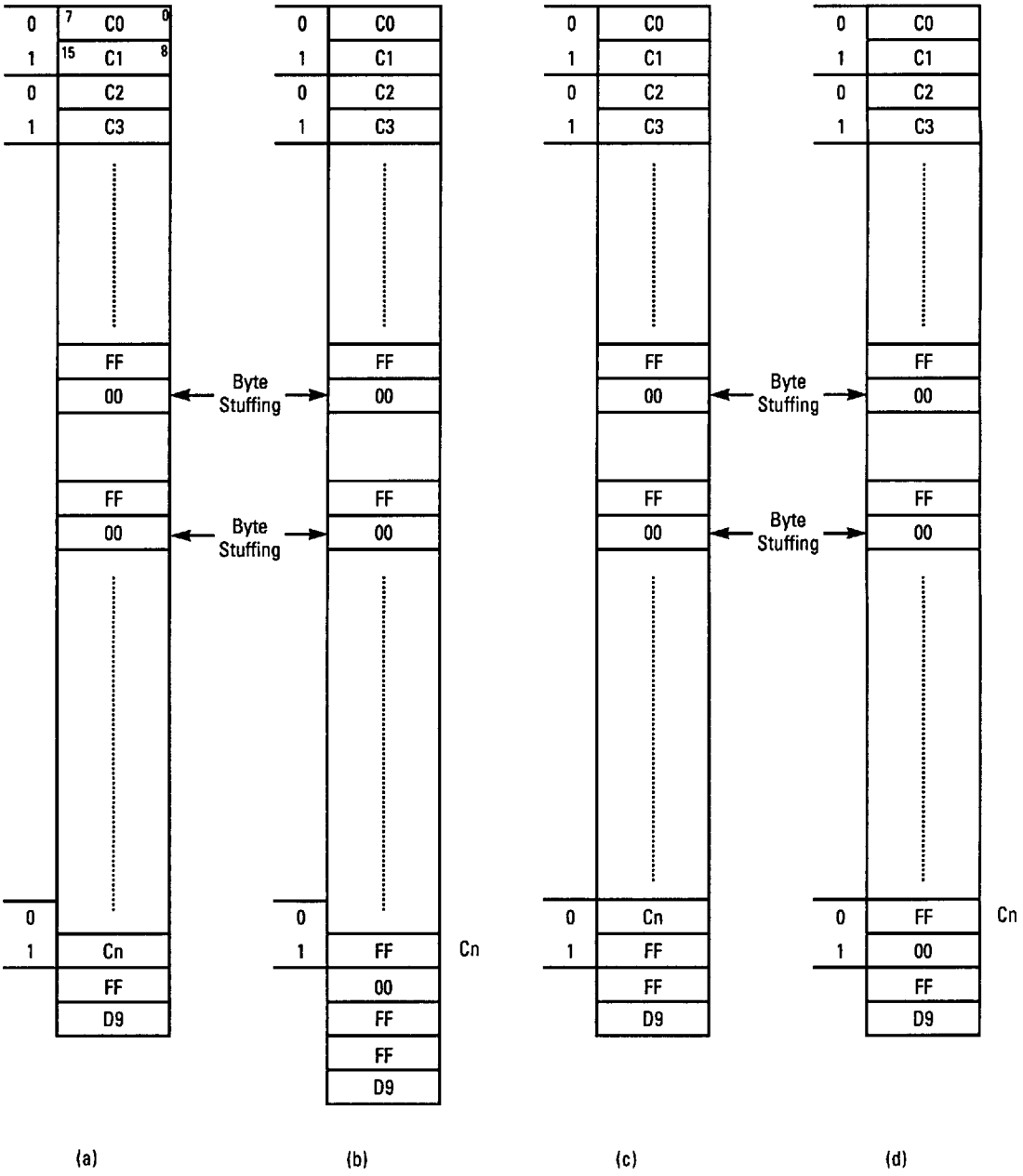


Figure 6.2a is an example of a code data stream that ended on a bit in the least-significant byte.

Figure 6.2b is an example of a code data stream that ended on a bit in the least-significant byte originally or as a result of padding ones. A naturally occurring 0xFF was created in the code data stream. As a result, a 0x00 was appended by byte stuffing. In order to end the data stream on the 16-bit boundary, a 0xFF was appended to the 0x00.

If padding with ones creates a 0xFF, a 0x00 byte is inserted as described above. The L64702 may pad the last byte with ones if the total code segment, including the stuffed zeroes that follow a naturally occurring 0xFF, does not end on a 16-bit word boundary.

Figure 6.2c is an example of a code data stream that ended on a bit in the most-significant byte. A 0xFF was appended to fill data to the 16-bit boundary.

Figure 6.2d is an example of a code data stream that ended on a bit in the most-significant byte originally or as a result of padding ones. A 0xFF was created in the code data stream and a 0x00 was appended. Because the code data ends on a 16-bit boundary, additional bytes are not appended.

6.3 Decompression Code Data Structure

When The L64702 decompresses an image, the data supplied to it must conform to the JPEG baseline standard (CD 10918-1). An external CPU must remove all header information from the code data stream before writing it into the FIFO.

The L64702 decompresses code segments that may contain restart marker codes (RST_m). An RST marker code consists of a 0xFF followed by a byte in the range 0xD0 through 0xD7. The L64702 also decompresses code segments that contain an end of image marker code (EOI). The EOI marker code consists of a 0xFF followed by a 0xD9.

During decompression, if a 0x00 is detected following a 0xFF, the L64702 decoder ignores the 0x00 byte and processes the 0xFF byte as a legal code byte. This process is known as *byte stripping*. If the byte following a 0xFF is non-zero, then this byte is assumed to be a marker code and is processed as described in the following subsections.

Reset Marker Code (RST_m)

If the L64702 detects a restart marker code (0xFF followed by 0xD0 - 0xD7) during decompression, the entire 16-bit word is ignored. The L64702 notifies the external system that it has detected a RST marker by asserting the RI bit in the STS Register. An interrupt may be generated if the corresponding bit in the SMR is unmasked.

If the L64702 is in a DMA transfer cycle when the RST marker is detected, it deasserts SDRQ on the rising edge of $\overline{SCS}$ or $\overline{SWR}$ and enters its internal reset sequence. At the end of the reset sequence, it resets the four DC predictors that temporarily maintain the differences between DC coefficients of sequential 8 x 8 data blocks and becomes ready to continue decoding the next segment of the image. If in DMA transfer mode the L64702 sets SDRQ.

End of Image Marker Code (EOI)

The EOI marker (0xFF, 0xD9) is inserted at the end of the compressed data. If the L64702 detects an EOI marker code, it assumes that the end of an image has been reached. The L64702 decoder ignores the EOI marker code and asserts the EOI flag in the STS Register to notify the external system. An interrupt may be generated if the corresponding bit in the SMR is unmasked.

With EOI flagged, the L64702 ignores any write attempts by the CPU or DMA controller. An end of active window (EOA) condition is reached after the preceding code data has been decoded and the reconstructed image data transferred. Only the EOA flag is set. If the defined active image is outside the compressed image, the EOA condition does not appear.

To restart operation, you should initiate a frame reset, then set the START bit again. If DMA service is required the DE bit should be set.

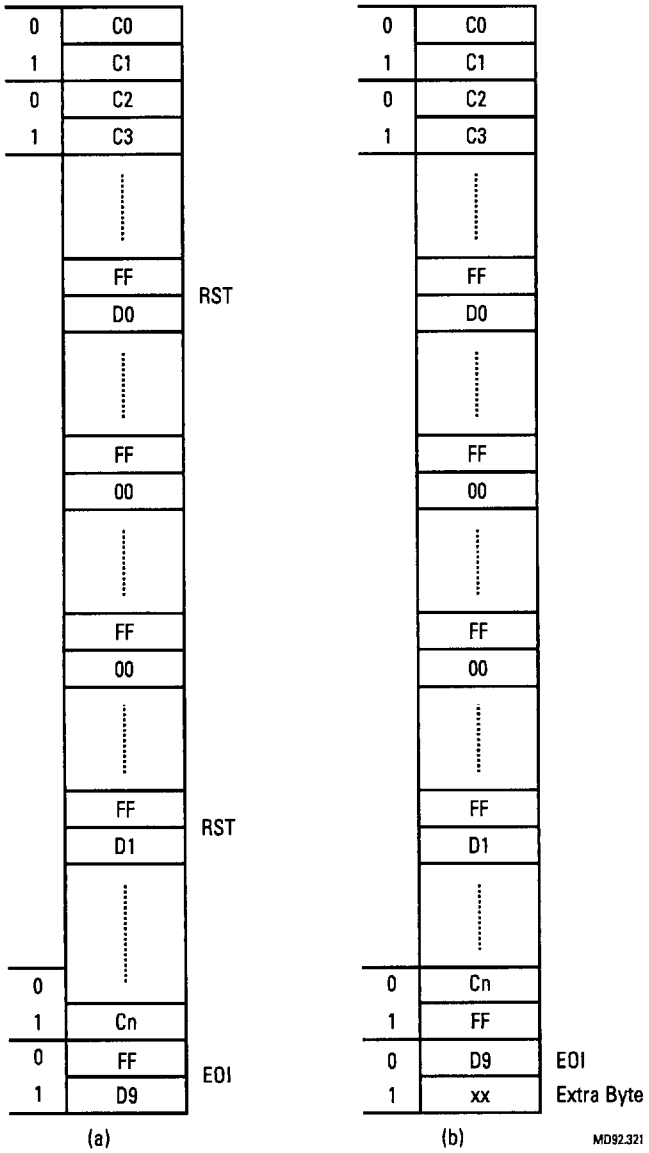
Other Marker Codes

The L64702 decoder ignores any other marker codes and, upon detection, asserts the illegal marker code (IMC) bit in the STS Register to notify the external system. The L64702 generates an interrupt if the corresponding bit in the SMR is not masked. The user must perform a frame reset in order to restart operation.

Figure 6.3 shows compressed file structures supplied to the L64702. The compressed data must be word-aligned on 16-bit boundaries.

Byte 0 represents the least-significant half of the 16-bit word, and byte 1 represents the most-significant half. In Figure 6.3a, the code data end on a word boundary. If the code data end on a byte boundary instead of a word boundary (Figure 6.3b), then a dummy byte (0xXX) must be inserted before the compressed data of the next image.

Figure 6.3 Compressed Data Structures Supplied to L64702



Chapter 7
Video Memory Organization

This chapter describes the organization of image data in external video memory used in conjunction with the L64702 and the associated video bus addressing scheme.

This chapter contains six sections:

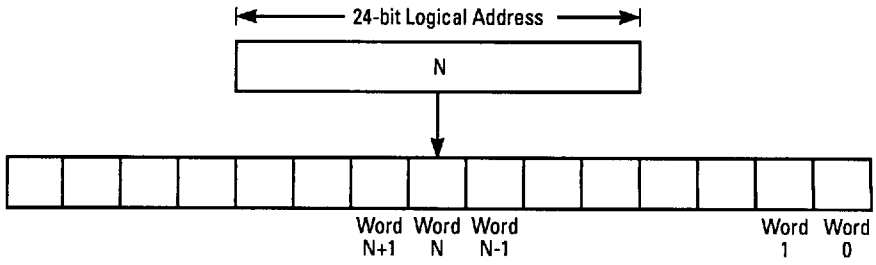
- Section 7.1, Video Memory Address Space
- Section 7.2, Raster-to-Block Conversion
- Section 7.3, Color Memory Organization
- Section 7.4, Image Subsampling
- Section 7.5, Scrolling in Compressed Files
- Section 7.6, Active Window and Two-Dimensional Addressing

7.1 Video Memory Address Space

In every video memory access (read or write), the L64702 transfers a complete 32-bit word between the external video memory and the internal MCU buffer. The 32-bit L64702 video bus is multiplexed between address and data. Twenty-four address bits directly access up to 64 Mbytes or 16M, 32-bit words.

Figure 7.1 shows the L64702 logical memory address space. Each word is 32 bits wide. Address values, N, range between 0x00_0000 and 0xFF_FFFF.

Figure 7.1
Logical Memory
Address Space

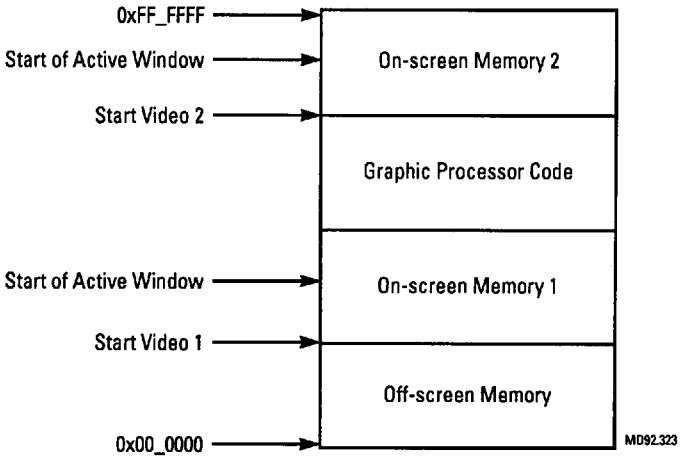


Note:
1. All words are 32 bits wide.

Figure 7.2 shows the physical memory mapping of video memory. Within the 24-bit address space of the L64702, video memory can be partitioned into several memory banks that can be used for different tasks. The L64702 may share the video memory with a graphics or DSP processor, which uses the video memory to store machine code and hold temporary data. In addition, display memory, defined as on-screen memory, can be used for image display. The programmability of the L64702 provides the flexibility of switching tasks in the memory address space.

When you initialize the L64702, you may configure the video control registers. These registers determine the start address of the active window area, the active window dimensions, the display pitch, and other video control functions. Refer to Chapter 3, "Internal Registers and Data Tables," for more details on the video control registers.

Figure 7.2
Physical Memory
Organization



7.2 Raster-to-Block Conversion

Raster-to-block conversion is a special-purpose operation, which is associated with the JPEG compression algorithm. JPEG compression is performed in such a way that pixels must be organized in 8 x 8 two-dimensional tiles. If the image exists initially in raster order, the front-end video interface unit should convert from raster order to block order. The L64702 integrates the raster-to-block conversion in its video address unit. This integration changes the definition of AW and AH from simple straightforward Active Width and Active Height parameters to a video organization in terms of Minimum Code Units (MCUs) in the horizontal and vertical directions.

An MCU is typically a two-dimensional area of image data containing 8 x 8 blocks for processing by the JPEG unit. The term *minimum* in MCU means that this MCU contains the smallest video structure for a particular sampling ratio, and that the entire image is constructed from an array of these MCUs.

The video address unit also notes when the actual image boundary does not correspond to an integral number of MCUs. If this is the case, the L64702 complements the image area with zeros in the compression direction. In the decompression direction, the area outside the active image area is not overwritten.

7.3 Color Memory Organization

This section describes the four different methods the L64702 uses to access color image data structures. These methods are listed below:

- Multisource
- Unpacked
- Packed
- Components sequential

The IDS field in the SCR Register determines which data structure is accessed. Table 7.1 shows the encoding of the IDS field.

Table 7.1
IDS Field Encoding

IDS	*Data Structure*
00[1]	Multisource Color Image
01[1]	Unpacked Color Image
10[1]	Packed Color Image
11	Components Sequential Color Image

1. Only sampling ratios of 1:0:0, 1:1:1, and 2:1:1 can be used.

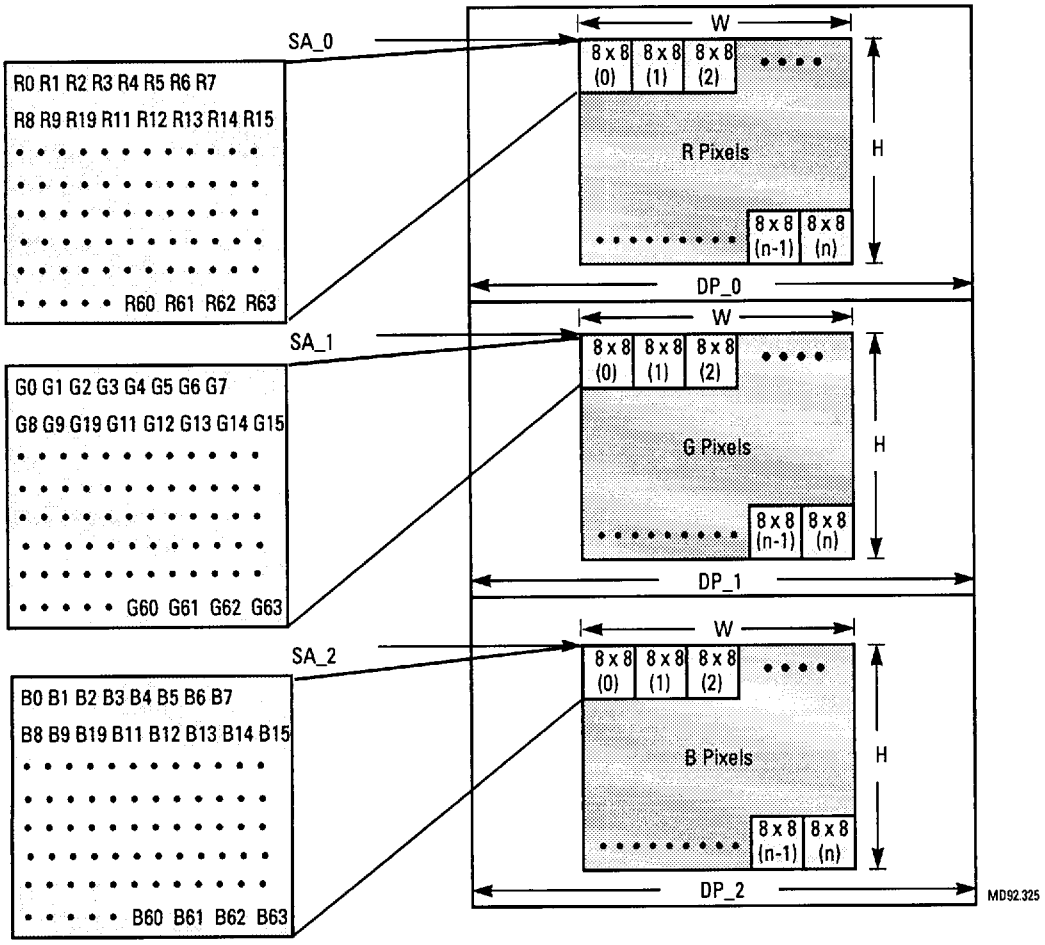
Each of the following four subsections provides a two-dimensional memory organization, describes the color addressing scheme, and describes the video bus activity for each data structure.

Multisource Color Memory Organization

Multisource color image data structures are accessed when the IDS field equals zero. In this configuration, the L64702 accesses three color components (RGB) from three separate memory regions. Each region is reconstructed from an array of pixels that are packed into a group of four pixels per address location. The total region size (in pixels) per color component is determined by multiplying the display pitch by the number of display lines allocated for the color component. The display pitch is specified per color component in the three display pitch registers: DP_0, DP_1, and DP_2. The active window region is the specified area of operation within the color region and is specified by three parameters: start address (SA_0, SA_1, and SA_2), Active Width (AW), and Active Height (AH).

Figure 7.3 shows the two-dimensional memory organization of an RGB multisource color image data structure. The figure is representative of an image that consists of data in three separate video memory regions according to color. The data in each region are organized as blocks eight bytes wide by eight lines high.

Figure 7.3
Multisource Color Memory Organization



As mentioned previously, the video bus address generator performs the raster-to-block and/or block-to-raster operations when transferring data from video memory to the L64702 or from the L64702 to video memory, respectively. Image data are transferred to or from the L64702 in 8 x 8 block order. The address generated for video memory is such that it is presented to a display device in raster order.

Every video memory cycle involves 32-bit transfers per cycle (four pixels). Therefore every active window parameter must be an integer multiple of four (4, 8, 12, 16, etc.). The starting address must also be aligned with a 32-bit boundary. You must specify the proper values in the starting address

registers as well as the width and height registers to prevent any overrides. Any mix of data between color regions results in image data corruption.

The addressing scheme implemented for this mode consists of two 32-bit memory transfer cycles for each color, which results in the transfer of eight pixels of each color component from consecutive addresses within that color's region. The bus cycles are interleaved as shown in Figure 7.4.

Figure 7.4 Multisource Color Video Bus Activity

Bus Cycle	32-bit 31 24	23 16	15 8	7 0
0	R3	R2	R1	R0
1	G3	G2	G1	G0
2	B3	B2	B1	B0
3	R7	R6	R5	R4
4	G7	G6	G5	G4
5	B7	B6	B5	B4
6	R11	R10	R9	R8
7	G11	G10	G9	G8
8	B11	B10	B9	B8
9	R15	R14	R13	R12
10	G15	G14	G13	G12
11	B15	B14	B13	B12
12	R19	R18	R17	R16
13	G19	G18	G17	G16
14	B19	B18	B17	B16
15	R23	R22	R21	R20
•	•	•	•	•
•	•	•	•	•

The first bus cycle transfers 32 bits of R; the second, 32 bits of G; the third, 32 bits of B. After two memory cycles have executed for each color component, eight complete pixels have been transferred for the three color components. The video bus address generator then skips to an address for the next R color component, which is the starting address plus the amount specified in the display pitch register. The new address points to the starting pixel for a particular 8 x 8 block on the next display line.

At this point, eight more pixels are transferred, and again, the video bus address generator skips ahead to point to the starting pixel for the block on a new line. Sets of eight pixels are transferred until an 8 x 8 block (64 pixels) transfer is complete. This operation is performed in an interleaved manner for all three color regions, as shown in Figure 7.4. The operation continues until all the 8 x 8 blocks are transferred in the order shown in Figure 7.3.

As shown in Figure 7.4, six memory cycles of 32 bits are required to transfer eight pixel components of a particular color. The value of the first complete pixel (R0, G0, B0) is contained in bits [7:0], the second (R1, G1, B1) in bits [15:8], the third (R2, G2, B2) in bits [23:16], and the fourth (R3, G3, B3) in bits [31:24]. This same process continues for pixels five through eight.

Figure 7.5 shows the raster-to-block addressing scheme as implemented to support the multisource color image data structure.

Figure 7.5
Multisource Color Addressing Scheme

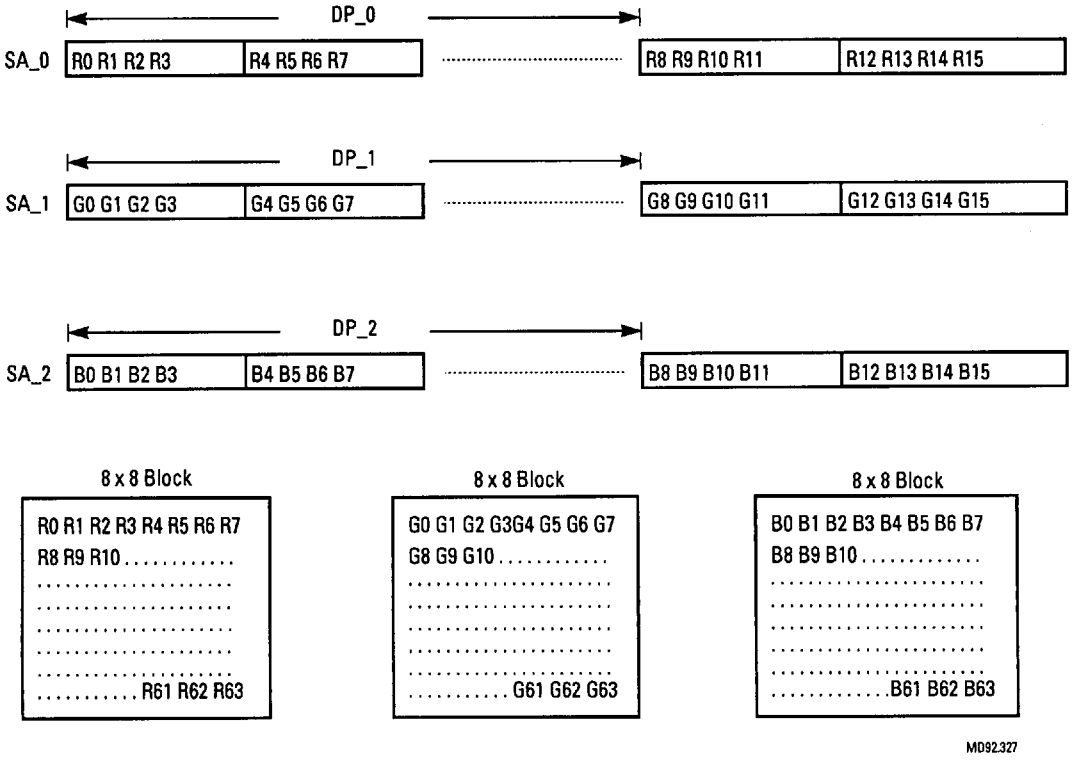
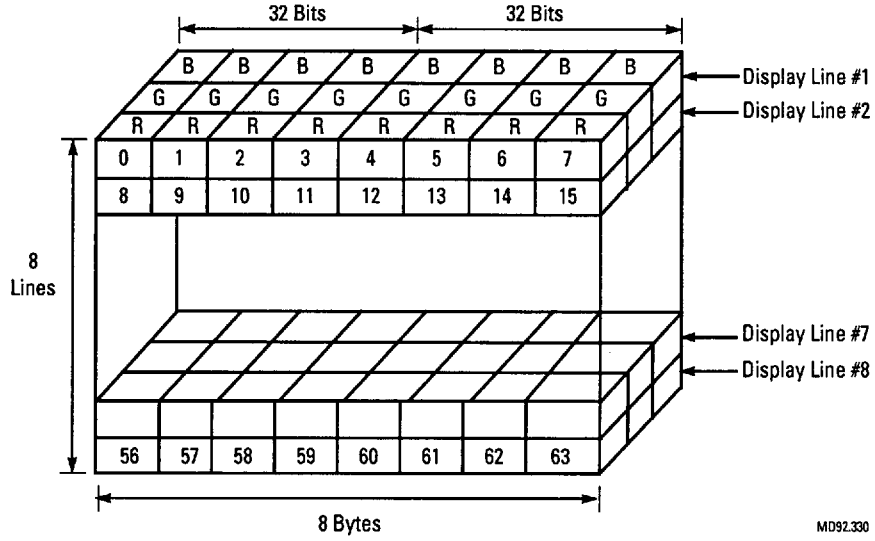


Image data are transferred in 32-bit words (four pixels) per memory cycle. Because the L64702 can execute video bus cycles faster than it can perform color space conversion on the image data, from time to time it must halt bus cycles in order to do conversions. Therefore on average, the L64702 transfers a word from a color region every other cycle.

Figure 7.6 is a three-dimensional representation of the display image in video memory. Complete three-color pixels are generated by accessing a

32-bit word from each color region of memory. Each access to a particular color region yields four values of the same color that are part of four distinct pixels. After two accesses to each color region, enough information is available to create one of the layers of the three-dimensional 8 x 8 block depicted in the figure. The subsequent two accesses to each region result in another layer that belongs to the next display line.

Figure 7.6 Three-dimensional 8 x 8 Multisource Display Image



Unpacked Color Memory Organization

Unpacked color image data structures are accessed when the IDS field is set to one. Three color components are assumed to be interleaved and to reside in the same memory region. Because each color pixel is comprised of three eight-bit components, pixel data are right-justified to the 24 least-significant bits (bits 0 to 23) of a 32-bit word. The eight most-significant bits (bits 24 to 31) are ignored in this configuration and are designated by "X."

Figure 7.7 shows the two-dimensional memory organization of an unpacked image data structure. The figure is representative of a image that consists of data in video memory organized as blocks eight bytes wide by eight bytes high.

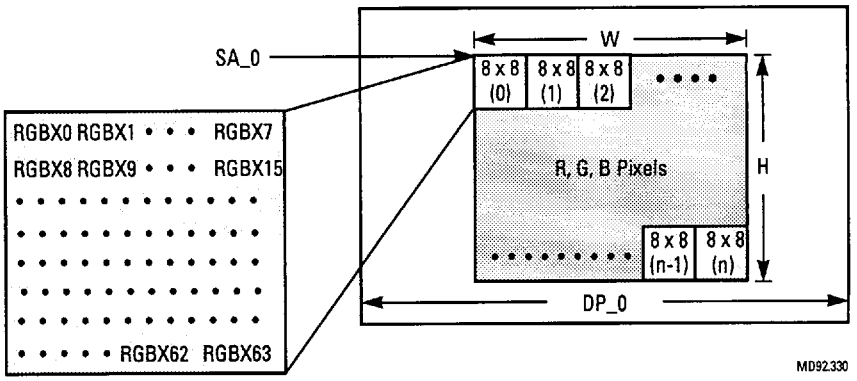


Figure 7.7 Unpacked Color Memory Organization

Data are transferred to and from the L64702 in this 8 x 8 block format. The highlighted block indicates the active window structure. The starting address of the active window is in the SA_0 Register, the active window width is in the W Register, and the active window height is in the H Register. The total region size (in pixels) is calculated by multiplying the specified display pitch by the number of display lines that are allocated for the display area. The display pitch is contained in the DP_0 Register.

The video bus address generator performs the raster-to-block and block-to-raster translation. Image data are transferred to or from the L64702 in 8 x 8 block order. The final placement of the data in video memory is such that it is presented to a display device in raster order.

Every video memory cycle involves 32-bit transfers, which, in this case, are equivalent to one complete pixel transfer containing an 8-bit value for each of three color components. This type of image must be entirely represented using complete 32-bit words. The starting address must also fall on a 32-bit boundary. The active window width and display pitch can be any 12-bit integer number. The display pitch must be greater than or equal to the value specified by the width.

Figure 7.8 shows the video bus activity for the unpacked color image data structure.

Figure 7.8 Unpacked Color Video Bus Activity

Bus Cycle	32-bit 31 24	23 16	15 8	7 0
0	x	B0	G0	R0
1	x	B1	G1	R1
2	x	B2	G2	R2
3	x	B3	G3	R3
4	x	B4	G4	R4
5	x	B5	G5	R5
6	x	B6	G6	R6
7	x	B7	G7	R7
8	x	B8	G8	R8
9	x	B9	G9	R9
10	x	B10	G10	R10
•	•	•	•	•
•	•	•	•	•

The addressing scheme implemented for this mode consists of eight consecutive memory transfer cycles, each of which transfers one complete pixel. Each pixel consists of three eight-bit color values, one for each color component. The video bus address generator then skips to an address that is the starting address plus the amount specified in the display pitch register, which points to the starting pixel for a particular 8 x 8 block on the next display line. At this point, eight more pixels are transferred, and again, the video bus address generator skips ahead to point to the starting pixel for the block on a new line. Sets of eight pixels are transferred until an 8 x 8 block (64 pixels) transfer is complete.

As shown in Figure 7.8, eight memory cycles are required to transfer eight complete pixels. The first complete pixel (R0, G0, B0, X) is contained in the first complete 32-bit word, the second pixel (R1, G1, B1, X) in the second 32-bit word, and so on.

Figure 7.9 shows the raster-to-block addressing scheme as implemented to support the unpacked color image data structure.

Figure 7.9
Unpacked Color
Memory
Organization

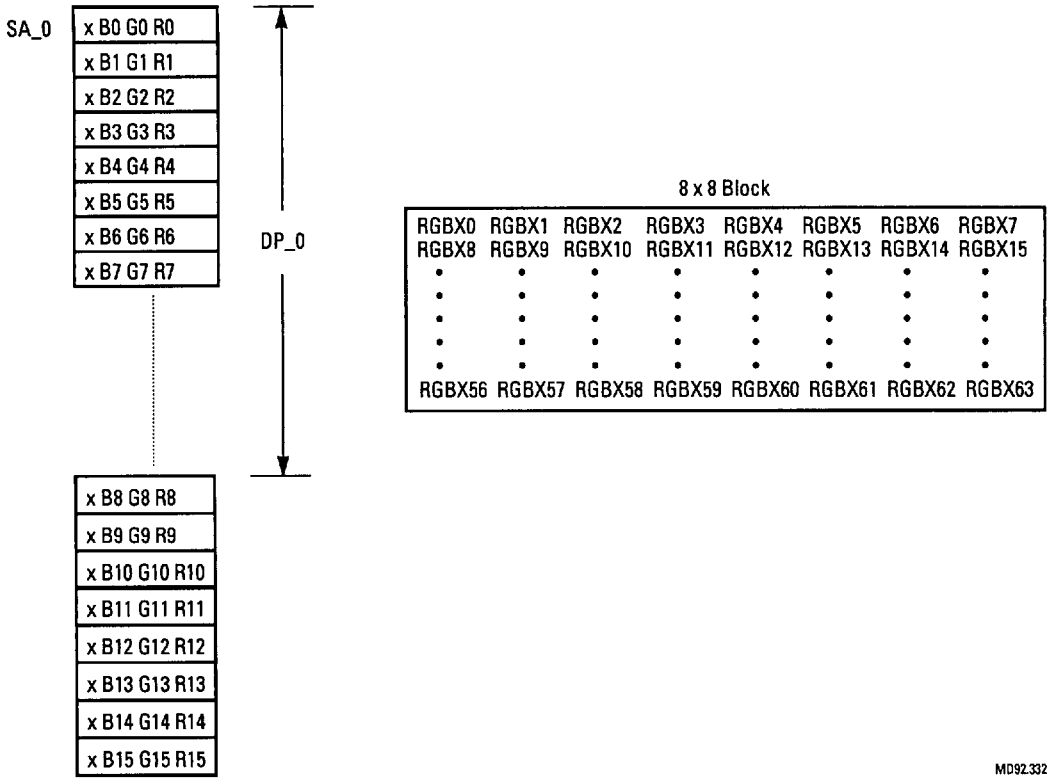


Figure 7.10 is a three-dimensional representation of the unpacked display image in video memory. Complete three-color pixels are generated as the CPU accesses a 32-bit word from memory. Each access yields a complete pixel. After eight memory accesses, enough information is available to create one of the layers of the three-dimensional 8 x 8 block depicted in the figure. The subsequent eight accesses result in another layer that belongs to the next display line.

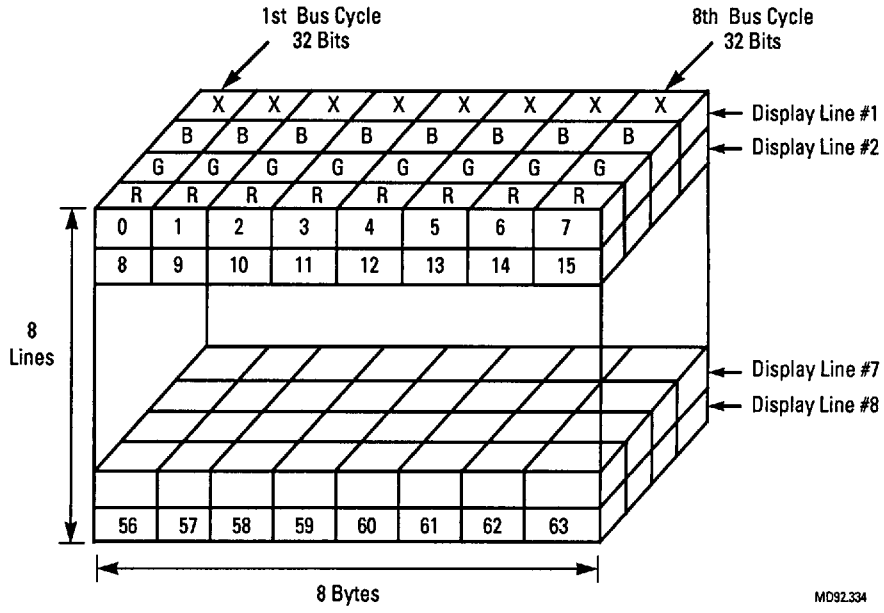


Figure 7.10 Three-dimensional 8 x 8 Unpacked Display Image

Packed Color Memory Organization

Packed color image data structures are accessed when the IDS field is set to two. Three color components are assumed to be interleaved and to reside in the same memory region. The packed color image data structure utilizes an entire 32-bit word for data storage. Because each color pixel is comprised of three 8-bit components, each 32-bit word contains portions of data from adjacent pixels. The advantage of this data-packing scheme is a 25% increase in memory utilization compared to the unpacked method.

Figure 7.11 shows the two-dimensional memory organization of a packed image data structure. The figure is representative of an image that consists of data in video memory organized as blocks eight bytes wide by eight bytes high. The starting address of the active window is in the SA_0 Register, and the active window width is in the AW Register.

Because pixel data are packed, every three 32-bit words contain data representing four color pixels. Packing data in this manner imposes a restriction on the width specification—the value must be an integer multiple of four. The active window height is in the AH Register. The total region size (in pixels) is determined by multiplying the specified display pitch by the number of display lines that are allocated for the display area. The display pitch value is limited to integer multiples of four is stored in the DP_0 Register.

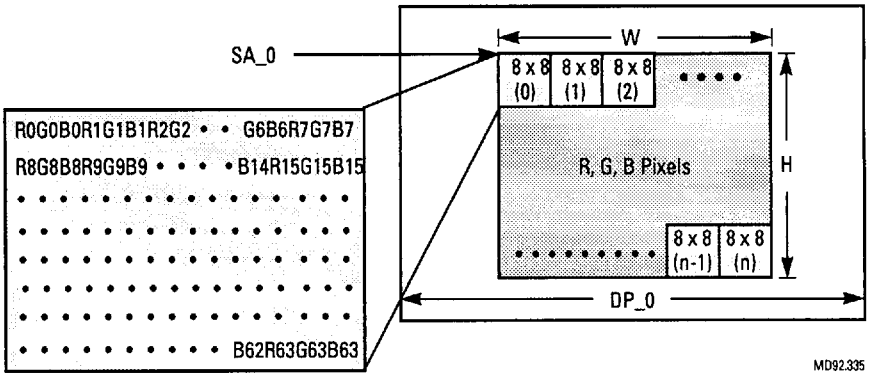


Figure 7.11 Packed Color Memory Organization

The video bus address generator performs the raster-to-block and block-to-raster translation. Image data are transferred to or from the L64702 in 8 x 8 block order. The final placement of the data in video memory is such that the data are presented to a display device in raster order.

Every video memory cycle involves 32-bit transfers, which, in this case, is equivalent to one and one-third pixel transfers. This type of image must be entirely represented using complete 32-bit words. The starting address must also fall on a 32-bit boundary. The active window width and display pitch can be any 12-bit integer. The display pitch must be greater than or equal to the value specified by the width.

Figure 7.12 shows the video bus activity for packed color image data structures.

Figure 7.12 Packed Color Video Bus Activity

	32-bit			
Bus Cycle	31 24	23 16	15 8	7 0
0	R1	B0	G0	R0
1	G2	R2	B1	G1
2	B3	G3	R3	B2
3	R5	B4	G4	R4
4	G6	R6	B5	G5
5	B7	G7	R7	B6
6	R9	B8	G8	R8
7	G10	R10	B9	G9
8	B11	G11	R11	B10

The addressing scheme implemented for this mode consists of eight consecutive memory transfer cycles, each of which transfers one and one-third eight-bit pixels, for a total of ten and two-thirds complete pixels every eight memory cycles. The video bus address generator then skips to an

address that is the starting address plus the amount specified in the display pitch register. The new address points to the starting pixel for a particular 8 x 8 block on the next display line. At this point, eight more pixels are transferred, and again, the video bus address generator skips ahead to point to the starting pixel for the block on a new line. Sets of eight pixels are transferred until an 8 x 8 block (64 pixels) transfer is complete.

As shown in Figure 7.12, eight memory cycles are required to transfer ten and two-thirds pixels. The first 32-bit word contains the first pixel (R0, G0, B0) plus one-third of the second pixel (R1). Each additional 32-bit word contains one and one-third pixels.

Figure 7.13 shows the raster-to-block addressing scheme as implemented to support the packed color image data structure.

Figure 7.13
Packed Color
Addressing Scheme

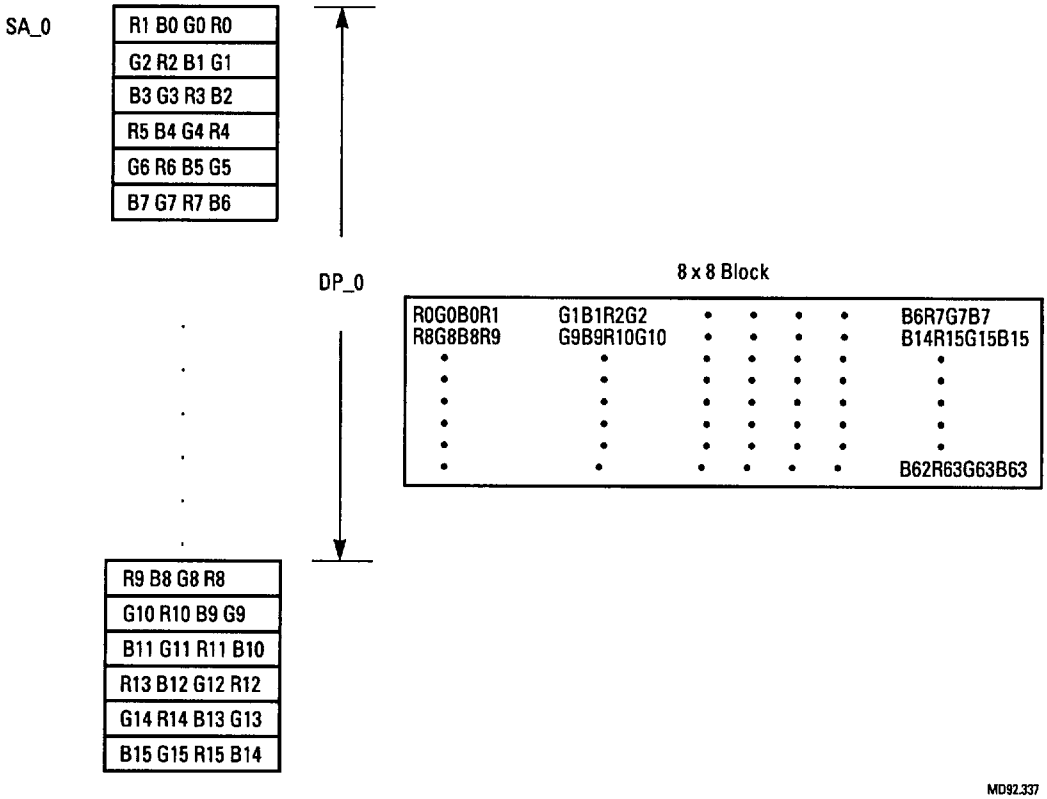
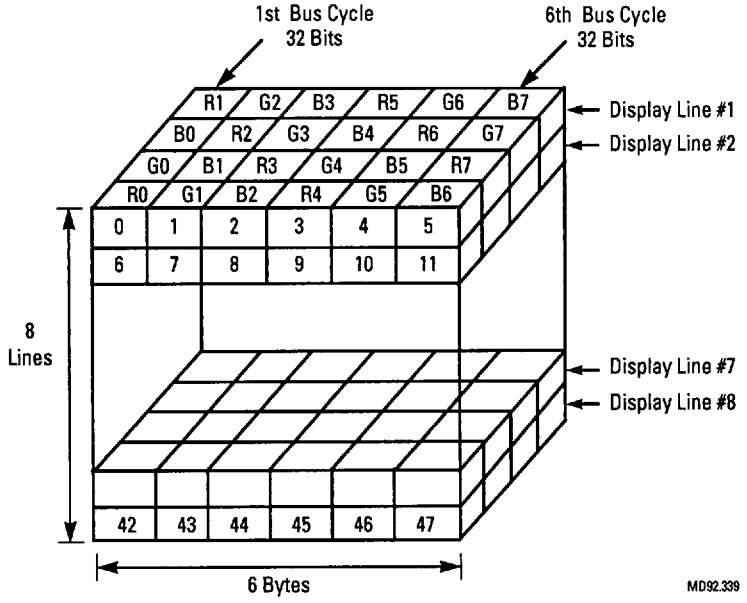


Figure 7.14 is a three-dimensional representation of the packed display image in video memory. Each access yields one and one-third pixels. After eight memory accesses, enough information is available to create one of the layers of the three-dimensional 8 x 8 block depicted in the figure. The subsequent eight accesses result in another layer that belongs to the next display line.

Figure 7.14 Three-dimensional 8 x 8 Packed Display Image



Components Sequential Memory Organization

Components sequential color image data structures are accessed when the IDS field is set to three. In this configuration, the L64702 accesses four color components from four separate memory regions. Each region is reconstructed from an array of pixels that are packed into a group of four pixels per address location. The total region size (in pixels) per color component is determined by multiplying the display pitch by the number of display lines allocated for the color component. The display pitch is specified per color component in the four display pitch registers: DP_0, DP_1, DP_2, and DP_3. The active window region is the specified area of operation within the color region and is specified by the following registers: start address (SA_0, SA_1, SA_2, and SA_3), width (W), and height (H).

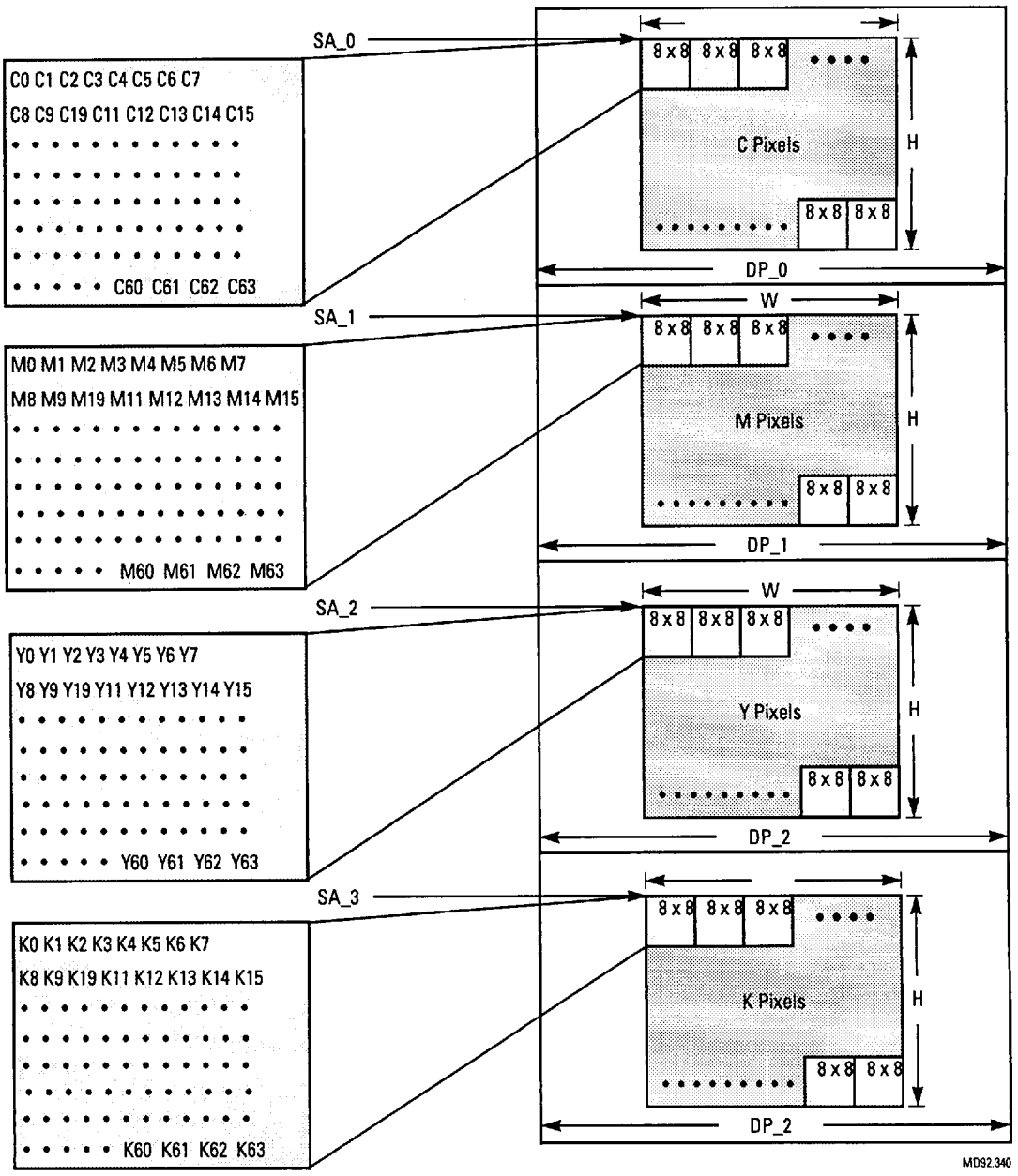
When operating in the components sequential image structure, the L64702 automatically bypasses the color space conversion unit, because the color components are assumed to be in the right color space for efficient compression or decompression. The JPEG baseline system standard dictates a certain sampling scheme that can be attached to each one of the color components with respect to the other. However, you must limit the sum of all the sampling schemes of all the participating color components to a value less than or equal to 10. The L64702 fully supports this method, and, in addition, the L64702 allows you to specify the sampling ratio in both the vertical and horizontal directions.

The sampling scheme is specified through the three registers MBS, MBS_X, and MBS_Y.

The width and the height dimensions of the active window are derived from the sampling factors of each color component. When the L64702 is compressing or decompressing image data in the components sequential mode, it uses the sampling values for each color component to generate the addresses that access the image data in the proper order.

Figure 7.15 shows the two-dimensional memory organization of components sequential image data structures. Note that the ordering of the 8 x 8 blocks is not indicated, because it is dependent on the sampling scheme chosen.

Figure 7.15
Components
Sequential Memory
Organization



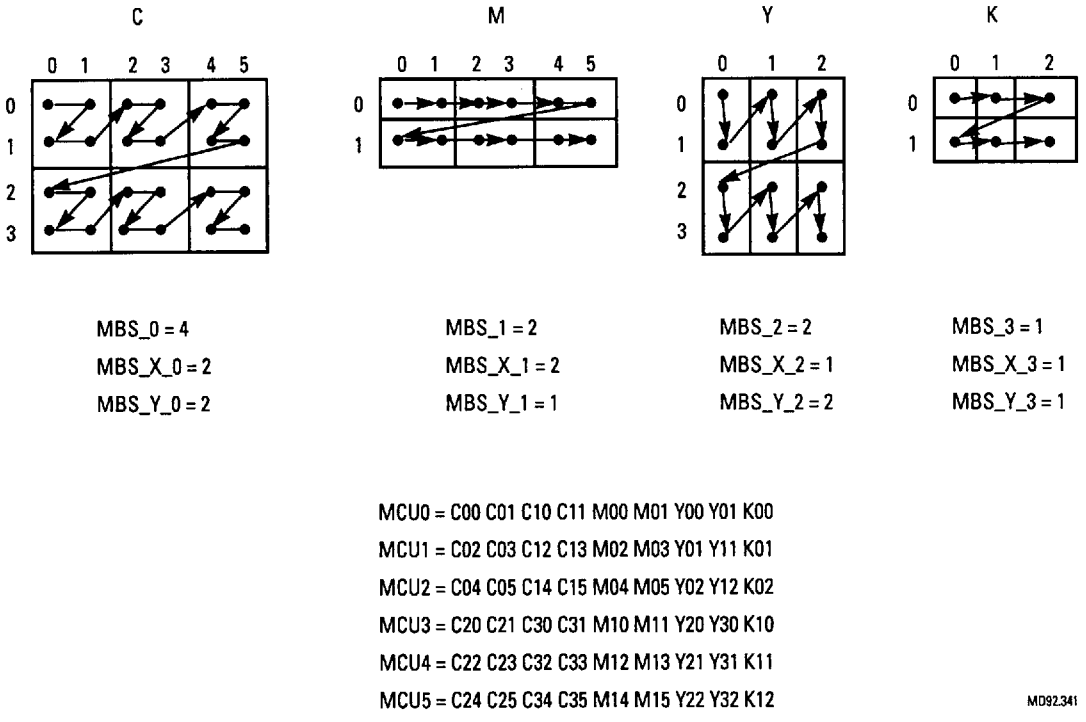
The video bus address generator performs the raster-to-block and block-to-raster operations when transferring data from video memory to the L64702 or from the L64702 to video memory, respectively. Image data are transferred to or from the L64702 in 8 x 8 block order. The data placement in video memory is such that the data are presented to a display device in raster order.

Every video memory cycle involves 32-bit transfers (four pixels). Therefore the specified active window width must be a multiple of four (4, 8, 12, 16, etc.). The start address must also be aligned with a 32-bit boundary. You must specify proper values in the starting address registers as well as the width and height registers to prevent any overrides. Any mix of data between color regions results in image data corruption.

Figure 7.16 shows the addressing scheme implemented for this mode. The basic concept of the addressing is that the image data are transferred one 8 x 8 block at a time in an order that is specified by the sampling scheme. The figure shows how MCUs are accessed according to the parameters specified in the MBS, MBS_X, and MBS_Y Registers.

In Figure 7.16, each color component has six regions. Each region has within it some dots that represent 8 x 8 data units. Each data unit is designated with a grid designation. Thus the top leftmost data unit is designated as C00. The next one to the right is C01 and so on.

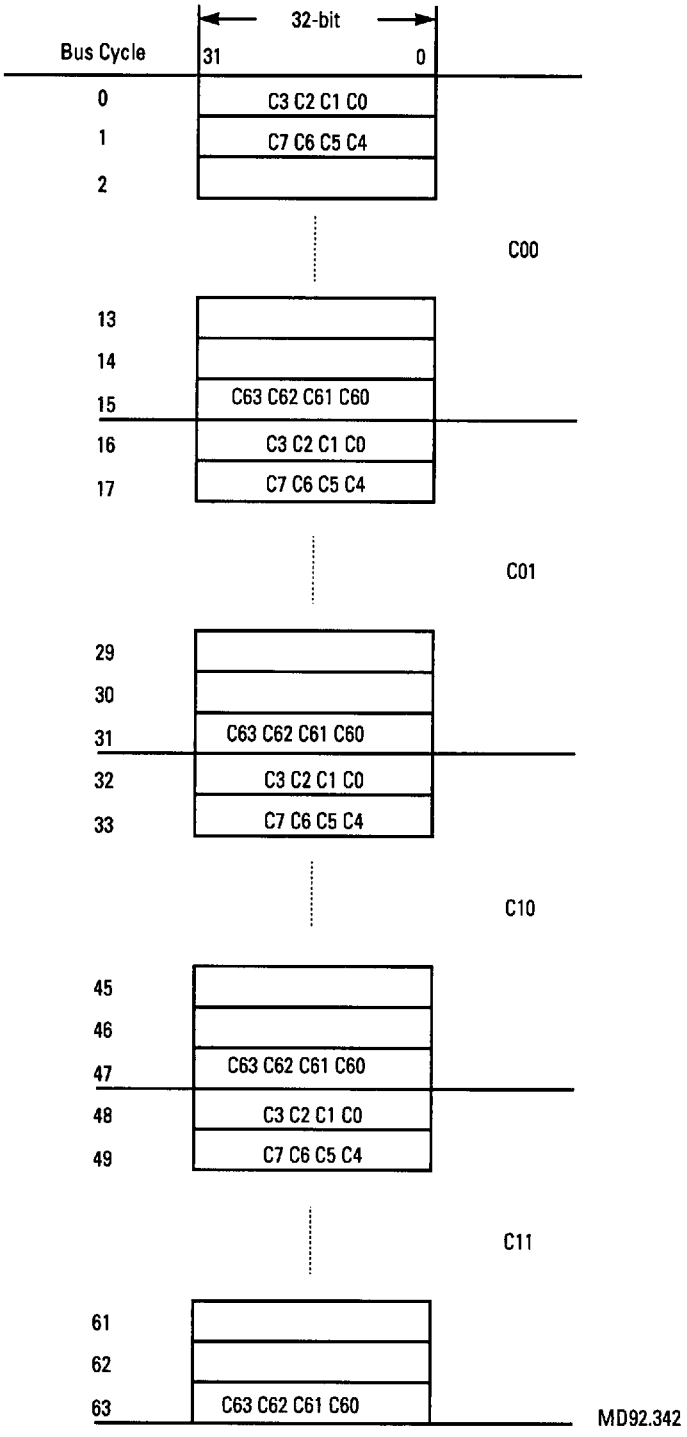
Figure 7.16
Components
Sequential
Addressing Scheme



MCU0 is constructed of 8 x 8 data units taken first from the top left-most region of the C color component, followed by data units from the same region of the M component, then from Y then from K. MCU1 is constructed following the same ordering for data taken from the next region to the right for the four components. This same procedure is followed for constructing MC0 through MC5.

Image data are transferred in 32-bit words (four pixels) per memory cycle. The L64702 transfers 16 words of 32-bit per 8 x 8 color component block. Figure 7.17 shows the video bus activity for color component C, according to the parameters specified in Figure 7.16. The bus activity for the other components follows that of the C component in the order C, Y, M, and K.

Figure 7.17 Components Sequential Video Bus Activity



7.4 Image Subsampling

The principle of subsampling is based on the fact that while each pixel of luminance is needed to maintain image fidelity, every other pixel of chrominance may be eliminated with no noticeable effect on the displayed results.

Subsampling ratios are specified in the L64702 Macro Block Size (MBS) Register. A sampling number for each component may be specified. For example, the ratio 1:1:1 specifies that for every sample of Y, there is also one sample of Cr and one of Cb. A 1:1:1 ratio does not really perform subsampling, because all of the original image data remains intact.

A ratio of 2:1:1 specifies that every pixel of Y is sampled, but only every other pixel of Cr and Cb is sampled. The result of 2:1:1 subsampling is that twice as many Y components are sampled than either of the Cr or Cb components. Using a 2:1:1 sampling ratio, the L64702 operates on a display segment 16 pixels wide by eight lines high and converts it to two 8 x 8 blocks of Y and one 8 x 8 block each of Cr and Cb.

A ratio of 1:1:1 specifies that every Y pixel is sampled, however, neither the Cr nor the Cb pixel is sampled.

Figure 7.18 illustrates how the components of an RGB image are converted by the Color Space Converter to YCrCb, then subsampled with a 2:1:1 sampling ratio into a final YCrCb MCU format consisting of two blocks of Y and one each of Cr and Cb. The net result of 2:1:1 subsampling is that for each block of 16 x 8 chrominance values, a single 8 x 8 block is produced.

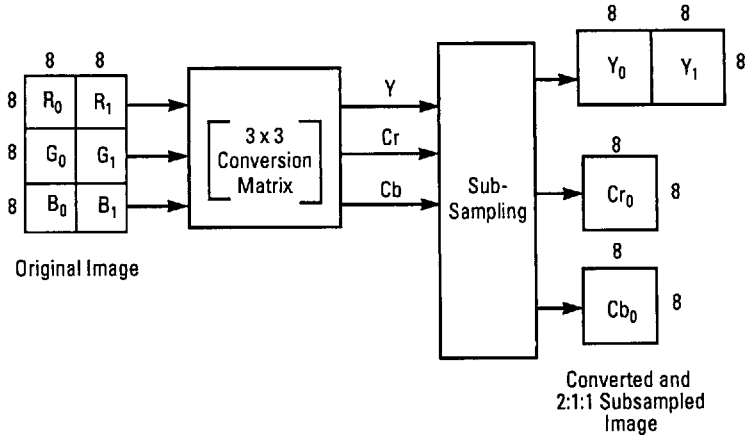
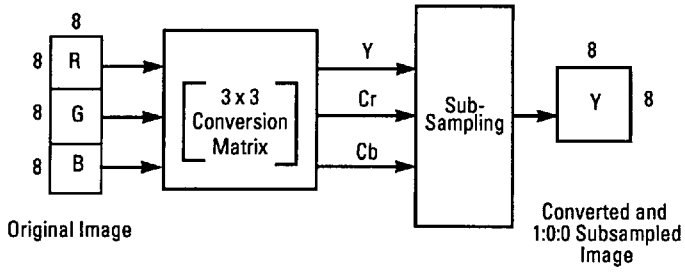


Figure 7.18 Color Conversion and Subsampling from RGB to YCrCb

Figure 7.19 illustrates how the components of an RGB image are converted by the Color Space Converter to YCrCb, then subsampled with a 1:0:0 sampling ratio into a final Y MCU format.

Figure 7.19 Color Conversion and Subsampling from RGB to Y



7.5 Scrolling in Compressed Files

The L64702 windowing capability allows you to define a window within a compressed file for decompressing a portion of an image and displaying this portion on a screen. This feature is useful for handling large compressed files. For example, a color scanner generates an image for an 8.5" x 11", 300-dpi page. This image contains approximately 25 Mbytes of display information, which represents an image resolution of 2550 x 3300 pixels. This resolution is far beyond what can be displayed on a regular monitor, which has a typical resolution of 640 x 480 pixels.

Using the L64702 windowing feature, you can specify a display window in the compressed file and dynamically scroll the window within the file. This feature allows previewing portions of an image prior to printing. Figure 7.20 shows the parameters that define the window.

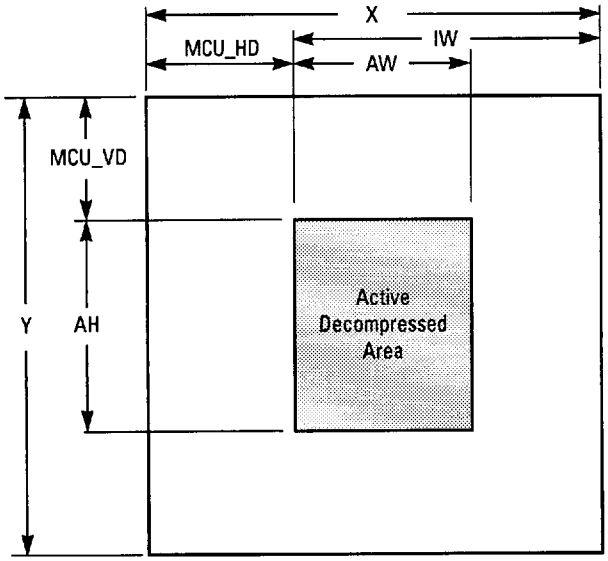


Figure 7.20 Active Decompressed Area Inside of a Compressed Image

Because the image is in compressed format, the image boundaries on the horizontal and vertical directions are assumed to be on MCU boundaries. The X and Y parameters define the total image resolution as specified in the compressed file header—they are not L64702 parameters. The Active Height (AH) and Active Width (AW) parameters are the active window parameters and, by definition, their values are in terms of MCUs. Image Width (IW) is the parameter that specifies how many MCUs lie between the left edge of the active window and the actual horizontal edge of the picture.

Two parameters, MCU_HD and MCU_VD, control the position of the active window in the compressed file. These negative, signed parameters define in terms of MCUs the specific starting point of the displayed portion of the compressed file. The software that controls L64702 initialization must be in accordance with Equation 7.1 and Equation 7.2 to prevent window violation.

Equation 7.1 Horizontal Scrolling

$$X \leq |MCUHD| + IW$$

Equation 7.2 Vertical Scrolling

$$Y \leq |MCUVD| + AH$$

Refer to Chapter 3, "Internal Registers and Data Tables," for more information regarding the window parameters mentioned above.

7.6 Active Window and Two-Dimensional Addressing

This section focuses on the video memory interface, because this interface employs most of the L64702 addressing capabilities. Note that the L64702 can access video memory composed of DRAMs and SRAMs with no restrictions.

The physical address space of image data in video memory is linear; that is, the pixel data of an image are stored in sequential memory locations. Pixel data may be organized as bytes (eight bits) for gray scale, or three bytes (24 bits) for color. However, for any image data format, the L64702 accesses 32-bit data, which may contain from one to four pixels.

When dealing with image information, it may be easier to conceptualize the image as two-dimensional. The L64702's on-board video addressing unit translates parameters from two-dimensional to linear address space. In addition, the video addressing unit performs the special function of raster-to-block or block-to-raster address conversion, which is essential for the L64702's JPEG Unit.

The display area is the total area in the memory that will be displayed on the screen. The active window area is the subsection of the display where the L64702 is actually accessing data.

Figure 7.21 illustrates in two dimensions a typical display screen, which is represented by the video memory contents and the active window portion within that display screen. Four parameters determine the active window area and the method the L64702 uses to operate within that window:

- Active Window Start Address (SA)
- Active Window Width (AW)
- Active Window Height (AH)
- Display Pitch (DP)

The Start Address (SA) is the absolute address of the pixel located in the upper left corner of the active window area. The Active Width (AW) is the number of MCUs in one active window line. The number of pixels can be computed from this parameter. The Active Height (AH) is the total number of lines in the active window. The Display Pitch (DP) is the total number of pixels in the scan line.

An external CPU may control these four parameters by writing to the L64702 video control registers.

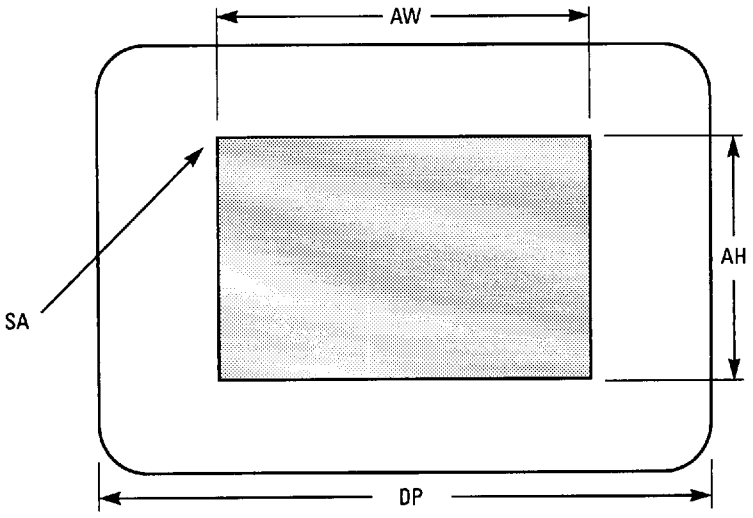


Figure 7.21 L64702 Active Window Area and 2-D Addressing Scheme

Active Window Start Address

The four starting address registers, SA_0, SA_1, SA_2, and SA_3, determine the active window start address. Each starting address register is associated with a specific color component (R, G, B or C, M, Y, K). The starting address register is used if the corresponding color plane is active. The sampling ratio specified in the MBS, MBS_X, and MBS_Y sampling control registers determines whether the plane is active or not. A sampling ratio is specified for each color component. See Chapter 3, "Internal Registers and Data Tables," for more information on the sampling control registers.

When the L64702 is programmed to process color images, the starting address of the image depends on whether the image is single- or multiple-source. For single-source color images, where the entire video image is contained in a single memory region, only the SA_0 Register is used. For multiple-source color RGB images, where each of the three color components of the image is contained in a separate memory region, SA_0, SA_1, and SA_2 are used.

For components sequential color CMYK images, where each of the four color components of the image is contained in a separate memory region, SA_0, SA_1, SA_2, and SA_3 are used. The 24-bit value specified in the starting address register is the physical memory address of the starting pixel of the color component and represents the starting address linear memory offset from zero.

Active Window Width

The AW Register in conjunction with the Height Width Register for a particular color component specify the total number of MCUs in the active window area, counted in the horizontal direction. Equation 7.3 and Equation 7.4 specify how the L64702 uses the contents of the AW and H0_W0 Registers to calculate the number of MCUs within the active window in the horizontal direction for color component zero. The same equations apply to all color components (0, 1, 2, or 3).

Equation 7.3 Active Width Calculation

$$AW = \lceil (X)/B \rceil - 1 \quad \text{where B = 8 x MBS\_X}$$

Equation 7.4 W0 Calculation

$$W0 = (X-1)\%B \quad \text{where W0 is the } \textit{remainder} \text{ resulting from the division operation}$$

An example for calculating the AW value is to use color mode, RGBx format, a sampling ratio of 1:1:2, a total number of pixels per line X = 640, and an MBS_X_0 value of two. See the next subsection for an explanation of sampling ratios and MCU construction.

Continuing with the example, we will assume that MBS_X_0 = 2 and MBS_Y_0 = 1. The MCU size in this case is eight lines by 16 pixels or words, so the value of AW is the *ceiling* of (640/16) – 1 = 39. Since the numbering of the value in the AW Register begins with zero, the total number of MCUs specified in the AW Register is 40 (0 through 39). The value of W0 is the remainder of (640 – 1) / 16, or 15. A value of 15 in the W0 Register indicates that all of the pixels in the last horizontal MCU are used, yielding a complete, final MCU.

When W is not on an MCU boundary, the Height Width Registers make up the difference. For example, if W = 641 pixels or words, the value of AW is the ceiling of (641/16) – 1 = 40. Since the numbering of the value in the AW Register begins with zero, the total number of MCUs specified in the AW Register is 41 (0 through 40). The value of W0 is the remainder of (641 – 1) / 16, or zero. A value of zero in the W0 Register indicates that only one display pixel or word is in the active window for the last MCU.

The W0 value determines the number of active display pixels or words in the last MCU as shown in Table 7.2.

Table 7.2 Number of Display Pixels or Words

W0	*Active Display Pixels or Words*
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8

Active Window Height

The AH Register in conjunction with the Height Width Register for a particular color component specify the total number of MCUs in the active window area, counted in the vertical direction. The following equations specify how the L64702 uses the contents of the AH and H0_W0 Registers to calculate the number of MCUs within the active window in the vertical direction for color component zero. The same equations apply to all color components (0, 1, 2, or 3).

Equation 7.5 Active Height Calculation

$$AH = \lceil Y/A \rceil - 1 \quad \text{where A = 8 x MBS\_Y}$$

Equation 7.6 H0 Calculation

$$H0 = (Y-1)\ \%A \quad \text{where H0 is the } \textit{remainder} \text{ resulting from the division operation}$$

An example for calculating the AH value is to use color mode, RGBx format, a sampling ratio of 2:1:1, a total number of lines Y = 480 in the active image that are to be displayed, and an MBS_Y_0 value of one. The sampling ratio of 2:1:1 indicates that two MCUs of color component 0 are sampled for every one MCU of the other color components.

In order to completely specify how the L64702 is to perform sampling, the MBS_0, MBS_X_0, and MBS_Y_0 Registers must be written. The MBS_0 Register specifies how many 8 x 8 data blocks of color component 0 are included in an MCU. The MBS_X_0 Register specifies how many 8 x 8 blocks in the horizontal direction to include in the sample for color component 0, and the MBS_Y_0 Register specifies how many blocks in the vertical direction to include in the sample for color component 0. In a similar fashion, the MBS_1, MBS_2, MBS_3, MBS_X_1, MBS_X_2,

MBS_X_3, MBS_Y_1, MBS_Y_2, and MBS_Y_3 Registers correspond to the components of the MCU included from the other color components.

In a 2:1:1 sampling scheme, MBS_0 = 2, MBS_1 = 1, and MBS_2 = 1. Since MBS_0 = 2, there are two possibilities of how the sampling is performed. One possibility is that two 8 x 8 blocks are sampled in the horizontal direction, in which case MBS_X_0 = 2, and MBS_Y_0 = 1. The other possibility is that two 8 x 8 blocks are sampled in the vertical direction, in which case MBS_X_0 = 1, and MBS_Y_0 = 2. The remaining MBS_X and MBS_Y Registers all contain a value of one, because there is only a single 8 x 8 block from each, and there is no vertical or horizontal sampling sequence.

Continuing with the example, we will assume that MBS_Y_0 = 1. The MCU size in this case is eight lines by 16 pixels, so the value of AH is the *ceiling* of (480/8) –1 = 59. Since the numbering of the value in the AH Register begins with zero, the total number of MCUs specified in the AH Register is 60 (0 through 59). The value of H0 in the H0_W0 Register is the remainder of (480 – 1) / 8, or 7. A value of 7 in the H0 Register indicates that all of the last MCU is active.

When Y is not on an MCU boundary, the Height Width Registers make up the difference. For example, if Y = 481 lines, the value of AH is the ceiling of (481/8) – 1 = 60. Since the numbering of the value in the AH Register begins with zero, the total number of MCUs specified in the AH Register is 61 (0 through 60). The value of H0 is the remainder of (481 – 1) / 8, or zero. A value of zero in the H0 Register indicates that only one display line in the 61st MCU is active.

The H0 value determines the number of active display lines in the last MCU as shown in Table 7.3.

Table 7.3 Number of Display Lines

H0	*Active Display Lines*
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8

Active Height and Width Example

This subsection calculates the values of W_n, H_n, AW, and AH for the four sampling modes assuming a 640 x 480 active image.

The values of W_n and AW for the three color modes for a 640 x 480 active image are shown in Table 7.4.

Table 7.4
W_n and AW Values for the Color Modes

	1:0:0, 1:1:1		*2:1:1*	
Image Format	***Wn***	***AW***	***Wn***	***AW***
Unpacked	7	79	15	39
Packed	5	79	11	39
Multisource	1	79	3	39

The values of W_n and AW for the components sequential mode are listed in Table 7.5.

Table 7.5
W_n and AW Values for Components Sequential Mode

Components Sequential				***W_0***	***W_1***	***W_2***	***W_3***	***AW***
4: (640 x 480)	2: (640 x 240)	1: (320 x 240)	1 (320 x 240)	3	3	1	1	39

The value of H_n for the three color modes is 7, and the value of AH is 59. For the components sequential mode, the values of H_n and AH are listed in the following table.

Table 7.6
H_n and AH Values for Components Sequential Mode

Components Sequential				***H_0***	***H_1***	***H_2***	***H_3***	***AH***
4: (640 x 480)	2: (640 x 240)	1: (320 x 240)	1 (320 x 240)	15	7	7	7	39

Active Window Display Pitch

The four display pitch registers, DP_0, DP_1, DP_2, and DP_3, determine the active window display pitch parameter. Each display pitch register is associated with a specific color component. The value in each register determines the total number of pixels per display line. In a display system, the display pitch parameter determines the memory distance between two adjacent vertical pixels. The actual width parameter of the color component must be less than or equal to the display pitch of the color component, unless such a value produces an invalid result.

The display pitch register is used if the color plane is actually active. The sampling ratio specified in the sampling control registers MBS, MBS_X, and MBS_Y determines whether the color plane is active.

When the L64702 is programmed to process color image structures, the use of the display pitch registers varies depending on whether the image is single-source, multiple-source, or components sequential. For a single-source color image, only the DP_0 register is used. For multiple-source RGB color images, the first three registers are used (DP_0, DP_1, and DP_2). For components sequential color images, all four registers are used.

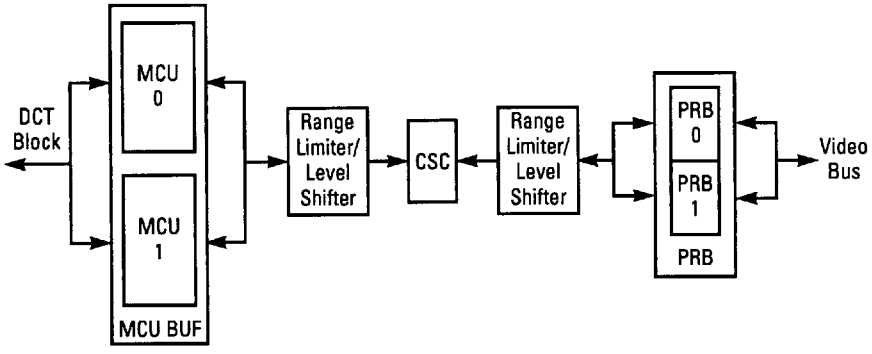
Chapter 8
Pixel Data Processing Overview

This chapter presents a functional overview of the image data pre- and post-processing, which is performed in the front-end video interface. Figure 8.1 shows the block diagram of the pixel data processing section.

This chapter contains four sections:

- Section 8.1, Pixel Data Reformatting Buffer (PRB)
- Section 8.2, Color Space Converter (CSC)
- Section 8.3, Level Shifting and Range Limiting
- Section 8.4, MCU Buffer

Figure 8.1 Pixel Data Processing Block Diagram

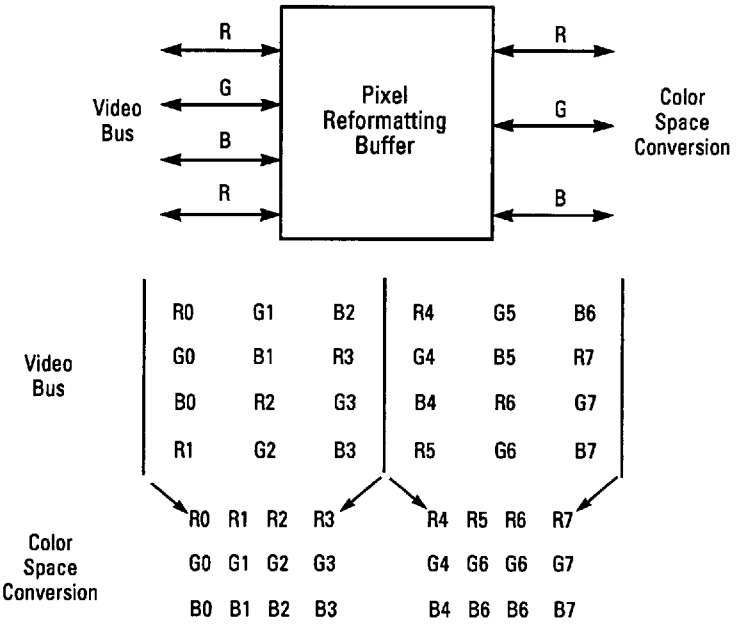


8.1 Pixel Data Reformatting Buffer (PRB)

The L64702 must convert the input data structures to a single internal format that can be used by the color space conversion unit and other processing modules. The PRB performs image data reformatting in real time in both direction. The IDS field of the SCR Register determines the image data format at the video memory interface and instructs the PRB on how to handle the image data. To maintain the pipeline data rate for the various image data structures, the PRB unit contains two buffers with storage for four pixels each. Figure 8.2 shows how the reformatting unit reformats

packed color image data into a format compatible with the color space conversion unit.

Figure 8.2 Pixel Data Reformatting Concept



8.2 Color Space Converter (CSC)

The Color Space Converter in the L64702 supports any linear transformation of a 3 x 3 matrix by multiplying the vector by a 3 x 1 vector. An external CPU or DMA Controller can download the nine coefficients of the 3 x 3 matrix. These coefficients are completely flexible, and allow a wide variety of conversions to take place. The matrix is bidirectional to allow color space conversion in either direction. The ENC bit in the SCR Register determines the direction of operation of the L64702 (compression or decompression). The values in the 3 x 3 matrix must be downloaded as appropriate for the operation being performed.

Figure 8.3 shows the color space conversion matrix.

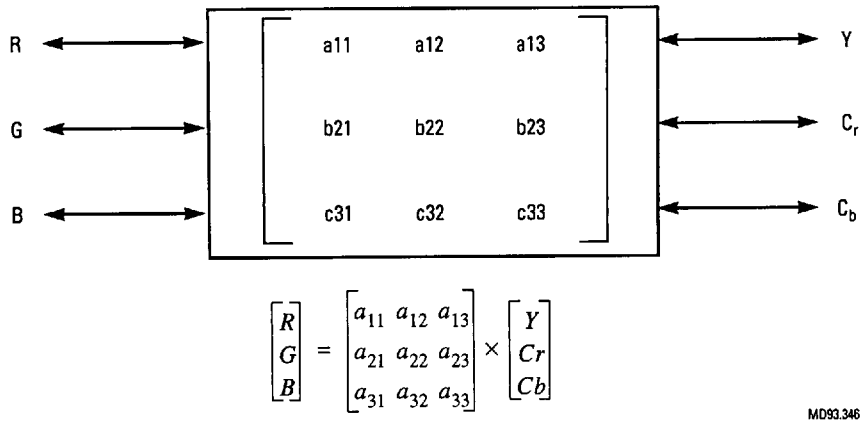


Figure 8.3
Color Space
Conversion Unit

The CSC interfaces to the PRB on one side and to the MCU buffer on the other. The SU0 – SU2 fields in the Video Bus Configuration Register determine whether data from the PRB are treated as signed or unsigned numbers. Unsigned values range from 0 to 255, while signed values range from -128 to 127. On the MCU buffer side, numbers are assumed to be eight-bit, signed numbers. Range limiting is performed automatically on both sides, depending on the operation, to prevent any data corruption due to mathematical overflow or underflow.

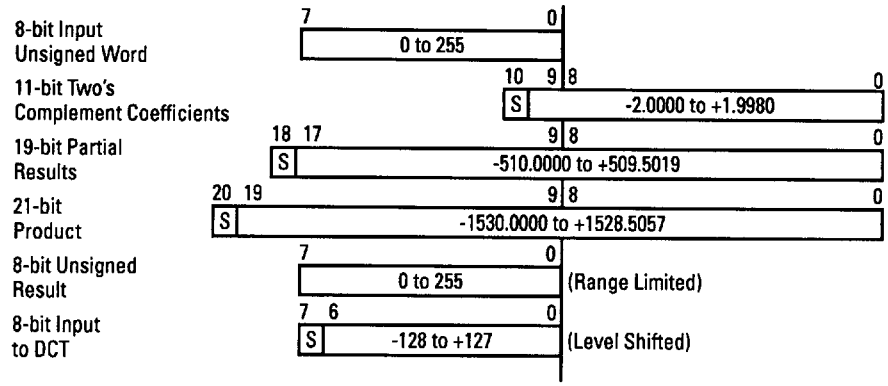
The coefficients in the matrix are 11-bit, two's-complement numbers, configured as a nine-bit fraction, a single integer bit, and a single sign bit. Coefficients range from -2.0000 to +1.9980.

Figure 8.4 shows the format of the color space conversion during the encoding process. The vertical dashed line represents the location of the binary point during the conversion process. The conversion steps are outlined below.

1. During compression, eight-bit image data from the video bus are multiplied by the 11-bit CSC coefficient.
2. The multiplication yields three partial product terms, or partial results, represented by 19-bit, signed numbers with values ranging from -510.0000 to +509.5019.
3. The three partial results are added together to produce the 21-bit product with values ranging from -1530.0000 to +1528.5057.
4. The eight bits situated immediately to the left of the binary point form the final result.

The CSC then processes the eight-bit number by level shifting and range limiting, if necessary, to deliver a final result with a value ranging from -128 to +127. During the process, the CSC maintains accuracy such that computational errors do not exceed one half of an LSB.

Figure 8.4 Color Space Conversion for Unsigned Numbers



The most common color space conversions are described in the following subsections. The values loaded into the conversion matrix are 11-bit numbers with values ranging from -2.000 to +1.998. Bit 10 is the sign bit. The binary point falls between bits 9 and 8. Therefore bit 9 is the integer part of the coefficient (1 or 0), and bits 8 through 0 make up the fractional part.

RGB-to-YC_bC_r

YC_bC_r color space has been found to be very efficient in achieving high compression ratios for most applications. This transformation is recommended by the CCIR 601 standard. Equation 8.1 shows the formula for transforming from RGB to YC_bC_r color space.

Equation 8.1 RGB-to-YC_rC_b

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

MD92.348

YC_bC_r-to-RGB

Equation 8.2 shows the formula for computing RGB components from YC_bC_r components.

Equation 8.2 YC_rC_b-to-RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 1.402 \\ 1.0 & -0.3437 & -0.7143 \\ 1.0 & 1.772 & 0.0 \end{bmatrix} \times \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix}$$

MD92.349

RGB-to-Y (Gray)

In this conversion, the color image is converted into a black-and-white image. Because the chrominance information, C_bC_r, is removed from the picture, the operation is not reversible. There are two methods of conversion in this case. The first method entails setting the chrominance components to zero in the conversion matrix and converting as if three components existed. In the second method, the Y component is compressed as a single component. The second method provides for a better compression ratio, because it does not contain the empty blocks of the chrominance components as does the first method.

Y-to-RGB

If a gray scale image needs to be displayed on a color monitor, the three color components must be the same. Equation 8.3 shows the matrix used to perform this conversion.

Equation 8.3 Y-to-RGB

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Color Space Conversion Bypass

To bypass color space conversion, the conversion matrix is programmed as a unit matrix. Equation 8.4 shows the formula for the color space conversion bypass.

Equation 8.4 Color Space Conversion Bypass

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \times \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix}$$

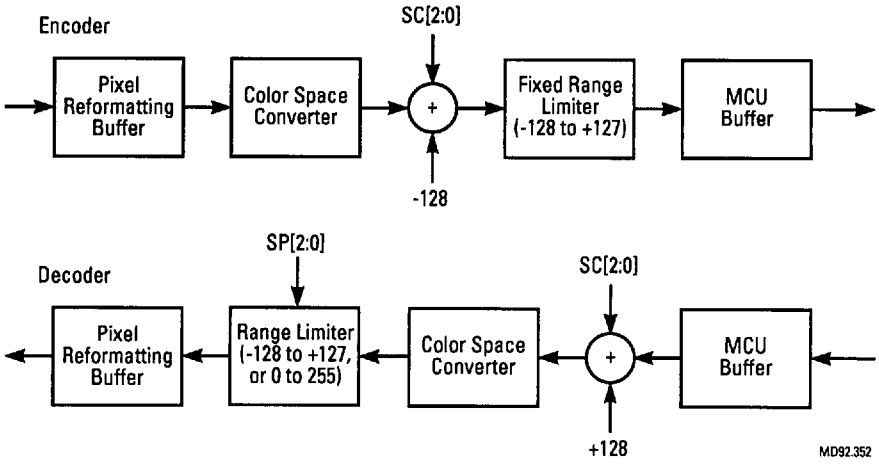
MD92.351

8.3 Level Shifting and Range Limiting

The L64702 includes two range limiters and a level shifter. The user can load matrix coefficients that could conceivably cause out-of-range values to result during multiplication. The range limiters correct for out-of-range numbers by clipping the values.

Figure 8.5 shows the range limiters and level shifter for both the encode and decode directions. SC[2:0] refers to the sign of the color space conversion components and SP[2:0] refers to the sign of the pixel components.

Figure 8.5
Range Limiters and Level Shifter



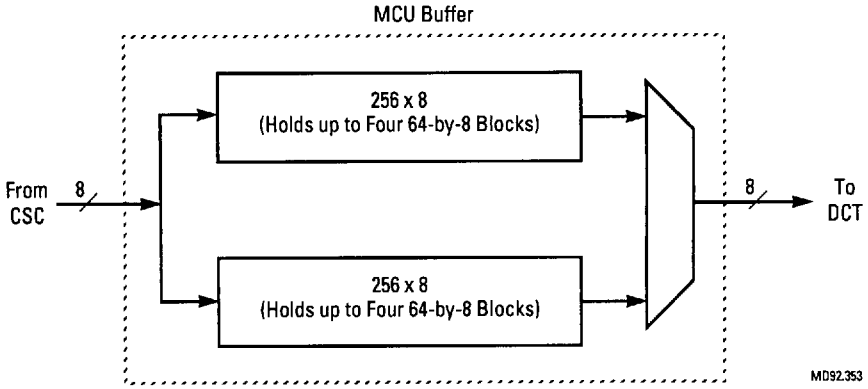
For a complete explanation of the function of these bits, refer to Chapter 3, "Internal Registers and Data Tables."

8.4 MCU Buffer

The MCU buffer interfaces the CSC with the DCT unit. The MCU buffers a complete set of data that reconstructs a complete MCU data structure. To maintain continuous data flow through the JPEG pipeline processor, the MCU buffer contains two sets of parallel buffers organized in a double-buffer architecture. When the DCT unit is accessing one buffer, the CSC can access the other buffer. Each internal buffer contains 256 eight-bit words; enough capacity to store four data units of 64 samples each. The L64702 switches between the buffers at a frequency determined by the MCU size. The MCU buffer generates wait states to ensure synchronization of the two sides because the two sides can be forced by the external system to function at different speeds.

Figure 8.6 shows the internal organization of the MCU buffer.

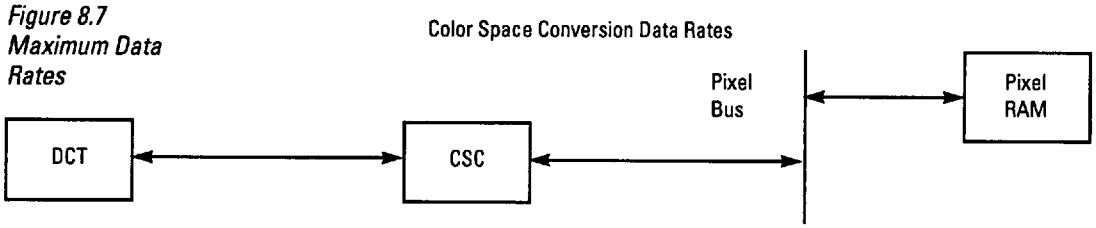
Figure 8.6
MCU Buffer
Configuration



The DCT unit processes data at the rate of one byte per four cycles. The MCU buffer supplies data at the appropriate rate to the CSC and the video bus.

Figure 8.7 summarizes the maximum data rates for the different sampling factors and image data structures. The values are given in bytes per clock cycle.

Figure 8.7 Maximum Data Rates



Y	1 B / 4 Cycles	1 : 0 : 0	4 B/16 Cycles	YYYY
Y	1 B / 8 Cycles	1 : 0 : 0	3 B/8 Cycles	RGB-
Y	1 B / 6 Cycles	1 : 0 : 0	4 B/8 Cycles	RGBRGBRGBRGB RRRRGGGGBBBB
YCrCb	1 B /4 Cycles	1 : 1 : 1	4 B/48 Cycles	YYYY
YCrCb	1 B /4 Cycles	1 : 1 : 1	3 B/12 Cycles	RGB-
YCrCb	1 B /4 Cycles	1 : 1 : 1	4 B/16 Cycles	RGBRGBRGBRGB RRRRGGGGBBBB
YYCrCb	1 B /4 Cycles	2 : 1 : 1	4 B/32 Cycles	YYYY
YYCrCb	1 B /4 Cycles	2 : 1 : 1	3 B/8 Cycles	RGB-
YYCrCb	1 B /4 Cycles	2 : 1 : 1	4 B/10 2/3 Cycles	RGBRGBRGBRGB RRRRGGGGBBBB

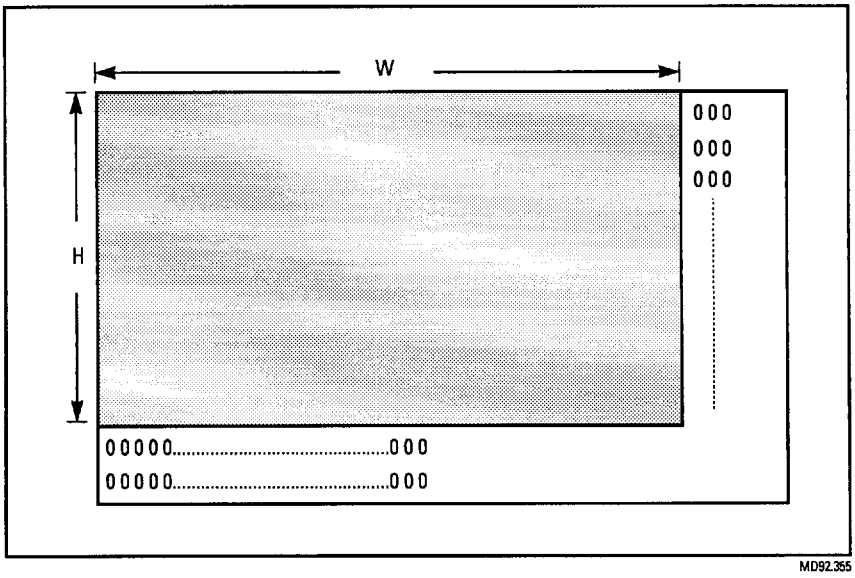
Note:
1. Maximum data rates: DCT: 1 byte/4 Cycles, Pixel Interface: 1 word/ 8 Cycles

The DCT unit requires a block of 8 x 8 pixels. The L64702 allows you to specify any integer number of words to be the image width size and any number of lines to be the height size. Sometimes the width and height specifications do not fall on an MCU boundary. For proper operation of the JPEG unit, the image size must fall on an exact MCU boundary. To fulfill this requirement, the L64702, when necessary, extends the boundaries of the image width and height parameters within the memory space frame, with certain restrictions for particular image data structures. Two parameters per color component (W0_H0, W1_H1, W2_H2, W3_H3) specify the number of words in the last MCU block in the horizontal direction, and the number of lines in the last MCU in the vertical direction. If the relevant parameter is zero for both the width and the height, no pixel or line filling is performed. When the active window width specified by the Width parameter is not an integer multiple of eight, the L64702 automatically fills the rightmost sample block with zeros as many times as required.

When the active window height specified by the Height parameter is not an integer multiple of eight, the L64702 adds lines filled with zeros as many times as required to fill the block.

Figure 8.8 shows an example of a boundary fill for the case where both the width and the height specification for the active window are not integer multiples of eight—in this case, the active window for the compression direction is extended to both the vertical and horizontal directions. During decompression there is no active window extension—the exact active window size is transferred to the video memory, and the pixels and lines outside the active window boundary are masked out.

Figure 8.8
MCU Boundary Schemes



Chapter 9
Video Memory Interface Port

This chapter describes the interface between the L64702 and the external video memory bus provided by the L64702 video interface.

Chapter 9 has three sections:

- Section 9.1, Video Bus Arbitration Protocol
- Section 9.2, Video Bus Request Criteria and Priorities
- Section 9.3, Video Bus Interface Timing

9.1 Video Bus Arbitration Protocol

The L64702 contains a Video Bus Control (VBC) block, which controls the operation of the video bus. The VBC block generates the addresses as well as all of the control signals that interface the chip to external VRAM or DRAM memory. The L64702 acts as master on the video bus and provides two control lines to provide bus sharing with another bus master, such as a graphics coprocessor or digital signal processor.

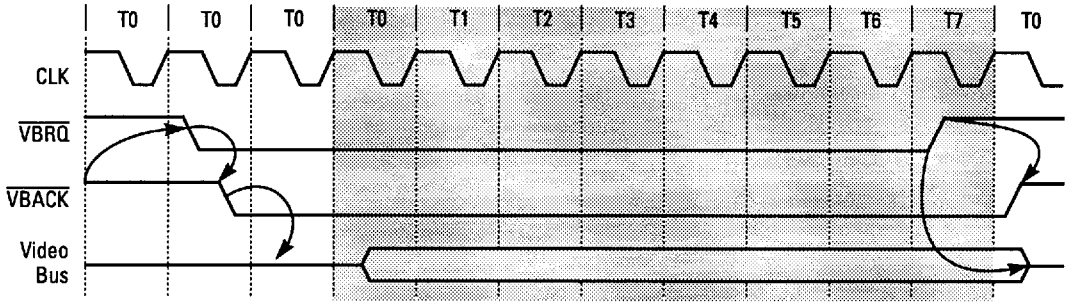
The L64702 asserts and holds LOW the video bus request signal, $\overline{\text{VBRQ}}$, to request the bus. An external video bus master asserts and holds LOW the video bus acknowledge signal, $\overline{\text{VBACK}}$, when the external bus master is ready to grant the bus to the L64702. As long as $\overline{\text{VBACK}}$ and $\overline{\text{VBRQ}}$ are asserted, the L64702 has control of the video bus, and generates the address, data, and control lines to provide data transfers to and from video memory. If the L64702 is the only bus master connected to the video bus, you may connect the $\overline{\text{VBRQ}}$ and $\overline{\text{VBACK}}$ signals together.

Figure 9.1 shows the regular video bus arbitration scheme. The minimum video bus cycle is eight CLKs (T0, T1, ..., T7). An external device may extend the bus cycle with the video bus wait signal, $\overline{\text{VWAIT}}$. If an internal request for a video bus cycle has been generated, the L64702 asserts the $\overline{\text{VBRQ}}$ signal. The L64702 asserts the $\overline{\text{VBRQ}}$ signal only if $\overline{\text{VBACK}}$ has

been previously inactive (HIGH) for at least one clock. When $\overline{\text{VBACK}}$ goes active (LOW), the L64702 has been granted the bus.

After the L64702 asserts $\overline{\text{VBRQ}}$, it waits for $\overline{\text{VBACK}}$ to be asserted. When the L64702 determines that $\overline{\text{VBACK}}$ is LOW, it starts a video bus cycle one CLK later. At this point, all of the L64702 video bus control and data signals change from 3-state to active. The L64702 deasserts the $\overline{\text{VBRQ}}$ signal if internally there is no demand for the video bus. The internal video bus request criteria are described in the following section. When the L64702 deasserts $\overline{\text{VBRQ}}$, it 3-states all video bus lines. An external master must deassert $\overline{\text{VBACK}}$ before the L64702 can assert $\overline{\text{VBRQ}}$ again to perform another video bus cycle.

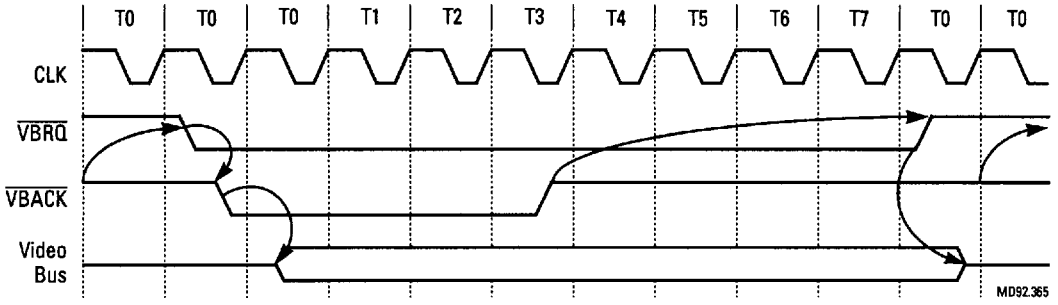
Figure 9.1
Regular Video Bus
Arbitration Protocol



An external video bus master has the right deassert the $\overline{\text{VBACK}}$ signal to force the L64702 to relinquish the bus at any time. If the external master processor deasserts $\overline{\text{VBACK}}$ while the L64702 is using the bus, the L64702 releases the bus after a maximum of eight CLKs. To notify the graphics processor that it is relinquishing the bus, the L64702 deasserts $\overline{\text{VBRQ}}$ for a minimum of two CLKs while $\overline{\text{VBACK}}$ is deasserted. The L64702 then reasserts $\overline{\text{VBRQ}}$ to indicate that the video bus activity has not been completed.

Figure 9.2 shows the bus preemption protocol.

Figure 9.2
Video Bus
Arbitration
Preemption Protocol



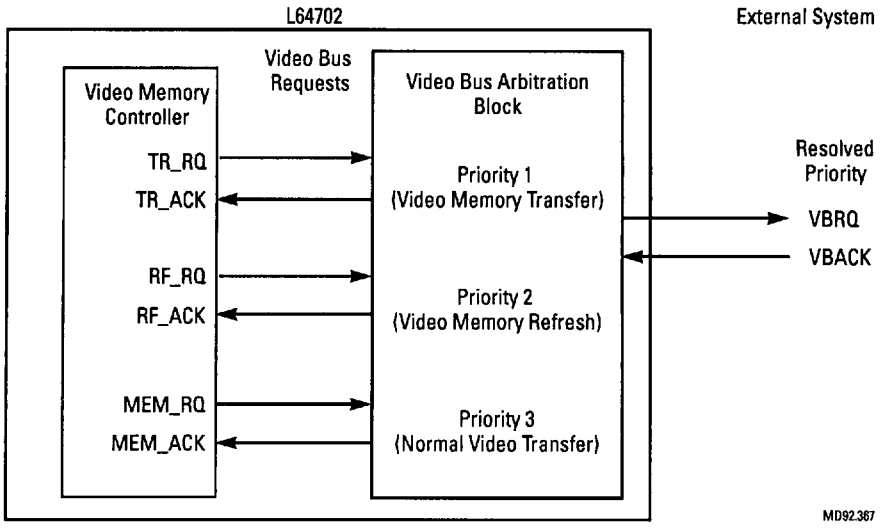
9.2 Video Bus Request Criteria and Priorities

Three internal sources may demand video bus cycles: video memory cycles (memory read or memory write), memory refresh cycles, or video memory transfers (DRAM core to shift register). All three sources may request service at the same time, so a priority scheme has been established to share the video bus among the different sources. The following sources may request the video bus, with 1 being the highest priority and 3 being the lowest:

1. Video Memory Transfer Cycle
2. Video Memory Refresh Cycle
3. Video Memory Cycle

Figure 9.3 shows the L64702 internal video bus arbitration block that controls the priority scheme. If the video bus is active with regular video memory cycles and refresh or transfer requests occur, the L64702 completes the current bus cycle, and then releases the bus in deference to higher priority requests. The L64702 resumes the memory read or write cycle only after higher priority requests have been serviced. Because refresh cycles and video memory transfer cycles are by definition higher priority than regular video memory cycles, and are serviced in a single memory cycle, the video bus cannot be stolen from them. The priority is resolved in the arbitration block before the start of any memory cycle.

Figure 9.3
Video Bus Internal Arbitration Block



Video Memory Transfer Request Criteria

The video transfer request input, $\overline{\text{VTRR}}$, is responsible for activating the video memory transfer request signal TR_RQ. An external video timing circuit produces $\overline{\text{VTRR}}$. The L64702 uses $\overline{\text{VTRR}}$ for either video memory display or video capture implementation. Once $\overline{\text{VTRR}}$ is activated, the internal TR_RQ signal is asserted and remains active until the video memory transfer cycle has been performed. If the next transfer request comes while the previous one is still pending, the new one may be lost because the transfer address register holds the address of the pending request. In actuality, it is not likely a transfer request will be lost, because $\overline{\text{VTRR}}$ occurs only once per video line.

Video Memory Refresh Request Criteria

The L64702 produces video memory refresh cycles if the RR field in the SCR Register is set to active mode. When refresh is active, an internal refresh counter asserts the RF_RQ signal to produce periodic refresh request indications. RF_RQ remains asserted until the video bus arbitration block acknowledges the refresh cycle. If the next refresh request occurs while the previous one is pending, the current request is lost.

Video Memory Read or Write Cycle Request Criteria

The video bus communicates with the L64702 through the two on-chip MCU buffers. The status of the MCU buffers determines when the L64702 is ready for data transfer, or in other words, when the L64702 will assert

$\overline{\text{VBRQ}}$. In fact, the MCU full and empty status flags activate the MEM_RQ signal and do not directly activate the $\overline{\text{VBRQ}}$ signal.

The assertion and deassertion of MEM_RQ is based on the full and empty flags of the MCU buffers and whether the operation being performed is in the compression or decompression direction. In the explanation below of the assertion or deassertion of MEM_RQ, MCU0_E is the MCU0 buffer empty flag, MCU1_E is the MCU1 empty flag, MCU0_F is the MCU0 full flag, and MCU1_F is the MCU1 full flag.

During compression, the L64702 asserts MEM_RQ when:

- the MCU0_E flag is set, or
- the MCU1_E flag is set

During compression, the L64702 deasserts MEM_RQ when:

- the MCU0_F flag is set, and
- the MCU1_F flag is set

During decompression, the L64702 asserts MEM_RQ when:

- the MCU0_F flag is set, or
- the MCU1_F flag is set

During decompression, the L64702 deasserts MEM_RQ when:

- the MCU0_E flag is set, and
- the MCU1_E flag is set

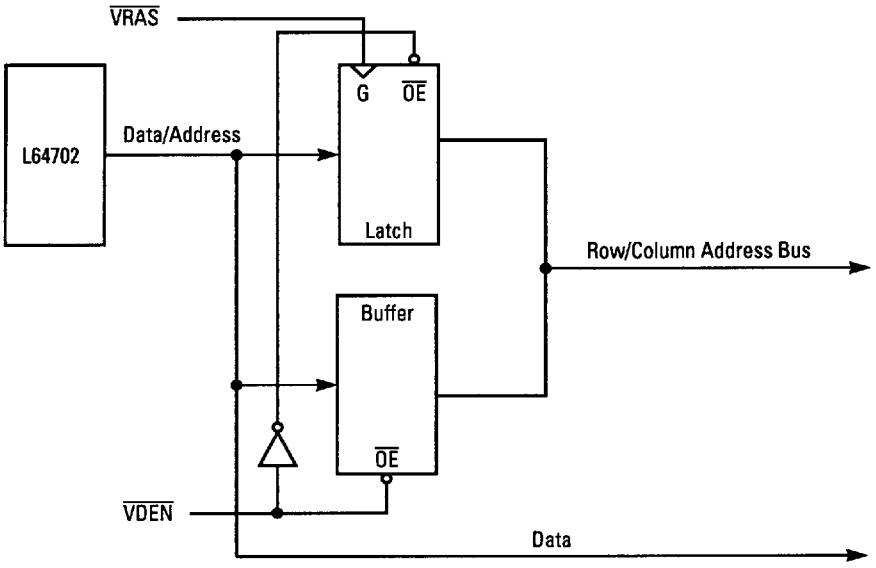
9.3 Video Bus Interface Timing

The video memory interface includes a multiplexed video address and data bus, VADB[31:0], on which addresses and image data are transmitted to external memory. The video memory interface also includes a set of control signals, which allows you to multiplex the address lines between rows and columns and to easily implement memory banks and interleaving.

The L64702 video memory row address strobe signal, $\overline{\text{VRAS}}$, latches addresses and drives the $\overline{\text{RAS}}$ inputs of the VRAM array. Use the video bus data enable signal, $\overline{\text{VDEN}}$, to generate multiplexed row and column addresses, and to indicate when data are actively being driven over the video bus. The video memory column address strobe, $\overline{\text{VCAS}}$, drives the

$\overline{CAS}$ inputs to the VRAM array. The video data transfer and output enable signal, $\overline{VTR/OE}$, drives the $\overline{TR/OE}$ inputs, and the video memory write enable signals, $\overline{VWE}[1:0]$, are the write strobes. $\overline{VWE0}$ is the write strobe for data bits 0 through 23, and $\overline{VWE1}$ is for bits 24 through 31. Figure 9.3 shows typical circuitry used to generate the multiplexed row/column addressing for external memory.

Figure 9.4 Typical Row/Column Address Multiplexing Circuitry



At the start of a memory cycle, the L64702 drives 24 bits of address on VADB[23:0] and also provides some status of the video bus cycle over the video refresh cycle, VRFC, and video transfer cycle, VTRC, lines. The VADB[27:24] lines also provide status information.

The L64702 drives the address and status information on the video bus for the first three CLK cycles. The address is valid before and after the falling edge of the $\overline{VRAS}$ signal to allow address latching using 74LS373 latches. The falling edge of the $\overline{VDEN}$ signal switches the video bus so that the bus contains data for a total of five clocks. If a memory write cycle is performed, the L64702 outputs video data on the VADB[31:0] bus as full 32-bit words. If a read cycle is performed, external memory drives data onto the bus. Either the $\overline{VCAS}$ or $\overline{VWE}[1:0]$ signals can strobe the data into external memory, depending on the type of the memory being used.

During a read cycle, the $\overline{VTR/OE}$ signal becomes active, indicating that the L64702 is performing an Auto Read from an external device. The

L64702 latches the data present on the VADB lines on the LOW-to-HIGH transition of $\overline{VTR/OE}$.

The following truth table shows the various video memory cycles performed by the L64702, and the state of the corresponding VRAM/DRAM control signals at the falling edge of the $\overline{VRAS}$ signal.

Table 9.1
VRAM/DRAM Control
Truth Table

		Control Outputs					*Status*	
Name	***Function***	***$\overline{VCAS}$***	***$\overline{VTR/OE}$***	***VWR***	***VDEN***	***VDSF***	***VRFC***	***VTRC***
REF	$\overline{CAS}$ before $\overline{RAS}$ Refresh	0	1	1	1	0	1	0
NR/NW	Normal DRAM Read or Write	1	1	1	1	0	0	0
NRT	Normal Read Transfer (DRAM -> SAM Transfer)	1	0	1	1	0	0	1
SRT	Split Read Transfer (Split DRAM -> SAM Transfer)	1	0	1	1	1	0	1
PWT	Pseudo Write Transfer (Serial Input Mode Enable)	1	0	0	1	0	0	1
AWT	Alternate Write Transfer (SAM -> DRAM Transfer)	1	0	0	1	1	0	1

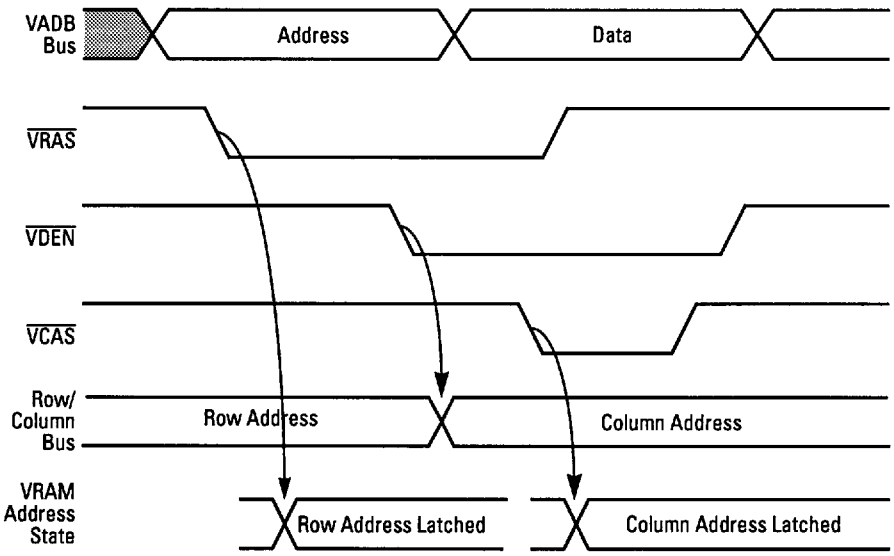
The various types of video memory cycles are summarized below:

- $\overline{CAS}$-before-$\overline{RAS}$ Refresh. The L64702 performs $\overline{CAS}$-before-$\overline{RAS}$ refresh cycles periodically according to internal requests generated by the programmable memory refresh counter.
- Normal DRAM Read or Write. The L64702 performs normal read or write operations in regular compression or decompression operation, or when the L64702 is in transfer-through mode.
- Normal Read Transfer. The L64702 performs a normal read transfer cycle to transfer an entire 32 bits of data from the DRAM core to serial access memory, (SAM), when the L64702 is in the display mode with the capture/display bit, C_D, in the SCR Register, set to zero. This cycle is performed in response to detection of the falling edge of the video transfer reset signal, $\overline{VTRS}$. You should perform a normal read transfer before any sequence of Split Read transfers, so that the SAM begins with a full 32 bits of data in it. If $\overline{VTRS}$ is driven by a VSync-type signal, the L64702 performs this cycle once per frame.

- Split Read Transfer. While in the display mode (C_D = 0), the L64702 performs a Split Read Transfer cycle after it detects the falling edge of the video transfer request signal, $\overline{\text{VTRR}}$. Split read transfers allow data to be transferred from the DRAM core to half of the SAM while the other half of the SAM is being clocked out. This method must be used for real-time data transfer.
- Pseudo Write Transfer. While in the capture mode (C_D = 1), the L64702 performs a pseudo write transfer cycle. This cycle is performed in response to the detection of the falling edge of the $\overline{\text{VTRS}}$ signal. As a result of this cycle the serial ports of the VRAM devices are switched to input mode. In normal capture operation, $\overline{\text{VTRS}}$ occurs once per frame.
- Alternate Write Transfer. While in the capture mode (C_D = 0), the L64702 performs an alternate write transfer after it detects the falling edge of $\overline{\text{VTRR}}$. The alternate write transfer is performed once per video line, as indicated by $\overline{\text{VTRR}}$.

Figure 9.5 shows the basic concept of address/data multiplexing and row/column address multiplexing. This timing reflects the function of the circuitry shown in Figure 9.3.

Figure 9.5 Row and Column Address Phases of Memory Cycle



VRAM Serial Port Control

The L64702 contains two input lines, $\overline{\text{VTRS}}$ and $\overline{\text{VTRR}}$, which control the data transfer between the VRAM serial shift register and the internal DRAM core. The C_D bit in the SCR Register determines the direction of the transfer. The C_D bit specifies whether the serial port of the VRAM device is used for image capture (C_D = 1) or for image display (C_D = 0).

The L64702 contains two programmable registers, Transfer Start Address (TSA) and Transfer Increment (TI), which determine the address that is output during the transfer cycle. The L64702 outputs all 24 bits of address during the first three clocks of the memory cycle. External circuitry, such as that shown in Figure 9.3, demultiplexes the row and column address information. In addition to TSA and TI, the L64702 contains a 24-bit Transfer Start Address Temp Register (TSAT), which holds the next transfer address for the memory transfer cycle. This register is updated prior to the update cycle.

When the L64702 is in the display mode (C_D = 0), the VRAM serial port is in the output state, and image data are clocked out from its shift register. The L64702 performs two types of read transfers, Normal Read Transfer cycle (NRT) and Split Read Transfer cycle (SRT).

The NRT is designed for initializing the two halves of the shift register with data, and the SRT is designed for clocking the data from the serial port from one half of the shift register while transferring data from the DRAM core to the second half of the shift register.

For proper operation, the NRT cycle should be initiated only one time at the start of the frame or field. When the falling edge of the $\overline{\text{VTRS}}$ signal occurs, the NRT cycle initiates. When the NRT cycle is performed, the output address is the TSA value. At the end of the NRT cycle, the TSAT Register is updated by the value of TSA + 2 * TI.

The SRT cycle should be performed periodically on a time interval equivalent to the number of clocks required to clock out half the shift register. The SRT cycle is initiated on the falling edge of the $\overline{\text{VTRR}}$ signal. When the SRT cycle is performed, the output address is the TSAT current value. At the end of the NRT cycle, the TSAT Register is updated by the value of TSAT + TI.

Figure 9.6 shows the transfer cycles during display mode.

Figure 9.6
Display Mode Serial Control Timing

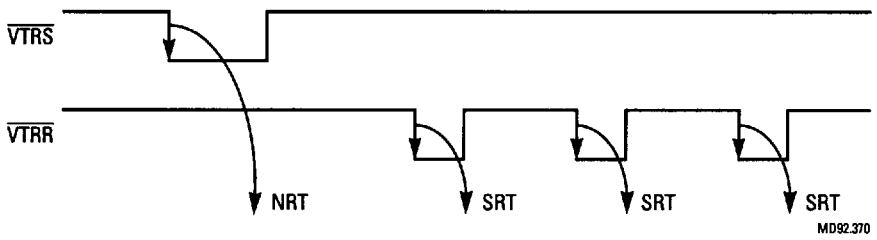


Table 9.2 shows an example of SRT cycles with their associated addresses.

Table 9.2
SRT Example

TSA = 0, TI = 256	
Cycle	*Address*
NRT	0
SRT	512
SRT	768
SRT	1024
SRT	1280
.	.
SRT	n x 256

When the L64702 is in the capture mode (C_D = 1), the VRAM serial port is in the input state, and image data are clocked into the VRAM shift register. The L64702 performs two types of write transfers: Pseudo Write Transfer (PWT) cycles and Alternate Write Transfer (AWT) cycles. The PWT cycle is initiated on the falling edge of the $\overline{\text{VTRS}}$ signal, while the AWT cycle is initiated on the falling edge of the $\overline{\text{VTRR}}$ signal.

The PWT cycle switches the VRAM serial port from an output device to an input device. When performing a PWT cycle, no actual transfer of the shift register to the DRAM core is performed. For proper image capture operation, the PWT cycle should be initiated only one time at the starting of the frame or field. When a PWT cycle is performed, the L64702 uses the TSA value as the output address, and the TSAT uses the value of the TSA Register.

The AWT cycle is designed for transferring the shift register into the DRAM core, and, unlike the SRT cycle, there is no split write transfer—the entire shift register contents are transferred during the AWT cycle. The AWT cycle should be performed periodically on a time interval equivalent to the number of clocks required to transfer all the image data per line into

the shift register. When an AWT cycle is performed, the address output is the TSAT current value. At the end of the AWT cycle, the TSAT Register is updated by the value of TSAT + TI.

Figure 9.7 shows the transfer cycles during display mode.

Figure 9.7 Capture Mode Serial Control Timing

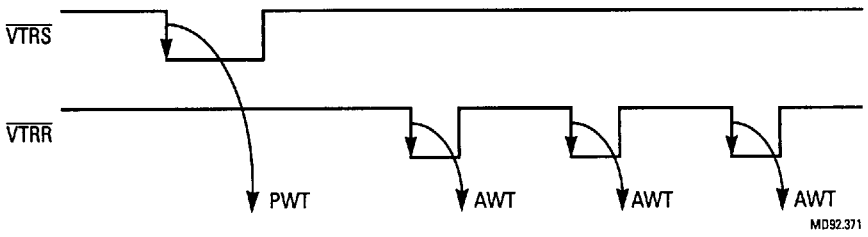


Table 9.3 shows an example of SRT cycles with their associated addresses.

Table 9.3 AWT Example

TSA = 0, TI = 256	
Cycle	***Address***
PWT	0
AWT	0
AWT	512
AWT	1024
AWT	1536
.	.
AWT	n x 512

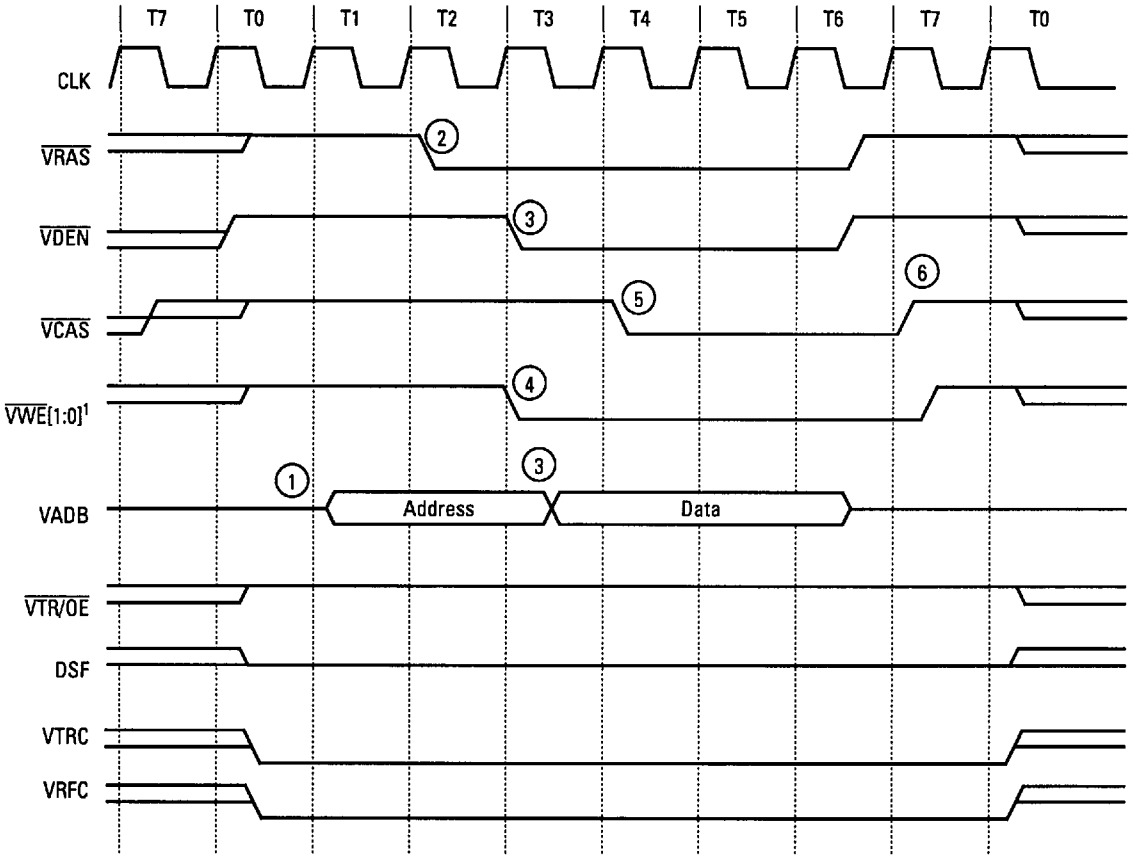
Video Memory Early Write Cycle

Figure 9.8 shows a video-memory, early-write cycle, which takes eight L64702 clocks. The L64702 executes video memory write cycles when it is in the operational mode performing image decompression, or when it is in transfer-through mode and image data are being transferred from the system memory to the video memory. The numbered steps below correspond to the annotations in the figure.

1. The address becomes active on VADB[23:0].
2. The L64702 asserts $\overline{\text{VRAS}}$ to indicate a valid row access on the row/columns bus shown in Figure 9.3.
3. The L64702 asserts $\overline{\text{VDEN}}$ to indicate the start of data on the VADB lines. $\overline{\text{VDEN}}$ has multiple purposes that are simultaneously operative: it multiplexes rows or columns in external logic and it drives the $\overline{\text{OE}}$ pin of external transceivers, such as the 74LS245.

4. $\overline{\text{VWE}}$ becomes active at the same time as $\overline{\text{VDEN}}$ and one clock before $\overline{\text{VCAS}}$ to indicate an early write cycle.
5. $\overline{\text{VCAS}}$ is asserted one CLK cycle later to indicate a valid column address.
6. For an early write, data are strobed into memory on the falling edge of $\overline{\text{VCAS}}$.

Figure 9.8
Video-Memory, Early-Write Operation Timing



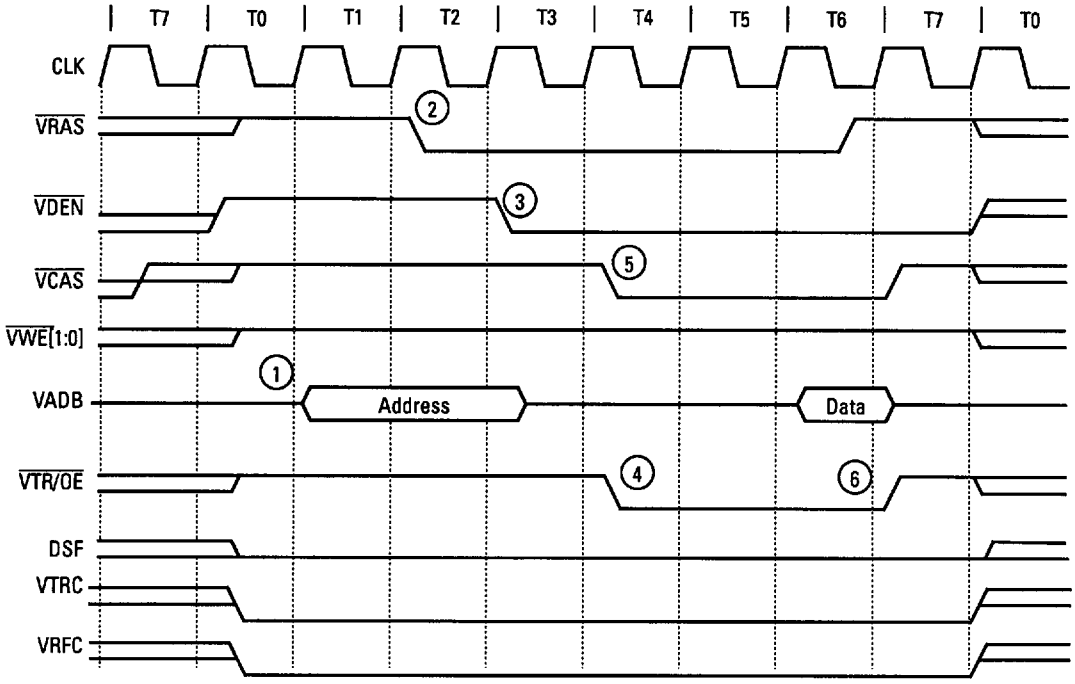
Note:
1. $\overline{\text{VWE1}}$ is not active, and stays high when the L64702 is programmed to operate in the RGBX image structure.

Video Memory Read Cycle

Figure 9.9 shows a video memory read cycle, which takes eight L64702 clocks. The L64702 executes video memory read cycles when it is in the operational mode performing image compression, or when it is in transfer-through mode and image data are being transferred from the video memory to the system memory. The numbered steps below correspond to the annotations in the figure.

1. The address becomes active on VADB[23:0].
2. The L64702 asserts $\overline{\text{VRAS}}$ to indicate a valid row access.
3. The L64702 asserts $\overline{\text{VDEN}}$ to indicate the start of data on the VADB lines. $\overline{\text{VDEN}}$ has multiple purposes that are simultaneously operative: it multiplexes rows or columns in external logic and it drives external transceivers, such as the 74LS245.
4. $\overline{\text{VTR/OE}}$ is asserted after the L64702 3-states the VADB bus. The external memory is expected to recognize a read cycle and to drive the VADB bus at this point.
5. The L64702 asserts $\overline{\text{VCAS}}$ to indicate a valid column address.
6. The L64702 latches the data from VADB[23:0] on the rising edge of the T7 CLK.

Figure 9.9
Video Memory Read
Operation Timing

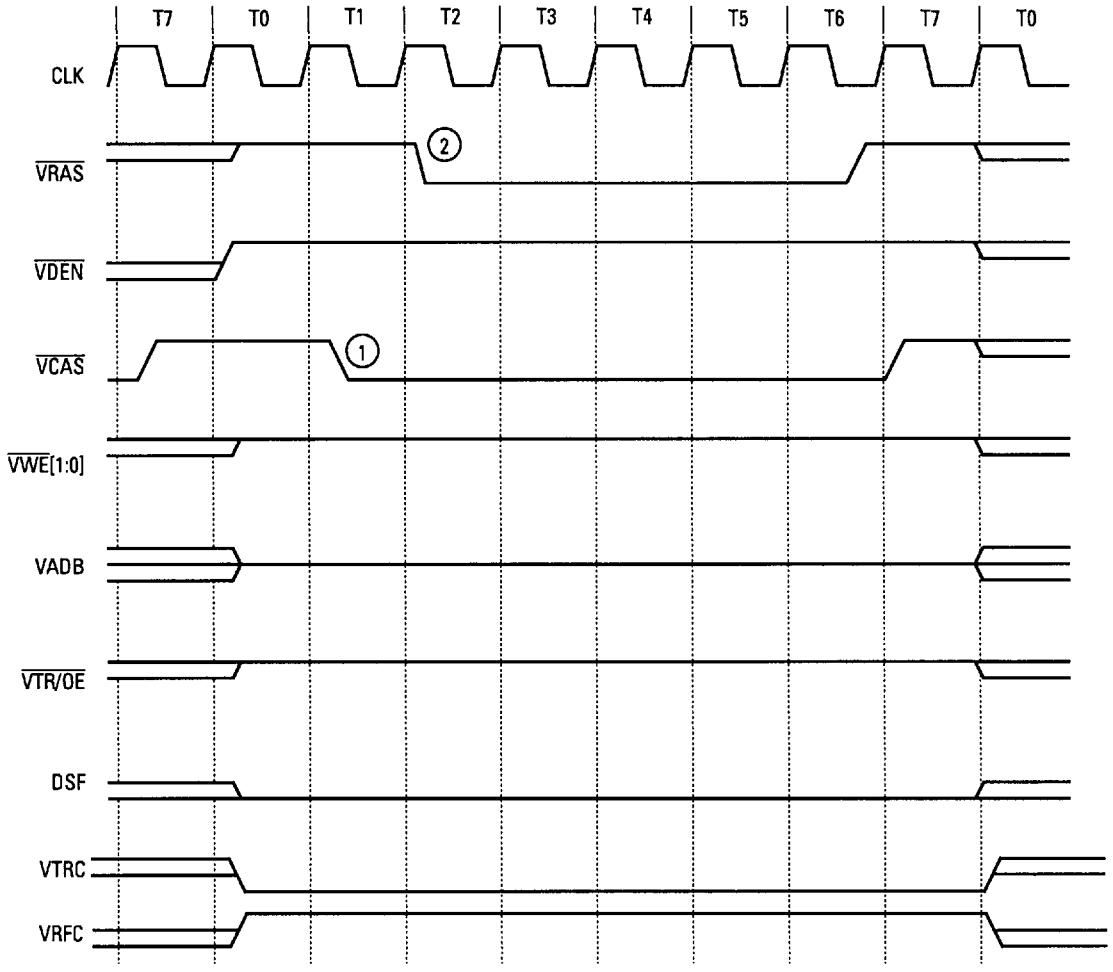


CAS-before-RAS Video Memory Refresh Cycle

The execution of video refresh cycles is user-selectable. You must set the RR field in the SCR Register to a specific valid refresh rate to initiate a video refresh cycle. Refer to Chapter 3, "Internal Registers and Data Tables," for more details regarding the RR field. An internal counter with a fixed period issues periodic requests for refresh cycles. The L64702 supports CAS-before-RAS refresh cycles only. With this scheme the L64702 relies on the internal row counter in the DRAM memory so that no refresh address is required. The basic sequence is shown in Figure 9.10. The numbered steps below refer to the annotations in the figure.

1. Assertion of VCAS when VRAS is not active indicates the start of a CAS-before-RAS refresh cycle.
2. The L64702 asserts VRAS after asserting VCAS to perform the refresh in the external device.
3. All other signals are 3-stated before, during, and after the refresh cycle.

Figure 9.10
$\overline{CAS}$-before-$\overline{RAS}$ Memory Refresh Cycle



Normal Read Transfer Cycle DRAM-TO-SAM (NRT)

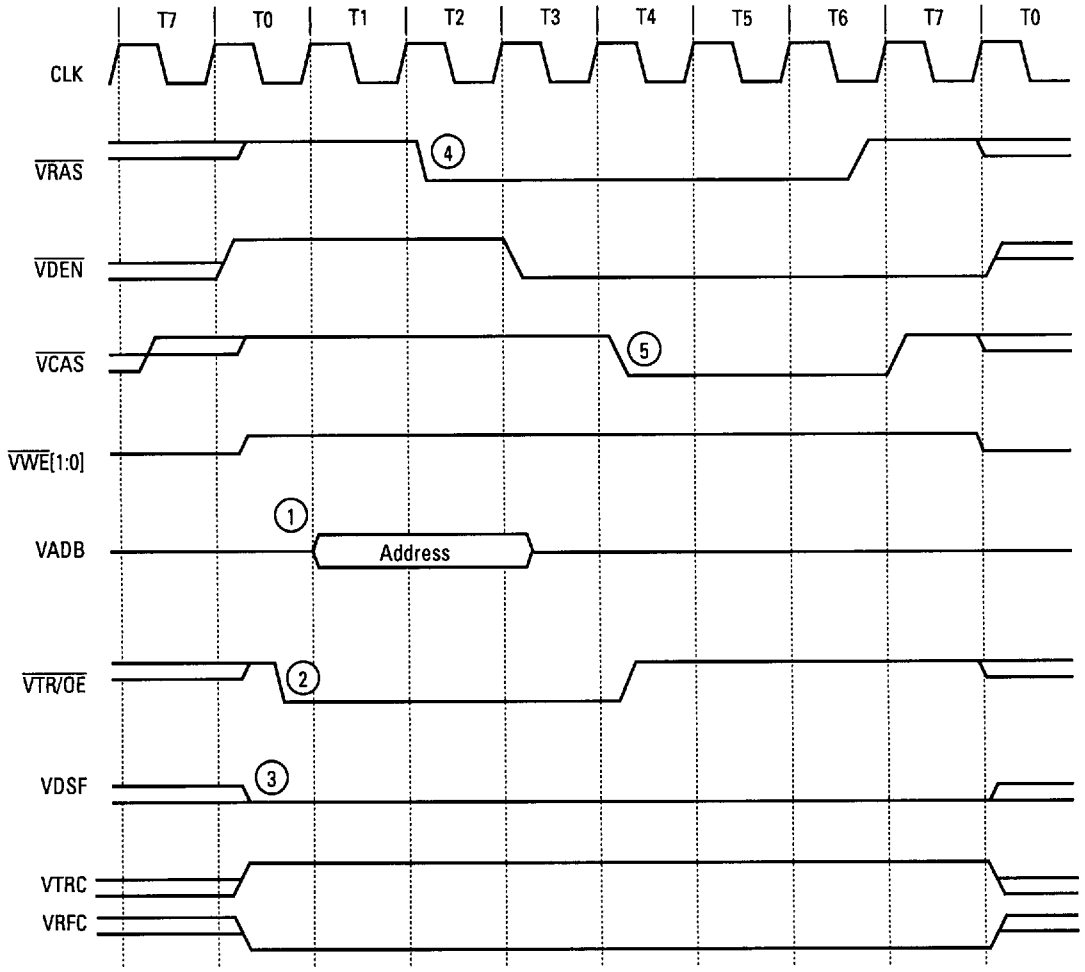
Normal read transfer cycles allow the L64702 to periodically refresh an external display device connected to the video memory. This operation results in the transfer of a complete row of data from the DRAM core to the serial shift register. The normal read transfer cycle works in conjunction with the Split Read Transfer cycle, because the VRAM device requires a normal read transfer cycle before any sequence of Split Read Transfer cycles. A normal read transfer cycle transfers a complete row of data from the DRAM to the SAM, as opposed to a Split Read Transfer that transfers half of the row.

The Read Transfer should be initiated every start of frame and is driven by the $\overline{\text{VTRS}}$ input signal. When the $\overline{\text{VTRS}}$ signal is asserted (LOW), the L64702 generates an internal request for a Read Transfer Cycle. When the request is granted the L64702 performs a Read Transfer Cycle. The address that appears on the VADB[23:0] bus is the value loaded in the Transfer Start Address Register (TSA). After completion of the cycle, the L64702 increments the contents of the TSA by the value in the Transfer Increment (TI) Register and stores the result in the Video Transfer Temp Register. If $\overline{\text{VTRS}}$ is driven from a Vertical Sync signal, the L64702 performs the read transfer cycle once per frame.

Figure 9.11 shows the timing of the memory to register cycle. The following steps describe the operation:

1. The address becomes active on VADB[23:0].
2. The L64702 asserts $\overline{\text{VTR/OE}}$ to indicate the memory to register transfer.
3. The L64702 asserts VDSF LOW, indicating a Normal Read Transfer operation.
4. The L64702 asserts $\overline{\text{VRAS}}$ LOW. On the $\overline{\text{VRAS}}$ falling edge, external memory may latch the row transfer address.
5. The L64702 asserts $\overline{\text{VCAS}}$ LOW. On the $\overline{\text{VCAS}}$ falling edge, external memory may latch the column transfer address.

Figure 9.11
Normal Read
Transfer Cycle
DRAM-to-SAM
(NRT)



Split Read Transfer Cycle DRAM-to-SAM (SRT)

The Split Read Transfer cycle is built into the L64702 and utilizes the VRAM device feature of reading the video shift register as two separate shift registers. While reading data from the DRAM core to one half of the shift register, the second shift register is clocked out. This feature provides better utilization of the memory for display, and reduces the timing constraints applied to the serial port of the VRAM device without the Split Read Transfer feature. In fact, this feature allows real time screen refresh of display screens with a display pitch larger than the VRAM shift register size.

A Split Read Transfer cycle is initiated on the falling edge of the $\overline{\text{VTRR}}$ signal. Every cycle transfers data from the DRAM core to half of the shift register, while the other part can be clocked out via the serial port of the VRAM device. The most-significant bit of the column address determines which half of the shift register the transfer cycle will affect. If the most-significant bit is zero, the least-significant half of the register is accessed.

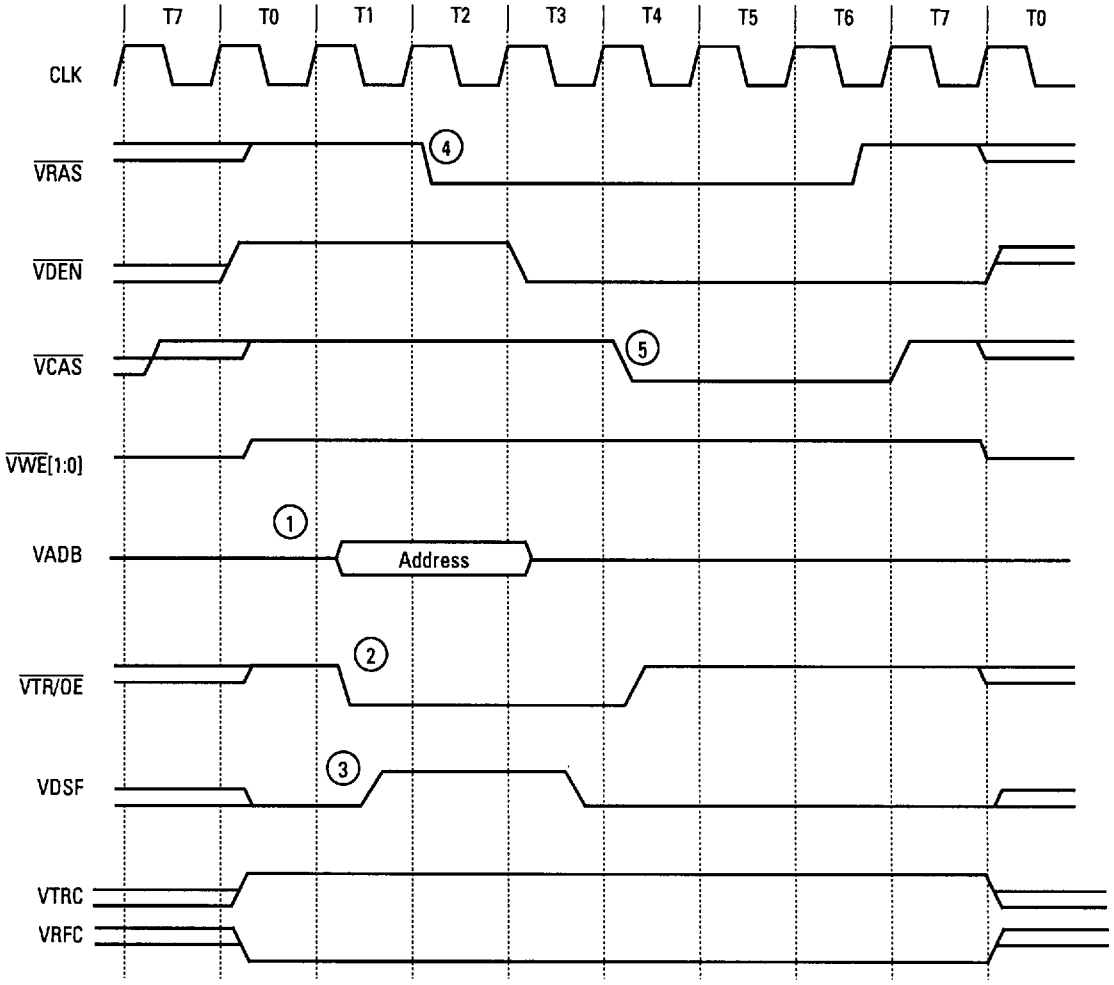
The most-significant half of the register is accessed if the most-significant column address bit is one. The logic level of this bit is determined on the falling edge of the $\overline{\text{VRAS}}$ signal. To maintain continuous data flow through the serial port, $\overline{\text{VTRR}}$ must be active during the time interval between the end of clocking one half of the register and the completion of clocking the second half. Prior to beginning a sequence of the Split Read Transfer cycles, the L64702 must perform a Normal Read Transfer cycle using the $\overline{\text{VTRS}}$ signal to initialize the two halves of the shift register.

The address that appears on the VADB[23:0] bus during the Split Read Transfer operation is contained in the Transfer Start Address Temp Register (TSAT). After the cycle has been performed, the L64702 updates the contents of the TSAT Register by the value of TSAT + TI. TI is the Transfer Increment Register. Figure 9.12 shows the timing of the memory to register cycle. The following steps describe the operation:

1. The address becomes active on VADB[23:0].
2. The L64702 asserts $\overline{\text{VTR/OE}}$ to indicate the memory to register transfer.
3. The VDSF signal is asserted HIGH, indicating start of the Split Read Transfer operation.
4. The L64702 asserts $\overline{\text{VRAS}}$ LOW. On the $\overline{\text{VRAS}}$ falling edge external memory may latch the row transfer address.

5. The L64702 asserts $\overline{\text{VCAS}}$ LOW. On the $\overline{\text{VCAS}}$ falling edge external memory may latch the column transfer address.

Figure 9.12
Split Read Transfer Cycle DRAM-to-SAM (SRT)



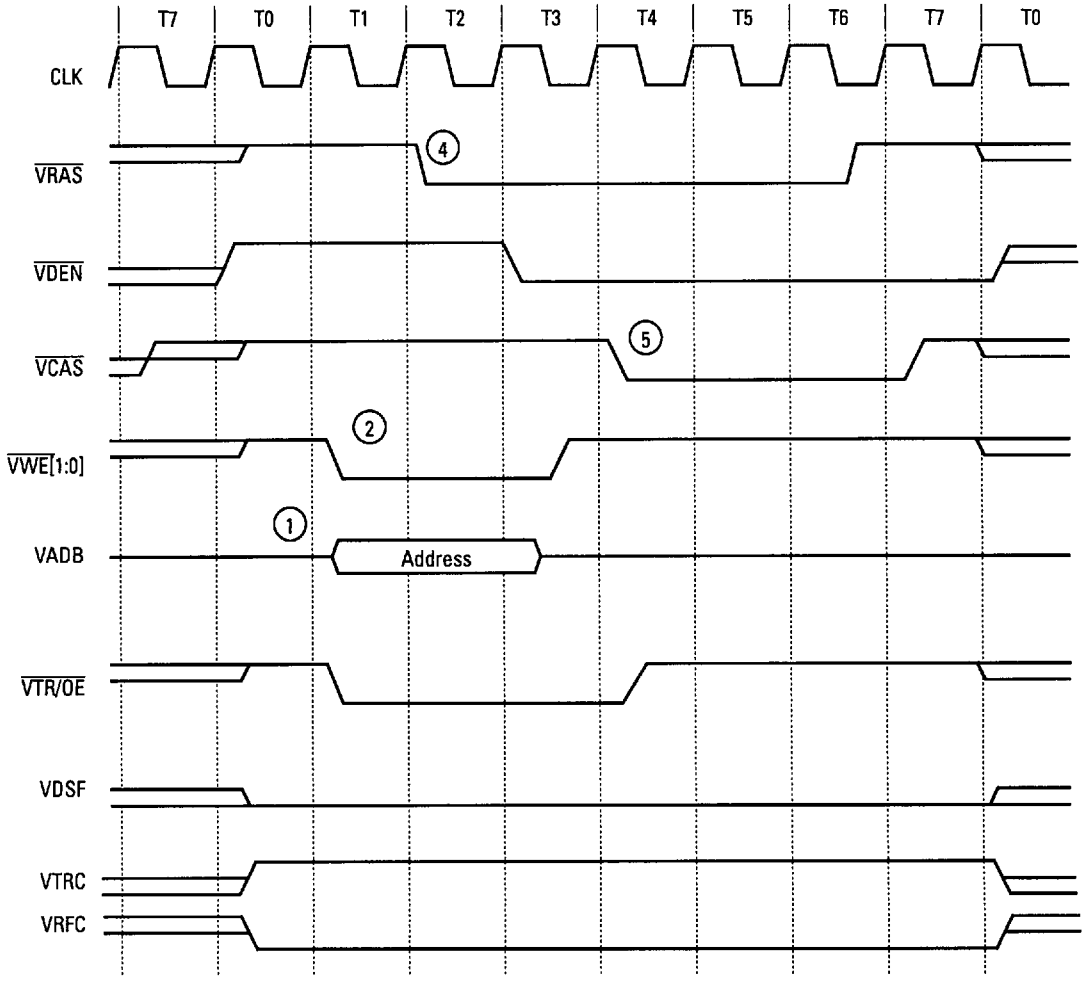
Pseudo-Write Transfer Cycle (PWT)

The Pseudo-Write Transfer cycle is performed when the L64702 is in capture mode (C_D = 1). The L64702 performs this cycle when it detects the falling edge of $\overline{\text{VTRS}}$. As a result of this cycle, the serial ports of the VRAM devices are enabled and the serial port direction is switched from output to input. No transfer of the shift register to the DRAM core is performed.

Figure 9.13 shows the timing of the memory to register cycle. The following steps describe the operation:

1. The address becomes active on VADB[23:0].
2. The L64702 asserts $\overline{\text{VWE}}$ to indicate the shift register to memory transfer.
3. The L64702 asserts VDSF LOW, indicating a Pseudo-Write Transfer operation.
4. The L64702 asserts $\overline{\text{VRAS}}$ LOW. On the $\overline{\text{VRAS}}$ falling edge external memory may latch the row transfer address.
5. The L64702 asserts $\overline{\text{VCAS}}$ LOW. On the $\overline{\text{VCAS}}$ falling edge external memory may latch the column transfer address.

Figure 9.13
Pseudo Write
Transfer Cycle
(PWT)



Alternate Write Transfer SAM-to-DRAM (AWT)

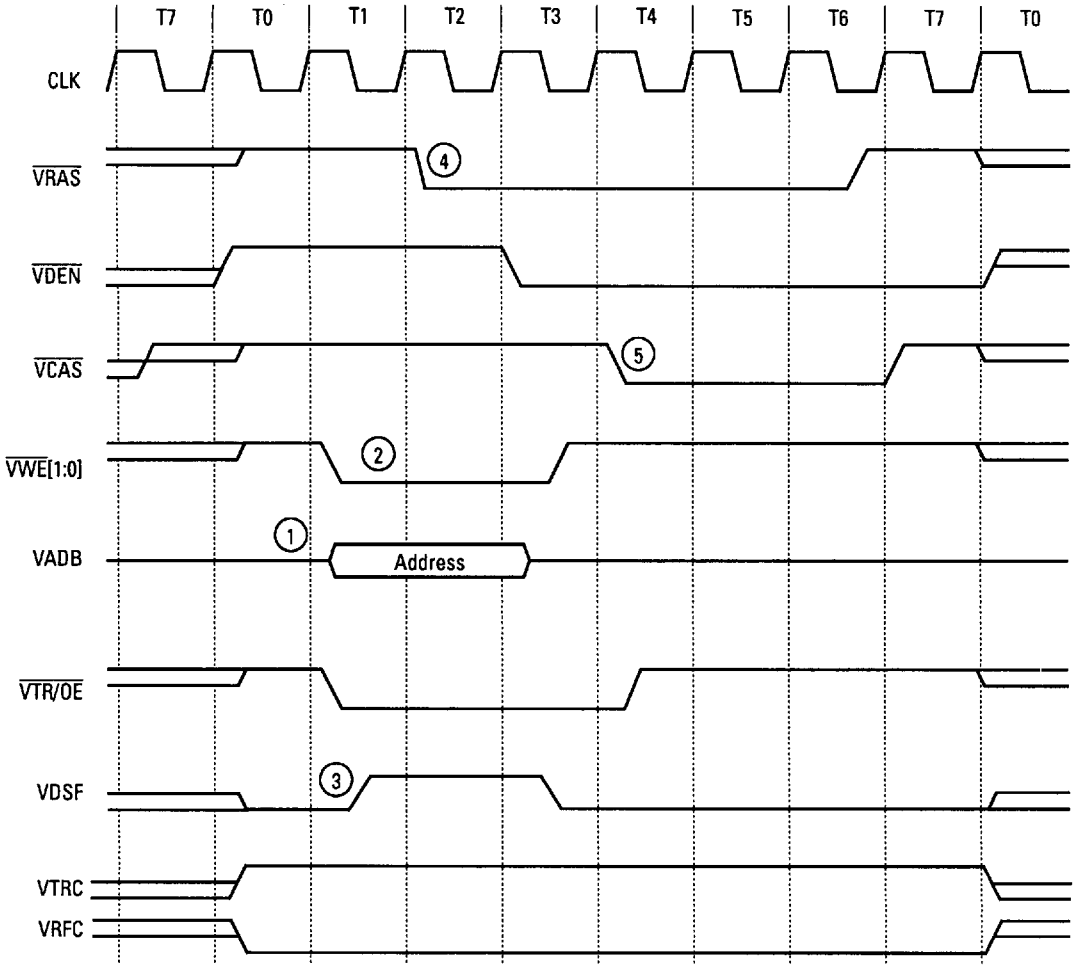
The Alternate Write Transfer cycle (AWT) allows the L64702 to transfer data from the VRAM shift register into the DRAM core. The L64702 performs this cycle when it is in the capture mode (C_D = 1) and detects the falling edge of $\overline{\text{VTRR}}$.

The address that appears on the VADB[23:0] bus during the Alternate Write Transfer operation is contained in the Transfer Start Address Temp Register (TSAT). After the cycle has been performed, the L64702 updates the contents of the TSAT Register by the value of TSAT + TI. TI is the Transfer Increment Register.

Figure 9.14 shows the timing of the memory to register cycle. The following steps describe the operation:

1. The address becomes active on VADB[23:0].
2. The L64702 asserts $\overline{\text{VWE}}$ to indicate the shift register to memory transfer.
3. The L64702 asserts VDSF HIGH, indicating an Alternate Write transfer operation.
4. The L64702 asserts $\overline{\text{VRAS}}$ LOW. On the $\overline{\text{VRAS}}$ falling edge, external memory may latch the row transfer address.
5. The L64702 asserts $\overline{\text{VCAS}}$ LOW. On the $\overline{\text{VCAS}}$ falling edge, external memory may latch the column transfer address.

Figure 9.14
Alternate Write
Transfer Cycle
(AWT)

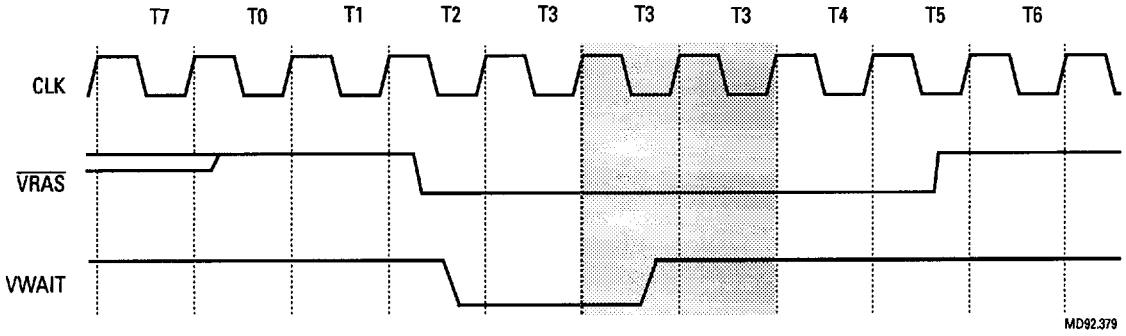


Video Memory Cycle with Wait States

All previous timing assumes that the $\overline{\text{VWAIT}}$ input signal is not asserted during the memory access cycle, which means that the memory cycles are executed without wait states. Assertion of $\overline{\text{VWAIT}}$ indicates to the L64702 video bus controller that external memory is not yet ready to complete a memory cycle, and that the current cycle should be extended. Similar to the other video control inputs, the L64702 synchronizes $\overline{\text{VWAIT}}$ with the L64702 CLK by sampling it on each rising edge of the T2, T3, and T4 CLKs (following the falling edge of the $\overline{\text{VRAS}}$ signal). $\overline{\text{VWAIT}}$ extends the memory cycle for all access types.

Figure 9.15 shows the basic operation of $\overline{\text{VWAIT}}$. When the external system asserts $\overline{\text{VWAIT}}$ LOW, the L64702 stretches the video bus cycle as long as $\overline{\text{VWAIT}}$ remains active, and when $\overline{\text{VWAIT}}$ goes inactive again, the bus cycle continues.

Figure 9.15
Video Memory Cycle With Wait States



Chapter 10
System Interface Port

The system interface port transfers compressed video image data as well as register and table data to the L64702. The system interface supports 16-bit data transfers between the external system and the L64702. The data transfers may be performed in either the DMA transfer mode or CPU transfer mode. The system interface port of the L64702 is designed to interface as a slave port with either an external CPU or DMA Controller.

The L64702 system interface operates asynchronously with respect to the external system; that is, the system interface signals are not related to the L64702 clock signal. $\overline{\text{SINT}}$ indicates that the L64702 has set one or more of its unmasked interrupt flags.

Chapter 10 has two sections:

- Section 10.1, CPU Transfer Mode
- Section 10.2, DMA Transfer Mode

10.1 CPU Transfer Mode

This section provides functional waveforms for the following CPU operations:

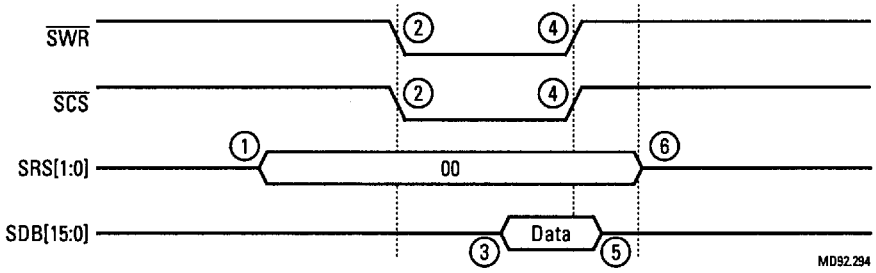
- CPU Write to the Address Pointer Register (APR)
- CPU Read from the Address Pointer Register (APR)
- CPU Read from the Status Register (STS)
- CPU Write to the System Mode Register (SMR)
- CPU Read from the Registers and Tables
- CPU Write to the Registers and Tables
- CPU Read from the FIFO
- CPU Write to the FIFO

CPU Write to the Address Pointer Register (APR)

Figure 10.1 shows the CPU write cycle timing when writing to the Address Pointer Register (APR). The APR is assigned to Group 0 and is accessed when SRS[1:0] = 0. The L64702 uses this register to address Group 2, the registers and tables group. The L64702 automatically increments the APR by one after reading or writing Group 2 tables and registers. The following sequence lists the steps the CPU must complete to write to the APR.

1. The CPU sets the SRS[1:0] lines to 00_2.
2. The CPU asserts the $\overline{SWR}$ and $\overline{SCS}$ signals.
3. The CPU drives the data lines SDB[15:0].
4. The CPU deasserts the $\overline{SWR}$ or $\overline{SCS}$ signals, indicating the end of the write cycle.
5. The CPU stops driving the SDB[15:0] lines, and either 3-states them or sets them to a don't care value.
6. The CPU stops driving the SRS[1:0] lines, and either 3-states them or sets to a don't care value.

Figure 10.1
CPU Write to the APR Register



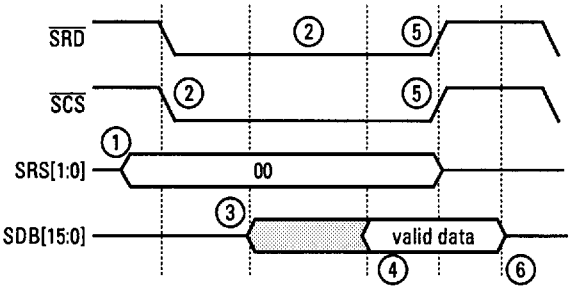
CPU Read from the Address Pointer Register (APR)

Figure 10.2 shows the CPU read cycle timing when reading from the Address Pointer Register (APR). The APR is assigned to Group 0 and is accessed when SRS[1:0] = 0. This register is used for addressing Group 2, the registers and tables group. The L64702 automatically increments the APR by one after reading or writing the Group 2 tables and registers. Reading the APR Register yields the current value of this register. The following sequence lists the steps the CPU must complete to read the APR.

1. The CPU sets the SRS[1:0] lines to 00_2.
2. The CPU asserts the $\overline{SRD}$ and $\overline{SCS}$ signals.

3. The L64702 begins to drive the data lines SDB[15:0] with the current value of the APR Register. At this point the data output from the L64702 are not yet stable or valid.
4. After a register access delay period, the data lines SDB[15:0] contain a valid value for the APR Register.
5. The CPU stops driving the $\overline{SCS}$, $\overline{SRD}$, and SRS[1:0] signals to indicate the end of the APR read cycle.
6. The L64702 stops driving the SDB[15:0] lines, and then either 3-states them or sets them to a don't care value.

Figure 10.2
CPU Read from the APR Register



CPU Read from the Status Register (STS)

Figure 10.2 shows the CPU read cycle timing when reading from the Status Register (STS). The Status Register (STS) is assigned to Group 1 and is accessed when SRS[1:0] = 1. The Status Register is a read-only register. Any attempt to write to this address changes the content of the SMR Register, which resides at the same address. The STS read cycle is internally synchronized with the L64702 clock, *which requires the read cycle active-LOW interval to be a minimum of five clocks*, and the read recovery time interval to be a minimum of one clock. Once a read cycle begins, the contents of the STS Register on the SDB bus are not allowed to change even if the internal L64702 status changes during the read cycle. STS represents the status of the L64702 at the beginning of the read cycle. The following sequence lists the steps the CPU must complete to read from the STS:

1. The CPU sets the SRS[1:0] lines to 01_2.
2. The CPU asserts the $\overline{\text{SRD}}$ and $\overline{\text{SCS}}$ signals for a minimum of five clocks.
3. The L64702 begins to drive the data lines SDB[15:0] with the current value of the STS Register. At this point the data output from the L64702 are not yet stable or valid.
4. After a register access delay period, the data lines SDB[15:0] contain a valid value for the STS Register.
5. The CPU deasserts the $\overline{\text{SCS}}$, $\overline{\text{SRD}}$, and SRS signals for a minimum of one clock, indicating the end of the STS read cycle.
6. The CPU stops driving the SDB[15:0] lines, and either 3-states them or sets them to a don't care value.

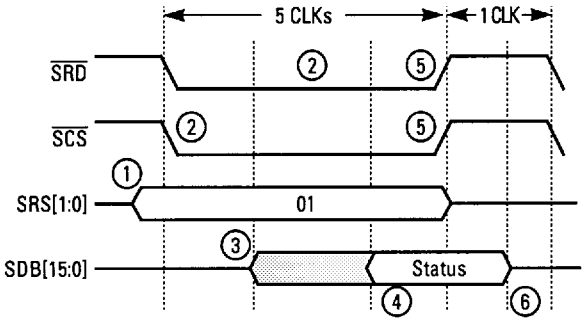


Figure 10.3
CPU Read from the STS Register

CPU Write to the System Mode Register (SMR)

Figure 10.4 shows the CPU write cycle timing when writing to the System Mode Register (SMR). The System Mode Register (SMR) is assigned to Group 1 and is accessed when SRS[1:0] = 1. The SMR Register is a write-only register, so any attempt to read from the SMR results in reading the STS Register, because the STS Register resides at the same address.

The SMR write cycle is internally synchronized with the L64702 clock, which requires the write-cycle, active-LOW interval to be a minimum of one clock, and the write recovery time interval to be a minimum of five clocks. The data value on the SDB bus is strobed into the L64702 on the LOW-to-HIGH transition of the $\overline{SWR}$ signal. The following sequence lists the steps the CPU must complete to write to the SMR:

1. The CPU sets the SRS[1:0] lines to 01_2.
2. The CPU asserts the $\overline{SWR}$ and $\overline{SCS}$ signals for minimum of one clock cycle.
3. The CPU drives the data lines SDB[15:0].
4. The CPU deasserts the $\overline{SWR}$ or $\overline{SCS}$ signals for a minimum write recovery time of five clocks, indicating the end of the SMR write cycle.
5. The CPU stops driving the SDB[15:0] lines, and either 3-states them or sets them to a don't care value.
6. The CPU stops driving the SRS[1:0] lines, and then either 3-states them or sets them to a don't care value.

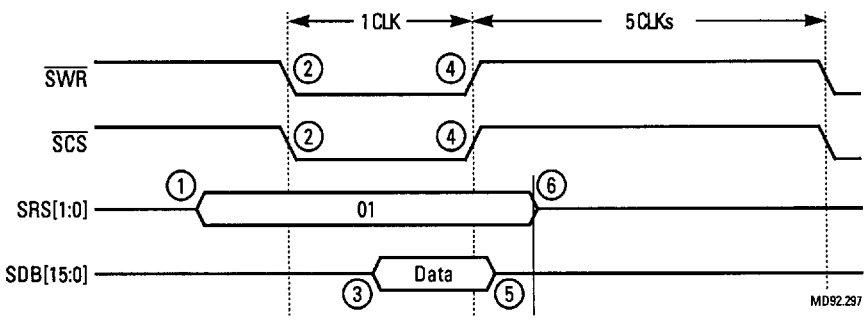


Figure 10.4 CPU Write to the SMR Register

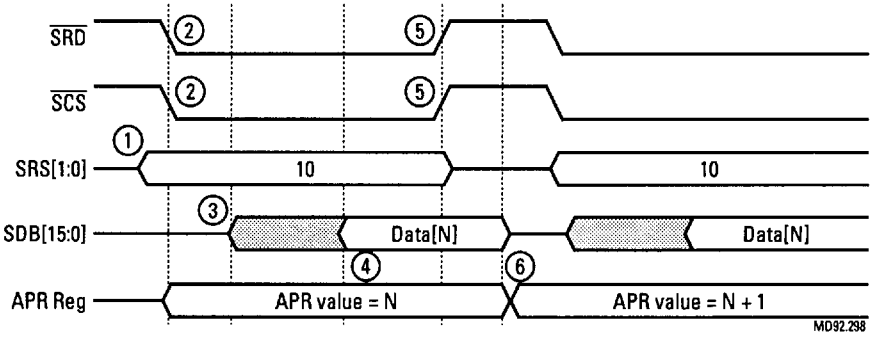
CPU Read from the Registers and Tables

Figure 10.5 shows the CPU read cycle timing when reading from the configuration registers and JPEG tables contained within the L64702. These registers and tables are assigned to Group 2 and are accessed when SRS[1:0] = 2. Each 16-bit word within Group 2 is located at a unique address in the L64702. Prior to the execution of a Group 2 read cycle, you must load the APR Register with the address of the first register or table location to be accessed. With each LOW-to-HIGH transition of the $\overline{SRD}$ signal, the L64702 increments the APR address information by one, so that the subsequent read cycle accesses the next sequential register or table location.

The following sequence lists the steps the CPU must complete to read the Group 2 registers and tables.

1. The CPU sets the SRS[1:0] lines to 10_2.
2. The CPU asserts the $\overline{SRD}$ and $\overline{SCS}$ signals.
3. The L64702 begins to drive the data lines SDB[15:0] with the current value of the Group 2 register or table being accessed. At this point the data output from the L64702 are not yet stable or valid.
4. After a delay period, the data lines SDB[15:0] contain a valid value for the register or table pointed to by the APR Register.
5. The CPU deasserts the $\overline{SCS}$, $\overline{SRD}$, and SRS signals, indicating the end of the read cycle. The APR increments by one.
6. The L64702 stops driving the SDB[15:0] lines, and then either 3-states them or sets them to a don't care value. To perform the next sequential read, repeat the above steps. The APR Register does not need not be reinitialized.

Figure 10.5
CPU Read from the Registers and Tables



CPU Write to the Registers and Tables

Figure 10.6 shows the CPU write cycle timing when writing to the configuration registers and JPEG tables contained within the L64702. The write cycle is similar to the read cycle in that you must load the APR Register with the address location of the table or register to be written. The L64702 increments the APR by one on every LOW-to-HIGH transition of the $\overline{SWR}$ signal. The following sequence lists the steps the CPU must complete to write the Group 2 registers and tables.

1. The CPU sets the SRS[1:0] lines to 10_2.
2. The CPU asserts the $\overline{SWR}$ and $\overline{SCS}$ signals.
3. The CPU drives the data lines SDB[15:0].
4. The CPU deasserts the $\overline{SCS}$ and $\overline{SRD}$ signals, indicating the end of the write cycle. The APR increments by one.
5. The CPU stops driving the SDB[15:0] lines, and then either 3-states them or sets them to a don't care value.
6. The CPU stops driving the SRS[1:0] lines, and then either 3-states them or sets them to a don't care value. To perform the next sequential read, repeat the above steps. The APR Register does not need not be reinitialized.

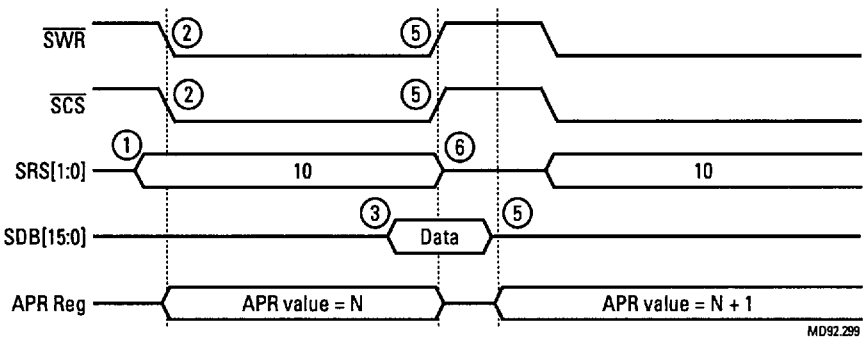


Figure 10.6
CPU Write to the Registers and Tables

CPU Read from the FIFO

Figure 10.7 shows the CPU read cycle timing when reading the L64702 FIFO. The Group 3 FIFO is read either during compression or in read-through mode.

The L64702 is operating in the data compression mode when ENC = 1, SRS[1:0] = 3, and the START bit in the SMR Register is set. With this setup, the CPU can read compressed data from the FIFO and then write it into system memory.

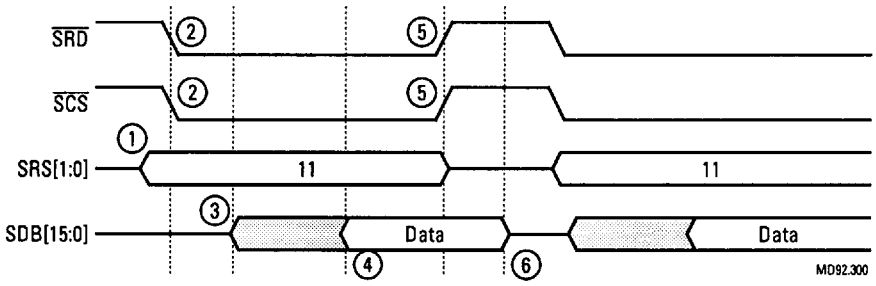
When ENC = 1 and the TRTH bit is set to one, the L64702 is in the read-through mode.

Bits in the SMR and STS Registers allow you to monitor the FIFO empty or FIFO full conditions. Before accessing the FIFO, you must initialize the burst level and check for the valid condition to start reading the FIFO. Refer to Section 3.5, "Group 3 FIFO," for more information on accessing the internal FIFO.

The following sequence lists the steps the CPU must complete to read the FIFO when the valid conditions for reading the FIFO are met:

1. The CPU sets the SRS[1:0] lines to 11_2.
2. The CPU asserts the $\overline{SRD}$ and $\overline{SCS}$ signals.
3. The L64702 begins to drive the data lines SDB[15:0] from the FIFO. At this point the data output from the L64702 are not yet stable or valid.
4. After a delay period, the data lines SDB[15:0] contain a valid value for the FIFO compressed data.
5. The CPU deasserts the $\overline{SCS}$, $\overline{SRD}$, and SRS signals to indicate the end of the FIFO read cycle.
6. The L64702 stops driving the SDB[15:0] lines, then 3-states them.

Figure 10.7 CPU Read from the FIFO



CPU Write to the FIFO

Figure 10.8 shows the CPU write cycle timing when writing the L64702 FIFO. The Group 3 FIFO is written to either during decompression on in write-through mode.

The L64702 operates in the data decompression mode when ENC = 0, SRS[1:0] = 3, and the START bit in the SMR Register is set. With this setup, the CPU can write decompressed data to the FIFO from the system memory.

When ENC = 0 and the TRTH bit is set to one, the L64702 is in the write-through mode.

Bits in the SMR and STS Registers allow you to monitor the FIFO empty or FIFO full conditions. Before accessing the FIFO, you must initialize the burst level and check for the valid condition to start reading the FIFO. Refer to Section 3.5, "Group 3 FIFO," for more information on accessing the internal FIFO.

The following sequence lists the steps the CPU must complete to write the FIFO when the valid conditions for writing the FIFO are met:

1. The CPU sets the SRS[1:0] lines to 11_2.
2. The CPU asserts the $\overline{\text{SWR}}$ and $\overline{\text{SCS}}$ signals.
3. The CPU drives the data lines SDB[15:0] with decompressed data to be written into the FIFO.
4. The CPU deactivates $\overline{\text{SWR}}$ or $\overline{\text{SCS}}$, which indicates the end of the write cycle.
5. The CPU stops driving the SDB[15:0] lines, and then either 3-states them or sets them to a don't care value.
6. The CPU stops driving the SRS[1:0] lines, and then either 3-states them or sets them to a don't care value.

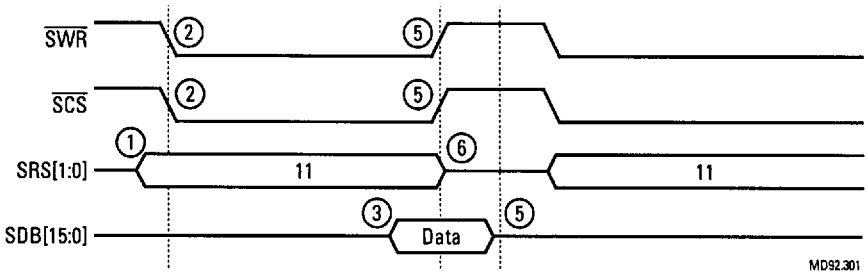


Figure 10.8 CPU Write to the FIFO

10.2 DMA Transfer Mode

This section provides functional waveforms for the following CPU operations:

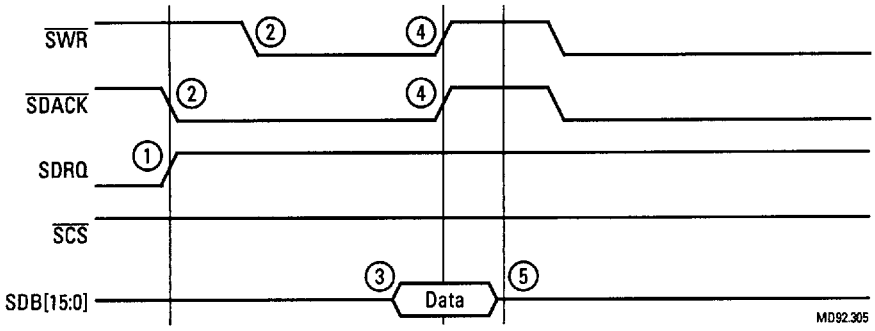
- DMA Write Operation
- DMA Read Operation

DMA Write Operation

Figure 10.9 illustrates a typical DMA write cycle. A DMA write cycle is defined as a read from system memory using the DMA controller's memory-mapped read signal, and a write to the L64702 using the DMA controller's I/O-mapped write signal. The read and write are simultaneous, performed under the direction of the DMA controller. A DMA write operation may consist of transfers to either Group 2 tables and registers or to the Group 3 FIFO. The following sequence lists the steps that take place during a DMA write operation:

1. The L64702 asserts SDRQ to indicate that the L64702 is requesting DMA service.
2. The DMA controller asserts the L64702 $\overline{\text{SDACK}}$ signal to acknowledge the request and activate $\overline{\text{SWR}}$ to start the write cycle.
3. The data that has been read from system memory are then presented on SDB[15:0] with sufficient setup time prior to the rising edge of $\overline{\text{SWR}}$.
4. The DMA controller writes data to the L64702 by asserting $\overline{\text{SWR}}$. The rising edge of $\overline{\text{SWR}}$ causes data to be written to the L64702 and indicates the end of the DMA transfer. $\overline{\text{SDACK}}$ may be deasserted or stay asserted for the next write cycle.
5. The DMA controller retains valid data on SDB[15:0] for the required hold time after the rising edge of $\overline{\text{SWR}}$.

Figure 10.9 DMA Write Cycle Timing

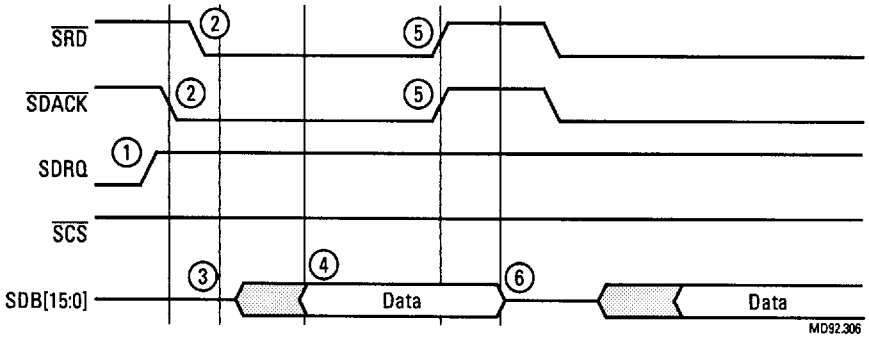


DMA Read Operation

Figure 10.10 illustrates a typical DMA read cycle. A DMA read cycle is defined as a read from the L64702 using the DMA controller's I/O-mapped read signal and a write to system memory using the DMA controller's memory-mapped write signal. The read and write are simultaneous, performed under the direction of the DMA controller. A DMA read operation may consist of transfers from either Group 2 tables and registers or from the Group 3 FIFO. The following sequence lists the steps that take place during a DMA read operation.

1. The L64702 asserts SDRQ, indicating that the L64702 is requesting DMA service.
2. The DMA controller asserts the $\overline{\text{SDACK}}$ signal to acknowledge the request and activates $\overline{\text{SRD}}$ to start the read cycle.
3. The L64702 starts to drive data onto the SDB[15:0] bus. At this time the data from the L64702 are not yet valid.
4. After a sufficient data access delay, valid data appear on the SDB bus.
5. On the rising edge of $\overline{\text{SRD}}$, the read operation is complete, indicating the end of the DMA read cycle. The DMA controller then writes the data just read from the L64702 into system memory. $\overline{\text{SDACK}}$ may be deasserted or stay asserted for the next read cycle.
6. The data that is output from the L64702 stay valid for the required hold time after the rising edge of $\overline{\text{SRD}}$.

Figure 10.10 DMA Read Cycle Timing



Chapter 11
System Software Control

This chapter summarizes the L64702 main control and status bits, which allow an easy interface between your software driver and the chip. The concept is basically similar to the interface between slave peripheral devices and a CPU or DMA controller in a PC system environment. The software driver that interfaces with the L64702 uses polling, interrupts, and DMA channel transfers for initializing the L64702 as well as monitoring the compression and decompression tasks. Flowcharts of the control program are provided for both compression and decompression.

This chapter contains three sections:

- Section 11.1, Initialization Task
- Section 11.2, Image Compression Task
- Section 11.3, Image Decompression Task

11.1 Initialization Task

The L64702 is in the initialization mode when you set the INIT bit in the SMR Register and set the INIT flag in the STS Register. Either CPU or DMA service can initialize the L64702. To initiate DMA service, set the DE bit in the SMR Register. The APR Register should be pre-initialized to the start address in Group 2 to make sure that the initialization data are written in the correct location. During image compression or decompression operation, occasional breaks in operation may be required to reprogram the L64702 tables or registers. Simply reset the START bit in the SMR Register to pause the operation. As a result of this pause, the L64702 sets the INIT flag in the STS Register. After the INIT flag is set, you can set the INIT bit in the SMR Register, which allows access to Group 2. Reading from Group 2 during operation is useful for testing and debugging purposes.

Do not write to Group 2 while operation is paused. You may change the initial values of tables or registers, and thus violate the operation in

progress, causing a data stream breakdown and corruption of the results. You must handle carefully the changing of tables and registers during operation.

The L64702 initialization procedure using the CPU transfer mode is outlined in the following steps:

Step 1. Read the INIT bit in the STS Register.

Step 2. If INIT = 1 then continue to Step 3, or else go back to Step 1.

Step 3. Set the INIT bit in the SMR Register.

Step 4. Write the start address to the APR Register.

Step 5. Write the necessary initialization words to Group 2.

Step 6. Repeat Steps 4 and 5 until initialization is finished.

The L64702 initialization procedure using the DMA transfer mode is outlined in the following steps:

Step 1. Read the INIT bit in the STS Register.

Step 2. If INIT = 1 then continue to Step 3, or else go back to Step 1.

Step 3. Initialize the system DMA Controller.

Step 4. Set the INIT and DE bits in the SMR Register.

Step 5. Write the start address to the APR Register.

Step 6. Wait for the TC interrupt from the DMA Controller.

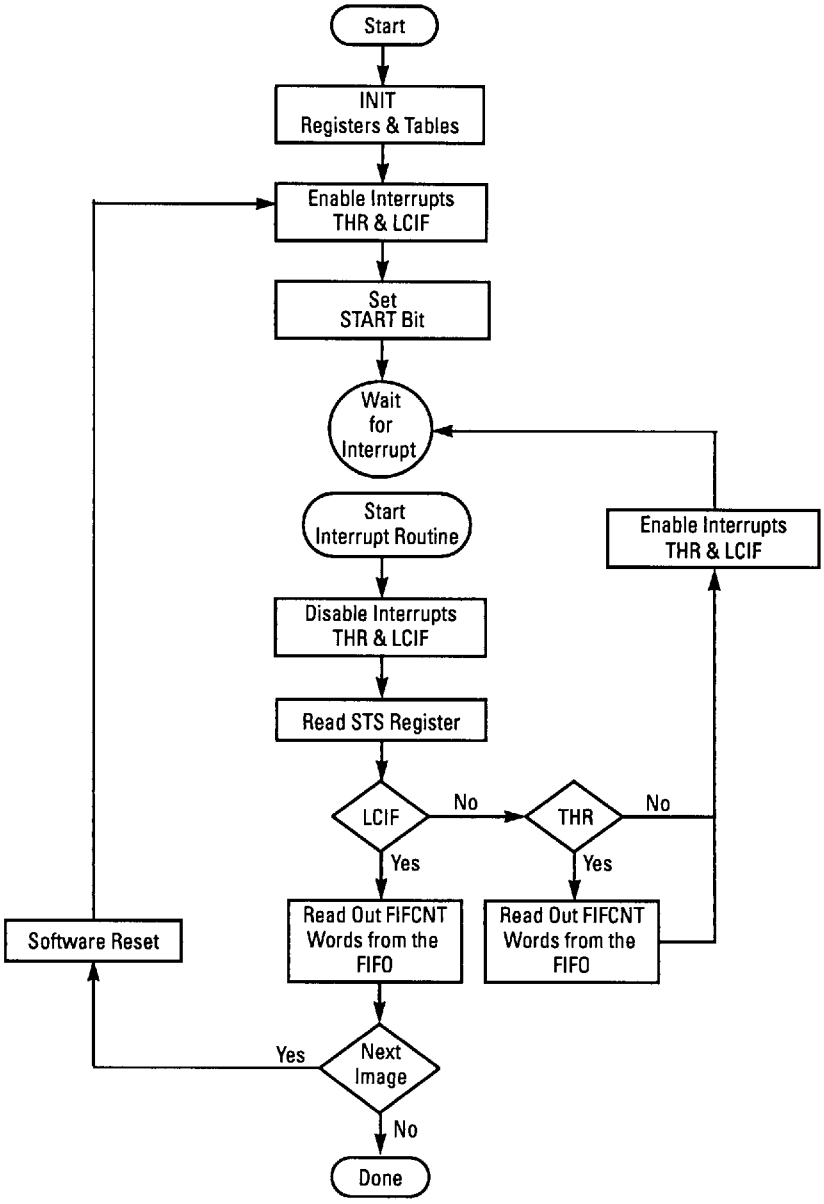
Step 7. Repeat Steps 5 and 6 until initialization is finished.

11.2 Image Compression Task

Figure 11.1 shows a sample program flowchart that uses the L64702 internal status flags and interrupt pin to control the flow of compressed data out of the L64702 FIFO.

The program uses the CPU for data transfer between the L64702 and the external system memory and shows how the L64702 interrupt pin handshakes with the CPU. The program first initializes the internal registers and tables, and then enables the proper interrupt bits in the SMR Register—in this case the THR and LCIF bits. After initializing the L64702, set the START bit, and the L64702 starts compression operation.

Figure 11.1
Compression
Program Flow
Example



When the external CPU receives an interrupt, the program jumps to the interrupt service routine, which determines what to do based on the state of the THR and LCIF bits of the STS Register.

If the L64702 sets the THR bit in the STS Register during compression, the CPU is interrupted and reads the number of words currently stored in the FIFO. The number of words read is equal to the value specified by the FIFCNT field in the STS Register.

When the CPU detects the L64702 LCIF interrupt, it signifies that the L64702 has written the last codeword into the FIFO. The CPU reads a number of words equal to the value of FIFCNT in the STS Register, and then becomes ready to compress the next image.

If compression of the next image is required, a software reset is performed by setting the RST bit in the SMR Register.

11.3 Image Decompression Task

Figure 11.2 shows a sample program flowchart that uses the L64702 internal status flags interrupt pin to control the flow of compressed data into the L64702 FIFO.

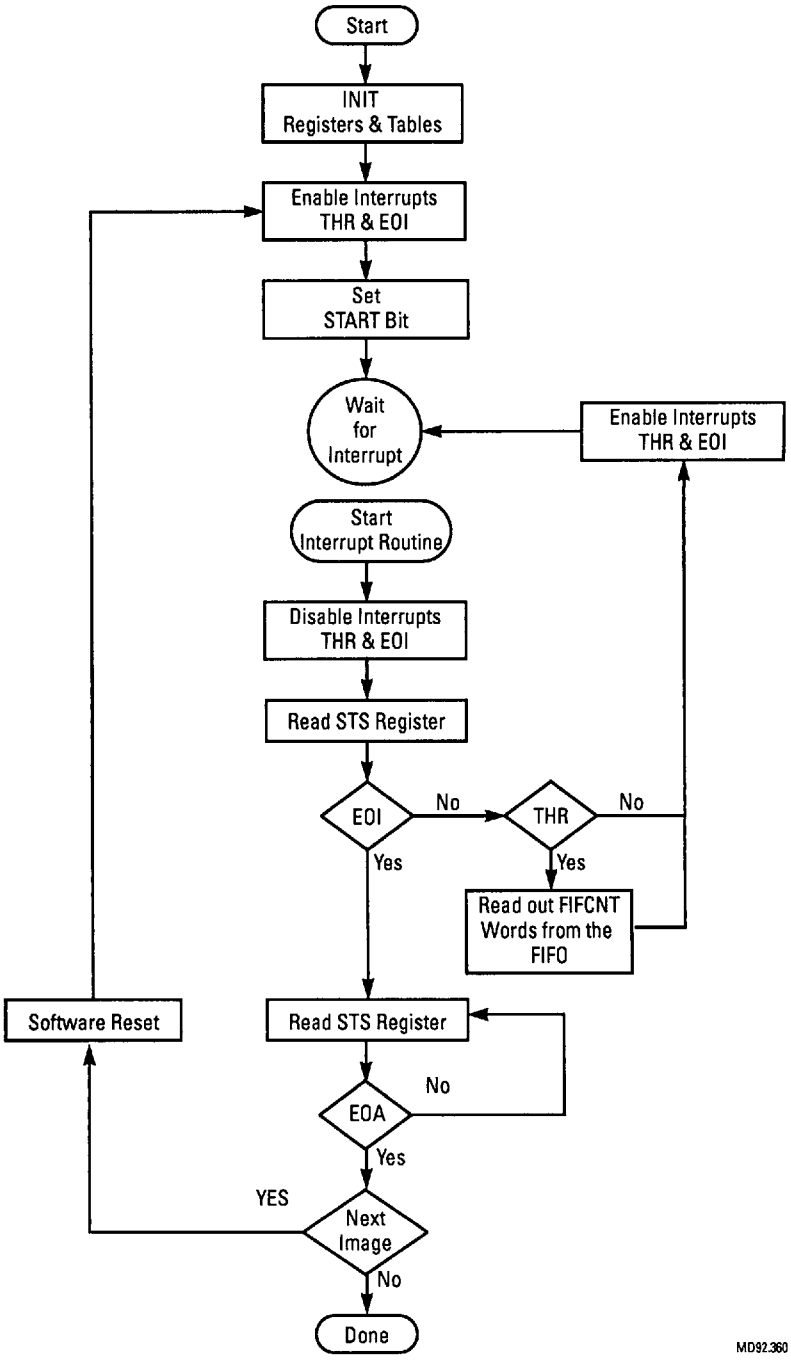
The program uses the CPU for data transfer between the L64702 and the external system memory. The program flow shows how the L64702 interrupt pin handshakes with the CPU. The program first initializes the internal registers and tables, and then enables the proper interrupt bits in the SMR Register—in this case the THR and the EOI bits. After initializing the device, set the START and the L64702 starts decompression.

When the external CPU receives an interrupt, the program jumps to the interrupt service routine, which determines what to do based on the status of the THR and EOI bits of the STS Register.

During decompression, the L64702 sets the THR bit periodically, which interrupts the program. The EOI bit also interrupts the program, but only once at the end of the compressed image. The CPU transfers to the FIFO the exact number of words specified by the FIFCNT level—no more words beyond this level should be transferred per burst transfer.

When the L64702 has detected the EOI marker in the code data stream, it sets the EOI bit, which generates a CPU interrupt. The L64702 then concludes decompression of the current image. The L64702 ignores any additional transfers to the FIFO after the EOI bit is set.

Figure 11.2 Decompression Program Flow Example



The EOA flag indicates that L64702 has finished decompression of the current image and that the L64702 is ready for the next image. If the CPU continues to decompress, the CPU can set the RST bit in the SMR Register. Setting this bit issues a software reset to the L64702. The CPU may then return to the same decompression procedure as for the previous image.

Chapter 12
Specifications

This chapter specifies the L64702 electrical and mechanical characteristics. Note that all specifications are preliminary and subject to change.

This chapter is divided into three sections:

- Section 12.1, AC Timing
- Section 12.2, Electrical Requirements
- Section 12.3, Packaging

12.1 AC Timing

This section presents AC timing information for the L64702 JPEG Coprocessor. The timing relationships between various signals are depicted in Figures 12.1 through 12.9. The figures depict the following timing relationships:

- Figure 12.1, L64702 CLK Timing
- Figure 12.2, L64702 RESET Timing
- Figure 12.3, CPU Write Cycle Timing
- Figure 12.4, CPU Read Cycle Timing
- Figure 12.5, CPU Write Cycle to the SMR Register
- Figure 12.6, CPU Read Cycle from the STS Register
- Figure 12.7, DMA Write Cycle Timing
- Figure 12.8, DMA Read Cycle Timing
- Figure 12.9, Video Port Timing

The numbers in Figure 12.1 through Figure 12.9 refer to the timing parameters listed in the first column of Table 12.1. Table 12.1 lists the AC timing values for the signals.

Table 12.1
AC Timing Values

Parameter	*Description*	*33 MHz* *Min*	*Max*	*Units*
Clock and Reset Timing				
1. t_C	CLK Cycle	30		ns
2. t_{CL}	CLK High Duration	14		ns
3. t_{CH}	CLK Low Duration	14		ns
4. t_{CR}	CLK Rise Time		10	ns
5. t_{CF}	CLK Fall Time		10	ns
6. t_{RH}	RESET High Duration	$2*t_C$		ns
7. t_{OVD}	RESET- to Valid Outputs		20	ns
CPU Write Cycle Timing				
8. t_{SRWS1}	SRS Setup before $\overline{SWR}$-	5		ns
9. t_{SRWH1}	SRS Hold from $\overline{SWR}$+	2		ns
10. t_{CSWS1}	$\overline{SCS}$- Setup before $\overline{SWR}$-	0		ns
11. t_{CSWH1}	$\overline{SCS}$+ Hold from $\overline{SWR}$+	0		ns
12. t_{SWW1}	$\overline{SWR}$ Pulse Width	22		ns
13. t_{SWR1}	$\overline{SWR}$ Recovery Time	22		ns
14. t_{SDWS1}	SDB Setup before $\overline{SWR}$+	22		ns
15. t_{SDWH1}	SDB Hold from $\overline{SWR}$+	5		ns
CPU Read Cycle Timing				
16. t_{SRRS1}	SRS Setup before $\overline{SRD}$-	5		ns
17. t_{SRRH1}	SRS Hold from $\overline{SRD}$+	2		ns
18. t_{CSRS1}	$\overline{SCS}$- Setup before $\overline{SRD}$-	0		ns
19. t_{CSRH1}	$\overline{SCS}$+ Hold from $\overline{SRD}$+	0		ns
20. t_{SRW1}	$\overline{SRD}$ Pulse Width	22		ns
21. t_{SRR1}	$\overline{SRD}$ Recovery Time	22		ns
22. t_{SDA1}	SDB Output Valid Delay from $\overline{SRD}$-		26	ns
23. t_{SDRH1}	SDB Hold from $\overline{SRD}$+	10		ns
CPU Write Cycle to SMR Register				
24. t_{SRWS2}	SRS Setup before $\overline{SWR}$-	5		ns
25. t_{SRWH2}	SRS Hold from $\overline{SWR}$+	2		ns
26. t_{CSWS2}	$\overline{SCS}$- Setup before $\overline{SWR}$-	0		ns
27. t_{CSWH2}	$\overline{SCS}$+ Hold from $\overline{SWR}$+	0		ns
28. t_{SWW2}	$\overline{SWR}$ Pulse Width	t_C		ns
29. t_{SWR2}	$\overline{SWR}$ Recovery Time	$4*t_C$		ns
30. t_{SDWS2}	SDB Setup before $\overline{SWR}$+	22		ns
31. t_{SDWH2}	SDB hold from $\overline{SWR}$+	5		ns

(Sheet 1 of 2)

Table 12.1 (Continued)
AC Timing Values

Parameter	*Description*	*33 MHz Min*	*33 MHz Max*	*Units*
CPU Read Cycle from STS Register				
32. t_{SRRS2}	SRS Setup before $\overline{SRD}$-	5		ns
33. t_{SRRH2}	SRS Hold from $\overline{SRD}$+	0		ns
34. t_{CSRS2}	$\overline{SCS}$- Setup before $\overline{SRD}$-	0		ns
35. t_{CSRH2}	$\overline{SCS}$+ Hold from $\overline{SRD}$+	0		ns
36. t_{SDA2}	SDB Output Valid Delay from $\overline{SRD}$-		$3*t_C + 26$	ns
37. t_{SDRH2}	SDB Hold from $\overline{SRD}$+	10		ns
38. t_{SRW2}	$\overline{SRD}$ Pulse Width	$5*t_C$		ns
39. t_{SRR2}	$\overline{SRD}$ Recovery Time	t_C		ns
DMA Write Cycle				
40. t_{AKWS}	$\overline{SDACK}$ Setup before $\overline{SWR}$-	0		ns
41. t_{AKWH}	$\overline{SDACK}$ Hold from $\overline{SWR}$+	3		ns
42. t_{SWW3}	$\overline{SWR}$ Pulse Width	22		ns
43. t_{SWR3}	$\overline{SWR}$ Recovery Time	22		ns
44. t_{SDWS3}	SDB Setup before $\overline{SWR}$+	22		ns
45. t_{SDWH3}	SDB Hold from $\overline{SWR}$+	5		ns
DMA Read Cycle				
46. t_{AKRS}	$\overline{SDACK}$ Setup before $\overline{SRD}$-	0		ns
47. t_{AKRH}	$\overline{SDACK}$ Hold from $\overline{SRD}$+	3		ns
48. t_{SRW3}	$\overline{SRD}$ Pulse Width	22		ns
49. t_{SRR3}	$\overline{SRD}$ Recovery Time	22		ns
50. t_{SDA3}	SDB Output Valid Delay from $\overline{SRD}$-		26	ns
51. t_{SDRH3}	SDB Hold from $\overline{SRD}$+	10		ns
Video Port Timing				
52. t_{VIW}	Video Input Signal Active Duration	$2*t_C$		ns
53. t_{VOD}	Video Valid Output Signal Delay from CLK		20	ns
54. t_{VAS}	VADB Input Setup	0		ns
55. t_{VAH}	VADB Input Hold	6		ns

(Sheet 2 of 2)

CLK and RESET Timing

Figure 12.1 and Figure 12.2 show the waveforms associated with the L64702 CLK and RESET signals.

Figure 12.1
L64702 CLK Timing

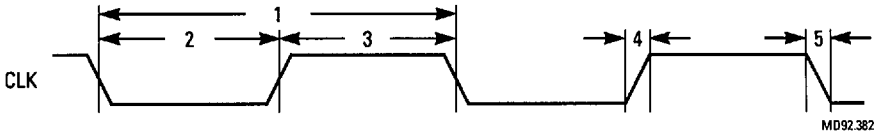
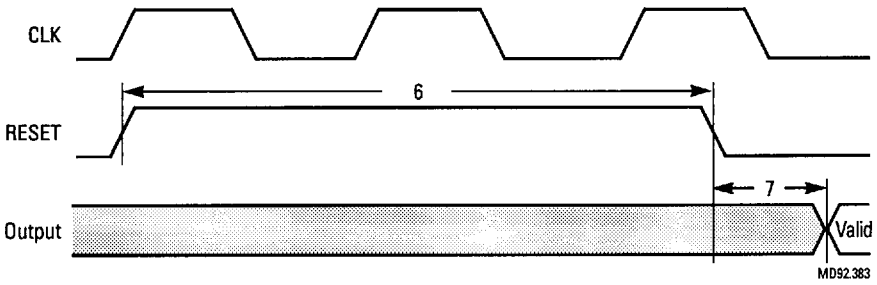


Figure 12.2
L64702 RESET Timing



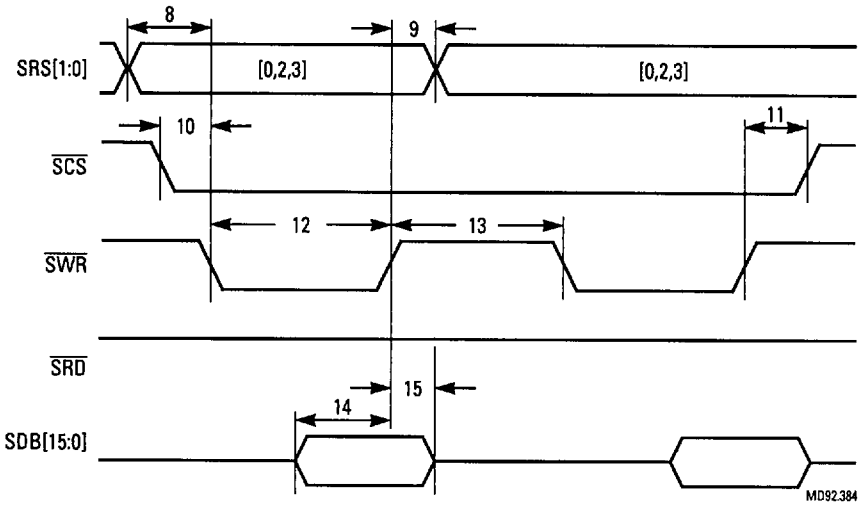
System Port Timing

The following waveforms are associated with the L64702 system port interface. The waveforms shown are the CPU Write Timing Cycle, CPU Read Timing Cycle, CPU Write Cycle to the SMR Register, CPU Read STS Register Timing Cycle, DMA Write Cycle, and DMA Read Cycle.

CPU Write Timing Cycle

Figure 12.3 illustrates the CPU write cycle timing when writing to the Code-FIFO, Group 2 tables and registers, and the APR Register. The CPU write to the Group 1 SMR Register has special timing considerations, which are illustrated later in Figure 12.5.

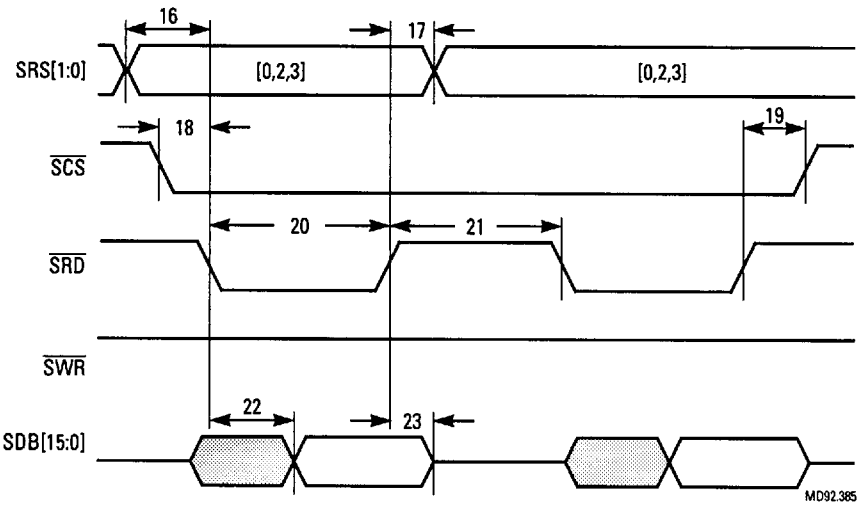
Figure 12.3
CPU Write Cycle Timing



CPU Read Timing Cycle

The CPU read cycle from the Code-FIFO, Group 2 tables and registers, and the APR Register is illustrated in Figure 12.4. The CPU read cycle from the Group 1 STS Register has special timing considerations, which are illustrated later in Figure 12.6.

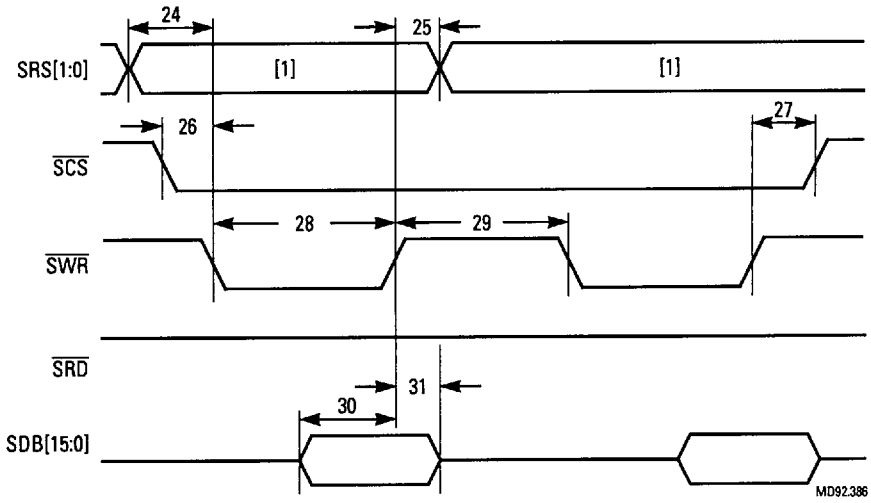
Figure 12.4
CPU Read Cycle Timing



CPU Write Cycle to the SMR Register

The CPU write cycle associated with the Group 1 SMR Register is illustrated in Figure 12.5.

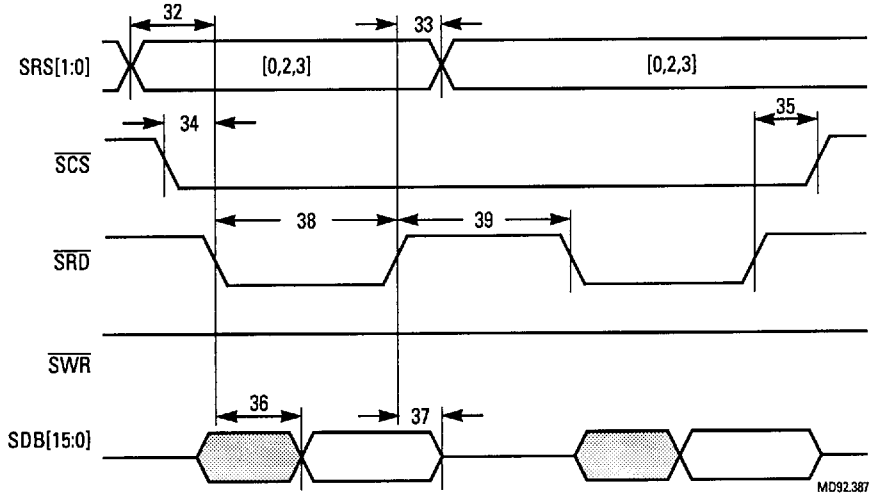
Figure 12.5 CPU Write Cycle to the SMR Register



CPU Read STS Register Timing Cycle

Figure 12.6 illustrates the CPU read cycle timing associated with the L64702 STS Register.

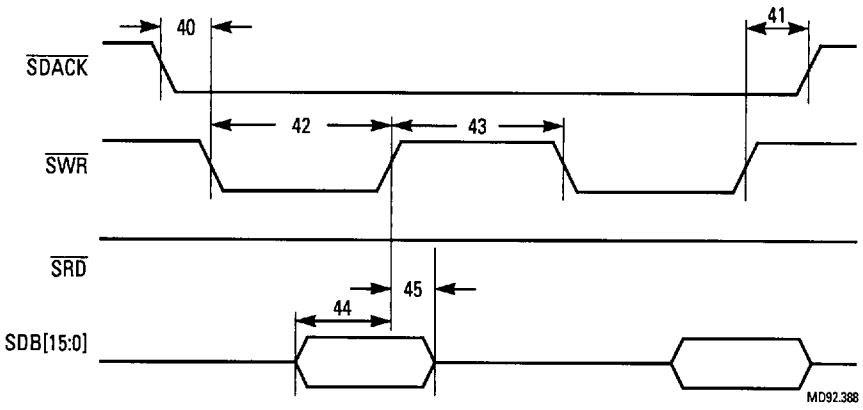
Figure 12.6 CPU Read Cycle from the STS Register



DMA Write Cycle Timing

Figure 12.7 illustrates the DMA write cycle timing.

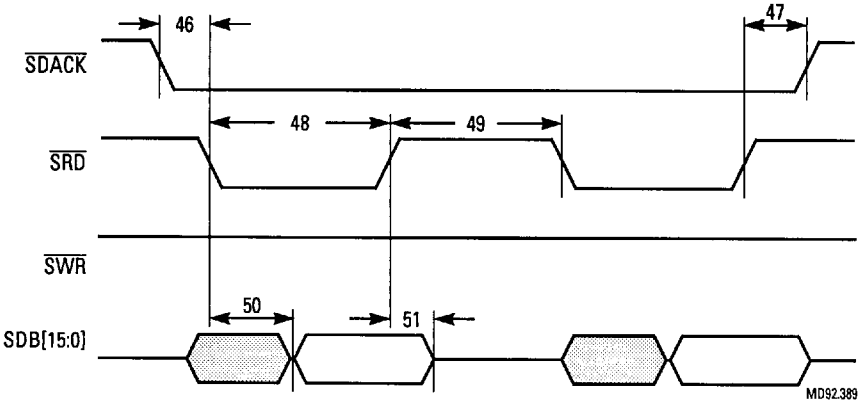
Figure 12.7 DMA Write Cycle Timing



DMA Read Cycle Timing

Figure 12.8 illustrates the DMA read cycle timing.

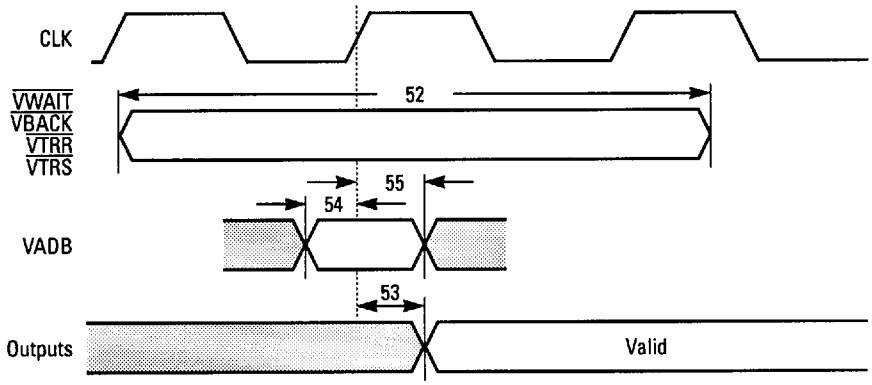
Figure 12.8 DMA Read Cycle Timing



Video Port Timing

Figure 12.9 provides the waveforms of the L64702 Video Port. The $\overline{\text{VWAIT}}$, $\overline{\text{VBACK}}$, $\overline{\text{VTRR}}$, and $\overline{\text{VTRS}}$ inputs are internally synchronized and should be active for a minimum of two clock cycles. All outputs are delayed from the rising edge of CLK. The cycle where the signal becomes active or inactive is according to the description given in Chapter 10, "Video Memory Interface Port." The outputs in the figure are: $\overline{\text{VBRQ}}$, $\overline{\text{VCAS}}$, $\overline{\text{VRAS}}$, $\overline{\text{VTR/OE}}$, $\overline{\text{VWR}}$, $\overline{\text{VDSF}}$, and $\overline{\text{VDEN}}$.

Figure 12.9
Video Port Timing



12.2 Electrical Requirements

This section specifies the electrical requirements for the L64702. Four tables list electrical data in the following categories:

- Absolute Maximum Ratings—Table 12.2
- Recommended Operating Conditions—Table 12.3
- Capacitance—Table 12.4
- DC Characteristics—Table 12.5
- Pin Description Summary—Table 12.6

Table 12.2
Absolute Maximum Ratings

Symbol	*Parameter*	*Limits*[1]	*Unit*
V_{DD}	DC Supply	-0.3 to +7	V
V_{IN}	Input Voltage	-0.3 to VDD +0.3	V
I_{IN}	DC Input Current	±10	mA
T_{STG}	Storage Temperature Range (Plastic)	-40 to +125	°C

1. Referenced to VSS

Table 12.3
Recommended Operating Conditions

Symbol	*Parameter*	*Limits*	*Unit*
V_{DD}	DC Supply	+4.75 to +5.25	V
T_A	Ambient Temperature	0 to +50	°C

Table 12.4
Capacitance

Symbol	*Parameter*[1]	*Min*	*Typ*	*Max*	*Units*
C_{IN}	Input Capacitance			15	pF
C_{OUT}	Output Capacitance			20	pF
C_{IO}	I/O Bus Capacitance			20	pF

1. Measurement conditions are V_{IN} = 5.0 V, T_A = 25°C, and clock frequency = 1 MHz.

Table 12.5
DC Characteristics

Symbol	*Parameter*	*Condition*[1]	*Min*	*Typ*	*Max*	*Units*
V_{IL}	Voltage Input Low		–	–	0.8	V
V_{IH}	Voltage Input High		2.0	–	–	V
V_{OH}	Voltage Output High	I_{OH} = -4.0 mA	2.4	4.5	–	V
V_{OL}	Voltage Output Low	I_{OL} = 4.0 mA	–	0.1	0.4	V
I_{IN}	Current Input					
	CMOS, TTL Inputs	V_{DD} = Max, V_{IN} = V_{DD} or V_{SS}	-10	±1	10	µA
	Inputs with Pulldowns	V_{DD} = Max, V_{IN} = V_{DD}	55	120	200	µA
	Inputs with Pullups	V_{DD} = Max, V_{IN} = V_{SS}	-60	-105	-185	µA
I_{OZ}	Current 3-State Output Leakage	V_{DD} = Max, V_{OUT} = V_{SS} or V_{DD}	-10	±1	10	µA
I_{OSP4}	Current P-Channel Output Short Circuit (4 mA Output Buffers)[2]	V_{DD} = Max, V_{OUT} = V_{SS}	-120	–	-30	mA
I_{OSN4}	Current N-Channel Output Short Circuit (4 mA Output Buffers)[2]	V_{DD} = Max, V_{OUT} = V_{DD}	30	–	105	mA
I_{DD}	Quiescent Supply Current	V_{IN} = V_{DD} or V_{SS}	–	–	10	mA
I_{CC}	Dynamic Supply Current[3]	V_{DD} = Max, f = 33 MHz	–	320	–	mA

1. Specified at V_{DD} equals 5V ± 5% over the specified ambient temperature range.
2. Not more than one output may be shorted at a time for a maximum duration of one second.
3. Typical I_{CC} can be calculated from the following formula: I_{CC} = 125 mA + (5.75 mA /f), where 125 mA is the static RAM current and f is the clock frequency.

Table 12.6
Pin Description Summary

Mnemonic	*Description*	*Type*	*Drive (mA)*	*Active*
CLK	Clock	Input		–
RESET	Reset	Input		High
$\overline{\text{SCS}}$	System Chip Select	Input		Low
$\overline{\text{SDACK}}$	System DMA Acknowledge	Input		Low
SDB[15:0]	System Data Bus	3-State Bidirectional	4	
SDREQ	System DMA Request	Output	4	High
$\overline{\text{SINT}}$	System Interrupt Request	Output	4	Low
$\overline{\text{SRD}}$	System Read	Input		Low
SRS[1:0]	System Register Select	Input		
$\overline{\text{SWR}}$	System Write	Input		Low
VADB[31:0]	Video Address and Data Bus	3-State Bidirectional	4	
$\overline{\text{VBACK}}$	Video Bus Acknowledge	Input		Low
$\overline{\text{VBRQ}}$	Video Bus Request	Output	4	Low
$\overline{\text{VCAS}}$	Video Memory Column Address Strobe	3-State Output	4	Low
$\overline{\text{VDEN}}$	Video Bus Data Enable	3-State Output	4	Low
VDSF	Video Special Function	3-State Output	4	High
$\overline{\text{VRAS}}$	Video Memory Row Address Strobe	3-State Output	4	Low
VRFC	Video Refresh Cycle	3-State Output	4	
VTRC	Video Transfer Cycle	3-State Output	4	
$\overline{\text{VTRR}}$	Video Transfer Request	Input		Low
$\overline{\text{VTRS}}$	Video Transfer Reset	Input		Low
$\overline{\text{VTR/OE}}$	Video Data Transfer and Output Enable	3-State Output	4	Low
$\overline{\text{VWAIT}}$	Video Bus Wait	Input		Low
$\overline{\text{VWE}}$[1:0]	Video Write Enable	3-State Output	4	Low

12.3 Packaging

This section contains three types of information for the 100-pin PQFP: an alphabetical pin list (Table 12.8), a pinout (Figure 12.10), and a mechanical drawing (Figure 12.11). Table 12.7 lists the ordering number for the L64702.

Table 12.7 L64702 Ordering Information

Order Number	*Clock Frequency (MHz)*	*Package Type*	*Operating Range*
L64702QC-33	33	100-pin PQFP	Commercial

When you order the L64702, you receive free of charge a 3 1/2" PC-format diskette with the Huffman Table source code (see Appendix B). The Huffman Table diskette is included free with your first L64702 order. Additional diskettes may be ordered by calling 800.828.4LSI extension 7468. You may also download the Huffman Table source code from the LSI bulletin board by calling 408.433.7578.

Table 12.8 Alphabetical Pin List for the 100-pin PQFP

Signal	*Pin*	*Signal*	*Pin*	*Signal*	*Pin*	*Signal*	*Pin*
CLK	67	VADB6	99	VDD	20	$\overline{\text{VWE0}}$	87
RST	74	VADB7	100	VDD	28	$\overline{\text{VWE1}}$	86
$\overline{\text{SCS}}$	73	VADB8	1	VDD	29	*reserved*[1]	57
$\overline{\text{SDACK}}$	75	VADB9	5	VDD	41	*reserved*	71
SDB0	36	VADB10	6	VDD	52		
SDB1	37	VADB11	7	VDD	53		
SDB2	38	VADB12	8	VDD	68		
SDB3	39	VADB13	9	VDD	78		
SDB4	40	VADB14	10	VDD	79		
SDB5	43	VADB15	13	VDD	90		
SDB6	44	VADB16	14	$\overline{\text{VDEN}}$	84		
SDB7	45	VADB17	15	VDSF	83		
SDB8	46	VADB18	16	$\overline{\text{VRAS}}$	89		
SDB9	47	VADB19	17	VRFC	58		
SDB10	48	VADB20	18	VSS	4		
SDB11	49	VADB21	21	VSS	12		
SDB12	50	VADB22	22	VSS	19		
SDB13	51	VADB23	23	VSS	27		
SDB14	55	VADB24	24	VSS	35		
SDB15	56	VADB25	25	VSS	42		
SDREQ	64	VADB26	26	VSS	54		
$\overline{\text{SINT}}$	81	VADB27	30	VSS	60		
$\overline{\text{SRD}}$	72	VADB28	31	VSS	66		
SRS0	65	VADB29	32	VSS	77		
SRS1	69	VADB30	33	VSS	85		
$\overline{\text{SWR}}$	70	VADB31	34	VSS	91		
VADB0	92	$\overline{\text{VBACK}}$	61	VSS	96		
VADB1	93	$\overline{\text{VBRQ}}$	62	VTRC	59		
VADB2	94	$\overline{\text{VCAS}}$	88	$\overline{\text{VTRR}}$	76		
VADB3	95	VDD	2	$\overline{\text{VTRS}}$	80		
VADB4	97	VDD	3	$\overline{\text{VTR/OE}}$	82		
VADB5	98	VDD	11	$\overline{\text{VWAIT}}$	63		

1. Leave all reserved pins unconnected.

Figure 12.10
100-Pin PQFP Pinout

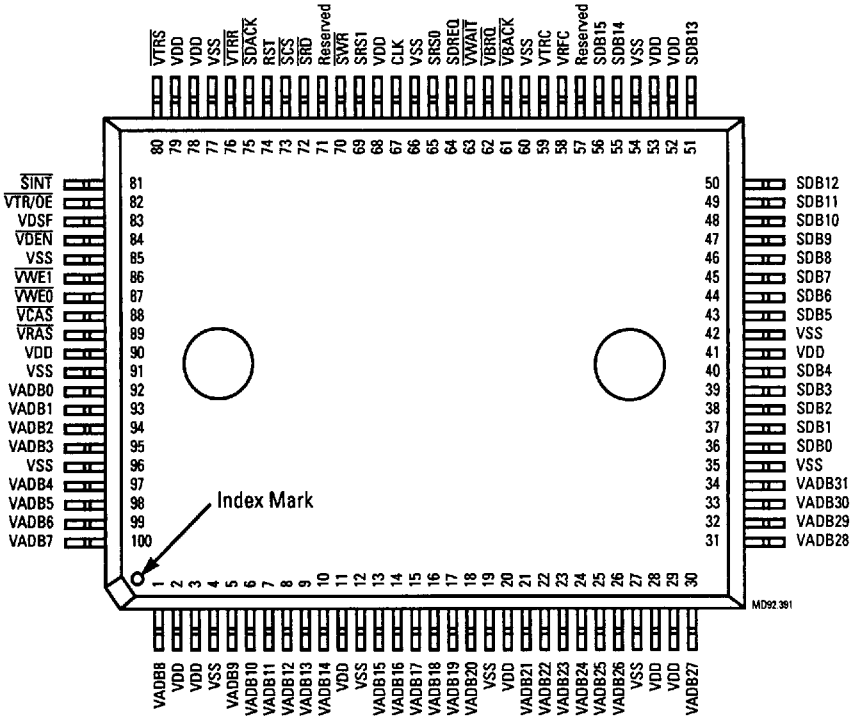
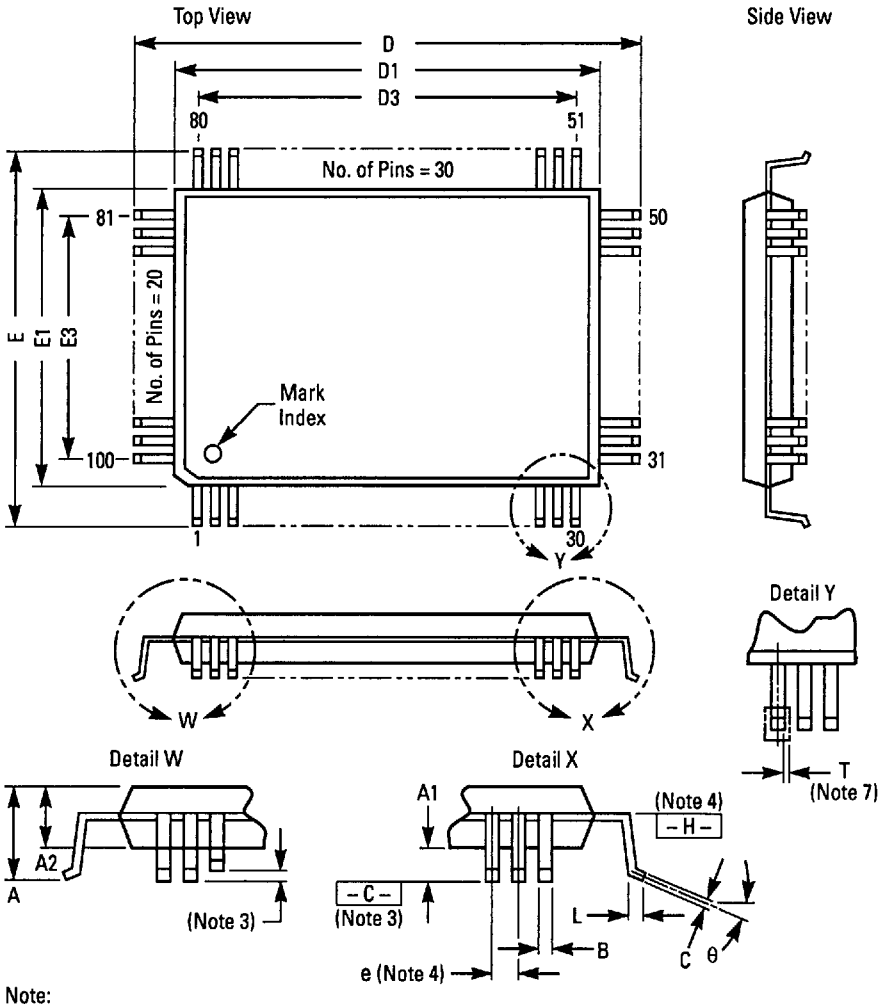


Figure 12.11
100-Pin PQFP
Mechanical Drawing



Dimension		mm
A	Max	3.40
A1	Min	0.25
	Max	0.35
A2	Min	2.55
	Nom	2.80
	Max	3.05
B	Min	0.22
	Nom	0.30
	Max	0.38
C	Min	0.12
	Max	0.25
D	Min	23.65
	Nom	23.90
	Max	24.15
D1	Min	19.90
	Nom	20.00
	Max	20.10
D3	Ref	18.85
e	BSC	0.65
E	Min	17.65
	Nom	17.90
	Max	18.15
E1	Min	13.90
	Nom	14.00
	Max	14.10
E3	Ref	12.35
L	Min	0.65
	Nom	0.80
	Max	0.95
T	Max	0.10
θ	Min	0°
	Max	7°

Note:
1. Total number of pins is 100.
2. Drawing is not to scale.
3. Coplanarity of all leads shall be within 0.10 mm (difference between the highest and lowest lead with seating plane – C – as reference).
4. Datum plane – H – is located at mold parting line and is coincident with the bottom of the lead, where the lead exits the plastic body. Lead pitch determined at – H –.
5. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. These dimensions to be determined at – H –.
6. Dimensions D3 and E3 to be centered relative to dimensions D1 and E1, respectively, ±0.200 mm.
7. Tolerance window for lead skew from true position is determined at seating plane – C –.
8. For board layout and manufacturing, you may obtain engineering drawings from your LSI Logic Products marketing representative by requesting the outline drawing for package code PB.

MD92.PB

Appendix A
The JVieW (JPEG Video in a Window) Evaluation Kit

The JVieW Kit converts a PC into a low-cost multimedia workstation. The JVieW (JPEG Video in a Window) Evaluation Kit consists of a 16-bit PC AT (ISA bus) board and a software driver that drives the JVieW board from Microsoft Video for Windows. The JVieW board is based on LSI Logic's L64702 JPEG Coprocessor.

The JVieW board records and plays full-motion video in a 1/4 screen window at 30 frames/second NTSC or 25 frames/second PAL. The software driver allows use of the Microsoft Video for Windows VidCap, VidEdit, and MediaPlayer functions to perform video capture, editing, and display.

This chapter has seven sections:

- Section A.1, "Overview"
- Section A.2, "Kit Contents"
- Section A.3, "System Requirements"
- Section A.4, "Features Summary"
- Section A.5, "JVieW Board in a Multimedia PC System"
- Section A.6, "JVieW Board Layout"
- Section A.7, "JVieW Board Functional Description"

See LSI Logic's *JVieW (JPEG Video in a Window) Evaluation Kit User's Guide* for more information on the JVieW Kit. Contact your local LSI Logic sales office for more information.

A.1 Overview

The JVieW Kit enables a regular PC to function as a full-powered multimedia workstation. The JVieW Kit comes with a software driver which allows users to use the Microsoft Video for Windows tools under Microsoft Windows 3.1.

Users can acquire, digitize, and compress analog video from either a VCR, video camera, or laserdisk, and then store the digital video data directly onto the PC hard disk. Video clips are stored in a JPEG compressed format embedded in an Audio Visual Interlude (AVI) file structure. Users can then edit these video clips and display them to an overlay window on a standard VGA monitor.

The JVieW Kit can capture 640 x 240, 320 x 240, 160 x 120, or 80 x 60 video clips in NTSC mode. The JVieW Kit can display AVI clips in whatever window size is specified in the header. The maximum board resolution allowed is 800 x 600.

The JVieW board and its software driver were designed to operate on a standard IBM PC AT compatible platform (either 80386 or 80486 microprocessors). The JVieW board can drive standard VGA or super VGA monitors, operating in conjunction with the VGA card through the VGA Feature Connector. The onboard frame buffer can be used as a workspace area for multimedia tools.

A.2 Kit Contents

The following items are included with the JVieW Evaluation Kit:

1. JVieW Evaluation Board
2. JVieW Software Driver on 3.5" Floppy Disk
3. VGA to JVieW Feature Connector Ribbon Cable
4. RCA Composite Video-In Cables
5. *JVieW (JPEG Video in a Window) Evaluation Kit User's Guide*
6. *L64702 JPEG Coprocessor Technical Manual*

A.3 System Requirements

The following is a list of hardware and software system requirements, as well as recommended equipment, used in setting up a JVieW Kit-based system for demonstration and evaluation.

1. IBM PC AT Compatible Computer
2. 80386 or 80486 Microprocessor
3. At least 4 Mbyte Installed Memory (8 Mbyte is recommended)
4. A hard disk with enough free space to install the software and some video clips (at least 100 Mbytes)

5. VGA or SVGA Card with Feature Connector
6. Standard VGA or SVGA Monitor
7. Video Source, Laser Disk, or Camcorder (for Video Capture Only)
8. MS-DOS 5.0
9. Microsoft Windows Operating System, Version 3.1
10. Microsoft Video for Windows, Version 1.0

A.4 Features Summary

This section lists the key hardware and software features of the JVieW Kit.

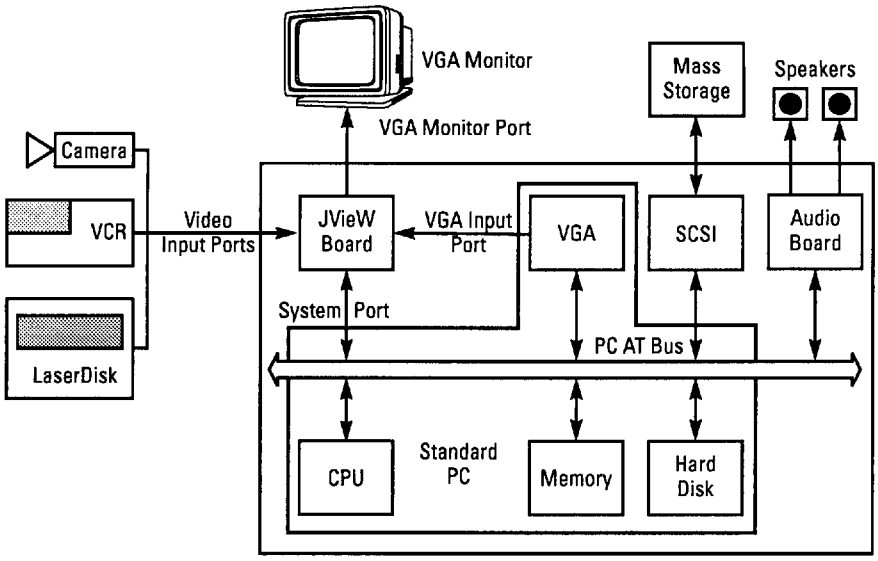
- JPEG baseline image compression and decompression
- Realtime video capture from composite or SVIDEO signals
- Realtime video recording and playback of 320 x 240 NTSC or 384 x 288 PAL images
- Full size video recording and playback rate – 640 x 480 pixels at 10 frames per second for NTSC or 8 frames per second for PAL
- Up to 8.25 Mbytes/second processing capability
- Four programmable image size options: Full, 1/2, 1/4, and 1/8
- Software frame rate control using an interrupt schedule scheme
- ISA bus interface utilizing I/O mapping only
- 16-bit DMA channel for fast DMA transfers
- Overlay of VGA display and video in a window
- 2 Mbyte frame buffer for display of high resolution pictures
- Direct access from the system bus to the frame buffer for support of still image applications and image processing
- Supports for AVI file format and JPEG AVI
- Software driver that runs under Video for Windows under Microsoft Windows (user can use MediaPlayer, VidCap, and VidEdit tools)

A.5 JVieW Board in a Multimedia PC System

The JVieW board provides the video compression and decompression capabilities that allow the user to record or playback video clips from the system hard disk or CDROM.

Figure A.1 shows a block diagram of a multimedia PC system containing the JVieW board. The figure shows the main blocks of a standard PC along with the main blocks that allow the PC to act as a multimedia system.

Figure A.1 PC Multimedia System with JVieW Board



The SCSI block is an optional enhancement that allows additional storage beyond the standard system hard disk drive capacity, and is fast enough to enhance record or playback speed. An audio board provides sound enhancement and is usually used in video game applications.

The JVieW board contains four ports that communicate with the PC system and external accessories.

Video Input Ports

There are two analog composite video input ports for NTSC or PAL signals and one for SVIDEO (Super VHS). These ports provide a connection to standard consumer video sources (VCR, video camera, or laserdisk). The JVieW board captures realtime video from these ports.

VGA Input Port

This port is a standard connector on most VGA cards that allows the JVieW board to receive digital VGA (data and sync) information and to overlay video.

VGA Monitor Port

The JVieW board overlays the digital video display information over the VGA digital display information and then outputs an analog video signal to the standard VGA or Super VGA monitor.

System Port

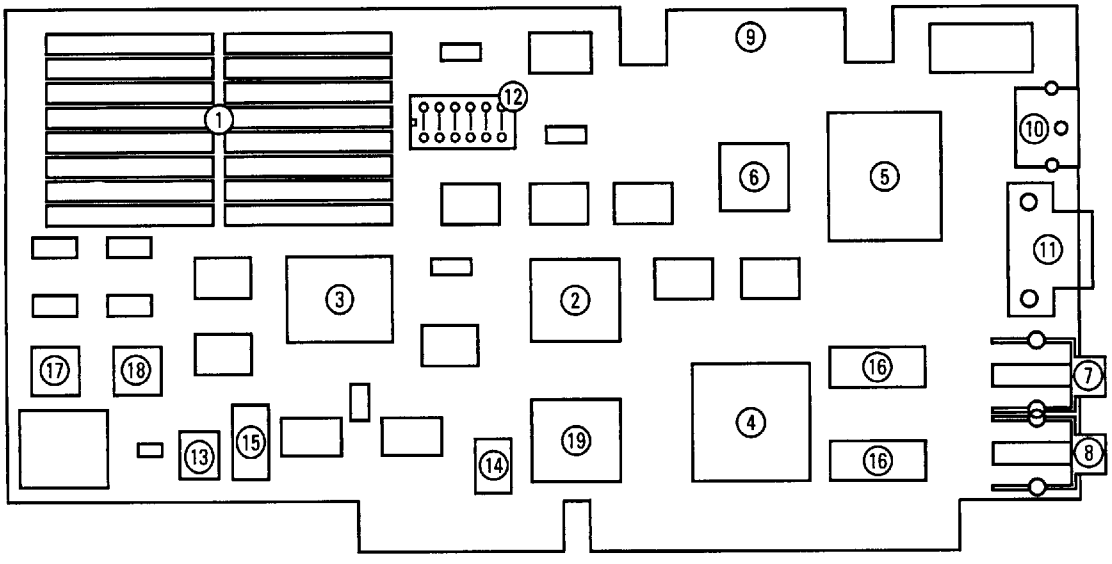
The PC transfers compressed data, image data or control words to or from the JVieW board through this ISA bus interface port.

The system port is bidirectional and transfers compressed data between the L64702 FIFO and the system hard disk. The system port is also used for L64702 initialization and still image applications involving direct image data transfer between the system hard disk and the display memory.

A.6 JVieW Board Layout

The JVieW board is a single-slot, 16-bit ISA bus card designed to operate in IBM PC ATs and compatibles. The JVieW board measures 9.9" wide and 4.3" tall, about 3/4 of a full size AT card. Figure A.2 shows the board layout with the main components. The numbers in the figure correspond to the numbers in the component list that follow the figure.

Figure A.2
JVieW Board Layout



The main components on the JVieW board are:

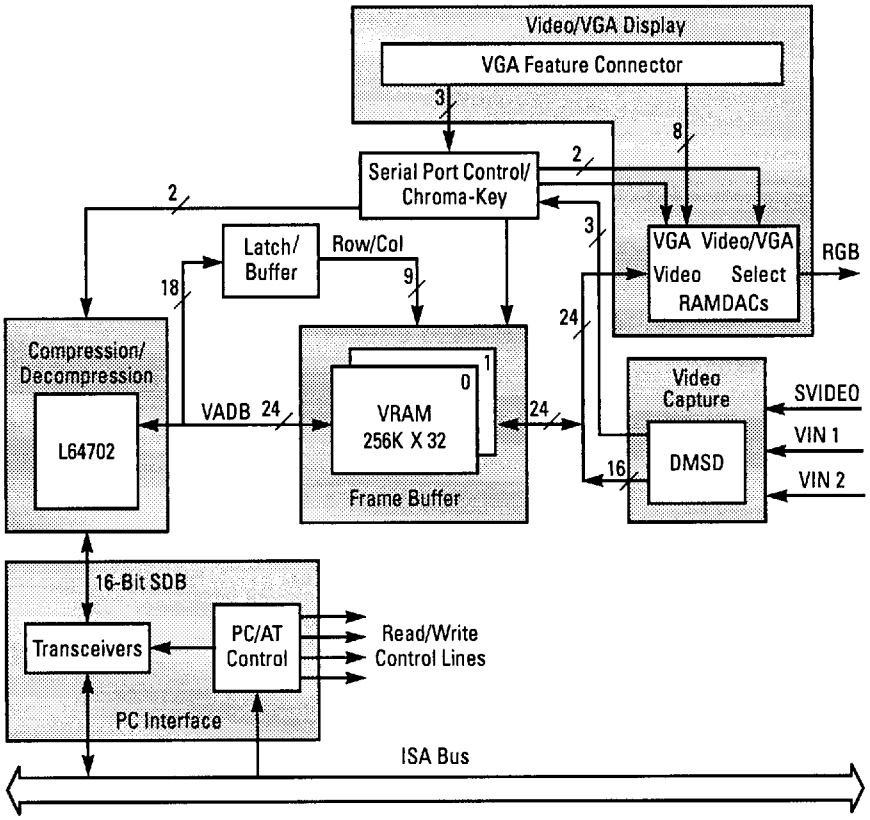
1. Frame Buffer (Video Memory) – two banks of 256K x 32 bits each.
2. Frame Buffer Serial Port Control Unit (SPC)
3. L64702 JPEG Coprocessor
4. Philips SAA7191 Digital Multistandard Decoder (DMSD) Chip
5. BrookTree Bt473 24-Bit True Color Video RAMDAC
6. BrookTree Bt478 8-Bit (256 Color Palette) VGA RAMDAC
7. RCA Composite Video Input Connector 1
8. RCA Composite Video Input Connector 2
9. VGA Feature Connector
10. SVIDEO Input Connector
11. VGA Output Connector
12. Six-Switch DIP Switch
13. DMA Request Jumpers
14. DMA Acknowledge Jumpers
15. Interrupt Request Jumpers
16. Two A/Ds
17. Special Feature to control VRAM for both read and write (QSF)
18. Parallel Port Controller (PPC)
19. ISA Bus Interface Control Unit (PC INT)

A.7 JVieW Board Functional Description

This section describes the functional hardware characteristics of the JVieW board. Figure A.3 shows a block diagram of the JVieW board and the main data and control lines. The following main functional blocks are described below:

- Video Capture
- Video/VGA Display
- Frame Buffer
- Compression/ Decompression
- PC Interface

Figure A.3
JVieW Hardware
Block Diagram



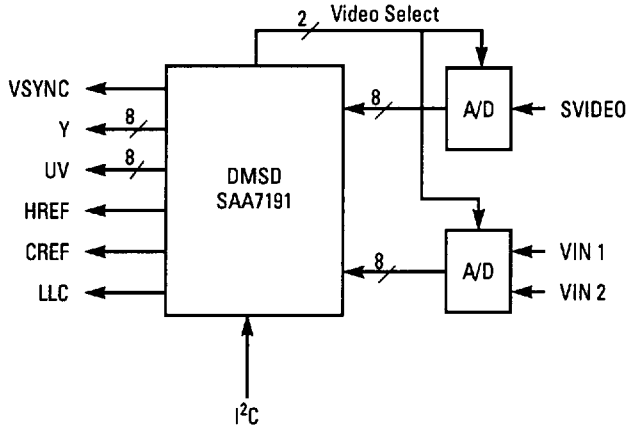
Video Capture

The video capture logic has been designed and implemented using the Philips Digital Multi-Standard Decoder (DMSD) chipset. This chipset contains two A/D converters, the TDA8708 and TDA8709, and the SAA7191 DMSD. The A/Ds convert the analog composite signal input into a digital composite signal. The pixel resolution is eight bits and is compatible with the NTSC or PAL broadcasting standards as well as with the SVIDEO (Y/C) format. The application software may select the video source by programming two general-purpose bits in the SAA7191.

A digital multistandard video decoder chip, the SAA7191, provides the capability to genlock (achieve synchronization) on the incoming digital composite video signal, output the digital image information in 16-bit YUV and 4:2:2 formats, and output all required video sync signals (VSYNC, HREF, CREF) used by other control circuits for synchronization purposes.

Figure A.4 shows the video capture logic.

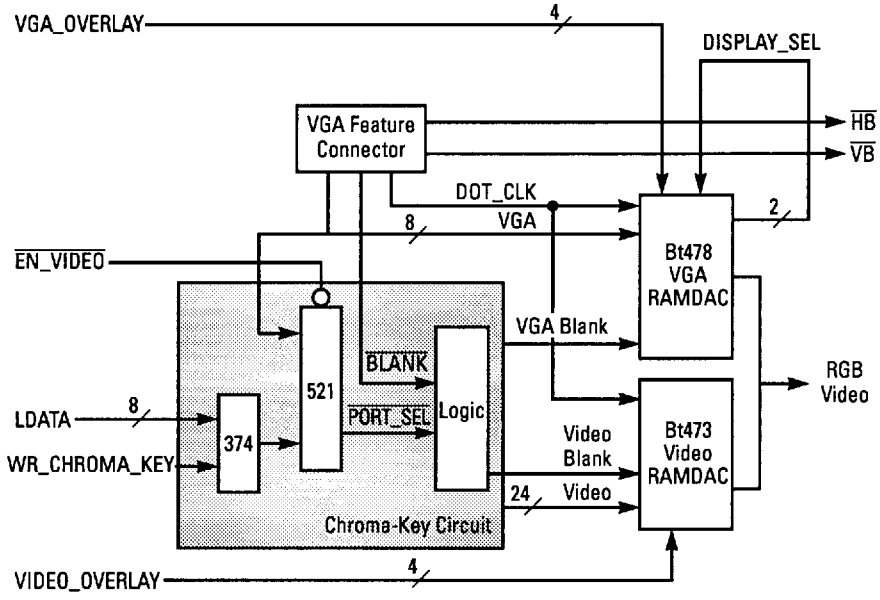
Figure A.4 Video Capture



Video/VGA Display

The video/VGA display logic consists of two Brooktree RAMDACs, the Bt473 and the Bt478, and a special purpose chroma-key circuit. The VGA feature connector transfers the exact VGA data on the PC VGA card to the Bt478 VGA RAMDAC. Synchronization signals are also provided through this connector. Figure A.5 shows the video/VGA display logic with the chroma-key circuit highlighted.

Figure A.5 Video/VGA Display with Chroma-Key



The Bt478 VGA RAMDAC functions exactly like the system VGA RAMDAC, which resides on the VGA card. Special circuitry intercepts the system write to the system VGA RAMDAC and maintains an exact copy of the color palette registers.

The Bt473 24-bit Video RAMDAC receives the video data directly from the frame buffer serial port. Additionally this RAMDAC allows the application to gamma correct the video data. In applications where gamma correction is not required, a low-cost triple DAC such as the Bt121 may be used. The Bt473 allows you to display the video in the following formats:

- 24-bit True Color Mode
- 24-bit Bypass Mode
- 8-bit Gray Scale
- 15-bit Color

The above selections are programmable through the Bt473 command register. The L64702 supports only the 24-bit true color and 24-bit bypass modes when displaying decompressed data.

The chroma-key circuit is implemented by a 74F374 register and a 74F521 comparator. The system writes the chroma-key value to the 74F374, and the 74F521 compares the value in the 74F374 to the eight-bit VGA data. When the two values are equal, the chroma-key logic enables the Bt473 and disables the Bt478. The blank signals provided to the VGA and video RAMDACs switch between VGA and video.

The JVieW board also uses the overlay registers of the RAMDACs. Each RAMDAC overlay is controlled by four bits. The two RAMDACs are controlled by the eight most-significant bits of the frame buffer—four bits are connected to the Video RAMDAC and four bits are connected to the VGA RAMDAC. This feature allows an additional layer of overlay information known as the α function may be used for special video effects.

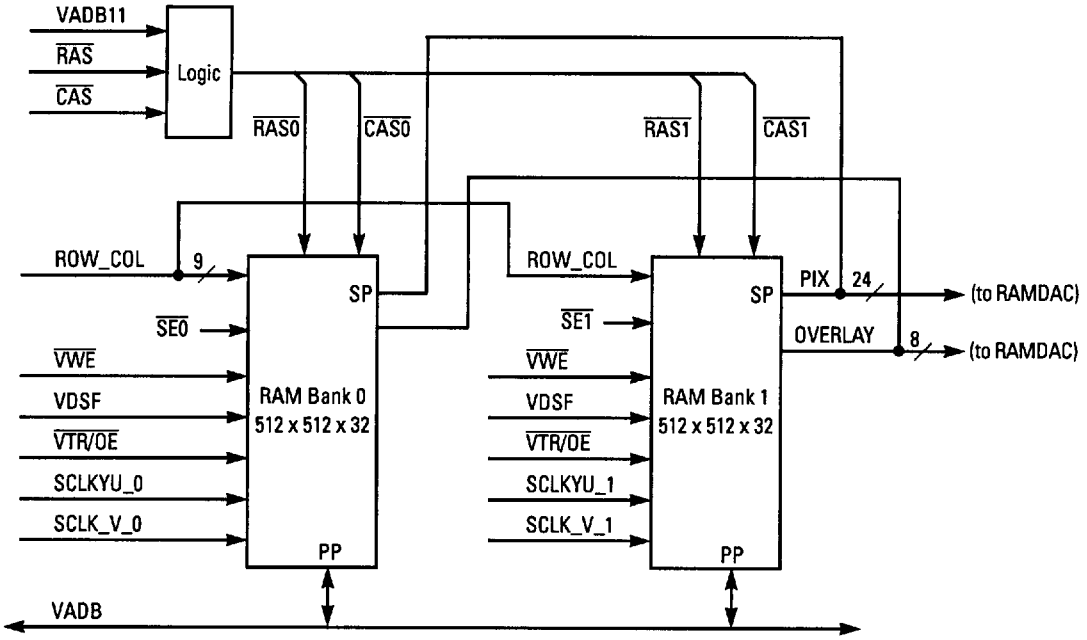
Frame Buffer

The frame buffer consists of two 256K x 32-bit VRAM banks. This amount of memory allows the capture or display of images of up to 512K pixels. The user may reduce the expense of the board by installing only one 256K x 24/32 bank, but this limitation compromises the ability of the JVieW board to support full resolution images in both moving video and still image applications.

Video memory is organized as xBGR (null, Blue, Green, and Red) in a 32-bit plane. Each of the three colors use eight bits with the most significant eight bits set to zero.

Figure A.6 shows the frame buffer logic.

Figure A.6
Frame Buffer Block Diagram



The frame buffer has been implemented using TMS44C251 256K x 4 VRAM components. The TMS44C251 is organized internally as a two-dimensional array of 512 x 512 x 4, and contains a 512 x 4 serial shift register. Currently, the JVieW board supports only the two-bank mode, in which the frame buffer is organized as 1024 x 512 x 24/32. In the single-bank mode, the frame buffer can be organized as 512 x 512 x 24/32. A special serial port control circuit controls the data flow in and out of the VRAM serial port and operates in full synchronization with the video capture and display sync signals.

The L64702 JPEG Coprocessor directly controls the VRAM parallel port. The serial port control generates two control signals, $\overline{\text{VTRS}}$, the new field indicator, and VTRR, the new line indicator. These signals initiate periodic video transfer cycles for both capture and display.

Compression/ Decompression

The L64702 JPEG Coprocessor performs image compression and decompression. The L64702 communicates by means of its video port with the frame buffer in bus master fashion, delivering video information to or from the frame buffer. The L64702 also supports frame buffer memory refresh and frame buffer serial control cycles, which allows realtime video capture and display.

As a slave peripheral, the L64702 communicates directly with the ISA bus from its system port. The PC software communicates with the L64702 through the CPU I/O port using the I/O-mapped IN and OUT instructions. The L64702 can also be programmed to initiate DMA service through the system DMA controller. DMA transfers are fast and efficiently deliver compressed data to and from the system memory.

The L64702 system port is an asynchronous port, which allows direct connection of the ISA bus signals to the L64702. Some of these signals are for I/O Read (IOR), I/O Write (IOW), and DMA Acknowledge (DACK). The Chip Select signal (CS), which enables L64702 read and write operations, originates at the PC interface logic.

Figure A.7 shows the L64702 logic.

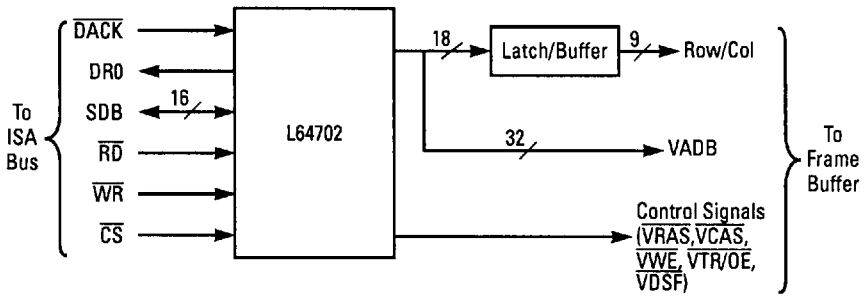
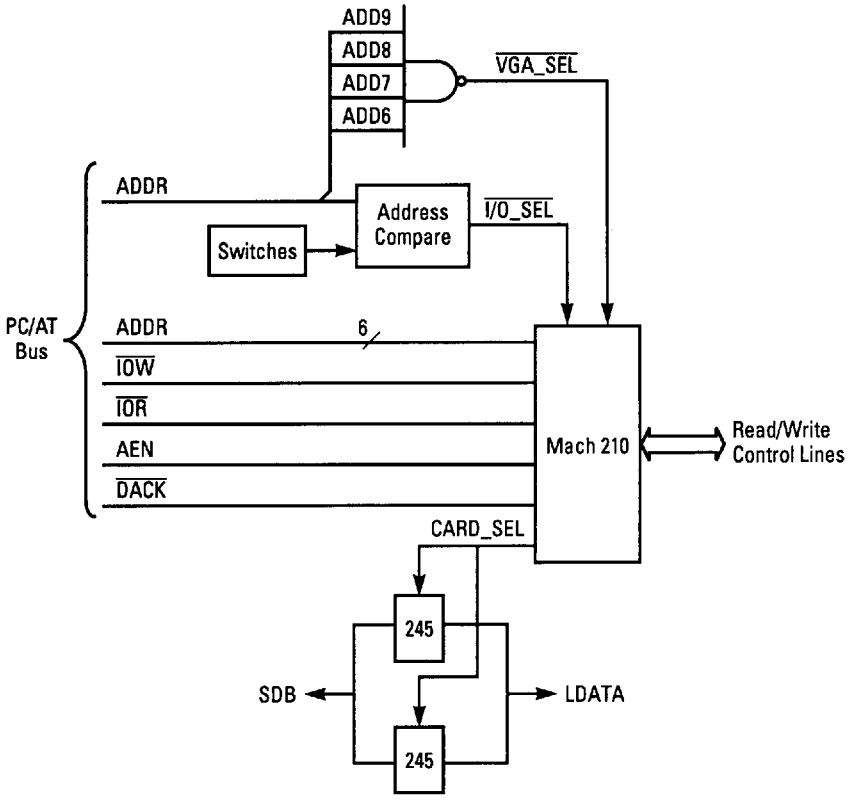


Figure A.7 L64702 Compression/ Decompression

PC Interface

The PC Interface logic, shown in Figure A.8, is responsible for connecting the JVieW local data bus (LDATA), to the PC ISA bus. The data interface is either 8-bit or 16-bit, depending on the memory-mapped resource being used on the JVieW board. For example, the L64702 uses 16-bit transfers, while the RAMDACs use 8-bit data transfers. The address decoding logic and control signals are implemented with a single MACH210 device. Two eight-bit 74F245 transceivers act as buffers between the ISA data bus and the JVieW LDATA bus.

Figure A.8
PC ISA Bus Interface



Appendix B
Huffman Table Software Listing

This section contains a listing of the software source code supplied by LSI Logic to generate the Huffman tables that reside in the L64702 RAM. When you order the L64702, you receive free of charge a 3 1/2" PC-format diskette with the Huffman Table source code. The Huffman Table diskette is included free with your first L64702 order. Additional diskettes may be ordered by calling 800.828.4LSI extension 7468. You may also download the Huffman Table source code from the LSI bulletin board by calling 408.433.7578.

```
/*
 * This file generates the ram contents for the RR640 and RR56 rams
 * in the L64702 VLC module.
 *
 * Copyright (C) 1992 LSI Logic Corporation.  All rights reserved.
 */

#include <stdio.h>
#include <strings.h>


#ifndef lint
static char rcsid[] = "$Header: huffRamGen.c,v 2.2 92/11/17 09:40:43 zorro!hjh
Exp $ ";
#endif

#define TRUE  1
#define FALSE 0

#define AC    1
#define DC    0

#define SEL_HUFF_DC01
#define SEL_HUFF_DC12
#define SEL_HUFF_AC04
#define SEL_HUFF_AC18
```

```
#define HUFF_RAM640_SIZE640
#define HUFF_RAM56_SIZE    56

int RR56[HUFF_RAM56_SIZE];
int RR640[HUFF_RAM640_SIZE];


/*
 * The Huffman code table definition sequences.
 * These are hard coded defaults for convenience.
 */
#define HuffDCLumSize 28
/* First 16 entries are code group size for code
 * length group from 1 to 16.
 * The remaining 12 entries are the actual code symbols.
 */
int HuffDC0[28] =
{
    /* length specifications */
    0x00, 0x01, 0x05, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* symbol definitions */
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B
};
int HuffDC1[28] =
{
    /* length specifications */
    0x00, 0x03, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* symbol definitions */
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B
};

#define HuffACLumSize 178
/* First 16 entries are code group size for code
 * length group from 1 to 16.
 * The remaining 162 entries are the actual code symbols.
 */
```

```
int HuffAC0[178] =
{
    /* length specifications */
    0x00, 0x02, 0x01, 0x03, 0x03, 0x02, 0x04, 0x03,
    0x05, 0x05, 0x04, 0x04, 0x00, 0x00, 0x01, 0x7D,

    /* symbol definitions */
    0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,
    0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,
    0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xA1, 0x08,
    0x23, 0x42, 0xB1, 0xC1, 0x15, 0x52, 0xD1, 0xF0,
    0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0A, 0x16,
    0x17, 0x18, 0x19, 0x1A, 0x25, 0x26, 0x27, 0x28,
    0x29, 0x2A, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
    0x3A, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
    0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
    0x5A, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
    0x6A, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
    0x7A, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
    0x8A, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
    0x99, 0x9A, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7,
    0xA8, 0xA9, 0xAA, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6,
    0xB7, 0xB8, 0xB9, 0xBA, 0xC2, 0xC3, 0xC4, 0xC5,
    0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xD2, 0xD3, 0xD4,
    0xD5, 0xD6, 0xD7, 0xD8, 0xD9, 0xDA, 0xE1, 0xE2,
    0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9, 0xEA,
    0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8,
    0xF9, 0xFA
};
int HuffAC1[178] =
{ /* length specifications */
    0x00, 0x02, 0x01, 0x02, 0x04, 0x04, 0x03, 0x04,
    0x07, 0x05, 0x04, 0x04, 0x00, 0x01, 0x02, 0x77,

    /* symbol definitions */
    0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,
    0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,
    0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,
    0xA1, 0xB1, 0xC1, 0x09, 0x23, 0x33, 0x52, 0xF0,
    0x15, 0x62, 0x72, 0xD1, 0x0A, 0x16, 0x24, 0x34,
    0xE1, 0x25, 0xF1, 0x17, 0x18, 0x19, 0x1A, 0x26,
    0x27, 0x28, 0x29, 0x2A, 0x35, 0x36, 0x37, 0x38,
    0x39, 0x3A, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
    0x49, 0x4A, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
    0x59, 0x5A, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
    0x69, 0x6A, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
    0x79, 0x7A, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
    0x88, 0x89, 0x8A, 0x92, 0x93, 0x94, 0x95, 0x96,
```

```
    0x97, 0x98, 0x99, 0x9A, 0xA2, 0xA3, 0xA4, 0xA5,
    0xA6, 0xA7, 0xA8, 0xA9, 0xAA, 0xB2, 0xB3, 0xB4,
    0xB5, 0xB6, 0xB7, 0xB8, 0xB9, 0xBA, 0xC2, 0xC3,
    0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xD2,
    0xD3, 0xD4, 0xD5, 0xD6, 0xD7, 0xD8, 0xD9, 0xDA,
    0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7, 0xE8, 0xE9,
    0xEA, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8,
    0xF9, 0xFA
};
int tableSelectMask= 0;

FILE *fpin, *fpout;
int linenum=0;
int ac;
char *av[20];
char line[256];
char filename[64]= "LOAD_RAM_DATA";
int encode_mode=1;

int error_flag=0;


/* Forward Declarations: */
void parse_table_def_file();
void read_huff_table_def();
void jpHuffBuildRam();
void jpGenDecodeRam();
int get_base_offset();
int get_num_leading1();
void jpGenOffset();
int StrIsWhite();
void StrRemoveTrSpace();
int scan();

main(argc, argv)
    int argc;
    char **argv;
{
    int element;

    /* Print title: */
    printf("\n\nL64702 Huffmann Ram Data Generation Program version 1.0:\n");
    printf("Copyright (C) 1992 LSI Logic Corporation.\n");
    printf("All rights reserved.\n\n");

    /* Parse command line arguments: */
    if (argc <2)
    {
```

```
            fprintf(stderr,"usage: %s SPEC_INPUT_FILE_NAME
[output_file_name] [dec]\n", argv[0]);
            exit(1);
    }
    else
    {
            if (!(fpin=fopen(argv[1], "r")))
            {
                fprintf(stderr,"%s error: cannot read file %s!\n", argv[0],
argv[1]);
                exit(1);
            }
            if (argc>=3)
                (void) strcpy(filename, argv[2]);

            if (!(fpout=fopen(filename, "w")))
            {
                fprintf(stderr,"%s error: cannot write file %s!\n", line);
                exit(1);
            }

            printf("reading Huffmann Table definitions from file %s\n",
argv[1]);
            printf("Huffmann Ram data will be in file %s\n", filename);
            if (argc >3 && !strcmp("dec",argv[3]))
            {
                printf("generating Huffmann ram contents for decoding
...\n");
                encode_mode = 0;
            }
            else
                printf("generating Huffmann ram contents for encoding
...\n");
    }

    /* Read input file for table definitions: */
    parse_table_def_file(fpin);

    if (error_flag)
    {
            fprintf(stderr,"%d errors found!  Ram data generation
aborted!\n", error_flag);
            exit(1);
    }

    /* Build the arrays for the Huffmann rams: RR640, RR56 */
    jpHuffBuildRam(encode_mode);
```

```
    /* Print out the ram data into output file: */
    fprintf(fpout,"HUFFMANN_RAM\nSTART_ADDRESS 000\nSIZE 696\n");
    for (element=0; element < HUFF_RAM640_SIZE; element++)
    {
            fprintf(fpout,"%4.4x\n", RR640[element]);
    }
    for (element=0; element < HUFF_RAM56_SIZE; element++)
    {
            fprintf(fpout,"%4.4x\n", RR56[element]);
    }


    fclose(fpin);
    fclose(fpout);
    printf("done.\n");
    return(0);
}

/* Read in the table definitions.  Huffmann tables
 * defined as in JPEG standard.
 * See sample input file for syntax.
 */
void
parse_table_def_file(fpin)
    FILE * fpin;
{

    /* Read in each table definitions one at a time: */
    while (!StrGetLine(fpin, line, 256, &linenum))
    {
            /* lines beginning with '*' are comments. */
            if (line[0] == '*')
                continue;

            scan(line, &ac, av, 12);
            if (!strcmp("#HUFF_DC", av[0]))
               read_huff_table_def(fpin, DC, HuffDC0, HuffDC1, atoi(av[1]),
&linenum);
            else if (!strcmp("#HUFF_AC", av[0]))
               read_huff_table_def(fpin, AC, HuffAC0, HuffAC1, atoi(av[1]),
&linenum);
            else
                fprintf(stderr,"error at line %d: unknown keyword %s!\n",
                          linenum, av[0]);
    }
```

```
    if (tableSelectMask != 15)
     {
               fprintf(stderr,"error: Table definitions are incomplete.\n");
               fprintf(stderr,"      All tables must be generated together!\n");
               error_flag++;
     }
}

void
read_huff_table_def(fp, ac_dc, table0, table1, table_num, linenum)
     FILE * fp;
     int ac_dc;
     int *table0, *table1;
     int table_num, *linenum;
{
     int item;
     int lineCt;
     int num_of_para;

     if (table_num >1)
     {
               table_num= 0;
               fprintf(stderr,"error at line %d: Illegal table number!  using 0
instead.\n",
                              *linenum);
     }

     if (ac_dc == DC)
     {
               if ((table_num==0 && (SEL_HUFF_DC0 & tableSelectMask)) ||
                   (table_num==1 && (SEL_HUFF_DC1 & tableSelectMask)))
               {
                   fprintf(stderr,"error at line %d: Table %d
redefined!\n",*linenum, table_num);
                   error_flag++;
               }
               else
                   tableSelectMask |= (1<<(table_num));
               num_of_para =28;
     }
```

```
    else
    {
            if ((table_num==0 && (SEL_HUFF_AC0 & tableSelectMask)) ||
                (table_num==1 && (SEL_HUFF_AC1 & tableSelectMask)))
            {
                fprintf(stderr,"error at line %d: Table %d
redefined!\n",*linenum, table_num);
                error_flag++;
            }
            else
                tableSelectMask |= (4<<(table_num));
            num_of_para =178;
    }

    for (lineCt=0; lineCt<num_of_para; )
    {
            if (!StrGetLine(fp, line, 256, linenum))
            {
                if (line[0]=='*') continue;   /* comments */
                if (line[0]=='#')        /* Beginning of next key word */
                {
                        fprintf(stderr,"error at line %d: insufficient
parameters in Huffmann spec!\n",
                                        *linenum);
                        error_flag++;
                        return;
                }
                scan(line, &ac, av, 12);
                for (item=0; item<ac ;item++)
                {
                        if (table_num)
                            sscanf(av[item],"%x", (table1+lineCt));
                        else
                            sscanf(av[item],"%x", (table0+lineCt));
                        lineCt++;
                }
            }
            else
            {
               fprintf(stderr,"error at line %d: insufficient parameters in
Huffmann spec!\n",
                                        *linenum);
                error_flag++;
                return;
            }
    }
}
```

```
/* ---------------------------------------------------
 *
 * jpHuffBuildRam --
 *           Construct the Rams for encoding or decoding.
 *
 * This procedure initializes all four tables (2 AC, and 2 DC)
 * according to the Huffman code table definition sequences
 * HuffDC0, HuffDC1, HuffAC0, and HuffAC1.
 *
 * Results:
 *           The required data to be loaded into the RR640 and RR56
 *           rams.  Results in array RR640, and RR56.
 *
 * Tricks/Warnings/Notes:
 *
 * Address map for the RR640 ram is as follows for decoding:
 * For AC
 * +-+-------+-------+-----+ to +-+-------+-------+-----+
 * |0|0|0|0|0|0|0|0|table|    |1|0|0|1|0|1|1|1|1|table|
 * +-+-------+-------+-----+    +-+-------+-------+-----+
 * For DC
 * +-+-------+-- ----+-----+ to +-+-------+-------+-----+
 * |1|0|0|1|1|0|0|0|0|table|    |1|0|0|1|1|1|1|1|1|table|
 * +-+-------+-------+-----+    +-+-------+-------+-----+
 * ---------------------------------------------------
 */
void
jpHuffBuildRam(encode)
    int encode;
{
    int max_tail_len[16];/* maximum number of trailing bits
                                                        * in the leading one
groups.
                                                        * Tail length exclude
leading one and
                                                        * first zero.
                                                        */
    int first_code_or_addr[16];/* group first code for encode and
                                                        * group first address
for decode.
                                                        */
    int huff_table_addr[304];
    int huff_table_content[304];
    int num_content;
    int max_code_len;
    int dc ;
    int table;
    int ct1, ct2;
```

```
int *bits;                              /* pointer to table definition. */

/* Initialize the arrays so that unused location
 * can be located easily.
 */
for (ct2=0; ct2<HUFF_RAM56_SIZE; ct2++)
        RR56[ct2] = 0xfff;
for (ct2=0;ct2<HUFF_RAM640_SIZE;ct2++)
        RR640[ct2] = 0xfff;

/* there are 4 tables */
for (ct1=0; ct1<4; ct1++)
{
        switch (ct1)
        {
            case 2:
                    bits = HuffDC0;
                    dc = 1;
                    table=0;
                    break;
            case 3:
                    bits = HuffDC1;
                    dc = 1;
                    table = 1;
                    break;
            case 0:
                    bits = HuffAC0;
                    dc = 0;
                    table=0;
                    break;
            case 1:
                    bits = HuffAC1;
                    dc = 0;
                    table = 1;
                    break;
        }
        max_code_len = (dc?12:16);
        if (encode)
        {
            jpGenOffset(bits, dc, first_code_or_addr,
                                  huff_table_addr,huff_table_content,
                                  &num_content);
            for (ct2=0; ct2 < max_code_len; ct2++)
            {
                    RR56[(((dc<<4)|ct2)<<1)|table] =
                                  first_code_or_addr[ct2];
            }
        }
```

```
            else
            {
                jpGenDecodeRam(bits, dc, first_code_or_addr,
                                        max_tail_len, huff_table_addr,
                                        huff_table_content, &num_content);

                for (ct2=0; ct2 < max_code_len; ct2++)
                {
                        RR56[(((dc<<4)|ct2)<<1)|table] =

((first_code_or_addr[ct2]&(0x3ff))<<3) |
                                        (max_tail_len[ct2]&7);
                }
            }

            for (ct2=0; ct2 <num_content; ct2++)
            {
                RR640[(huff_table_addr[ct2]<<1)|table] =
                            huff_table_content[ct2];
            }
    }
}

/* Generate the contents of the 2 rams for decoding according to
 * the huffmann table sequence bits, and events.
 */
void
jpGenDecodeRam(bits, dc,
            grp_first_addr, max_tail_len,
            huff_table_addr, huff_table_content, num_content)
    int bits[];
    int dc;
    int max_tail_len[16];
    int grp_first_addr[16];
    int huff_table_addr[];
    int huff_table_content[];
    int *num_content;
{
    int *events;/* symbols represented by code words. */
    int code[162];      /* Huffmann code words. */
    int code_len[162];
    int code_leading1_grp[162];/* Leading group that the code
                                                    * word is in.
                                                    */
    int ct, ct1, num;
    int max_code_len;
    int num_leading1, new_max;
    int base_code, num_code;
```

```
int base_addr;
int base_offset;
int table_content;
int code_tail_len;
int num_addr_to_fill;

events = bits+16;
for (ct=0; ct < 16; ct++)
        max_tail_len[ct] = 0;

max_code_len = (dc?12:16);

num_code= 0;
base_code= 0;
num_leading1= 0;

/* Generate all codewords and update the leading one group
 * size and their maximum tail length.
 */
for (ct=0; ct<max_code_len; ct++)
{
        for (num=bits[ct]; num>0; num--)
        {
            num_leading1 = get_num_leading1(base_code, ct);
            /* taking off the first 0 */
            if ((new_max=ct-num_leading1) > max_tail_len[num_leading1])
                    max_tail_len[num_leading1] = new_max;
            code[num_code] = base_code++;
            code_leading1_grp[num_code] = num_leading1;

            /* code len is actually code_len -1 */
            code_len[num_code++] = ct;
        }
        base_code <<=1;
}

/* Compute the leading one group first addresses. */
base_addr = (dc?0x130:0);
for (ct=0; ct<max_code_len; ct++)
{
        grp_first_addr[ct] = base_addr;
        base_addr += (1<<max_tail_len[ct]);
}
```

```
    *num_content= 0;
    for (ct=0; ct<num_code; ct++)
    {
            /* possibly more than one address space to fill for one code. */
            base_offset = get_base_offset(code[ct], code_len[ct],
                                                    code_leading1_grp[ct],
1);
            base_offset = base_offset +
grp_first_addr[code_leading1_grp[ct]];

            /* excluding first 0 */
            code_tail_len = code_len[ct] - code_leading1_grp[ct];
            num_addr_to_fill = 1 <<(max_tail_len[code_leading1_grp[ct]] -
                                                    code_tail_len);
            table_content = (events[ct]<<4) | code_len[ct];
            for (ct1=0; ct1<num_addr_to_fill; ct1++)
            {
                /* need to be << by 1 and bit or with table # */
                huff_table_addr[*num_content] = base_offset +
(ct1<<code_tail_len);
                huff_table_content[*num_content] = table_content;
                (*num_content)++;
            }
    }
}

/* The offset is the trailing bits of the codeword.
 * HOWEVER, the significance of the trailing bits are
 * reversed for hardware reason (reduces hardware).
 */
int
get_base_offset(code, code_len, num_leading1, flag)
    int code;
    int code_len;
    int num_leading1;
    int flag; /* two usages */
{
    int ct;
    int mask=0;
    int base_offset=0;

    /* Use this only if the leading1 and first zero hasn't been
     * shifted yet.
     * Shity!  But I don't care ...
     */
    if (flag)
            mask = 1<< (code_len-num_leading1-1);
```

```
    else
            mask = 1<< 15;
    for (ct=0; ct <code_len-num_leading1; ct++)
    {
            if (mask & code)
                base_offset |= (1<<ct);
            mask >>= 1;
    }
    return base_offset;
}

int
get_num_leading1(code, code_len_minus1)
    int code;
    int code_len_minus1;
{
    int mask;
    int ct;

    mask = 1<< code_len_minus1;
    for (ct=0; ct<= code_len_minus1; ct++)
    {
            if (!(mask & code))
                break;
            mask >>= 1;
    }
    if (ct >= 16)
    {
            printf("ERROR: DETECTED 16 or more leading ones.\n");
    }
    return (ct);
}
```

```
/* ------------------------------------------------
 *
 * jpGenOffset --
 *      Generate the ram contents for encoding.
 *
 * ------------------------------------------------
 */
void
jpGenOffset(bits, dc, firstcode,
                huff_table_addr, huff_table_content, num_offset)
    int    *bits;
    int    dc;
    int *firstcode;
    int    huff_table_addr[];
    int    huff_table_content[];
    int    *num_offset;
{
    int tmp;
    int addr;
    int i, j;
    int *events;
    int code;
    int tail_bit;
    int num_lead1;
    int shift;
    int offset;

    *num_offset = 0;
    events = bits + 16;
    tmp = 0;
    code = 0;
    for (i=0; i<16; i++)
    {
            firstcode[i] = 0xfff;  /* default */
            for (j=0; j<bits[i]; j++)
            {
                /* The firstcode for encoding is stored with 12 bit.
                 * 4 msb indicating the number of leading 1s,
                 * and the last 8 bits for the second part of the
                 * cordword.  The second part is stored aligned to msb.
                 */
                if (!j)
                {
                        num_lead1 = get_num_leading1(code, i);
                        shift = 8-(i+1-num_lead1);
                        tail_bit = 0xff & (code << shift);
                        firstcode[i] = (tail_bit<<4) | num_lead1;
                }
```

```
                code++;

                if (dc)
                        addr = 0x100|events[tmp+j];
                else
                        addr = events[tmp+j];

                /* address need to be << by 1 and bit or with table # */
                huff_table_addr[*num_offset] = addr;
                /* i is the codelength and j is the offset.
                 * j is stored in the top 8 bits; aligned towards msb.
                 * Need to know number of leading 1.
                 */
                offset = 0xff & (j << shift);
                huff_table_content[*num_offset] = (offset<<4) | i;
                (*num_offset)++;
            }
            /* update code */
            code <<=1;
            tmp += bits[i];
    }
}


/* ------------------------------------------------------------------
 *
 * scan --
 *          Obtain arguments from a buffer.  Arguments are separated by
 *          spaces.  This code is plaigarized from ~justin/dp/v4/scan.c
 *
 * Results:
 *          0 if done, else 1.
 *
 * Side Effects:
 *
 * ------------------------------------------------------------------
 */
int
scan(buff, ac, av, maxac)
    char *buff;
    int *ac;
    char **av;
    int maxac;
{
    register char *p;

    *ac = 0;
    *av = (char *) 0;
```

```
    p = buff;

scan_loop:
    if (*ac >= maxac) return;

    /* skip over blanks and tabs to the next word */
    /* This is for reading scl tpt input format only.
    while ((*p == ' ') || (*p == '\t') || (*p == '.'))
    */
    while ((*p == ' ') || (*p == '\t'))
    {
        *p = '\0';
        ++p;
    }

    /* return if we reached the end of the buffer */
    if (*p == '\n') *p = '\0';
    if (*p == '\0') return;
    *ac += 1;
    *av++ = p;
    *av = (char *) 0;

    /* advance pointer past the end of the word */
    do
    {
            ++p;
    } while ((*p != ' ')&&(*p != '\t')&&(*p != '\n')&&(*p != '\0'));
    /* This is for reading scl tpt input format only.
   } while ((*p != ' ')&&(*p != '\t')&&(*p != '\n')&&(*p != '\0')&&(*p != '.'));
    */
    goto scan_loop;
}   /* scan */


/* ---------------------------------------------------------------------
 * StrGetLine ##
 *   Get a new non-blank line from fp.
 *
 * Return Value:
 *   -1 if eof reached before a non-blank line can be found.
 *    0 otherwise.
 *
 * Side effects / Tricks / Warnings / Notes:
 *   File pointer moved.  Linenum is updated.
 * ---------------------------------------------------------------------
 */
```

```
StrGetLine(fp, line, size, linenum)
    FILE *fp;
    char *line;
    int size;
    int * linenum;
{
    while (1)
    {
            if (!fgets(line, size, fp))
                return 1;
            (*linenum)++;
            if (!StrIsWhite(line, TRUE))
                break;
    }
    StrRemoveTrSpace(line);
    return 0;
}

int
StrIsWhite(line, commentok)
    char *line;
    short commentok;/* TRUE means # comments are considered all-white */
{
    if ( (*line == '*') && (*(line+1) != '/') && commentok)
            return TRUE;
    while(*line)
    {
            if ( !isspace(*line) && (*line != '\n') )
                return FALSE;
            line++;
    }
    return TRUE;
}

void
StrRemoveTrSpace(s)
    char *s;
{
    int ct;
    char *ptr;

    ct = strlen(s)-1;
    ptr = s+ct;
    while (*ptr == ' ' || *ptr == '\n' || *ptr == '\t')
            ptr--;
    *(ptr+1) = '\0';
}
```

Appendix C
Customer Feedback

We would appreciate your feedback on this document. Please copy the following page, add your comments, and fax it to us at:

LSI Logic Corporation
Microprocessor Publications
M/S G-812
Fax 408.433.8989

If appropriate, please also fax copies of any marked-up pages from this document.

IMPORTANT: Please include your name, phone number, FAX number, and company address so that we may contact you directly for clarification or additional information. Thank you for your help in improving the quality of our documents.