CL450 MPEG Video Decoder
User's Manual

C-Cube
Microsystems

C-Cube Microsystems
1778 McCarthy Boulevard
Milpitas, CA 95035
Telephone (408) 944-6300
Fax (408) 944-6314

Customer Comments and Feedback:

If you have any comments about this document, send them to the C-Cube Technical Publications Department at the address listed above, or send e-mail to:

techpubs@c-cube.com

C-Cube Part # 90-1450-101

Preface

This manual is the primary source of technical information for the operation and programming of the C-Cube CL450 MPEG Video Decoder.

Audience

This manual is intended for:

- System designers and managers who are evaluating the CL450 for possible use in a system.
- Design, software and system engineers developing a video decoding system using the CL450 for whom a comprehensive programming background as well as a detailed understanding of MPEG compression is assumed.

Related Publications

In particular, readers should understand the MPEG standard: *Coded Representation of Picture, Audio and Multimedia/Hypermedia Information*, ISO/IEC JTC 1/SC 29, December 6, 1991.

As an aid to developing systems and applications based on the CL450, readers may also wish to obtain C-Cube's *CL450 MPEG Video Decoder Sample Kit*, which includes a sample development board, microcode and user's guide.

Organization

This manual is divided into three main sections:

- Section I, General Information, including an introduction to the CL450 and the MPEG standard that it implements.
- Section II, Hardware Interface, including signal descriptions, operational information for the main interfaces of the CL450, registers, and detailed electrical and mechanical specifications.

- Section III, Software Interface, including an overview of the fundamental mechanisms used to communicate between the host system and the CL450's microapplication, and an alphabetical listing of all macro commands and interrupts available to the programmer. Note: the microapplication information in this manual applies to CL450 versions 2.00 through 2.FF *only*.

Conventions

Please note the following notation examples and conventions used in this manual:

Notation Examples	Explanation
0x1c3	"0x" prefix indicates a hexadecimal number.
11011_2	"2"subscript indicates a binary number.
IMEM	Four-letter mnemonics indicate on-chip memories, starting with a letter to indicate function of memory, and ending with "MEM."[1] For example, IMEM is the CL450's instruction memory.
HOST_control CMEM_control	This format indicates a register name. The first part of a register name is a group specifier, given in all upper-case. The second part (separated from the first by an underscore and given in all lower-case) is a register specifier, indicating the function performed by the register within the group. In the examples shown, HOST_control and CMEM_control are separate registers, even though both have the same register specifier, "control."
VIE, Res	Italicized acronyms or abbreviations (initial or all caps) indicate bit field names within registers or data words.
HMEM[3] CPU_control[0] HMEM[0][15] HMEM[3][2:0]	Square bracket notation similar to C language array subscripting indicates words within memories, and bits within words and registers. HMEM[3], for example, is the 16-bit word at address 3 within HMEM, while CPU_control[0] is bit 0 of the CPU_control register. Similarly, HMEM[0][15] is the most significant bit of the word at HMEM address 0. Ranges of bits are indicated by numbers separated by a colon such as the three bits HMEM[3][2:0]. Ranges of words within a memory are indicated by numbers separated by a dash such as the eight words HMEM[0-7].
RESERVED or *Res*	Indicates bit fields within registers which are not defined. RESERVED bit fields may return any value when read and must be written with 0 (or 1 if so specified). Writing the incorrect value to a RESERVED CL450 register bit will cause indeterminate behavior. In addition, all CL450 registers which are not explicitly given names are also RESERVED, and accessing these registers may cause indeterminate results on current or future CL450 implementations.
leftBorder	Bold-face type represents macro command arguments.
`return();`	C-style syntax presented in courier typeface represents program pseudocode and equations.
`vbv_delay`	Names presented in courier represent names of items within the MPEG bitstream taken from the MPEG standard.

1. In some cases, this current reference scheme departs from the method used by the first edition of the *CL450 MPEG Video Decoder User's Manual* which, for example, referred to CMEM as both the "Coded Data FIFO" and "C-FIFO."

Contents

Section I. General Information

Section II. Hardware Interface

Section III. Software Interface

13 Audio/Video Synchronization

14 Host Access of DRAM Variables

15 CL450 Host-independent Operations

Appendix A CL450 DRAM-Variable Allocation

Figures

Tables

Chapter 1
Introduction

1.1 General Description

The C-Cube CL450™ is the world's first single-chip MPEG video decoder. It is designed to provide full-motion video capability for cost-sensitive consumer electronics products. The CL450 decompresses constrained-parameters MPEG bitstreams (typically SIF resolution) in real time. SIF resolution is 352 x 240 pixels at 30 Hz or 352 x 288 pixels at 25 Hz, with typical compressed data rates of 1.2 to 3 Mbits per second.

The CL450 interpolates decompressed pictures horizontally (typically to 704 pixels per scan line) before outputting them. The CL450 outputs decompressed pictures multiple times to increase the frame rate from 24, 25, or 30 Hz (coded frame rate) to 50 or 60 Hz (display frame rate). The CL450 can change the display position of decompressed pictures relative to its $\overline{\text{HSYNC}}$ and VSYNC inputs.

1.2 The CL450 Product Family

The CL450 is a fully-integrated MPEG decoding engine whose functionality is an intertwined combination of hardware and microapplication set provided by C-Cube, as explained in the software interface section of this manual, Section III.

1.2.1 CL450 Features

The basic CL450 product includes these features:

- Fully complies with all requirements of the MPEG standard (ISO CD 11172)
- Performs real-time decoding of SIF-resolution bitstreams (352 x 240 pixels at 30 Hz or 352 x 288 pixels at 25 Hz)
- Performs real-time horizontal pixel interpolation and frame duplication to produce output formats of 704 x 240 pixels at 60 Hz or 704 x 288 pixels at 50 Hz
- Provides either RGB or YCbCr video output using an on-chip color-space converter
- Supports NTSC and PAL video output timing formats
- Interfaces to 680x0 processors and DRAM with no external logic
- Requires only 4 Mbits of 80-ns DRAM to decode SIF-resolution MPEG bitstreams
- Provides hardware and microapplication support for audio/video synchronization
- Decodes Huffman variable-length codes at a peak rate of 4 bits per clock (160 Mbits/second at 40 MHz)
- Allows the active display window to be positioned relative to $\overline{\text{HSYNC}}$ and VSYNC inputs with one-pixel and one-$\overline{\text{HSYNC}}$ accuracy, respectively
- Displays all or part of decompressed pictures; displayed section can be selected to one-pixel accuracy (within the decoded picture) and can be changed every coded frame for panning motion video
- Supports host access of local DRAM including byte writes
- Supports both programmed I/O and DMA transfers of compressed bitstreams from the host
- Is fabricated in a CMOS process
- Is supplied in a 160-pin plastic quad flat-pack (PQFP)

1.2.2 Microapplication Features

The CL450 microapplication includes these features:

- 18 macro commands, allowing the host to control interactive playback, CL450 configuration, and audio/video synchronization:
  - AccessSCR()
  - DisplayStill()
  - FlushBitstream()
  - InquireBufferFullness()
  - NewPacket()
  - Pause()
  - Play()
  - Reset()
  - Scan()
  - SetBlank()
  - SetBorder()
  - SetColorMode()
  - SetInterruptMask()
  - SetThreshold()
  - SetVideoFormat()
  - SetWindow()
  - SingleStep()
  - SlowMotion()
- 11 interrupts, providing the host feedback on bitstream transition, display, and decoding processes
- Bit rate up to 5.0 Mbits per second
- Transcoding between NTSC and PAL input and output frame rates
- System memory expansion capability
- Support for 24-Hz film format input
- Display window positioning within decoded picture to single-pixel accuracy
- Flagged and unflagged error concealment
- MPEG double vertical-resolution still pictures

1.3 Functional Description

Figure 1-1 shows a block diagram of the CL450, which has three interfaces:

- *The host interface*: Connects directly to 680x0 processors with no external logic. It can also be easily connected to 80x86 processors.
- *The DRAM interface*: Reads from and writes to the local DRAM with no external logic.
- *The video interface*: Outputs pixel data to a video monitor or other video processing device.

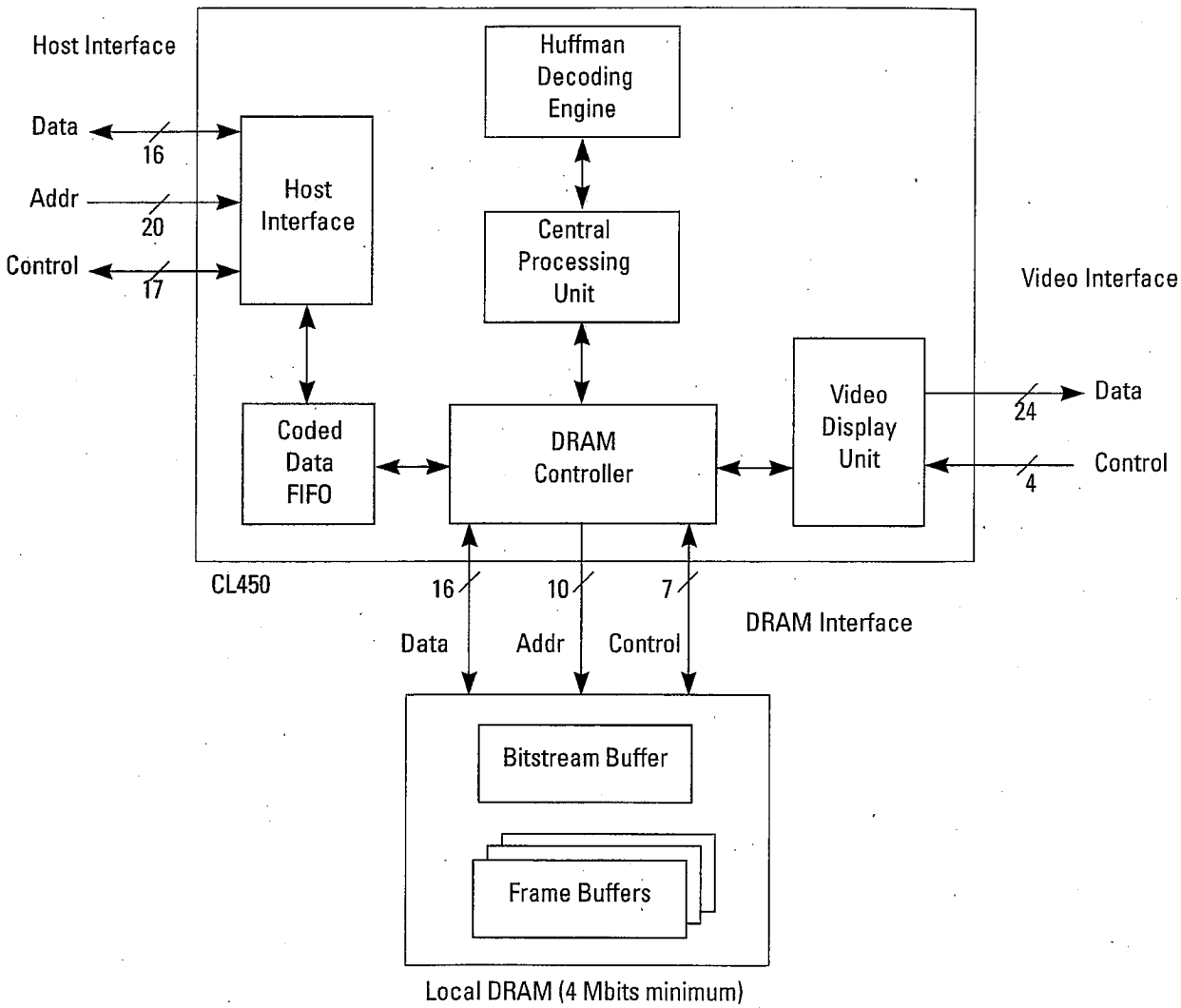


Figure 1-1 Block Diagram of the CL450

The host computer supplies compressed (coded) data to the CL450 via the host interface. The CL450 buffers up to 16 coded data words in an internal coded data FIFO, called CMEM, from which the data words are read by the DRAM controller and written into the bitstream buffer in the local 4-Mbit DRAM.

The on-chip central processing unit (CPU) interprets commands issued by the host processor, works with the Huffman decoding engine to perform the decompression process, and writes decompressed pixel data into the frame buffer in the local DRAM.

The video display unit reads decompressed pixel data from the frame buffer, sends it through the color-space converter if necessary, and outputs the pixel data on the video bus.

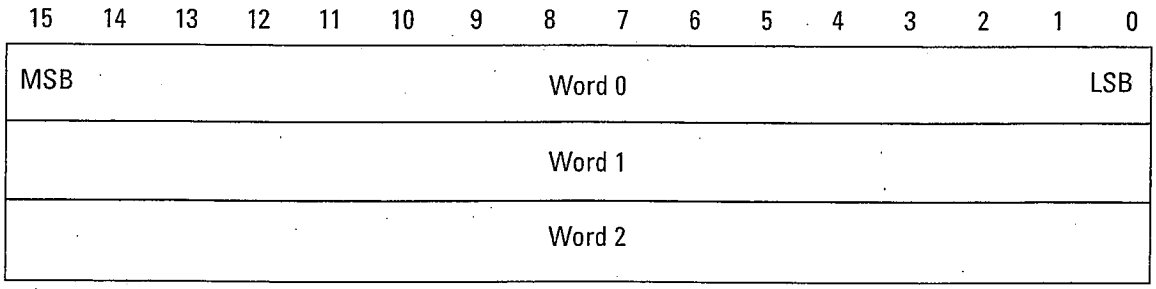
1.4 Data Organization

Though the CL450 can be used with many different processors, it has been optimized for use with the Motorola 68070 processor.

1.4.1 CL450 Registers

Each of the 34 registers of the CL450 is 16 bits wide and is accessed by word operations which cover the entire 16-bit width. The least significant bit of each address is bit 0, and the most significant bit is bit 15.

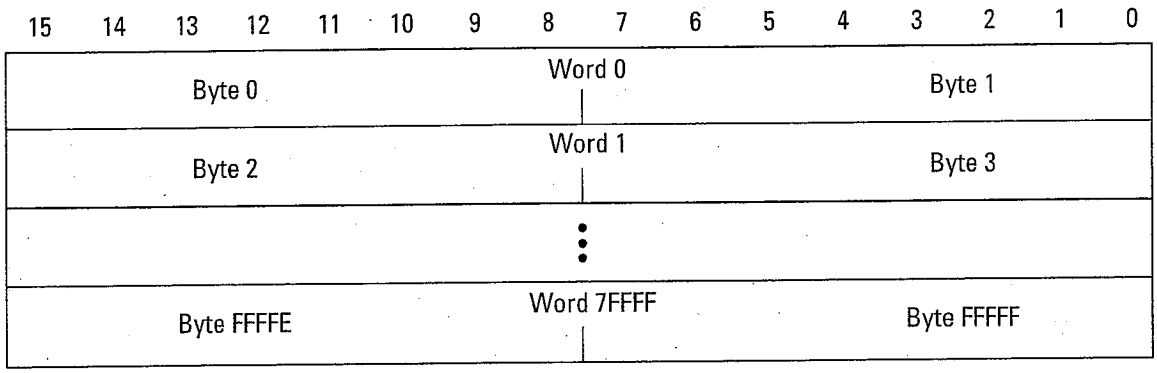
Even though CL450 register addresses may be expressed as byte addresses, the CL450 does not support single-byte access to its internal registers; instead, all CL450 register accesses must be made using 16-bit operations (i.e., 16 bits at a time) as shown below.



1.4.2 CL450 Memories

All on-chip memory transactions on the CL450 must be 16 bits (as shown in the previous figure), while the local (off-chip) DRAM may be accessed by the host through the CL450 using single-byte access.

When byte-addressable access to local DRAM is performed, the higher-order byte has an even address twice (2x) that of the word (shown below), and the low-order byte has an odd address that is one count higher than twice the word address (2x + 1). The $\overline{\text{UDS}}$ (Upper Data Strobe) and the $\overline{\text{LDS}}$ (Lower Data Strobe) signals distinguish byte operations from word operations as explained in Chapter 4.



1.5 Address Size

The size of CL450 addresses within a memory (on-chip memories or the off-chip DRAM) are given in units which are the same as the word size of the RAM in question. For example, the off-chip DRAM is configured as 256K or 512K words of 16 bits each, with DRAM addresses therefore being within the range 0 to 0x3ffff or 0x7ffff words and each distinct address containing a 16-bit value.

The word size of the CL450 memories are shown in Table 1-1.

Table 1-1 Address Size of CL450 Memories

Memory	Address Unit Size
On-chip registers	16 bit
Off-chip (local) DRAM	16 bit
HMEM	16 bit
IMEM	32 bit
TMEM	24 bit
CMEM	16 bits

Note: Writing host software to interact with the CL450 on byte-addressable processors may necessitate multiplying the given DRAM and register addresses by 2.

1.6 Programming Overview

The CL450 performs its higher-level functions by executing a microapplication. The microapplication must be loaded into the CL450 by a software driver before the CL450 is fully operational. C-Cube provides C-language source code for a loader, which users may modify and compile on their target system.

Application programs access the CL450 in two ways:

- *Registers*: Some of the CL450's internal registers are accessed by application programs to set key operating parameters during initialization and operation. Other CL450 registers are used to load command IDs and arguments for the macro commands described below and to determine the status of microapplication execution. Chapter 8, Registers, describes the use and format of the CL450's registers.
- *Macro commands*: Use of these commands is the primary method for communicating between the host software and the microapplication executing on the CL450's CPU. Section 1.2.2 on page 1-3 shows the macro commands implemented in the CL450 microapplication set. Section III of this manual, Software Interface, describes the syntax and use of each macro command.

1.7 Typical Applications

The CL450 MPEG decoder is designed for low-cost consumer applications such as:

- CD–I systems
- Video games
- Interactive multimedia systems
- Point-of-sale/information kiosks
- Interactive TV

Figure 1-2 shows the CL450 in a typical system application.

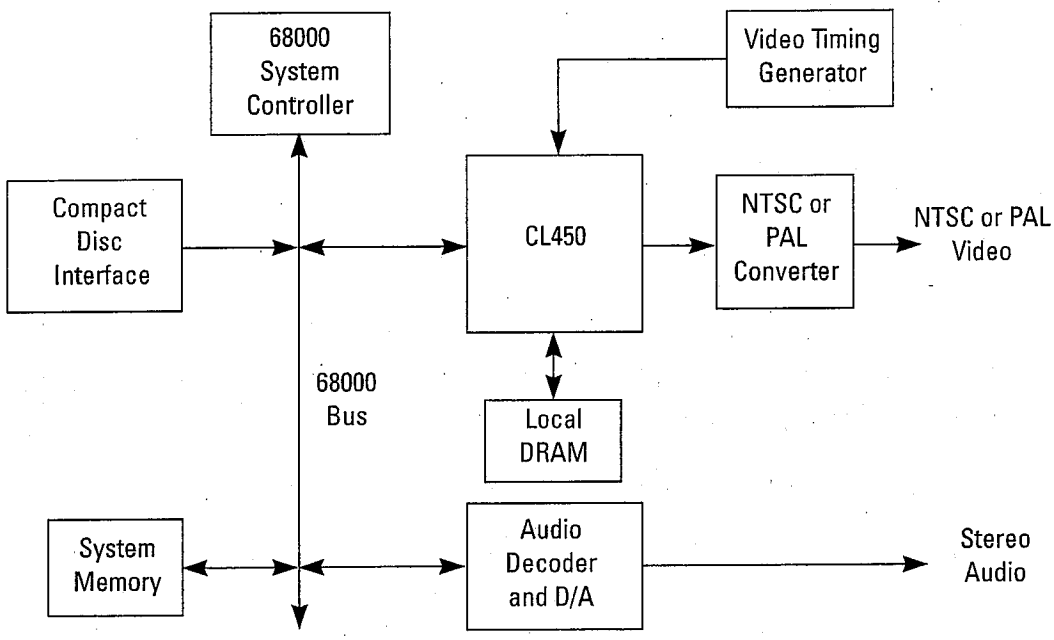


Figure 1-2 Typical CL450 System Application

2
MPEG Overview

This chapter presents an overview of the Moving Picture Experts Group (MPEG) standard that is implemented by the CL450. The standard is officially known as ISO/IEC Draft Standard *Coded representation of picture, audio and multimedia/hypermedia information*, CD 11172, December 6, 1991. It is more commonly referred to as the *MPEG standard*.

MPEG addresses the compression and decompression of video and audio signals and the synchronization of audio and video signals during playback of decompressed MPEG data. The MPEG video algorithm can compress video signals to an average of about 1/2 to 1 bit per coded pixel. At a compressed data rate of 1.2 Mbits per second, a coded resolution of 352 x 240 at 30 Hz is often used, and the resulting video quality is comparable to VHS.

2.1 MPEG Decoding

This section explains the general structure of an MPEG stream and introduces some basic concepts used in the rest of the chapter.

2.1.1 MPEG Stream Structure

In its most general form, an *MPEG stream* is made up of two layers:

- The *system layer* contains timing and other information needed to demultiplex the elementary audio and video streams and to synchronize audio and video during playback.
- The *compression layers* include the elementary audio and video streams.

2.1.2 General Decoding Process

Figure 2-1 shows a generalized decoding system using the CL450 as the decoder for the elementary video stream.

The *system decoder* extracts the timing information from the MPEG system stream and sends it to the other system components. (Section 2.4, Synchronization, has more information about the use of timing information for audio and video synchronization.) The system decoder also demultiplexes the elementary video and audio streams from the system stream and sends each to the appropriate decoder. In many applications, the system decoder function is implemented as a software program on the host computer.

The *video decoder* decompresses the elementary video stream as specified in Part 2 of the MPEG standard. (See Section 2.2, Inter-picture Coding, and Section 2.3, Intra-picture Coding, for more information about video compression.) The CL450 performs the video decoding function.

The *audio decoder* decompresses the audio stream as specified in Part 3 of the MPEG standard.

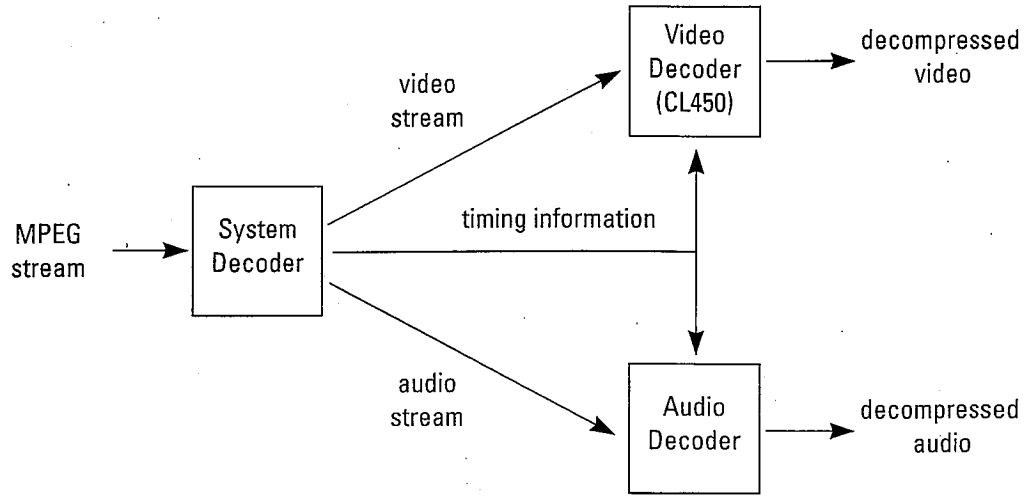


Figure 2-1 General MPEG Decoding System

2.1.3 Video Stream Data Hierarchy

The MPEG standard defines a hierarchy of data structures in the video stream as shown schematically in Figure 2-2.

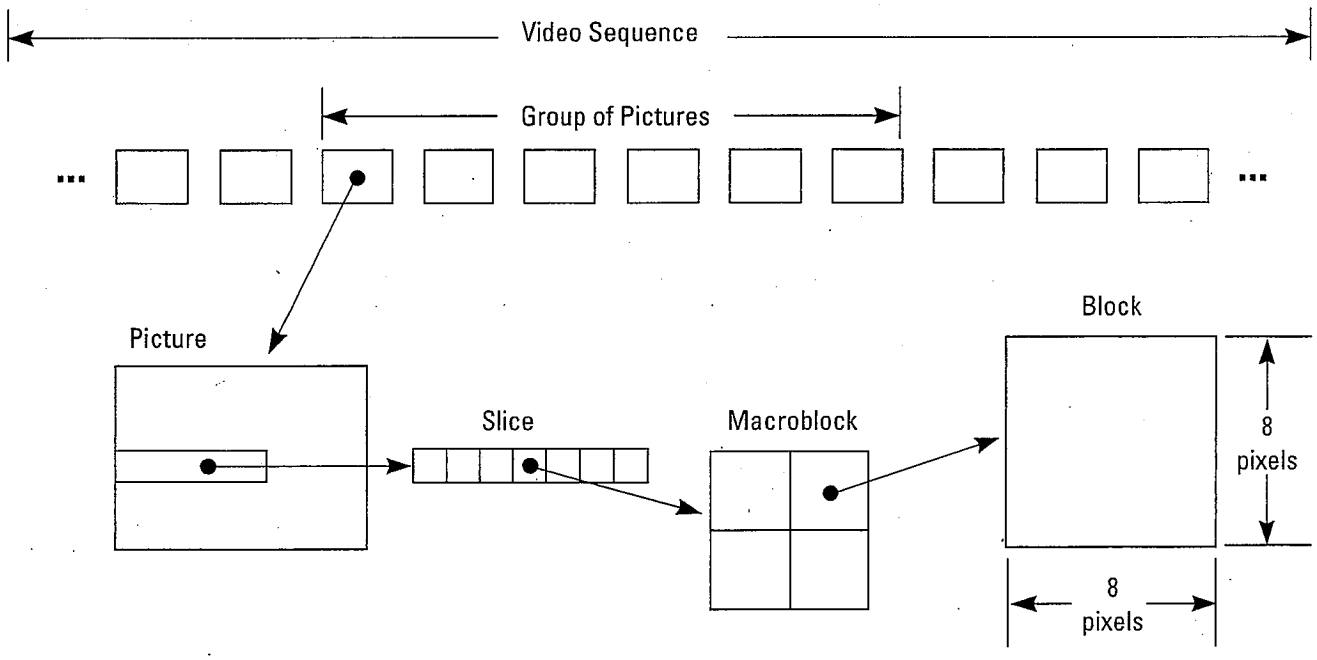


Figure 2-2 MPEG Data Hierarchy

Video Sequence

A sequence begins with a sequence header (and may contain additional sequence headers), includes one or more groups of pictures, and ends with an end-of-sequence code.

Group of Pictures (GOP)

A header followed by a series of one or more pictures; intended to allow random access into the sequence.

Picture

The primary coding unit of a video sequence. A picture consists of three rectangular matrices representing luminance (Y) and two chrominance (Cb and Cr) values. The Y matrix has an even number of rows and columns. The Cb and Cr matrices are each one-half the size of the Y matrix in both directions, horizontal and vertical.

Figure 2-3 shows the relative x-y locations of the luminance and chrominance components. Note that for every four luminance values, there are two associated chrominance values: one Cb value and one Cr value. (The location of the Cb and Cr values is the same, so only one circle is shown in the figure.)

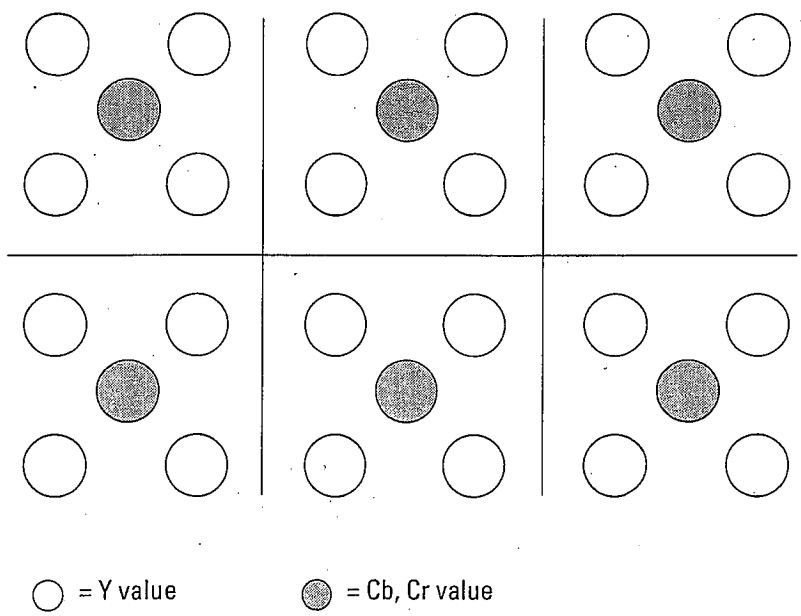


Figure 2-3 Location of Luminance and Chrominance Values

Slice

One or more "contiguous" macroblocks. The order of the macroblocks within a slice is from left to right and top to bottom.

Slices are important in the handling of errors. If the bitstream contains an error, decoders may skip to the start of the next slice. Having more slices in the bitstream allows better error concealment but uses bits that could otherwise be used to improve picture quality.

Macroblock

A 16-pixel by 16-line section of luminance components and the corresponding 8-pixel by 8-line sections of the two chrominance components. See Figure 2-3 for the spatial location of luminance and chrominance components. A macroblock contains four Y blocks, one Cb block and one Cr block as shown in Figure 2-4. The numbers correspond to the ordering of the blocks in the data stream, with block 1 first.

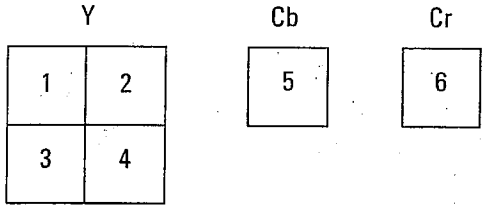


Figure 2-4 Macroblock Composition

Block

A block is an 8-pixel by 8-line set of values of a luminance or a chrominance component. Note that a luminance block corresponds to one-fourth as large a portion of the displayed image as does a chrominance block.

2.2 Inter-picture Coding

Much of the information in a picture within a video sequence is similar to information in a previous or subsequent picture. The MPEG standard takes advantage of this temporal redundancy by representing some pictures in terms of their differences from other (reference) pictures, or what is known as *inter-picture coding*. This section describes the types of coded pictures and explains the techniques used in this process.

2.2.1 Picture Types

The MPEG standard specifically defines three types of pictures used for inter-coding: intra, predicted, and bidirectional.

Intra Pictures

Intra pictures, or I-pictures, are coded using only information present in the picture itself. I-pictures provide potential random access points into the compressed video data. I-pictures use only transform coding (as explained in Section 2.3 on page 2-10) and provide moderate compression. I-pictures typically use about two bits per coded pixel.

Predicted Pictures

Predicted pictures, or P-pictures, are coded with respect to the nearest previous I- or P-picture. This technique is called *forward prediction* and is illustrated in Figure 2-5.

Like I-pictures, P-pictures serve as a prediction reference for B-pictures and future P-pictures. However, P-pictures use *motion compensation* (see Section 2.2.3) to provide more compression than is possible with I-pictures. Also unlike I-pictures, P-pictures can propagate coding errors because P-pictures are predicted from previous reference (I- or P-) pictures.

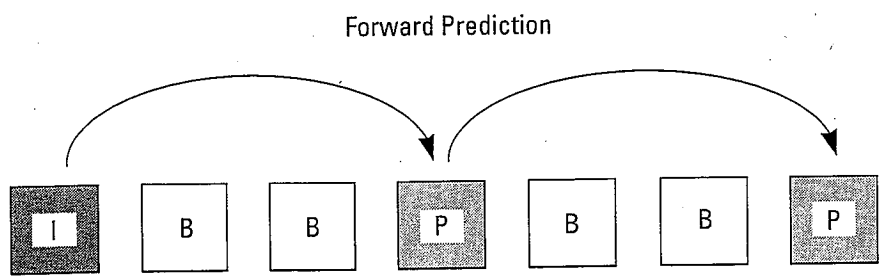


Figure 2-5 Forward Prediction

Bidirectional Pictures

Bidirectional pictures, or B-pictures, are pictures that use both a past and future picture as a reference. This technique is called *bidirectional prediction* and is illustrated in Figure 2-6. B-pictures provide the most compression and do not propagate errors because they are never used as a reference. Bidirectional prediction also decreases the effect of noise by averaging two pictures.

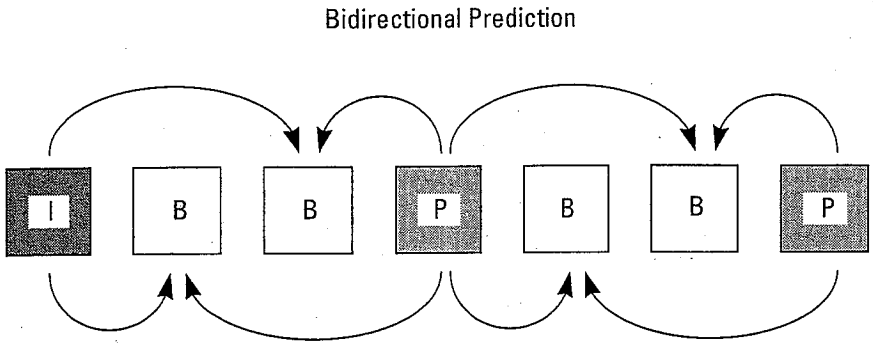


Figure 2-6 Bidirectional Prediction

2.2.2 Video Stream Composition

The MPEG standard allows the encoder to choose the frequency and location of I-pictures. This choice is based on the application's need for random accessibility and the location of scene cuts in the video sequence. In applications where random access is important, I-pictures are typically used two times a second.

The encoder also chooses the number of B-pictures between any pair of reference (I- or P-) pictures. This choice is based on factors such as the amount of memory in the encoder and the characteristics of the material being coded. For a large class of scenes, a workable arrangement is to have two bidirectional pictures separating successive reference pictures. A typical arrangement of I-, P-, and B-pictures is shown in Figure 2-7 in the order in which they are displayed.

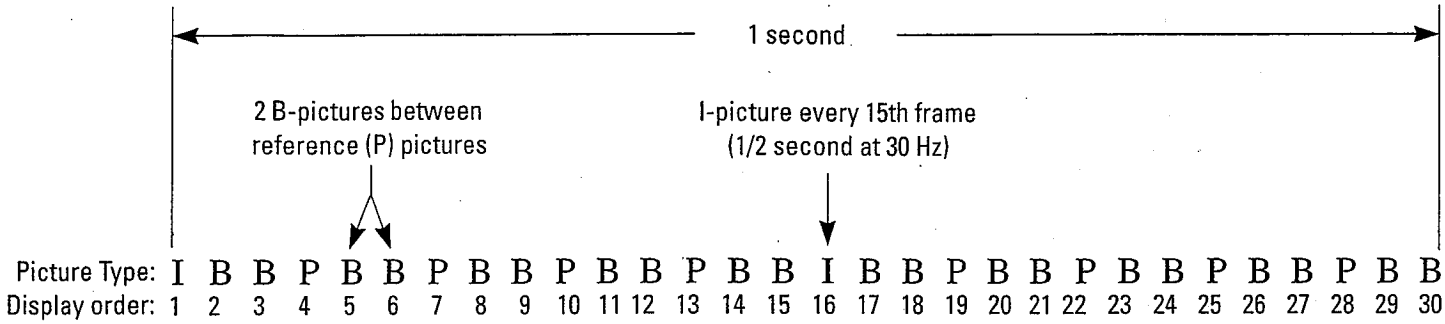


Figure 2-7 Typical Display Order of Picture Types

The MPEG encoder reorders pictures in the video stream to present the pictures to the decoder in the most efficient sequence. In particular, the reference pictures needed to reconstruct B-pictures are sent *before* the associated B-pictures. Figure 2-8 demonstrates this ordering for the first section of the example shown above.

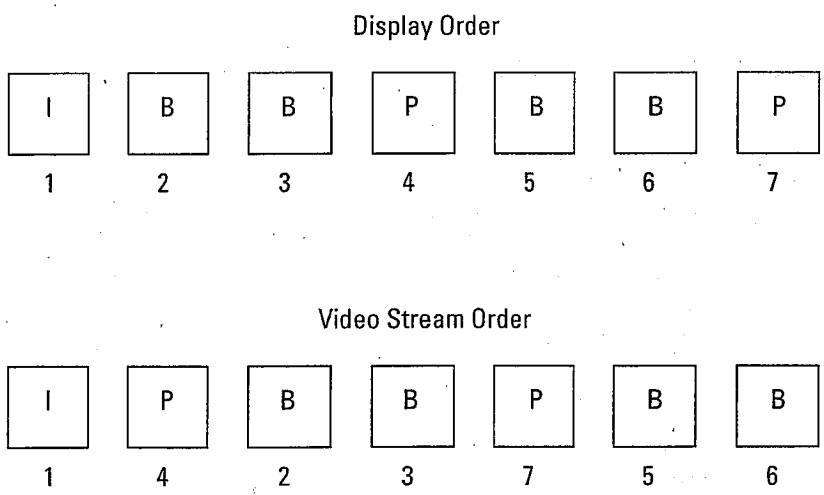


Figure 2-8 Video Stream versus Display Ordering

2.2.3 Motion Compensation

Motion compensation is a technique for enhancing the compression of P- and B-pictures by eliminating temporal redundancy. Motion compensation typically improves compression by about a factor of three compared to intra-picture coding. Motion compensation is performed at the macroblock level.

When a macroblock is compressed by motion compensation, the compressed file contains this information:

- One or two spatial vectors between the reference macroblock area and the macroblock being coded (*motion vectors*)
- The content differences between the reference macroblock area and the macroblock being coded (*error terms*)

However, not all information in a picture can be predicted from a previous picture.

Consider a scene in which a door opens: The visual details of the room behind the door cannot be predicted from a previous frame in which the door was closed. When a case such as this arises—i.e., a macroblock in a P-picture cannot be efficiently represented by motion compensation—the picture is coded in the same way as a macroblock in an I-picture using transform coding techniques (see Section 2.3, Intra-picture Coding).

The difference between B- and P-picture coding is that macroblocks in a P-picture are coded using the previous reference (I- or P-picture) *only*, while macroblocks in a B-picture are coded using any combination of a previous and/or future reference picture.

Four codings are therefore possible for each macroblock in a B-picture:

- Intra coding: no motion compensation
- Forward prediction: the previous reference picture is used as a reference
- Backward prediction: the next reference picture is used as a reference
- Bidirectional prediction: two reference pictures are used, the previous reference picture and the next reference picture

Backward prediction can be used to predict uncovered areas that do not appear in previous pictures.

2.3 Intra-picture (Transform) Coding

The MPEG transform, or intra-picture, coding algorithm includes these steps:

- Discrete cosine transform (DCT)
- Quantization
- Run-length encoding

Both image blocks and prediction-error blocks have high spatial redundancy. To reduce this redundancy, the MPEG algorithm transforms 8 x 8 blocks of pixels or 8 x 8 blocks of error terms from the spatial domain to the frequency domain with the Discrete Cosine Transform (DCT).

Next, the algorithm quantizes the frequency coefficients. Quantization is the process of approximating each frequency coefficient as one of a limited number of allowed values. The encoder chooses a quantization matrix that determines how each frequency coefficient in the 8 x 8 block is quantized. Human perception of quantization error is lower for high spatial frequencies, so high frequencies are typically quantized more coarsely (i.e., with fewer allowed values) than low frequencies.

The combination of DCT and quantization results in many of the frequency coefficients being zero, especially the coefficients for high spatial frequencies. To take maximum advantage of this, the coefficients are organized in a zigzag order to produce long runs of zeros (see Figure 2-9). The coefficients are then converted to a series of run-amplitude pairs, each pair indicating a number of zero coefficients and the amplitude of a non-zero coefficient. These run-amplitude pairs are then coded with a variable-length code, which uses shorter codes for commonly occurring pairs and longer codes for less common pairs.

Some blocks of pixels need to be coded more accurately than others. For example, blocks with smooth intensity gradients need accurate coding to avoid visible block boundaries. To deal with this inequality between blocks, the MPEG algorithm allows the amount of quantization to be modified for each macroblock of pixels. This mechanism can also be used to provide smooth adaptation to a particular bit rate.

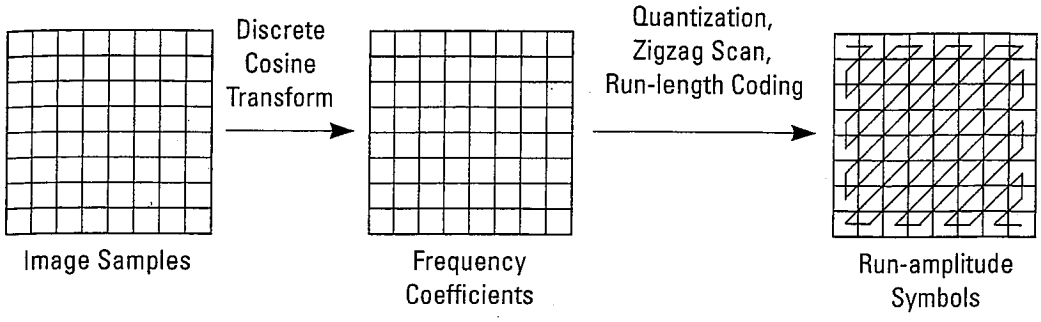


Figure 2-9 Transform Coding Operations

2.4 Synchronization

The MPEG standard provides a timing mechanism that allows for synchronization of audio and video. The standard includes two parameters in the systen layer of a bitstream which are used by the CL450: the *system clock reference (SCR)* and the *presentation time stamp (PTS)*.

The MPEG-specified "system clock" runs at 90 kHz and generates 7.8 x 10^9 clock ticks in a 24-hour day. System clock reference and presentation time stamp values are coded in MPEG bitstreams using 33 bits, which can represent any clock period in a 24-hour period.

2.4.1 System Clock References

A system clock reference is a snapshot of the encoder system clock which is placed into the system layer of the bitstream, as shown in Figure 2-10. During decoding, these values are used to update the clock counter(s) in the system decoder before being sent to the audio and video decoders. This is typically done in a system in which MPEG data is delivered at a fixed bit rate, and effectively locks the decoder's time base with that of the encoder and the bitstream transport mechanism.

The CL450 contains an on-chip counter which it uses as its clock for performing synchronization. The host may update the CL450's clock with SCR information from the bitstream (in fixed bit-rate systems) or from some other accurate, decoder-wide time base.

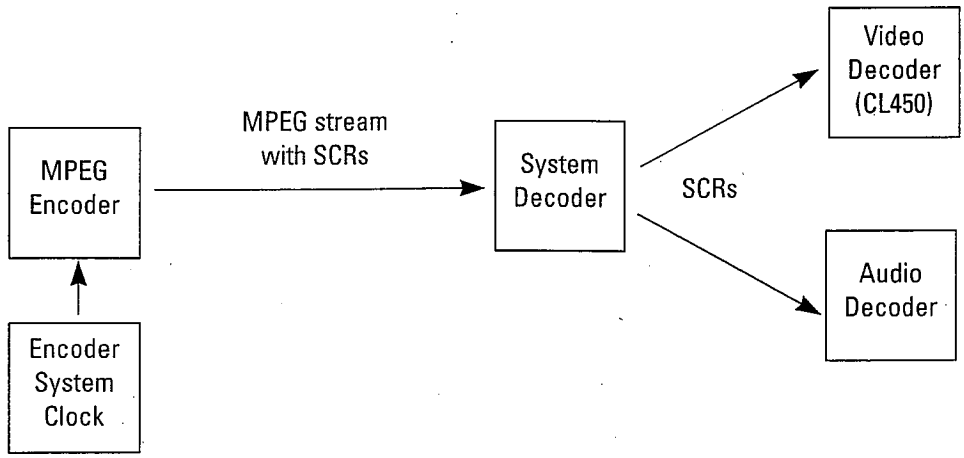


Figure 2-10 SCR Flow in MPEG System

2.4.2 Presentation Time Stamps

Presentation time stamps are samples of the encoder system clock that are associated with video or audio *presentation units*. A presentation unit is a decoded video picture or a decoded audio sequence. The PTS represents the time at which the video picture is to be displayed or the starting playback time for the audio time sequence.

Elementary video or audio decoders use the PTS values associated with each presentation unit to adjust their decoding rate so that each presentation unit is presented (displayed or played) at the correct time as measured by the decoder's system clock.

3
Signal Descriptions

This chapter describes the signals that comprise the external physical interface to the CL450. The information presented for each signal includes the signal mnemonic and name, type (input, output, or bidirectional), and description. For information about the functional operation of the CL450, including functional waveforms, see Chapters 4, 5, and 6. For timing information, see Chapter 7.

This chapter is divided into three sections that correspond to the components that interface to the CL450:

- 3.1: Host Interface
- 3.2: DRAM Interface
- 3.3: Video Interface

Figure 3-1 shows a diagram of the CL450 with the host, DRAM and video signals grouped together.

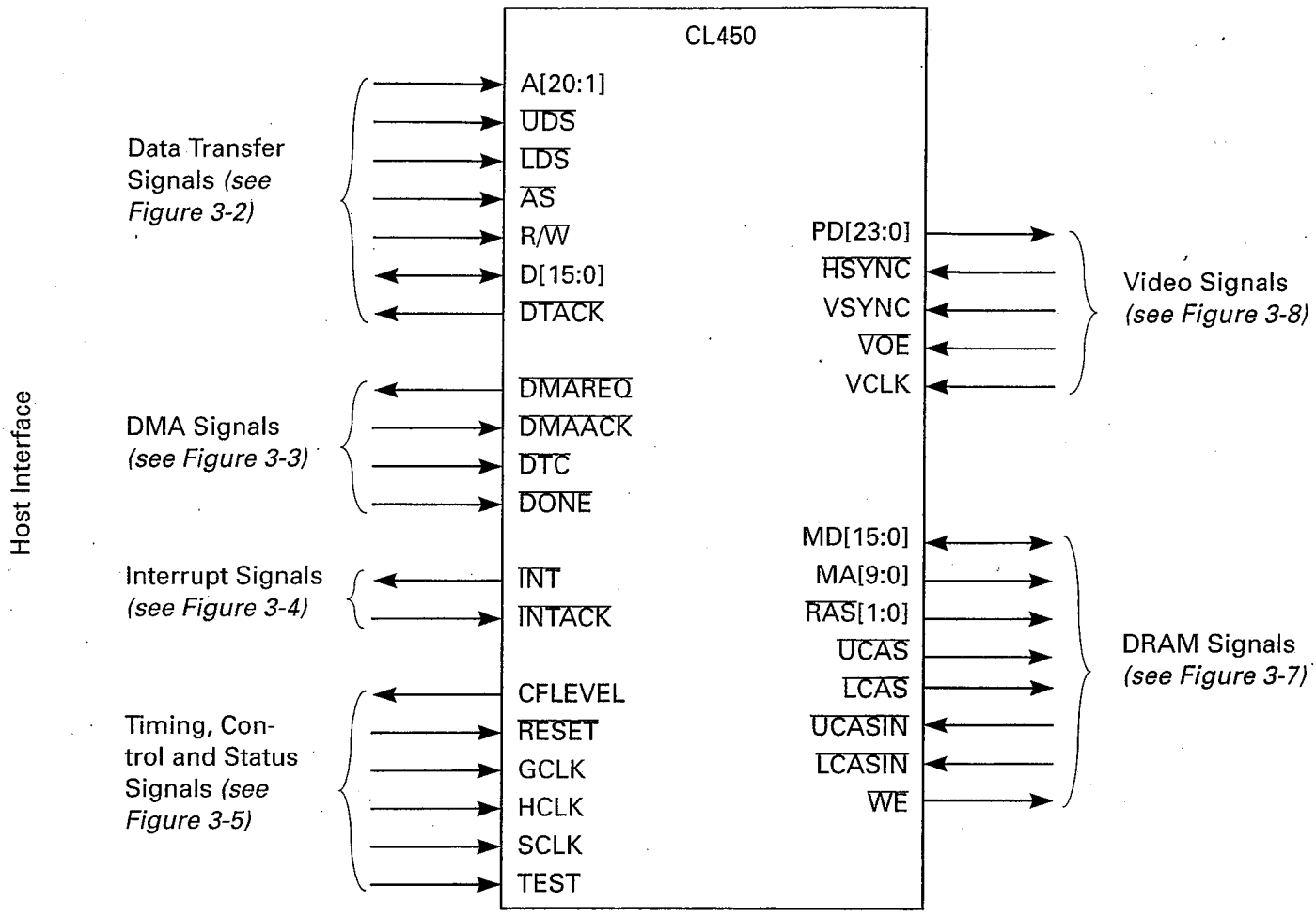


Figure 3-1 Bus Connection Diagram

3.1 Host Interface

The host interface signals divide logically into these functional groups:

- *Data transfer signals*: These signals comprise the address and data buses and the control signals used for data transfer handshaking.
- *DMA signals*: The CL450 uses these signals to implement the SCC68070 DMA protocol. This is a subset of that used by the 68000 family of DMA controllers.
- *Interrupt signals*: These signals provide a request-acknowledge handshake used by the CL450 to request vectored interrupt service from the host.
- *Timing, control, and status signals*: These signals include the clocks and reset signals, and the CMEM status signal.

Note: The following description of host interface signals is based on the Motorola MC680X0 family of microprocessors, even though the CL450 works with a wide range of other processors and corresponding address widths.

3.1.1 Data Transfer Signals

These signals are used to communicate between the CL450 and the host processor. Figure 3-2 shows how the data transfer signals of the CL450 connect to the host processor. The various modes of transferring data are discussed in Sections 4.2 through 4.5.

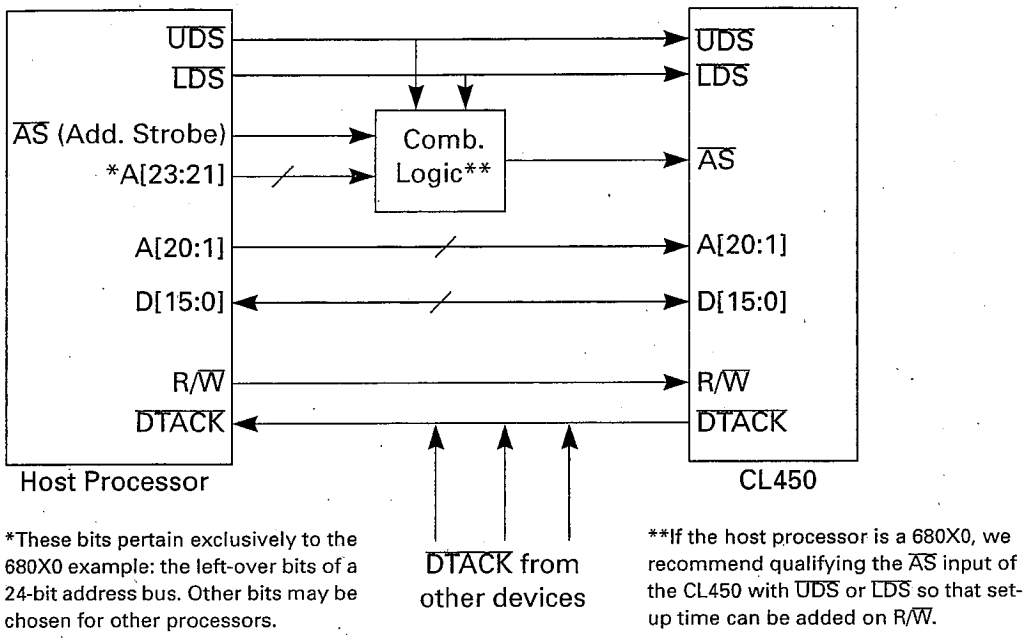


Figure 3-2 Data Transfer Signals

$\overline{\text{UDS}}$ — Upper Data Strobe **Input**
$\overline{\text{LDS}}$ — Lower Data Strobe **Input**

The host processor uses $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ to indicate the byte validity and transfer size of D[15:0] as shown in Table 3-1. The CL450's local DRAM (accessed when A[20] is 0) is the *only* CL450 resource which supports byte-wide accesses; at all other times $\overline{\text{UDS}}$ must equal $\overline{\text{LDS}}$.

Table 3-1 Meaning of $\overline{UDS}$ and $\overline{LDS}$

$\overline{UDS}$	$\overline{LDS}$	Valid	Size
High	High	D[15:0] not valid	none
Low	High	D[15:8] (DRAM only)	byte
High	Low	D[7:0] (DRAM only)	byte
Low	Low	D[15:0]	word

$\overline{AS}$ — Address Strobe — Input

The host processor asserts $\overline{AS}$ (active low) to select the CL450 for a non-DMA read or write operation. In a 68070-based system, $\overline{AS}$ is derived by the combination of the host signal $\overline{AS}$ and the address bits above A[20].

A[20:1] — Host Address Bus — Inputs

A[20:1] are the address lines that the host processor uses to address the CL450's CMEM, internal registers, and the local DRAM memory. A[20:19] selects which of the CL450 internal modules will be addressed as shown in Table 3-2.

Table 3-2 Internal Module Selected by A[20:19]

A20	A19	Internal Module
0	0	DRAM bank 0
0	1	DRAM bank 1 (optional)
1	0	Internal registers
1	1	CMEM (write only)

A[0] and the address bits above A[20] are not connected to the CL450. Instead of A[0], the host processor uses $\overline{UDS}$ and $\overline{LDS}$ to indicate which byte is valid on the data bus. (Note that $\overline{UDS}$ and $\overline{LDS}$ must both be asserted for host access to the CL450 when A[20] = 1.) Address bits above A[20] can be used to generate $\overline{AS}$ to indicate that the CL450 is selected.

D[15:0] — Host Data Bus — Bidirectionals

D[15:0] comprises the 16-bit bidirectional host data bus. The host processor uses D[15:0] to write data to the CL450's CMEM, internal registers, and local DRAM; and to pass data to the CL450 for DMA writes. The CL450 uses D[15:0] to send requested data to the host processor.

R/$\overline{W}$ — Read/Write **Input**

The host processor asserts R/$\overline{W}$ (high) to initiate a read operation, and deasserts R/$\overline{W}$ (low) to initiate a write operation.

$\overline{DTACK}$ — Data Transfer Acknowledge

Open-Drain Output

The CL450 asserts $\overline{DTACK}$ (active low) when it is ready to receive or output data on D[15:0]. When the CL450 responds to a read request, it holds $\overline{DTACK}$ deasserted (high) until the requested data is ready. When the CL450 responds to a write request, it asserts $\overline{DTACK}$ when it has received and latched the write data.

$\overline{DTACK}$ is an open-drain signal that allows it to be wire-ORed with other components on the host bus. It requires a pull-up resistor of no less than 470 ohms.

3.1.2 DMA Signals

The CL450 operates as a DMA slave only. An external DMA controller performs the DMA transfer of bitstream data to CMEM according to the SCC68070 DMA protocol, which is a subset protocol of the 68000 family of DMA controllers.

Figure 3-3 shows how the CL450 DMA signals connect to an external DMA controller. DMA data transfers are discussed in full detail in Section 4.7.

Note: Be sure to consult the diagram shown in Section 4.6 before connecting the $\overline{DMAREQ}$ line.

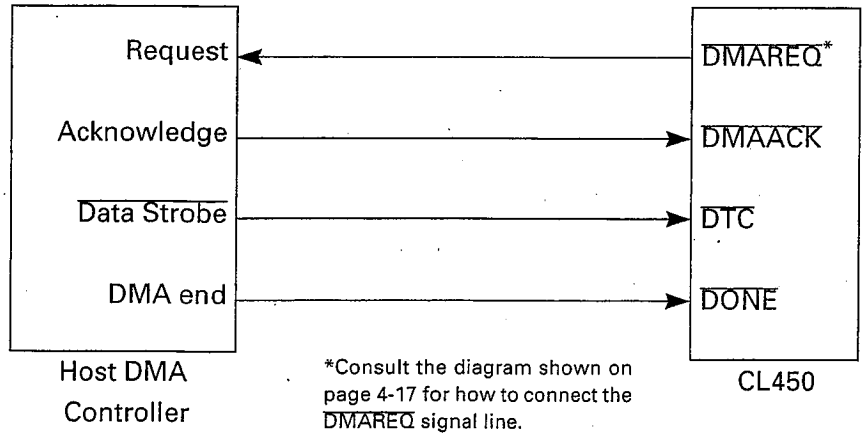


Figure 3-3 DMA Signals

$\overline{\text{DMAREQ}}$ — DMA Request **Output**

The CL450 asserts $\overline{\text{DMAREQ}}$ (active low) to request a DMA transfer. The CL450 asserts $\overline{\text{DMAREQ}}$ whenever CMEM has at least one free space, and the DMA enable flag (*DE*) in the CMEM_dmactrl register has been set correctly. (See the description of the *DE* bit in the CMEM_dmactrl register in Section 8.3.1, CMEM Registers, for more information about enabling DMA.)

$\overline{\text{DMAACK}}$ — DMA Acknowledge **Input**

The external DMA controller asserts $\overline{\text{DMAACK}}$ (active low) in response to a DMA request from the CL450 on $\overline{\text{DMAREQ}}$.

$\overline{\text{DTC}}$ — DMA Transfer Complete **Input**

The external DMA controller asserts $\overline{\text{DTC}}$ (active low) when a valid data word is present on D[15:0]. The CL450 latches the data word on the falling (active) edge of $\overline{\text{DTC}}$.

$\overline{\text{DONE}}$ — DMA Done **Input**

The external DMA controller asserts $\overline{\text{DONE}}$ (active low) to indicate that the DMA sequence is complete. When $\overline{\text{DONE}}$ is asserted and CMEM is not full, the CL450 resets *DE* in the CMEM_dmactrl register and deasserts $\overline{\text{DMAREQ}}$.

3.1.3 Interrupt Signals

The CL450 uses these signals to indicate an interrupt to the host. The CL450 can be programmed to implement an optional request-acknowledge handshake for vectored interrupts. Figure 3-4 shows how the CL450 interrupt signals connect to a host or interrupt processor. Section 4.8, Interrupt Cycle Timing, discusses interrupt processing in more detail.

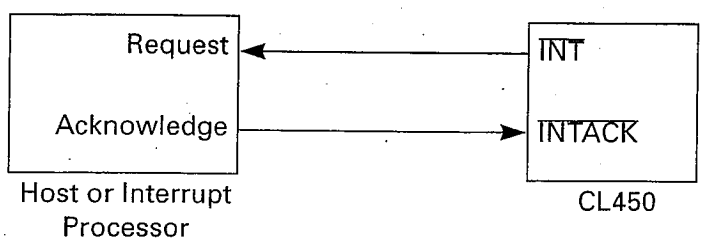


Figure 3-4 Interrupt Signals

$\overline{\text{INT}}$ — Interrupt Request **Open-Drain Output**

The CL450 asserts $\overline{\text{INT}}$ (active low) to request an interrupt from the host processor. $\overline{\text{INT}}$ is an open-drain signal. This allows it to be wire-ORed with other components on the host bus. It requires a pullup resistor of no less than 470 ohms.

$\overline{\text{INTACK}}$ — Interrupt Acknowledge **Input**

The host processor asserts $\overline{\text{INTACK}}$ (active low) in response to $\overline{\text{INT}}$ and begins the vectored interrupt sequence. The $\overline{\text{INTACK}}$ pin is ignored unless vectored interrupts are enabled (see the *VIE* bit description in the HOST_control register, page 8-12, and the HOST_intvecw register, page 8-13). $\overline{\text{INTACK}}$ is also used in conjunction with the TEST pin to 3-state the output pins for diagnostic purposes.

3.1.4 Timing, Control, and Status Signals

These signals provide the clock inputs, reset control, and CMEM status. Figure 3-5 shows these signals. The timing characteristics of these signals is described in Section 7.2.

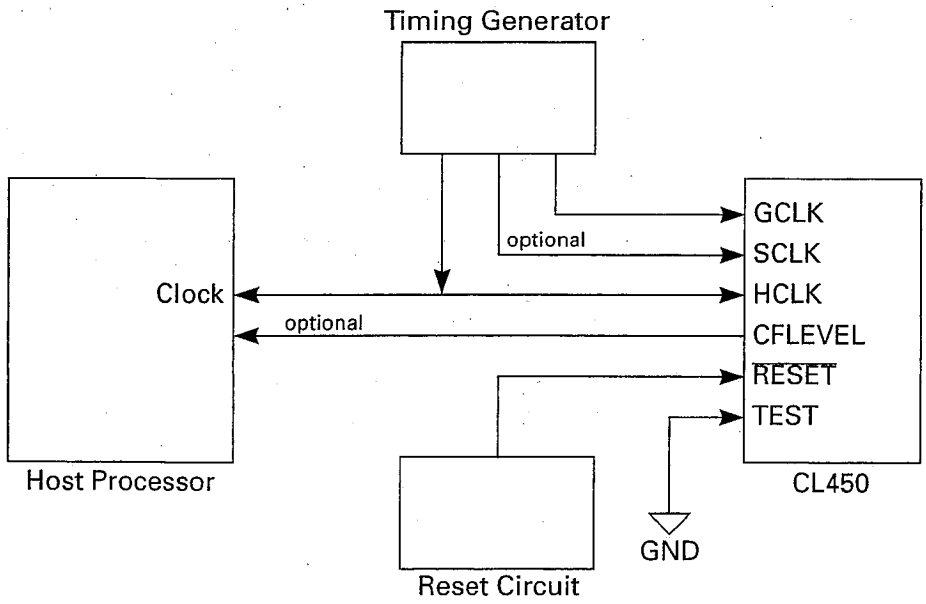


Figure 3-5 Timing, Control, and Status Signals

GCLK — Global Clock **Input**

The CL450 uses GCLK to clock the internal processor and the DRAM controller. For proper operation of the CL450, this signal must have a nominal frequency of 40 MHz. All local DRAM accesses are synchronized to this clock. HCLK and VCLK are limited to half the frequency of GCLK.

Note: The use of a precision 40.000 MHz crystal for GCLK is strongly recommended. Other statements in this manual regarding the performance and speed of the CL450 hardware and microapplication are based on a 40-MHz GCLK frequency.

SCLK — System Clock **Input**

SCLK is an optional external clock input for the CL450 system timer (SCR counter). This is typically a 90-kHz signal, although higher frequencies can be used with internal prescaler values greater than 1.

The system timer (Figure 3-6) is a 33-bit counter that is used by the host processor to synchronize video generated in the CL450 with other outside devices. These devices could include other CL450s, audio decoders, and other display devices within a multimedia system.

The CL450 system timer contains a 9-bit prescaler that can be used to divide the selected source clock prior to incrementing the system timer. The source clock can be either the SCLK signal or (typically) the GCLK signal.

In a typical system application using a 40-MHz GCLK, the divider should be set to divide GCLK by 444. If an external 90-kHz signal is used for the SCLK frequency, the divider can be set to divide by one. When the GCLK signal is used as a clock source, the SCLK pin must be pulled either *HIGH* or *LOW*.

Note: The divider is hardware-programmable and may therefore be written to a value other than 444 if the microapplication is not running; otherwise, the microapplication fixes the divider at a value of 444.

The frequency of SCLK must be less than half the frequency of GCLK, and both the SCLK-*LOW* and SCL*K-HIGH* periods must be greater than the GCLK clock period.

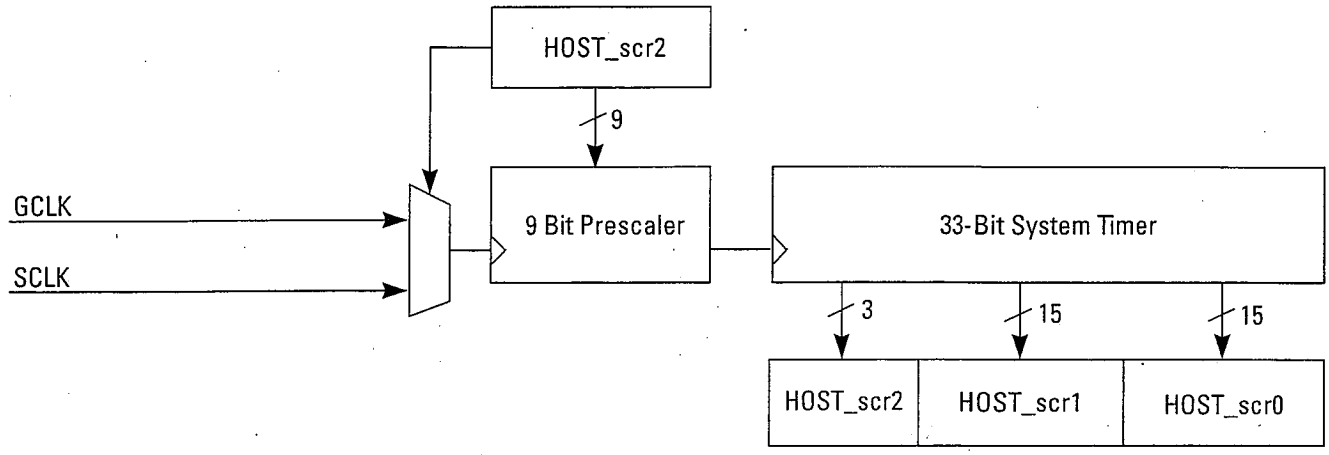


Figure 3-6 System Timer Block Diagram

HCLK— Host Clock **Input**

The clock signal HCLK (host clock) is the host bus interface reference signal. It is asynchronous with respect to both GCLK and VCLK. All host bus transfers to and from the CL450 are internally synchronized to GCLK. The CL450 supports a host processor with an HCLK of up to 20.0 MHz.

CFLEVEL — CMEM Level Status **Output**

The CL450 asserts CFLEVEL (active high) when CMEM becomes as full or more full than the threshold selected by the *1QE* to *4QE* bits in the CMEM_dmactrl register (see Chapter 8).

$\overline{\text{RESET}}$ — Hardware Reset **Input**

An external device asserts $\overline{\text{RESET}}$ (active low) to force the CL450 to execute a hardware reset. To be fully recognized, $\overline{\text{RESET}}$ must be asserted for at least 50 GCLK cycles. After a reset, the CL450 registers are in an indeterminate state and a complete re-initialization of the CL450 should be performed.

TEST — Test **Input**

When TEST (active high) and $\overline{\text{INTACK}}$ (active low) are asserted simultaneously, the CL450 three-states all outputs for performance of system diagnostics. For normal operation, TEST must be held LOW (deasserted).

3.2 DRAM Interface

The CL450's internal DRAM controller generates the addressing and control signals to control up to one Mbyte of local DRAM. A minimum DRAM size of 512 Kbytes is required for MPEG decoding. This DRAM is typically configured as one or two banks of 256K by 16 bits.

Figure 3-7 shows the signals that comprise the CL450's DRAM interface. The signal descriptions are presented following the figure. See Section 5.2, Memory Bus Interface, for more information about DRAM memory architecture.

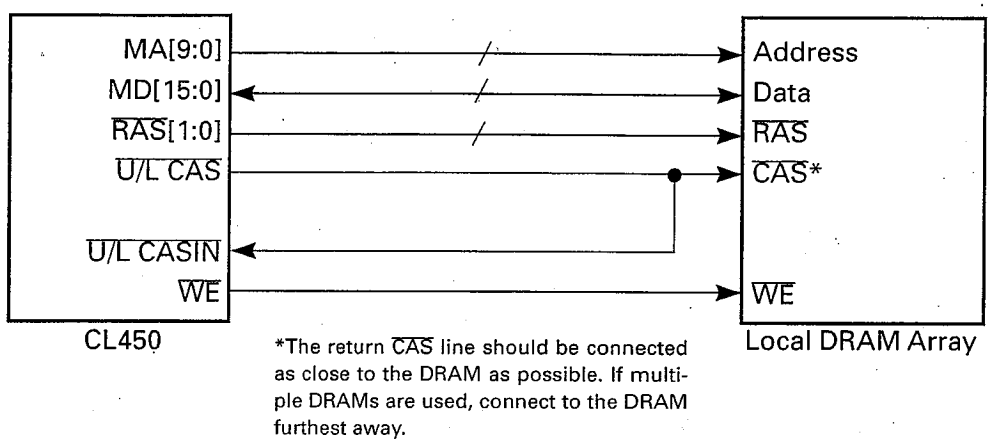


Figure 3-7 DRAM Interface Signals

MA[9:0] — Memory Address Bus **Outputs**

The CL450 multiplexes the row and column addresses on these signals to address up to one Mbyte of DRAM. See Section 5.2, Memory Bus Interface, for more information about how the address signals are used with different DRAM components and DRAM array sizes.

MD[15:0] — Memory Data Bus **Bidirectionals**

These signals comprise the memory data bus by which data is transferred between the CL450 and the local DRAM array. The direction of the data transfer is determined by the state of $\overline{WE}$.

$\overline{\text{RAS}}$[1:0] — Row Address Strobe **Outputs**

The CL450 asserts these signals to latch the row address into the local DRAM array. $\overline{\text{RAS}}$[1] (active low) latches the row address for bank 1, and $\overline{\text{RAS}}$[0] (active low) latches the row address for bank 0.

$\overline{\text{UCAS}}$ — Upper Column Address Strobe **Output**
$\overline{\text{LCAS}}$ — Lower Column Address Strobe **Output**

The CL450 asserts these signals to latch the column address into the local DRAM array. $\overline{\text{UCAS}}$ (active low) latches the column address for the upper memory data byte, MD[15:8], and $\overline{\text{LCAS}}$ (active low) latches the address for the lower byte, MD[7:0]. For local DRAM accesses performed by the CL450 for the host, $\overline{\text{UCAS}}$ is generated in response to the $\overline{\text{UDS}}$ input, and $\overline{\text{LCAS}}$ is generated in response to the $\overline{\text{LDS}}$ input.

$\overline{\text{UCASIN}}$ — Upper Data Latch Enable **Inputs**
$\overline{\text{LCASIN}}$ — Lower Data Latch Enable **Inputs**

When the CL450 reads data from the local DRAM array, the data on MD[15:0] is latched into the CL450 on the rising edge of the two $\overline{\text{CASIN}}$ signals. $\overline{\text{UCASIN}}$ latches data coming from the high data byte, MD[15:8], and $\overline{\text{LCASIN}}$ latches data coming from the low data byte, MD[7:0]. Typically, these are connected to the $\overline{\text{CAS}}$ pin(s) of the local DRAM array.

$\overline{\text{WE}}$ — Write Enable **Output**

The CL450 asserts $\overline{\text{WE}}$ (active low) to request a write operation (data transfer from CL450 to DRAM). The CL450 deasserts $\overline{\text{WE}}$ to request a read operation (DRAM to CL450).

3.3 Video Interface

The CL450's video interface outputs pixel data to the video display subsystem in RGB or YCbCr format. Figure 3-8 shows the signals in the CL450's video interface. Operation of the video interface is discussed in Chapter 6.

Note: CL450 video output is compatible with MPEG, CCIR 601, and CD-I players and therefore uses a digital output range of 16-235 as opposed to 0-255. (See Section 6.1, Digital Video Standards, for more information.)

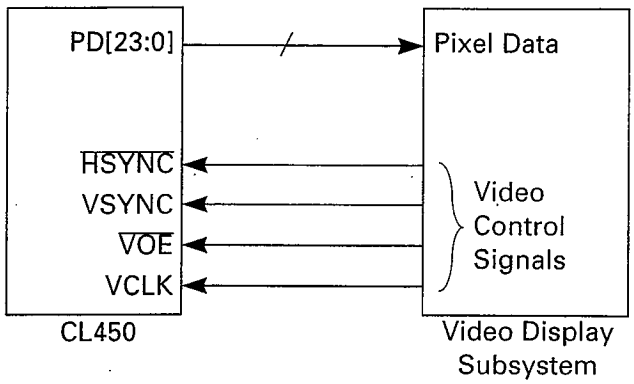


Figure 3-8 Video Interface Signals

PD[23:0] — Pixel Data Bus **Outputs**

The CL450 transmits pixel data to the video display subsystem using these signals. The definition of the signal lines differs for RGB and YCbCr formats as shown in Figure 3-9 and Figure 3-10. The format used is determined by the microapplication and can be changed using the SetColorMode() macro command described in Chapter 11.

When RGB format is used, each 24-bit word on PD[23:0] contains eight bits each of blue, green, and red. The highest bit of each component is most significant; for example, BLUE[7] corresponds to PD[23].

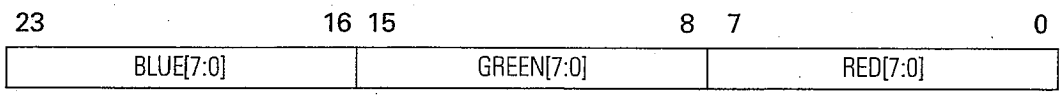


Figure 3-9 Pixel Bus Definition, RGB Format

When YCbCr format is selected, each 24-bit word on PD[23:0] contains eight bits of auxiliary register data, eight bits of luminance (Y), and eight bits of one of the chrominance components (Cb or Cr). The Cb and Cr coefficients alternate on successive VCLK periods.

The AUX[7:0] signals have no predefined purpose and can be used for whatever the designer wishes. For example, they could be used to load the color map in a RAMDAC. The bits are controlled by writing the desired data to the VID_selaux register (see Section 8.6.2).

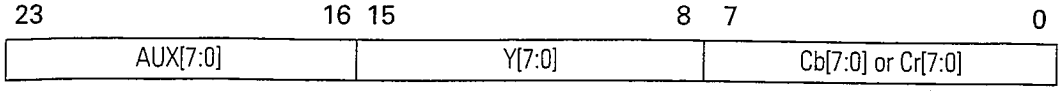


Figure 3-10 Pixel Bus Definition, YCbCr Format

$\overline{\text{HSYNC}}$ — Horizontal Synchronization **Input**

The CL450 begins counting the left border for a new horizontal line on the second VCLK after the rising (inactive) edge of $\overline{\text{HSYNC}}$ (active low). $\overline{\text{HSYNC}}$ must be synchronous to VCLK as shown in Section 6.3, Video Synchronization.

VSYNC — Vertical Synchronization **Input**

The CL450 begins outputting the top border of a new field on the first $\overline{\text{HSYNC}}$ after the rising edge of VSYNC. VSYNC is asynchronous with respect to VCLK, and is active high. See Section 6.3, Video Synchronization, for more information on the relationship of VSYNC to $\overline{\text{HSYNC}}$.

$\overline{\text{VOE}}$ — Video Output Enable **Input**

$\overline{\text{VOE}}$ must be asserted (active low) to enable the CL450 to drive the pixel bus, PD[23:0]. When $\overline{\text{VOE}}$ is deasserted, the CL450 holds the pixel bus in a high-impedance state.

VCLK — Video Clock **Input**

The CL450 outputs one pixel on PD[23:0] for each cycle of VCLK. VCLK cannot run faster than one-half the frequency of GCLK, although at frequencies below 15 MHz it does not need to be synchronous with GCLK.

3.4 Miscellaneous

Reserved — Future Expansion **Input**

C-Cube has reserved these pins for possible future use. They should be either pulled HIGH or LOW but should not be allowed to float.

4
Host Interface

The host interface on the CL450 is designed to interface to a variety of general-purpose microprocessors. While it is optimized for connection to members of the 680X0 microprocessor family, it also connects easily to the 80X86 family and other host processors.

The host processor can directly access any CL450 register or local DRAM location by reading and writing to specific addresses. Compressed video data transfers are performed by writing the data into the CMEM on-chip memory using either processor writes or DMA transfers. The CL450 host bus interface supports both vectored and polled interrupts.

This chapter is organized into the following sections which describe how the host interface is used:

- 4.1: Overview
- 4.2: Local DRAM or Register Read
- 4.3: Local DRAM or Register Write

- 4.4: CMEM Write Timing
- 4.5: CMEM DMA Write Timing
- 4.6: CMEM DMA Write Application Guidelines
- 4.7: CMEM Level
- 4.8: Interrupt Cycle Timing

Figure 4-1 shows the pinout diagram of the CL450 host bus interface. The host interface signals include a 20-bit address bus, a 16-bit data bus, and the necessary control signals for performing data transfers and interrupt handling.

Note: Be sure to consult the diagram shown in Section 4.6 before connecting the DMAREQ and CFLEVEL lines.

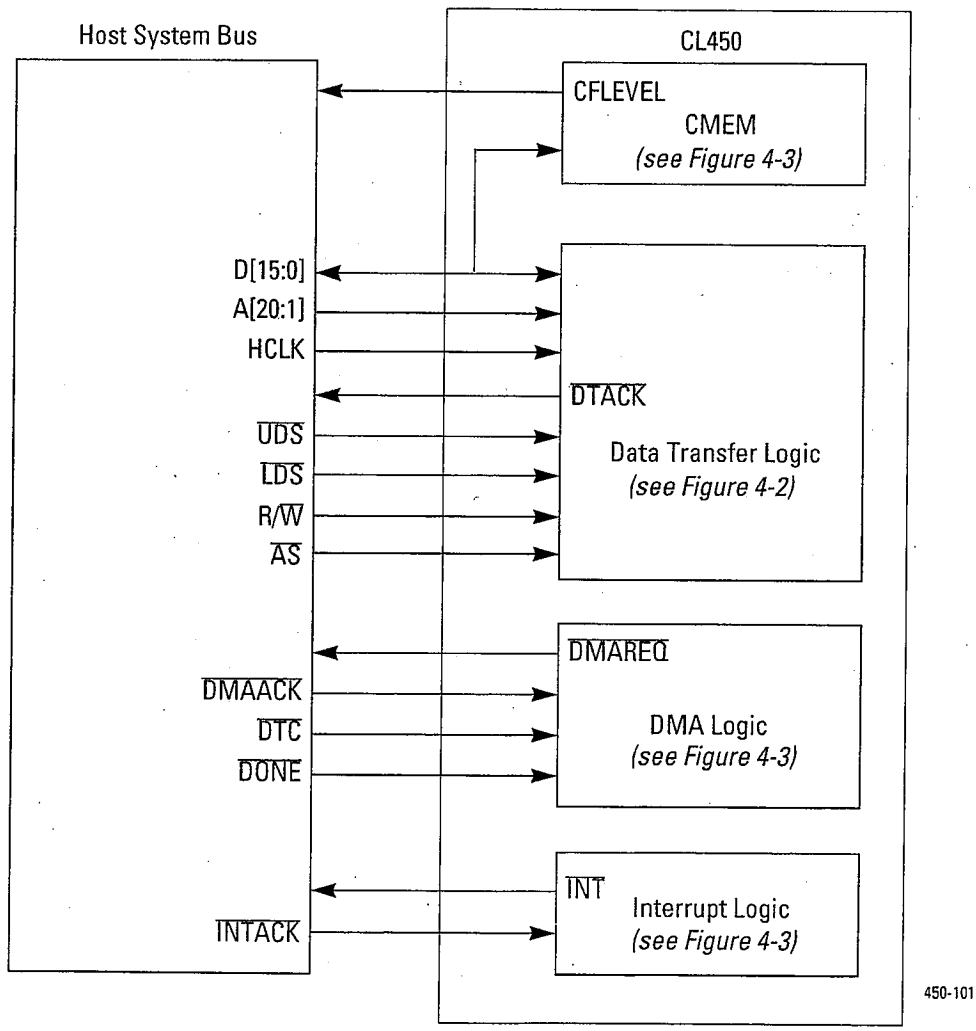


Figure 4-1 Host Interface Block Diagram

4.1 Overview

The host processor uses memory reads and writes to address the CL450's local DRAM, the internal registers, and CMEM. The host processor indicates that a memory access is being performed by putting the address for the desired CL450 location on the address bus and asserting the $\overline{AS}$ (Address Strobe) pin. Then the CL450's internal address decoder determines which resource to access by decoding address lines A[20:19] and the R/$\overline{W}$ line as shown in Table 4-1.

Table 4-1 CL450 Memory Access Address Bits

A[20]	A[19]	R/$\overline{W}$	Byte Addressable	Description
0	0	X	Yes	DRAM Bank 0 Access (Read/Write)
0	1	X	Yes	DRAM Bank 1 Access (Read/Write)
1	0	X	No	Register Access (Read/Write)
1	1	0	No	CMEM Access (Write Only)

The $\overline{AS}$ (Address Strobe) signal on the CL450 performs like a chip select. In a typical system application, $\overline{AS}$ is a function of the address lines greater than A[20] and a signal indicating that a valid address has been placed on the address bus. ($\overline{AS}$ is the signal used in the 68070 systems.)

Byte-wide memory accesses are performed by asserting either the $\overline{LDS}$ (Lower Data Strobe) or $\overline{UDS}$ (Upper Data Strobe) pin when $\overline{AS}$ is asserted as shown in Table 4-2. Word-wide (16-bit) accesses are performed by asserting both $\overline{LDS}$ and $\overline{UDS}$ simultaneously. Whenever the CL450 is the target of a byte-wide read (R/$\overline{W}$=1), only the drivers for the byte being read are driven. The data read on the byte not selected is indeterminate. The local DRAM is the only CL450 resource which can be addressed using byte-wide accesses.

Table 4-2 $\overline{LDS}$ and $\overline{UDS}$ Decoded Values

$\overline{LDS}$	$\overline{UDS}$	Definition
H	H	Data Not Valid
H	L	DRAM only: D[15:8] Valid (Upper byte, A[0] = 0)
L	H	DRAM only: D[7:0] Valid (Lower Byte, A[0] = 1)
L	L	D[15:0] Valid (Word access, A[0] = 0)

Byte accesses are provided to make the CL450 DRAM available as system memory for the host when the CL450 is not being used. The $\overline{LDS}$ and $\overline{UDS}$ inputs to the CL450 may be tied together if this feature is not required.

Note: In systems based on the 68000, 68010, and 68070 processors, the $\overline{LDS}$ and $\overline{UDS}$ signals will be asserted in roughly the same cycle as $\overline{AS}$; but when a write is being performed, $\overline{LDS}$ and $\overline{UDS}$ will be delayed one or two cycles from the assertion of $\overline{AS}$ to allow setup time for the data.

4.1.1 Register Access

When address lines A[20:19] = 10_2, the CL450 addresses its internal registers. The lower seven address lines, A[7:1], determine which register is being addressed, and the R/$\overline{W}$ signal determines whether the operation is a read or a write. (The upper eleven address lines, A[18:8], are in a "don't care" state at this time; the only time they matter is during a DRAM read or write, at which time they form part of the local DRAM address.) Internal register operations should always be full-word accesses as indicated by the assertion of both $\overline{LDS}$ (Lower Data Strobe) and $\overline{UDS}$ (Upper Data Strobe). Two UDSsynchronous state machines inside the host interface—one using HCLK, the other using GCLK—control the data transfer as shown in Figure 4-2. Inputs to the two state machines are registered to synchronize them to the local clock.

4.1.2 Local DRAM Access

When address line A[20] is LOW, the CL450 addresses the local DRAM array. The DRAM array is divided into two banks typically organized as 256K addresses by 16 bits. Address line A[19] selects whether Bank 0 or Bank 1 is being addressed, and address lines A[18:1] select the location to be accessed. The local DRAM array can be accessed using either byte-wide or word-wide reads and writes. The same two synchronous state machines inside the host interface that are used for register read/writes are also used for local DRAM read/writes. The DRAM controller drives GSEL (see Figure 4-2) to complete the read or write operation.

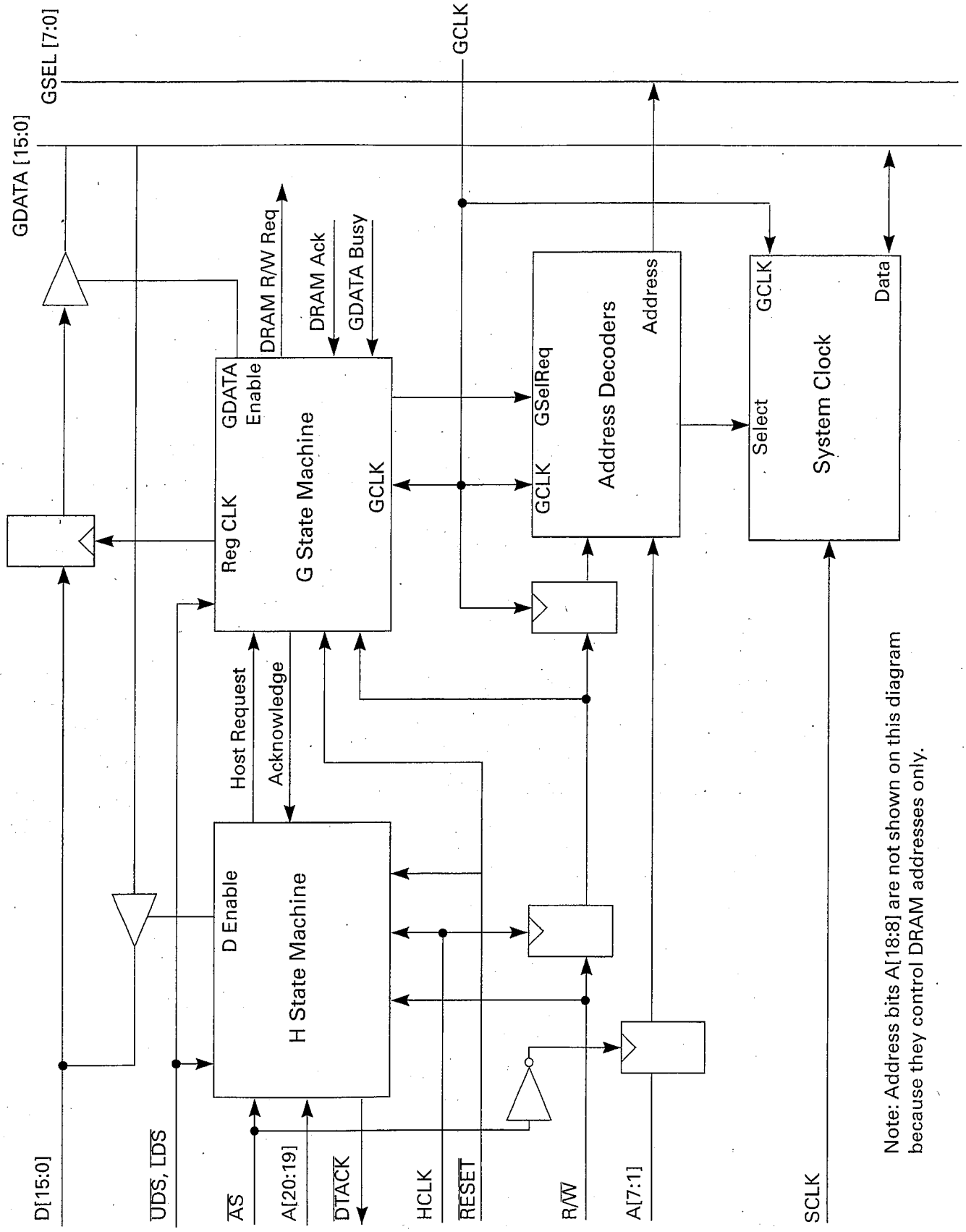


Figure 4-2 Host Interface Diagram (detailed)

4.1.3 CMEM Access

When address lines A[20:19] = 11_2, the CL450 addresses CMEM for writes; A[20:19] must be 11_2 before $\overline{AS}$ goes LOW. Writing a word to any address within the address range puts data into CMEM as shown in Figure 4-3.

For CMEM writes, the input data is registered on the falling edge of the data strobes ($\overline{LDS}/\overline{UDS}$), and an internal CMEM write request is generated on the rising edge of the data strobes. The data is written into the CMEM on the first rising edge of GCLK following the activation of the write request. The CMEM address counters in CMEM_status are incremented at the end of the write.

> *Note: CMEM writes are asynchronous with respect to both HCLK and GCLK; the state machines used for CL450 register read/write and direct DRAM read/write are not used in either DMA or CMEM operations. Also, CMEM is a true dual-port memory; therefore, internal read and host write operations can occur in the same GCLK cycle. HCLK has no effect on CMEM writes.*

4.1.4 DMA Operation

DMA writes are very similar to CMEM writes except that the $\overline{DMAACK}$ and $\overline{DTC}$ pins control the operation as shown in Figure 4-3. For DMA operation to work correctly, $\overline{AS}$ and R/$\overline{W}$ must both = 1. (HCLK, D[15:0] and A[20:1] do not affect DMA operation.) The input data is registered on the falling edge of $\overline{DTC}$, and an internal CMEM write request is generated on the rising edge of $\overline{DTC}$. The data is written into CMEM on the first rising edge of GCLK following the activation of the write request. The CMEM address counters in CMEM_status are incremented at the end of the write.

4.1.5 Interrupt Vector Operation

For interrupt vector operation to occur, HOST_control bit 14 (Vectored Interrupt Enable, *VIE*) must be 1. For a vectored interrupt operation, host address A[3:1] must be set up before $\overline{INTACK}$ goes low and hold until $\overline{INTACK}$ goes high. In addition, $\overline{INT}$ from the CL450 must be low. (Asserting $\overline{INTACK}$ while $\overline{INT}$ is inactive will cause indeterminate behavior.) All of these operations are asynchronous with respect to GCLK and, like the other access modes described in Sections 4.1.3 and 4.1.4, do not affect the GDATA bus shown in Figure 4-3.

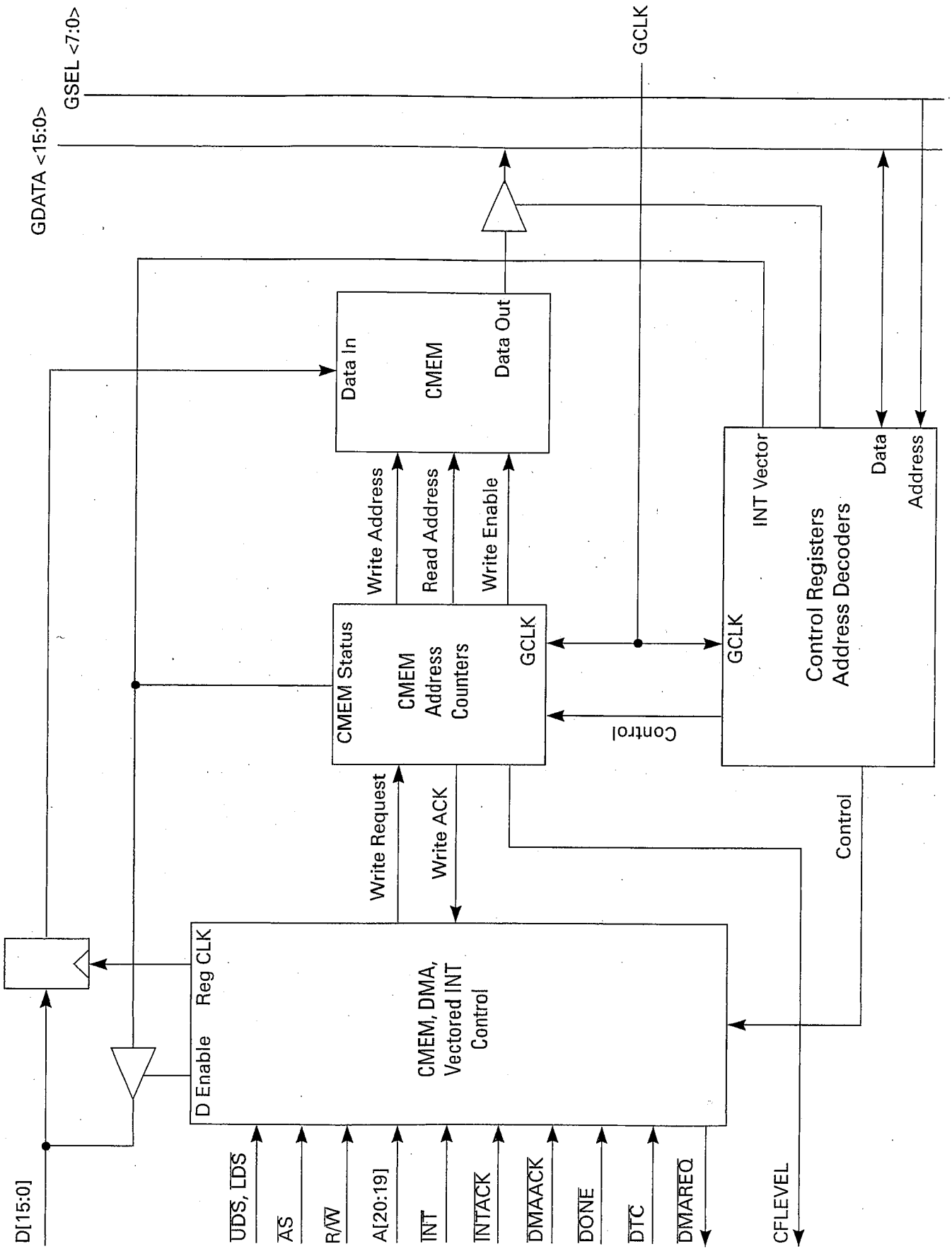


Figure 4-3 CMEM Block Diagram

4.2 Local DRAM or Register Read

The host processor accesses the internal registers of the CL450 and the CL450 local DRAM array by reading them as if they were memory locations. Because the resource being read might not be immediately available, the CL450 can insert wait states by delaying assertion of the $\overline{\text{DTACK}}$ signal until the read can be completed.

Register and DRAM reads have a minimum delay of two GCLK cycles. Maximum DRAM latency is dependent on the operation of the microapplication.

Figure 4-4 shows the sequence of signals for a local DRAM or internal register read. The circled numbers in the figure refer to the steps below.

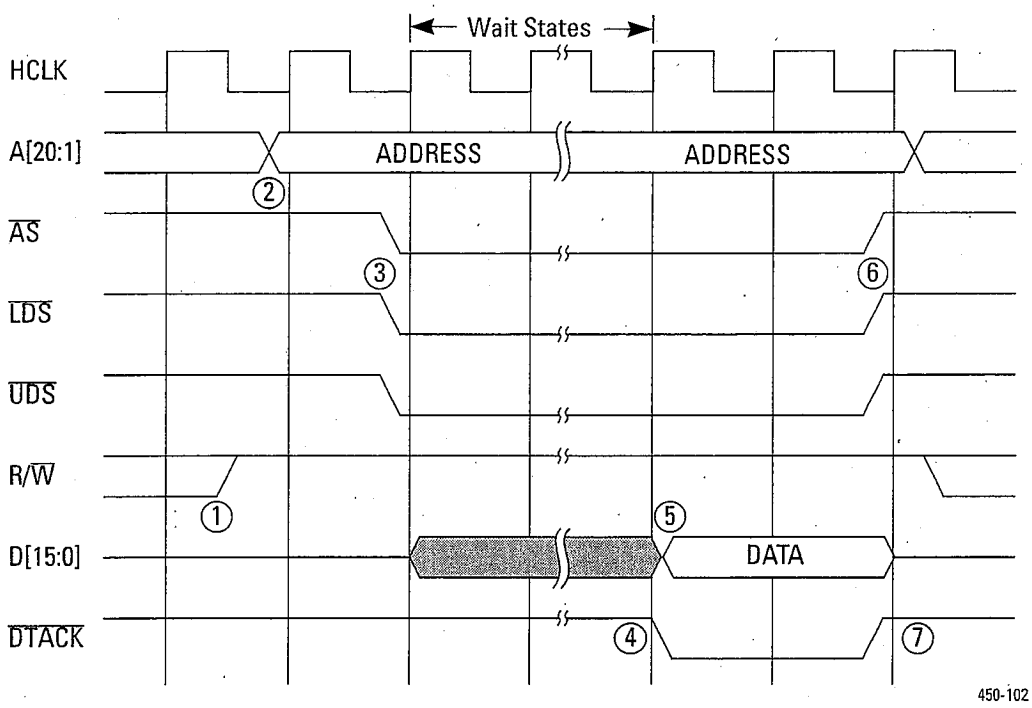


Figure 4-4 Local DRAM or Register Read Timing

1. The host processor sets the R/$\overline{\text{W}}$ line to the proper state for the read operation (*HIGH*).
2. The host processor outputs the desired address on address lines A[20:1].
3. After the R/$\overline{\text{W}}$ line and the address lines have settled, the host

processor asserts the control lines $\overline{AS}$, $\overline{LDS}$, and $\overline{UDS}$ to indicate to the CL450 that a host bus read access is in progress. The data bus (D[15:0]) should be in a high-impedance state at this time.

4. The CL450 holds $\overline{DTACK}$ deasserted (high) until it can respond to the read request. Because the CL450 is performing arbitration of its internal data buses during memory or register accesses, $\overline{DTACK}$ assertion is delayed by at least one HCLK cycle. When the data is available, the CL450 asserts $\overline{DTACK}$.
5. The requested data becomes available on the data bus at the same time that $\overline{DTACK}$ is asserted.
6. Once the host processor has read the data, it deasserts $\overline{AS}$, $\overline{LDS}$, and $\overline{UDS}$. This completes the bus cycle.
7. When the CL450 detects the deassertion of $\overline{AS}$, $\overline{LDS}$, and $\overline{UDS}$, it releases the $\overline{DTACK}$ signal and the data bus.

4.3 Local DRAM or Register Write

Figure 4-5 shows the sequence of signals for a local DRAM or internal register write. The circled numbers in the figure refer to the steps below.

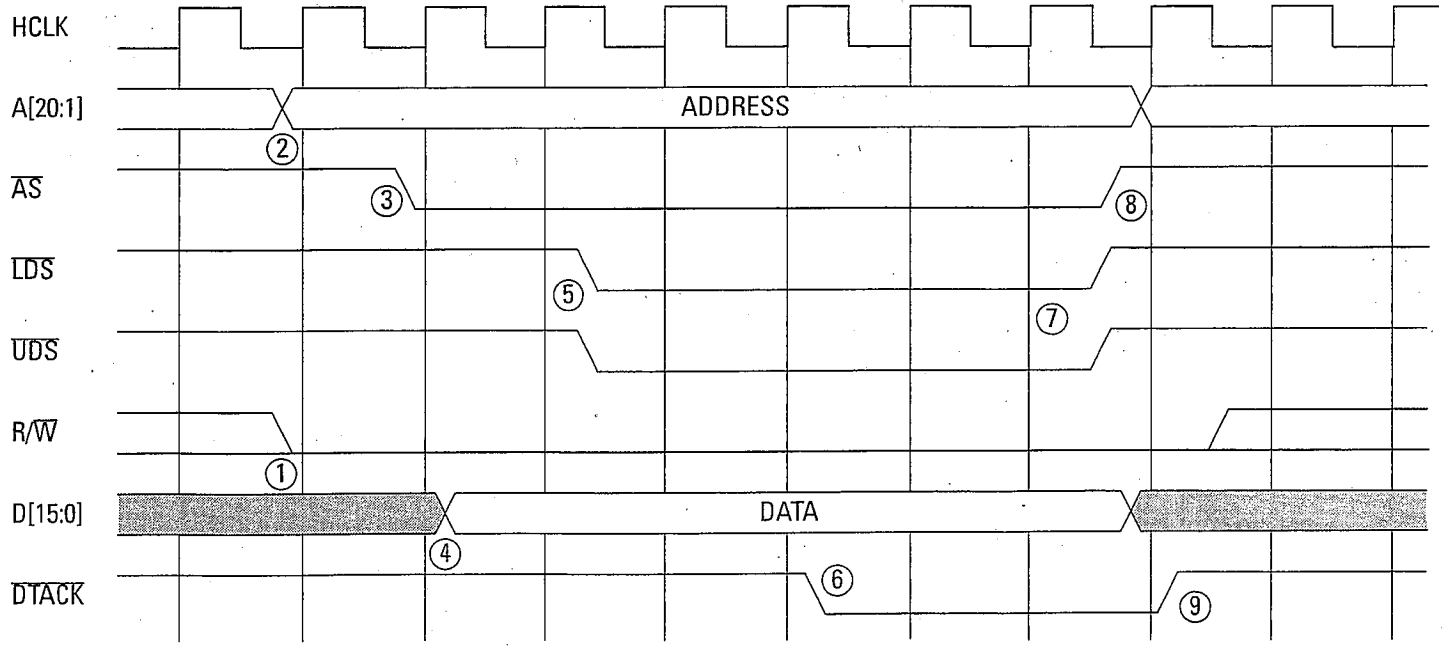


Figure 4-5 Local DRAM or Register Write Timing with Delayed $\overline{UDS}$ and $\overline{LDS}$ Timing Shown

1. The host processor sets the R/$\overline{W}$ line to the proper state for a write operation (*LOW*).
2. The host processor outputs the desired address on address lines A[20:1].
3. After the address lines have settled, the host processor asserts $\overline{AS}$. This indicates to the CL450 that a memory transfer is in progress.
4. The host processor then puts the data to be written to the CL450 on the data bus.
5. $\overline{UDS}$ and $\overline{LDS}$ are asserted. (The host processor asserts $\overline{LDS}$ or $\overline{UDS}$ if a byte-wide write is taking place, or both if it is a word-wide write.)
6. The CL450 generates a $\overline{DTACK}$ signal after $\overline{UDS}$ and $\overline{LDS}$ are asserted.
7. The CL450 latches the data within 2 GCLK cycles after $\overline{DTACK}$ goes low.
8. The host processor releases $\overline{AS}$.
9. The CL450 responds by releasing $\overline{DTACK}$. This completes the write cycle.

Note: For register reads, the $\overline{DTACK}$ output of the CL450 provides a handshaking response to indicate to the host CPU that the CL450 is ready to terminate the current read or write operation.

For local DRAM reads, the local DRAM is selected by host A[20], and host bits A[19:1] are the DRAM word address. However, $\overline{DTACK}$ will not be asserted as quickly because the DRAM controller must acknowledge host interface request signals.

The state diagrams for $\overline{DTACK}$ generation logic are shown in Figure 4-6. Note that the states are drawn as event-driven and are not keyed to a particular clock. In the case of both reads and writes, if $\overline{AS}$ is deasserted before $\overline{DTACK}$ is asserted, $\overline{DTACK}$ will not be asserted. On a read operation, $\overline{DTACK}$ can be delayed due to internal DRAM arbitration.

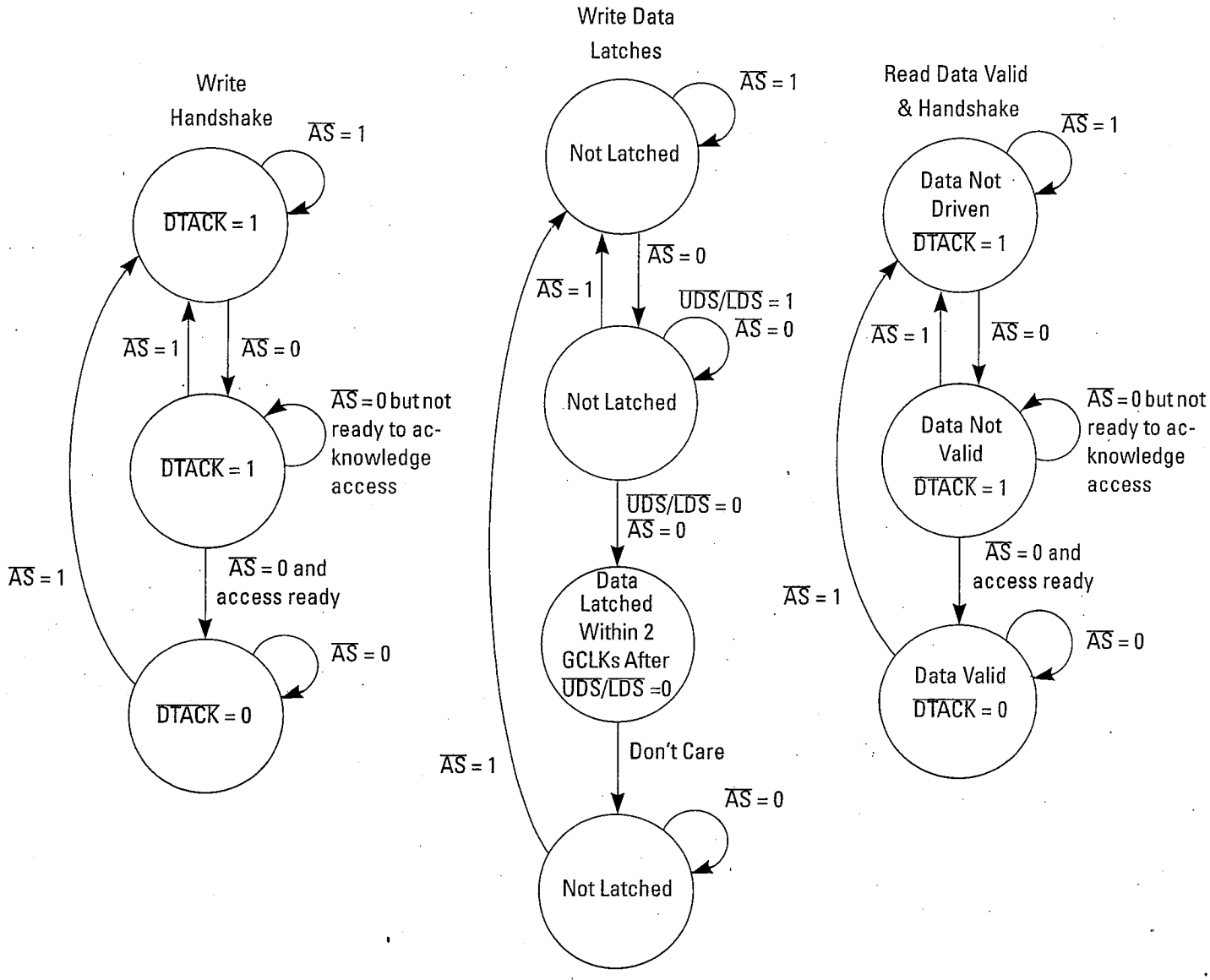


Figure 4-6 DTACK State Logic Diagram

4.4 CMEM Write Timing

While the CL450 is decompressing video, the bitstream buffer in the CL450's local DRAM must be supplied with a constant stream of MPEG video data. To speed up the transfer of compressed video data, CMEM (a coded data FIFO) has been included in the input data path. Write operations to CMEM can be performed with no added wait-states. The CL450 also supports no-wait-state Direct Memory Access (DMA) transfers of compressed video data into CMEM. The CL450 automatically transfers the compressed data from CMEM into the local DRAM

bitstream buffer whenever space is available. Writes to CMEM must always be 16-bits wide. Byte writes are *not* supported.

The host processor can control the transfer of data to CMEM using one of the following two methods:

- *Polled:* The host CPU monitors either the CFLEVEL signal or the empty status bits in the CMEM_dmactrl register; it writes data when CMEM has space.
- *DMA:* The CL450 asserts the DMA Request signal ($\overline{\text{DMAREQ}}$) whenever there is space in CMEM. The host DMA controller responds by acknowledging the request and transferring a word of data to the CL450. This method is desirable because the host does not need to be involved in the transfer of each data word or polling CMEM's fullness, leaving it free to perform other tasks.

Direct CMEM write and DMA CMEM write operations are done by a GCLK-synchronous state machine in the host interface (see Figure 4-3), and the inherent speed of this interface is much faster than the maximum average transfer speed of 5.0 Mbits per second. The data is registered on the falling edge of $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$, and three GCLK cycles after the rising edge of $\overline{\text{AS}}$ the data is written into CMEM. $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ must not go LOW again during these 3 GCLK periods, and R/$\overline{\text{W}}$ must be LOW when $\overline{\text{UDS}}$/$\overline{\text{LDS}}$ go LOW with a set-up time of T9 (Table 7-4).

For back-to-back transactions, $\overline{\text{AS}}$ HIGH from one cycle must precede $\overline{\text{UDS}}$/$\overline{\text{LDS}}$ LOW in the following cycle by the sum of T129 and T130 (see Table 7-5). For back-to-back CMEM writes, it would be best to leave R/$\overline{\text{W}}$ LOW. (The timing diagrams that follow are for a 68070 in which the normal state of R/$\overline{\text{W}}$ is high.) $\overline{\text{AS}}$ must stay HIGH for at least two GCLK cycles, and $\overline{\text{AS}}$ LOW must set up a short time before $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ go LOW (T129).

In direct DMA mode, $\overline{\text{DMAACK}}$ has the same function as $\overline{\text{AS}}$ in direct CMEM writes, and $\overline{\text{DTC}}$ has the same function as $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ in direct CMEM writes. This means that $\overline{\text{DMAACK}}$ LOW time can be very short and that $\overline{\text{DMAACK}}$ HIGH time must be two GCLK periods.

Figure 4-7 shows the timing diagram for CMEM writes, and Figure 4-8 shows the timing diagram for DMA transfers to CMEM. The circled numbers in each figure refer to the steps that follow each figure.

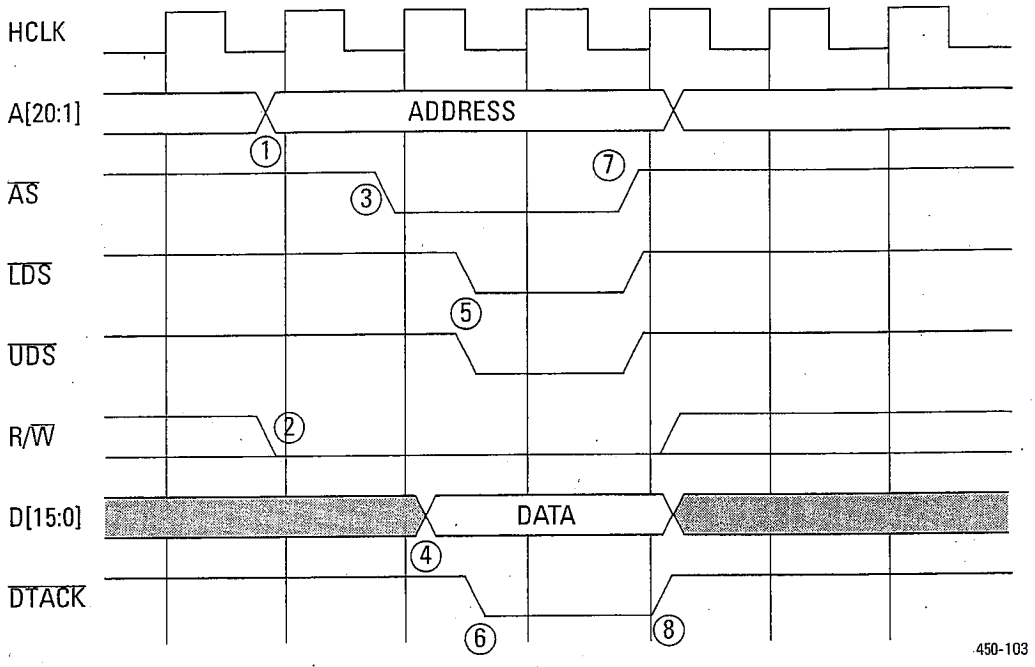


Figure 4-7 CMEM Write Timing

1. The host processor outputs the desired address on address lines A[20:1]. To access CMEM, A[20:19] must be 11_2 and A[18:1] may be any value.
2. The host processor sets the R/$\overline{W}$ line to the proper state for the write operation (LOW).
3. After the address lines have had time to settle, the host processor asserts $\overline{AS}$. This assertion indicates to the CL450 that a memory transfer is in progress.
4. The host processor then puts the data to be written to the CL450 on the data bus.
5. After allowing time for the data bus to become valid, the host processor asserts both $\overline{LDS}$ and $\overline{UDS}$. CMEM writes must always be 16-bits wide.
6. The CL450 asserts the $\overline{DTACK}$ signal after the assertion of $\overline{AS}$ (LOW). Because no arbitration is necessary, no wait states are necessary for CMEM writes.

7. When the host processor recognizes the $\overline{\text{DTACK}}$ signal, it releases $\overline{\text{AS}}$, $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$.
8. The CL450 responds by releasing $\overline{\text{DTACK}}$. This action completes the write cycle.

4.5 CMEM DMA Write Timing

The CL450 also supports DMA transfers of data into CMEM using the $\overline{\text{DMAACK}}$, $\overline{\text{DTC}}$ and $\overline{\text{DONE}}$ control inputs. DMA transfers minimize the burden that the CL450 puts on the host processor.

The host processor starts a DMA transaction by configuring the host DMA controller (to set up the DMA channel) and writing a 1 to the DMA enable bit (*DE*) in the CL450 CMEM_dmactrl register. The act of setting the *DE* bit while CMEM is *at* or *below* the threshold in CMEM_dmactrl will enable the CL450's internal DMA logic. Subsequently, when CMEM fullness is below the threshold, the CL450 asserts the $\overline{\text{DMAREQ}}$ (DMA Request) signal.

When the host detects $\overline{\text{DMAREQ}}$ active, it asserts the $\overline{\text{DMAACK}}$ (DMA Acknowledge) signal and begins a block write to CMEM. Each time that a word is transferred, the DMA controller checks to see if another $\overline{\text{DMAREQ}}$ pulse has been generated. If a $\overline{\text{DMAREQ}}$ pulse has been generated, the DMA controller generates another $\overline{\text{DMAACK}}$ pulse and writes a word of data.

When CMEM reaches threshold before all of the data has been transferred, the CL450 releases the $\overline{\text{DMAREQ}}$ signal until space becomes available again. This continues until the transfer of data is complete.

When the transfer is complete, the host may assert the $\overline{\text{DONE}}$ signal, which stops the transaction and clears the DMA enable bit (*DE*) in the CMEM_dmactrl register if the $\overline{\text{DONE}}$ pulse occurred when CMEM was not full. The host can abort the transaction at any point by clearing this bit.

> *Note: The CL450 supports a host processor with an HCLK frequency of up to 20.0 MHz, or one-half the GCLK frequency, whichever is less. DMA writes can provide two bytes every six host processor clock cycles.*

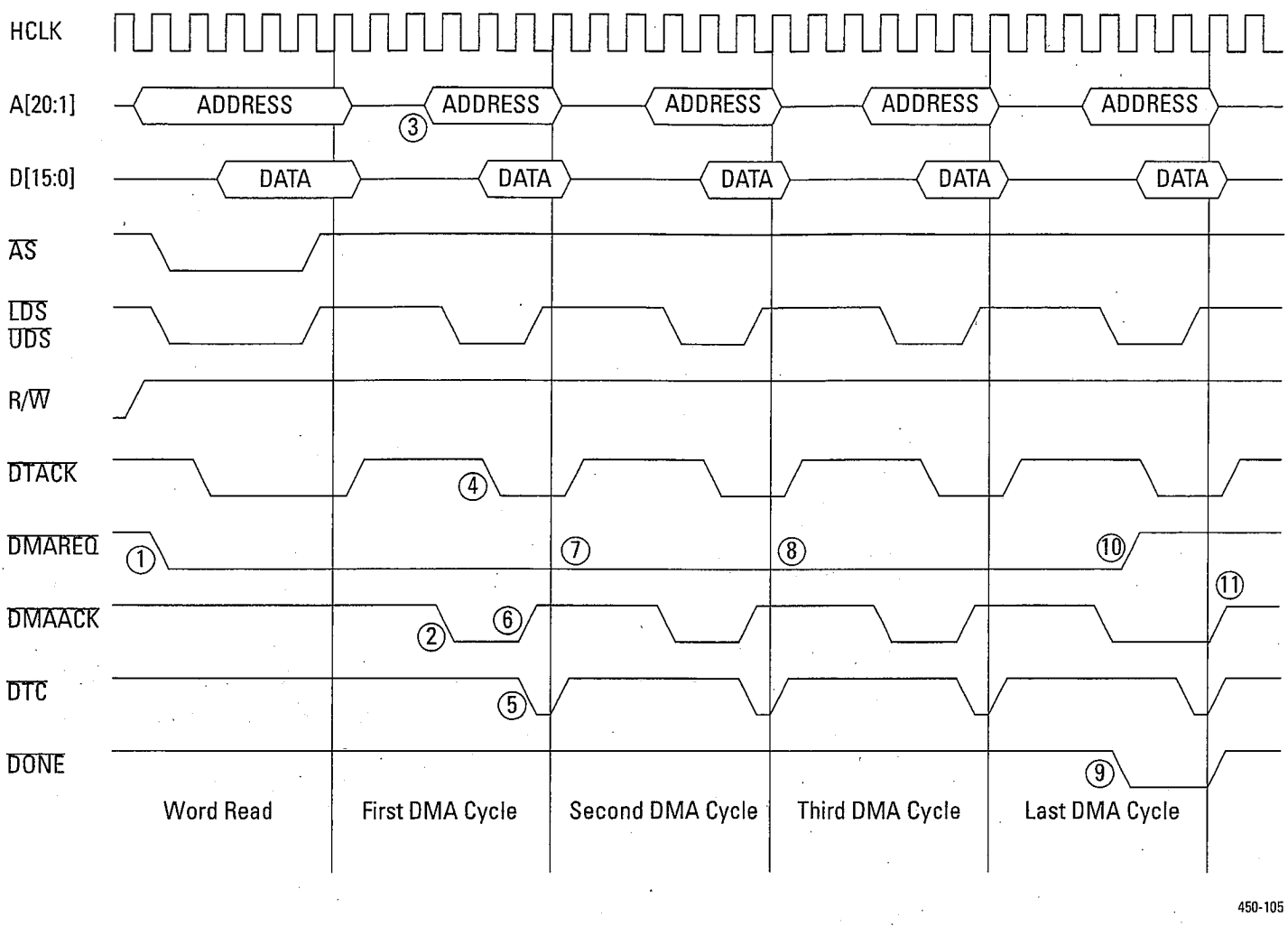


Figure 4-8 CMEM DMA Write Timing

Figure 4-8 shows the sequence of signals during a typical DMA transfer. During this transfer, four words of data are transferred into CMEM, and the host terminates the transfer by asserting $\overline{\text{DONE}}$. The circled numbers in the figure refer to the steps below.

1. The CL450 asserts $\overline{\text{DMAREQ}}$, indicating that space is available in CMEM. This request is asynchronous, and may occur during another CL450 access cycle. In this example, it occurs during a word read from the CL450.
2. After completing the current operation, the host DMA controller responds by asserting $\overline{\text{DMAACK}}$ and starting a word transfer.

3. The DMA controller drives the address bus and strobes the system $\overline{AS}$ (*not* the CL450's $\overline{AS}$ input), causing the host memory subsystem to drive the data onto the bus.
4. When the data is ready, the system memory responds by asserting $\overline{DTACK}$. Note that the host system is driving the address bus, R/$\overline{W}$, $\overline{AS}$, and $\overline{DTACK}$ during a DMA cycle, but that the CL450 is not using them. The CL450's $\overline{AS}$ and R/$\overline{W}$ input signals must remain *HIGH* during a DMA transfer. (In single-address mode, the CL450's $\overline{AS}$ and R/$\overline{W}$ signals will stay high without special logic due to the reads from system memory.)
5. The host DMA controller asserts Data Transaction Complete ($\overline{DTC}$) to indicate to the CL450 that the word is available. The CL450 stores the data on the falling edge of $\overline{DTC}$.
6. The host DMA controller completes the cycle by releasing $\overline{DMAACK}$. At this point, the first word has been transferred into CMEM.
7. Because the $\overline{DMAREQ}$ signal is still asserted, a second word of data is transferred into CMEM using timing identical to that of the first (repeat steps 2 through 6).
8. The transfer of the third word into CMEM is identical to that of the second.
9. During the transfer of the last word, the DMA controller in the host system asserts the $\overline{DONE}$ signal.
10. When the CL450 detects the active $\overline{DONE}$ signal (CMEM is assumed to not be full at this time), it clears the DMA enable bit (*DE*) in the CMEM_dmactrl register and releases the $\overline{DMAREQ}$ signal.
11. The DMA controller releases $\overline{DMAACK}$ and $\overline{DTC}$. This completes the DMA transfer.

4.6 CMEM DMA Write Application Guidelines

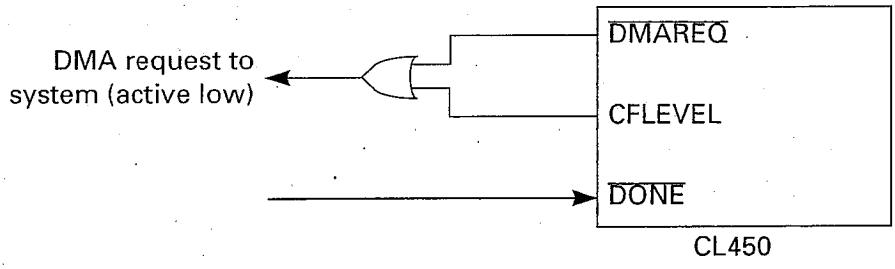
When designing a DMA controller to be used with the CL450, the following guidelines should be followed:

- Both R/$\overline{W}$ and $\overline{AS}$ must remain *HIGH* throughout a DMA transfer. If the 68070 dual-address DMA transfer mode is used, external logic must be provided to ensure that these conditions are met.

- The behavior of both the $\overline{\text{DMAREQ}}$ and $\overline{\text{DONE}}$ signals is conditional on the fullness of internal CMEM. In systems in which the $\overline{\text{DONE}}$ signal does not have to be used, the deterministic operation of $\overline{\text{DMAREQ}}$ can be assured by correct operation of the host software. Alternately, the host may program the CL450 to output the CFLEVEL signal so that it can be used to condition the $\overline{\text{DMAREQ}}$ signal before it is supplied to the host.

  For the CL450 to produce pulses on the $\overline{\text{DMAREQ}}$ signal, DMA operation must be enabled by the host writing the DMA Enable (*DE*) bit while CMEM is *not* full. This condition can be ensured by the host software polling the CMEM_dmactrl register prior to writing the DMA Enable bit in it.

  Alternately, if the $\overline{\text{DMAREQ}}$ signal is conditioned before being sent to the host, it can be ensured in hardware that CMEM never becomes full at all. To accomplish this, a circuit similar to the following should be used:



  For this circuit to operate correctly, CFLEVEL must be programmed to become active if the *1Q* bit (in CMEM_dmactrl) becomes 1. This action can be accomplished by writing a value of $10000000X_2$ to CMEM_dmactrl. Doing so reduces the effective size of CMEM from 16 to 12 words, so that the *actual* CMEM will never become completely full. Limiting the size of CMEM in this way ensures that writing a 1 to the DMA Enable bit *(DE)* in CMEM_dmactrl *always* activates the CL450's internal DMA logic, and that a pulse on the $\overline{\text{DONE}}$ input signal will *always* cause *DE* to be cleared.

- A DMA transfer allows a word of data to be written into CMEM every six HCLK cycles. Using direct memory writes, a word can be transferred every four HCLK cycles (8.75 Mbytes per second). If extremely high (instantaneous) data transfer rates are required,

external hardware could be designed to take advantage of this. An example would be a state machine that loads the data from an external FIFO directly into the CL450's CMEM.

- $\overline{AS}$ has a minimum pulse width of 40ns, and $\overline{UDS}/\overline{LDS}$ have a minimum pulse width of 5ns (not tested).
- $\overline{AS}$ can be de-asserted on the same cycle where $\overline{UDS}/\overline{LDS}$ get asserted if $\overline{DTACK}$ is asserted.
- During direct host writes to CMEM, data is registered on the falling edge of $\overline{UDS}/\overline{LDS}$.
- No timing restriction exists on how fast back-to-back write transactions can occur during CMEM data transfers; however, $\overline{UDS}$ and $\overline{LDS}$ should not go LOW until data has been written into CMEM on the third GCLK cycle after $\overline{AS}$ goes HIGH.
- $\overline{DTACK}$ is combinatorial during direct CMEM writes and vectored interrupts but synchronous otherwise.

4.7 CMEM Level

CFLEVEL (CMEM level) is an output signal that the CL450 generates to inform the host processor of the fullness of CMEM. The host programs control bits in the CMEM_dmactrl register to set the point at which the CFLEVEL signal is asserted by the CL450. Table 4-3 shows how these bits are programmed. This signal can either be polled by the host processor or used to generate an external interrupt.

Table 4-3 CMEM Level Control Bits

Bit 4	Bit 3	Bit 2	Bit 1	Assert CFLEVEL when FIFO is:
1	0	0	0	1/4 empty
0	1	0	0	1/2 empty
0	0	1	0	3/4 empty
0	0	0	1	Empty[1]

1. During normal operation, CMEM is not necessarily drained to empty.

Note: The CMEM level must always be polled before data is written to it; otherwise, data may be lost. Typically, the CFLEVEL pin is set to 3/4 empty. The host polls for this condition and sends 12 words before polling again.

4.8 Interrupt Cycle Timing

The CL450 supports both polled and vectored interrupts using the 680X0 vectored interrupt sequence.

Note: $\overline{INT}$ is an open-drain output so that it can be wire-ORed with other interrupt signals in the system. It should be pulled up to V_{CC} with a resistor no smaller that 470 ohms.

4.8.1 Polled Interupts

Polled interrupt operation is straightforward. The CL450 microapplication generates an interrupt by writing the *$\overline{Int}$* bit in the HOST_control register with zero, which asserts Interrupt Request ($\overline{INT}$). (The interrupt bit in the HOST_control register directly connects to the $\overline{INT}$ output pin.) Typically, the host first services the interrupt and then clears it by setting the interrupt bit in the HOST_control register. This action automatically deasserts $\overline{INT}$.

Note: The interrupt must also be logically cleared at some point by clearing the Interrupt Status location maintained by the microapplication in HMEM (see Section 14.2, Interrupt Status Location).

4.8.2 Vectored Interrupts

The vectored interrupt sequence is more complex. The microapplication sets the *$\overline{Int}$* bit in the HOST_control register, asserting $\overline{INT}$. The host responds by asserting Interrupt Acknowledge ($\overline{INTACK}$) and broadcasting the interrupt priority on address lines A[3:1].

The CL450 compares the priority of this interrupt to the priority previously written into the *IPID* bits of the HOST_intvecw register by the host. If the priorities match, the CL450 outputs the Interrupt Vector programmed into the *IVect* bits of the HOST_intvecw register on D[7:0] and signals its validity by asserting the $\overline{DTACK}$ signal. If the *IPID* and *IVect* bit fields are not written, the initial value of *IPID* is 0 and *IVect* = 0xf.

The host processor receives the Interrupt Vector and ends the cycle by releasing the $\overline{INTACK}$ signal. This action automatically clears the *$\overline{Int}$* bit in the HOST_control register if the *AIC* (Auto Interrupt Clear) bit is 1. If *AIC* is not 1, the host must clear *$\overline{Int}$* by writing to the HOST_control

register. For more information on interrupts and interrupt servicing, see Section 14.2.2, Interrupt Control Registers.

Note: The CL450 requires $\overline{AS}$ to be inactive (HIGH) during an interrupt acknowledge cycle. This differs from the 680X0 bus protocol in which $\overline{AS}$ is active (LOW) during an interrupt acknowledge cycle. The 680X0 system designer should therefore use $\overline{INTACK}$ in the $\overline{AS}$ generation combinatorial logic to make sure that $\overline{INTACK}$ and $\overline{AS}$ are mutually exclusive.

Alternatively, the CL450 can be mapped into an address range that has at least one zero in the upper address bits (A[23:21]). By doing this, the decoder for $\overline{AS}$ will not activate during an interrupt acknowledge cycle since the upper address bits must be all ones during this time.

This problem does not exist when using non-vectored interrupts.

Figure 4-9 shows the timing for a vectored interrupt cycle with automatic interrupt clearing.

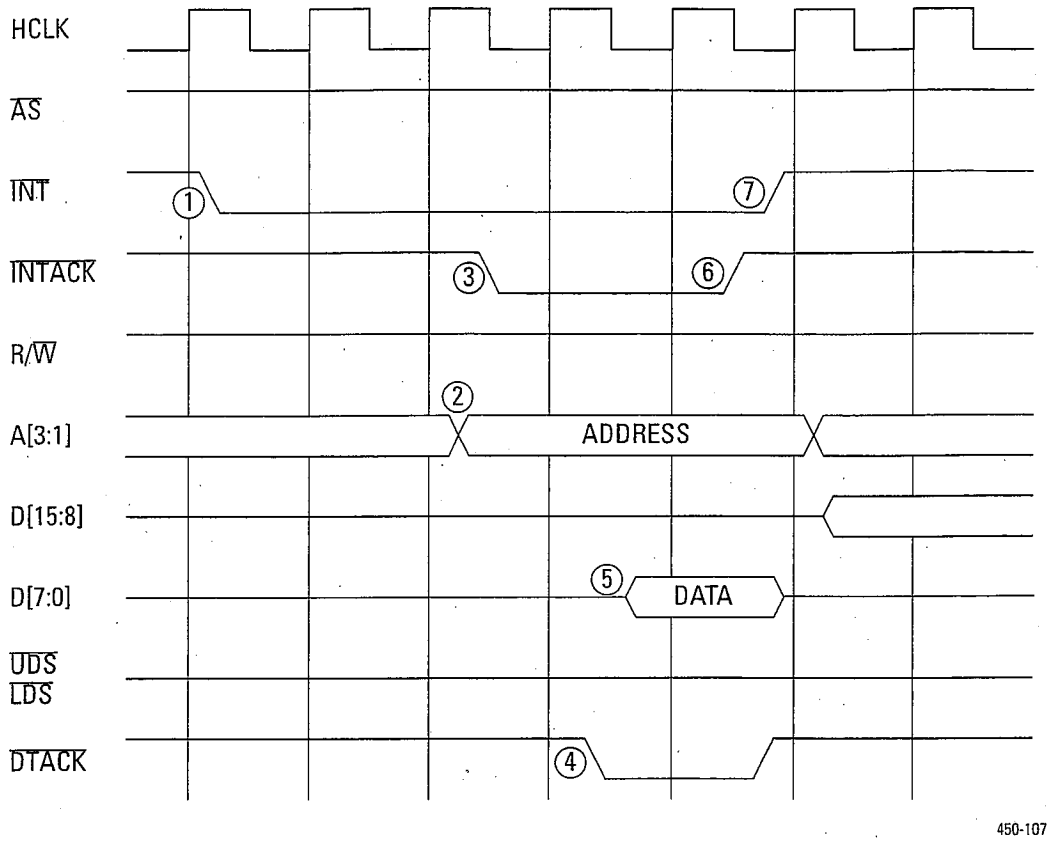


Figure 4-9 Vectored Interrupt Cycle (with auto interrupt clearing)

1. The CL450 requests an interrupt by asserting the $\overline{\text{INT}}$ signal.
2. The host system broadcasts the Interrupt Priority ID on address lines A[3:1].
3. The host system asserts the $\overline{\text{INTACK}}$ signal to acknowledge the interrupt.
4. The CL450 determines that the Interrupt Priority ID matches *IPID*, and drives the $\overline{\text{DTACK}}$ signal to indicate to the host processor that the vector interrupt address is valid.
5. The CL450 drives the vector interrupt address byte (*IVect*) on D[7:0].
6. The host releases the $\overline{\text{INTACK}}$ line. (The host latches the interrupt address on the rising edge of $\overline{\text{INTACK}}$.) This action signals the CL450 that the interrupt acknowledge cycle is complete.
7. The CL450 completes the cycle by releasing the interrupt request line (if *AIC*=1), $\overline{\text{DTACK}}$, and the address and data buses.

5
Local DRAM Interface

This chapter describes the local DRAM interface bus and how it is used. It details all of the signals necessary to connect the CL450 to a DRAM array and shows several example applications. The sections in this chapter are:

- 5.1: General Description
- 5.2: Memory Bus Interface
- 5.3: Memory Bus Timing

5.1 General Description

In addition to the CL450's internal registers, the host has direct access to the CL450's external DRAM buffer. Typically, DRAM is initialized by the host with the CL450's microapplication (as described in Section 10.2.2, Loading Sequence). After initialization, the host may access variables stored in DRAM (see Appendix A). The microapplication does not support any other host access to DRAM.

> *Note: Access to off-chip DRAM is the only operation in which byte-wide rather than 16-bit access to the CL450 is possible. All other transactions with the CL450 must be 16 bits.*

5.1.1 Amount of DRAM

The CL450 can be connected to either four Mbits or eight Mbits of local dynamic RAM. However, *only four Mbits of DRAM are needed.* If a second four Mbits of DRAM are added, it may be used by the system but it will be ignored by the microapplication.

The DRAM array is used to store:

- Compressed video data waiting to be decoded by the bitstream buffer
- The decoded video frame that is currently being displayed
- Future and past decoded reference frames
- CL450 microapplication instructions
- Bitstream header parameters
- The Command FIFO

5.1.2 Type and Organization of DRAM

The DRAM is organized as one or two banks of memory, each 256K locations deep by 16 bits wide, using 80ns fast-page-mode DRAMs. The CL450 can decode any MPEG constrained-parameter bitstream with a single bank of DRAM. However, designing the system so that it can be expanded to support the full amount of memory could allow the system to accommodate larger bitstream buffers and/or future enhanced microapplications.

> *Note: DRAMs which are "write per bit" or have a write mask should not be used with the CL450. More specifically, the DRAM used must ignore the state of the $\overline{WE}$ signal dur-*

ing a $\overline{CAS}$-before-$\overline{RAS}$ refresh cycle. (See the next section for an explanation of these signals.) Aside from this restriction, any normal 256K x 16 fast-page-mode DRAM can be used, and either 8/10 or 9/9 $\overline{RAS}$/$\overline{CAS}$ may be used.

Although this chapter primarily addresses the connections between the DRAM controller and external memory, it is also useful in understanding some of the other functions that the controller performs. For example, the DRAM controller acts as an intelligent peripheral to the CL450's internal CPU. It can be programmed in the microapplication to:

- Extract data from CMEM and place it in DRAM
- Move data from DRAM to the MPEG decoding engine
- Move data from the MPEG decoding engine back into the DRAM
- Move a decoded video frame from the DRAM to the video display unit
- Arbitrate requests from the host interface for access to the local DRAM

Once the CL450 CPU has instructed the controller to perform one or more of these functions, no more intervention is required until the process is complete. The CL450 CPU is thus free to perform other important functions.

5.2 Memory Bus Interface

The memory interface on the CL450 consists of a 10-bit multiplexed address bus, a 16-bit data bus, and the control lines necessary for reading and writing the DRAM: Write Enable ($\overline{WE}$), Column and Row Address Strobes ($\overline{LCAS}$, $\overline{UCAS}$ and $\overline{RAS}$[1:0]), and the data latch enable signals ($\overline{LCASIN}$ and $\overline{UCASIN}$). The high output drive of the CL450 allows it to directly drive the DRAMs without external buffers or logic. This lowers the system parts count and allows slower DRAMs to be used.

Memory arrays designed to use 256K x 4 DRAMs should only use nine of the ten address lines, MA[8:0]. Memory arrays designed to use 256K x 16 DRAMs should use all ten bits, MA[9:0], because some 256K x 16 DRAMs have asymmetrical row and column addresses (a ten-bit row address and an eight-bit column address). The CL450 supports both of

these memory organizations by duplicating column address bit nine in the tenth location of the row address as shown in Figure 5-1.

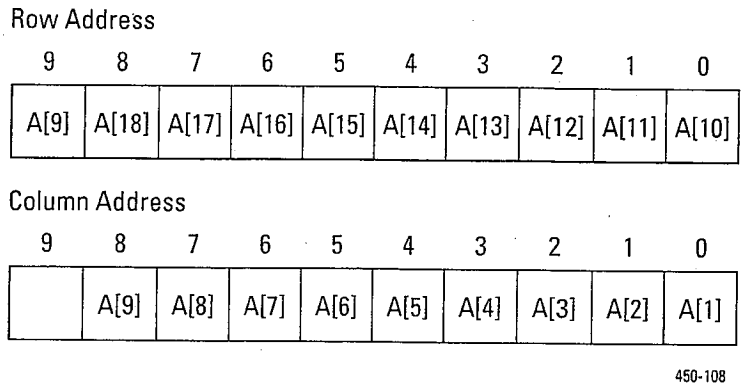


Figure 5-1 DRAM Address Bus Configuration

5.2.1 Memory Interface Connections

Figure 5-2 shows how the memory bus interface is connected when using 256K x 16 DRAMs, and Figure 5-3 show how the interface is connected when using 256K x 4 DRAMs.

The host processor addresses the local DRAM whenever address line A[20] is *LOW*. A[19] is used to select Bank 0 or Bank 1 through assertion of the $\overline{\text{RAS}}$[0] or $\overline{\text{RAS}}$[1] signal, respectively. Host address signals A[18:1] map directly into the multiplexed row and column DRAM addresses. The host bus interface signal R/$\overline{\text{W}}$ maps directly to the DRAM interface $\overline{\text{WE}}$ signal.

The host bus interface supports byte-wide accesses to the local DRAM by using the signals $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$ to select $\overline{\text{LCAS}}$ and $\overline{\text{UCAS}}$, respectively. $\overline{\text{LCASIN}}$ and $\overline{\text{UCASIN}}$ are used to latch the data coming back from the DRAMs.

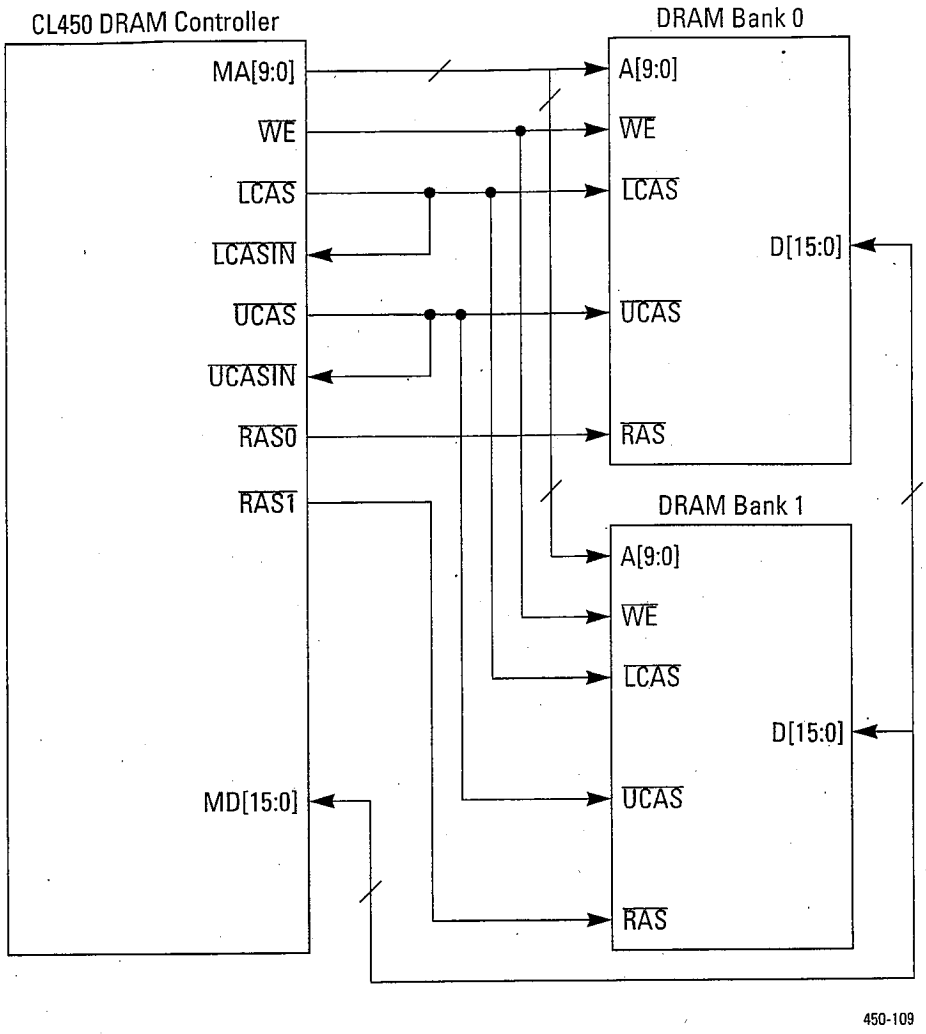


Figure 5-2 Local DRAM Implementation with 256K x 16 DRAMs

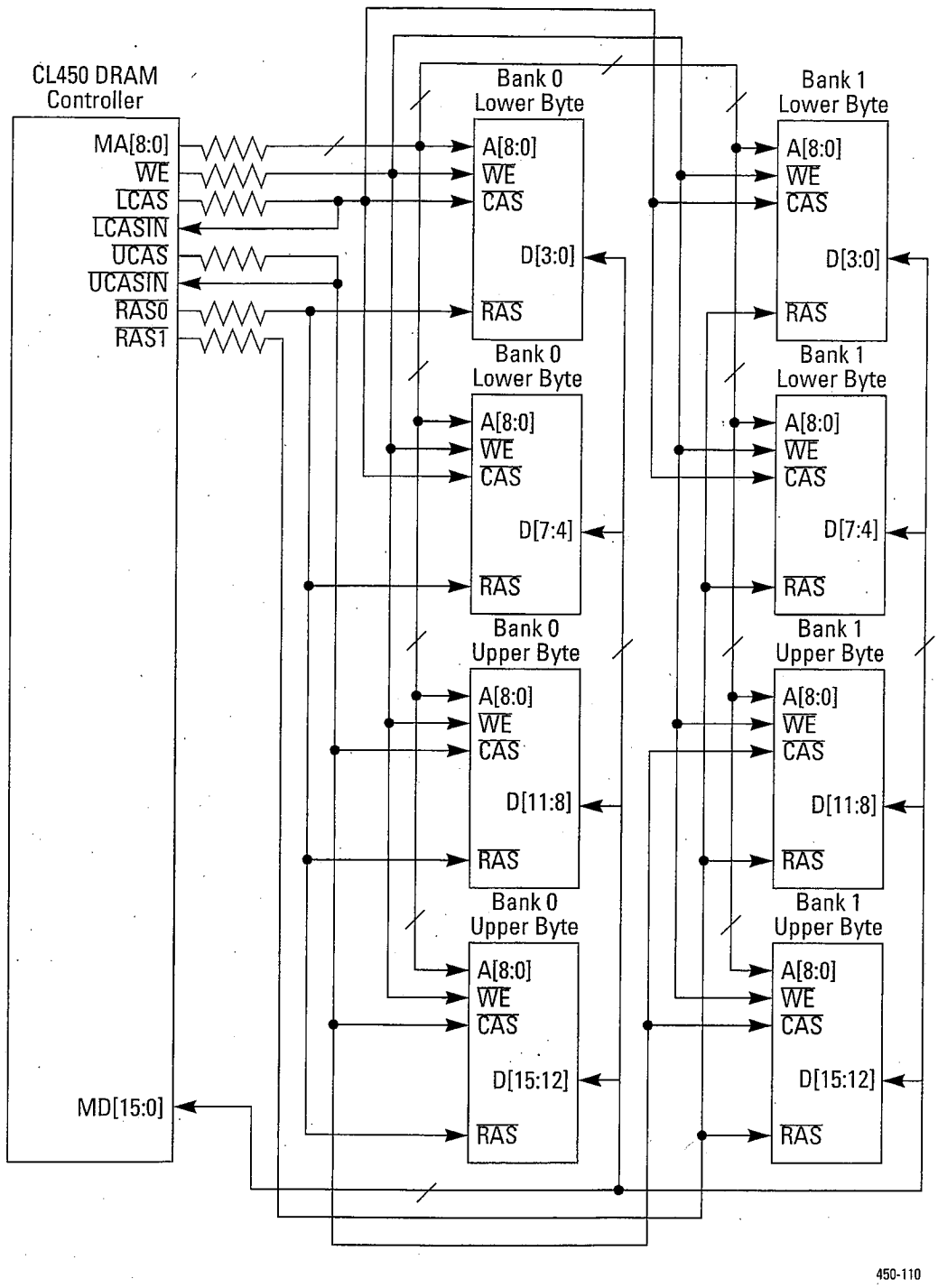


Figure 5-3 Local DRAM Implementation with 256K x 4 DRAMs

5.2.2 Memory Design Guidelines

When designing DRAM arrays for use with the CL450, the following guidelines should be followed:

- When using 256K x 16 DRAMs, connect all 10 address lines to the DRAMs. This allows DRAMs with either ten row and eight column address lines or nine row and nine column address lines to be used. The 10th address bit on 256K x 16 DRAMs is located on pin 15 of SOJ packages, pin 17 of TSOP packages and pin 25 of ZIP packages. This pin has no effect on DRAMs with nine row and nine column addresses.
- The $\overline{\text{LCAS}}$ and $\overline{\text{UCAS}}$ signals should be routed from the DRAM farthest from the CL450 to the $\overline{\text{LCASIN}}$ and $\overline{\text{UCASIN}}$ inputs. This gives the $\overline{\text{CASIN}}$ signal a delay path similar to the path of the DRAM read data to the CL450.
- Series damping resistors may need to be placed between the CL450 and the DRAMs to reduce overshoot and undershoot on the address lines and control signals. This problem can occur when there is a large array with long traces, such as when eight 256K x 4 DRAMs are used. The resistors should be connected as close to the CL450 as possible. The best value for the resistors depends on the board characteristics. The resistors are typically between 20 and 40 ohms with 33 ohms being most common.
- If the host bus byte-wide write capability is not used, only one of the $\overline{\text{CAS}}$ signals needs to be connected to the DRAMs. Either $\overline{\text{LCAS}}$ or $\overline{\text{UCAS}}$ can be used but must be connected to *both* the $\overline{\text{LCASIN}}$ and the $\overline{\text{UCASIN}}$ inputs. This allows 256K x 16 DRAMs with a single $\overline{\text{CAS}}$ input to be used.

  Note: The elimination of all possible DRAM noise is critical to proper DRAM layout for the CL450. Therefore, it is important that you keep the DRAM very close to the CL450, use terminations, and have a very "clean" ground and V_{CC} plane below the DRAM and those portions of the CL450 that connect to it. If possible, do not pass any other signals or clocks anywhere under the DRAMs or their traces.

5.3 Memory Bus Timing

The DRAM interface on the CL450 is designed to work with 80ns fast-page-mode DRAMs. All DRAM accesses are synchronized to GCLK. Fast-page-mode DRAMs allow shorter read and write cycle periods within a DRAM row. During a set of fast-page-mode accesses, a row address is strobed in using the $\overline{RAS}$ line, followed by several column addresses strobed in by $\overline{CAS}$. Releasing the $\overline{RAS}$ and $\overline{CAS}$ lines simultaneously completes the set of cycles.

Refresh is accomplished by asserting the $\overline{CAS}$ signal before the $\overline{RAS}$ signal (opposite of normal operation). This sequence starts a refresh cycle within the DRAM using the DRAM's internal address counter.

Refresh is performed at regular intervals determined by the value programmed into the DRAM_refcnt register (see page 8-23). Typical DRAM specifications require that all pages be refreshed each 8ms. The CL450 meets this specification by refreshing one page at a time.

5.3.1 DRAM Page-Mode Read Timing

Figure 5-4 shows the timing for a page-mode read. The circled numbers in the figure refer to the steps on the next page.

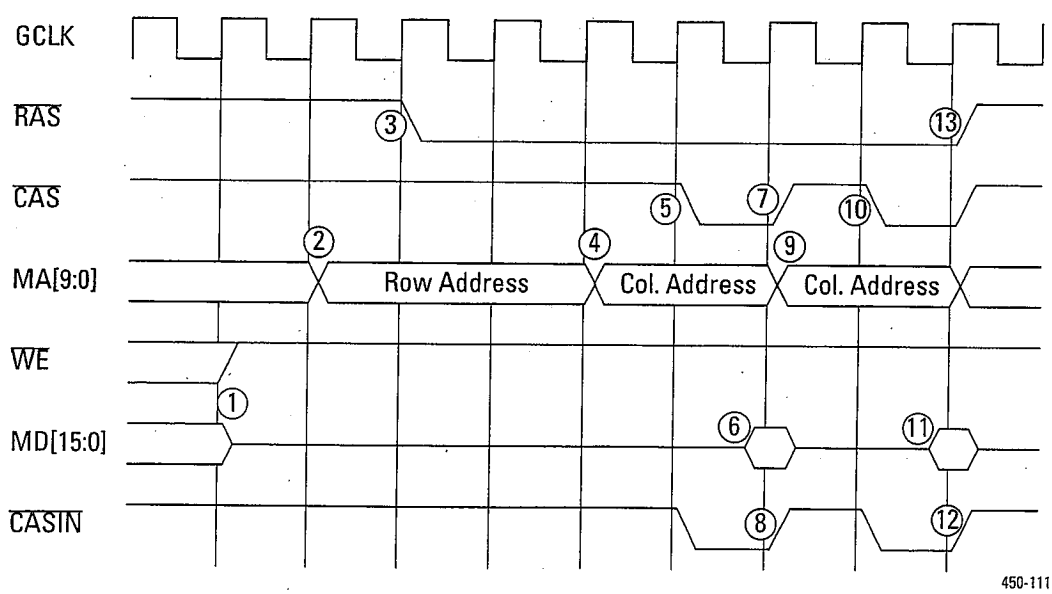


Figure 5-4 Page-Mode Read Timing

1. The CL450 starts a read cycle by three-stating the memory data bus, MD[15:0], and setting the write enable ($\overline{WE}$) line HIGH.
2. The CL450 then drives the row address onto the memory address (MA) bus.
3. The CL450 asserts the $\overline{RAS}$ line (LOW) to latch the row address into the DRAM.
4. The CL450 then outputs the column address of the first word to be read on MA.
5. The CL450 latches the column address into the DRAM by asserting the $\overline{CAS}$ line(s) (LOW).
6. The DRAMs output the data from the selected address.
7. The CL450 deasserts $\overline{CAS}$.
8. The data is loaded into the input latch of the CL450 on the rising edge of $\overline{CASIN}$. For a single-location read, the CL450 ends the cycle by deasserting $\overline{RAS}$ at this time.
9. For a page-mode read cycle, the CL450 outputs the column address of the next word to be read on MA[9:0].
10. The CL450 latches the address into the DRAM by asserting the $\overline{CAS}$ line.
11. The DRAMs output the data from the selected address.
12. The data is loaded into the input latch of the CL450 on the rising edge of $\overline{CASIN}$.
13. The CL450 deasserts $\overline{RAS}$ and $\overline{CAS}$ to complete the cycle.

5.3.2 DRAM Page-Mode Write Timing

Figure 5-5 shows the timing for a page-mode write. The circled numbers in the figure refer to the steps that follow.

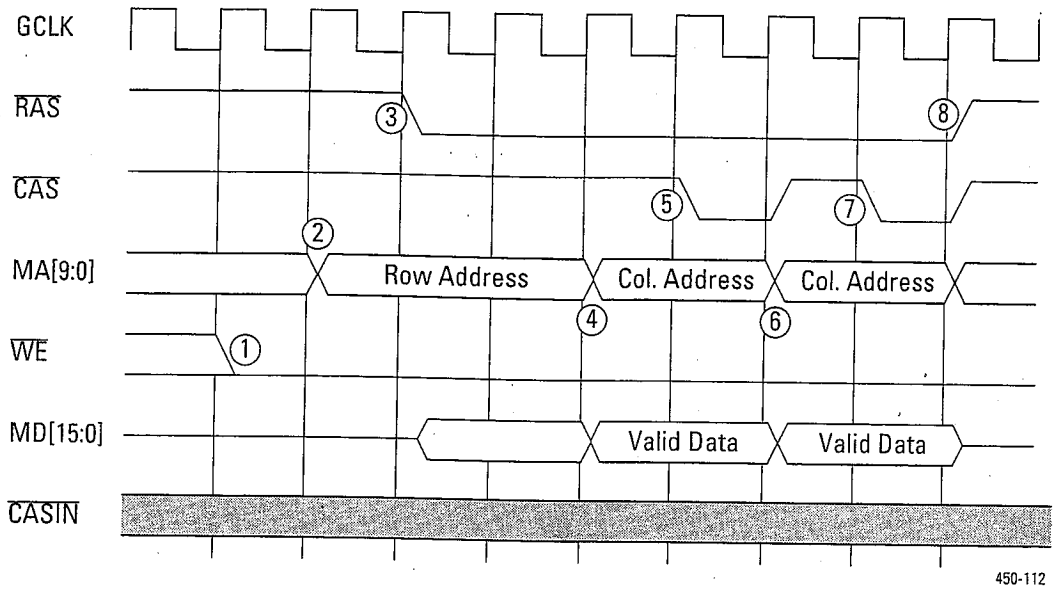


Figure 5-5 Page-Mode Write Timing

1. The CL450 starts a write cycle by asserting the write enable ($\overline{WE}$) line LOW.
2. The CL450 then drives the row address on the memory address (MA) bus.
3. The CL450 asserts the $\overline{RAS}$ line (LOW) to latch the row address into the DRAM.
4. The CL450 then outputs the column address of the first word to be written on the MA bus, and outputs the data to be written into the first address on the MD bus.
5. The CL450 asserts the $\overline{CAS}$ line (LOW) to write the data into the selected location of the DRAM.
6. For a single-location write, the write operation is terminated by driving the $\overline{RAS}$ and $\overline{CAS}$ lines inactive (HIGH) at this point. For a page-mode write, the address and data for the next word to be written are output on the MA and MD buses, respectively.
7. The CL450 asserts the $\overline{CAS}$ line to write the data into the selected location of the DRAM.
8. When the last word is written, the write operation is terminated by driving the $\overline{RAS}$ and $\overline{CAS}$ lines HIGH.

5.3.3 DRAM Refresh Timing

Figure 5-6 shows the timing for a memory refresh cycle. The circled numbers in the figure refer to the steps below.

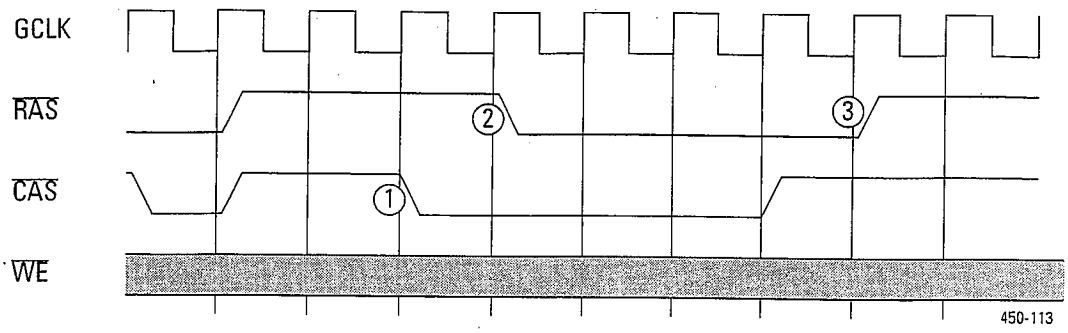


Figure 5-6 CAS-before-RAS Refresh Cycle

1. The CL450 initiates a DRAM refresh cycle by asserting the CAS line while the RAS line is inactive.
2. The CL450 asserts the RAS line to enable the refresh at the address pointed to by the DRAM's internal address counter.
3. The CL450 completes the refresh cycle by releasing RAS and CAS.

Note: The CL450 WE signal does not necessarily go inactive during DRAM refreshes, so use DRAMs that do not acknowledge the state of WE during DRAM refresh timing (i.e., when CAS is asserted while the RAS line is inactive).

6
Video Display Interface

This chapter describes the operation of the video display unit. It assumes that the reader is familiar with the way that television pictures are received, synchronized, and displayed.

The purpose of the video display unit is to output the decompressed video data in a form that can be either displayed on a television or video monitor, or mixed with computer-generated graphics to provide a video signal within a graphics window.

To perform these functions, the video display horizontally interpolates decompressed video, optionally converts pixel data from YCbCr to RGB color spaces, synchronizes it to the video clock, and outputs it to a display device.

The sections in this chapter are:

- 6.1: Digital Video Standards
- 6.2: Video Display Unit
- 6.3: Video Synchronization

- 6.4: Video Output
- 6.5: Video Output Enable Signal

6.1 Digital Video Standards

There are two primary standards for analog video transmission used in the world today:

- *NTSC (National Television Systems Committee) standard*: Used by North America, Japan, Korea, and Taiwan.
- *PAL (Phase Alternating Line) standard*: Used by Europe, China, and most of Africa.

The NTSC video standard specifies a scanning system of 525 horizontal lines per frame. Each frame consists of two interlaced fields of 262.5 horizontal lines. The field rate is 59.94 Hz.

The start of the frame is synchronized with a vertical sync pulse followed by an interval of 42 lines with no video, called *vertical blanking*. This leaves 480 lines in a frame for picture information (240 in a field).

The horizontal lines are synchronized with a horizontal synchronization pulse followed by an interval with no video, called *horizontal blanking*. Horizontal synchronization pulses occur at a rate of 15,734 Hz.

PAL video operates in much the same way, with these distinctions:

- The field rate is 50 Hz.
- There are 625 lines in a frame and 312.5 lines in a field.
- The horizontal sync pulses occur at a rate of 15,626 Hz.
- There are 576 lines of active video in a PAL frame and 288 lines in a field.

The SIF (source input format) specification reflects these standards by defining two *digital* video formats: 352 pixels x 240 lines x 30 Hz for compatibility with NTSC, and 352 pixels x 288 lines x 25 Hz for compatibility with PAL. Because the SIF frame rates are half those of the standard field rates, the CL450 displays each decoded SIF picture nominally for two field periods.

The actual video information may be represented using two data formats:

- *YCbCr data*: Consists of a Y value for the luminance, or brightness, of a pixel and two color values, Cb and Cr, for the chrominance. The Cb and Cr values represent color difference signals that can be converted into red, green, and blue values.
- *RGB (red, green, blue)*: The data output has a digital value that corresponds to the amplitude of the red, green, and blue signals. This digital value is converted to an analog signal before being displayed.

In accordance with the MPEG standard, CL450 compressed video data is always stored in YCbCr format, with Y values ranging from 16 to 235 as specified by MPEG. If RGB-encoded output video is needed, the YCbCr data is converted by the CL450 using a matrix which preserves the 220 discrete levels encoded by MPEG (as required by the CD-I standard), in which 16-16-16 RGB indicates the color black, while 235-235-235 indicates the color white.

Note: Your DAC should be set to the 16-235 values. If your DAC expects black at 0-0-0 and white at 255-255-255, then your output colors will not be as brilliant as expected.

On the CL450 development board or similar designs using the DAC in which the output of the CL450 is being stored into memory, the following steps may be taken to convert from 16-235 to the full, dynamic range of 0-255:

1. Subtract 16 to shift the RGB range to 0-219.

2. Multiply by 256/220 (1.1634) to expand the range to 256 levels.

RGB will now equal (RGB-16) x (1.1634).

6.2 Video Display Unit

The CL450's video display unit accepts decompressed YCbCr data from the local DRAM and outputs it as either horizontally interpolated YCbCr or RGB pixels. The pixels are clocked out by the VCLK (video clock) signal and are synchronized to external devices using the

$\overline{\text{HSYNC}}$ (Horizontal Sync) and VSYNC (Vertical Sync) input signals. Figure 6-1 shows the video bus connections.

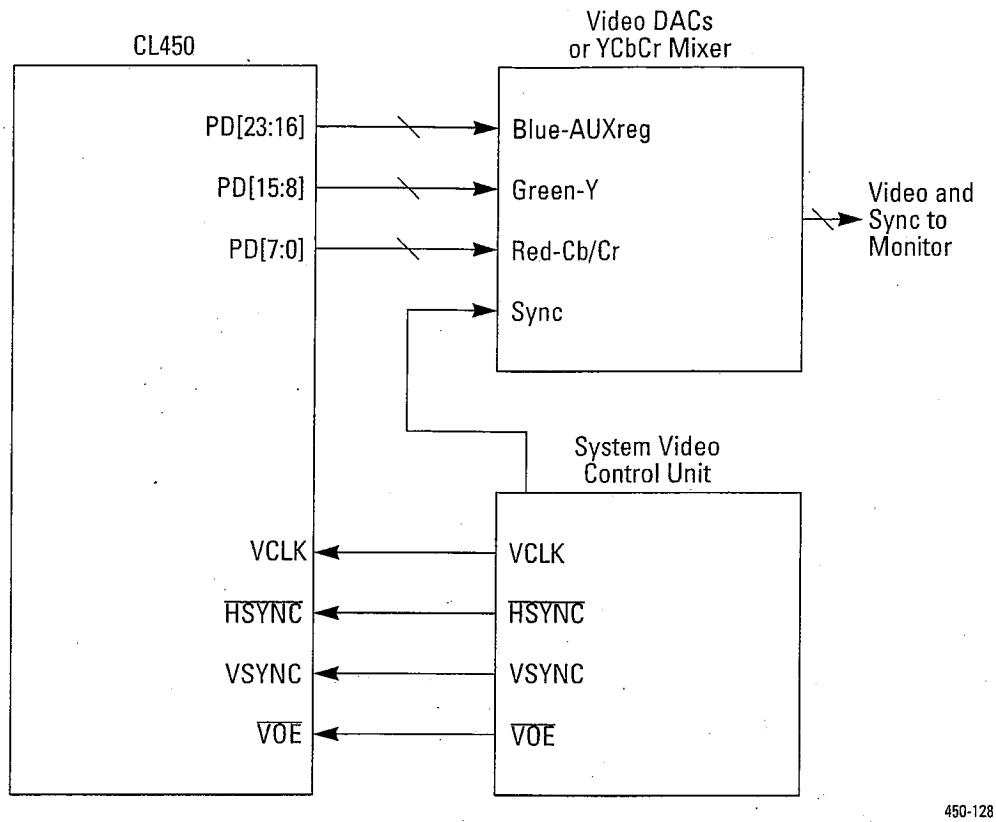


Figure 6-1 Typical Video Bus Connections

6.3 Video Synchronization

The CL450 does not generate video synchronization signals, but rather is designed to be synchronized with external horizontal and vertical sync signals. This allows it to be easily integrated with other video components in a multimedia system.

In a typical video display, the video information is surrounded by a colored border. The size and color of the border are programmed using the microapplication's SetBorder() command described in Chapter 11.

If the CL450 is directly driving the video generation logic, the border can be programmed to emulate the horizontal and vertical blanking intervals by selecting the appropriate border widths and setting the border color to black.

If the CL450 is being used to display video within a computer graphics window, the borders can be used to position the video window horizontally and vertically within the graphics frame. The border color is then programmed to a value that can easily be multiplexed with the computer-generated video color values.

A video field example is shown in Figure 6-2. This example field has a 15-line top border, 240 lines of displayed video, and a 12-line bottom border. Horizontally it has a 13-pixel left border, 320 pixels of displayed video, and a 15-pixel right border.

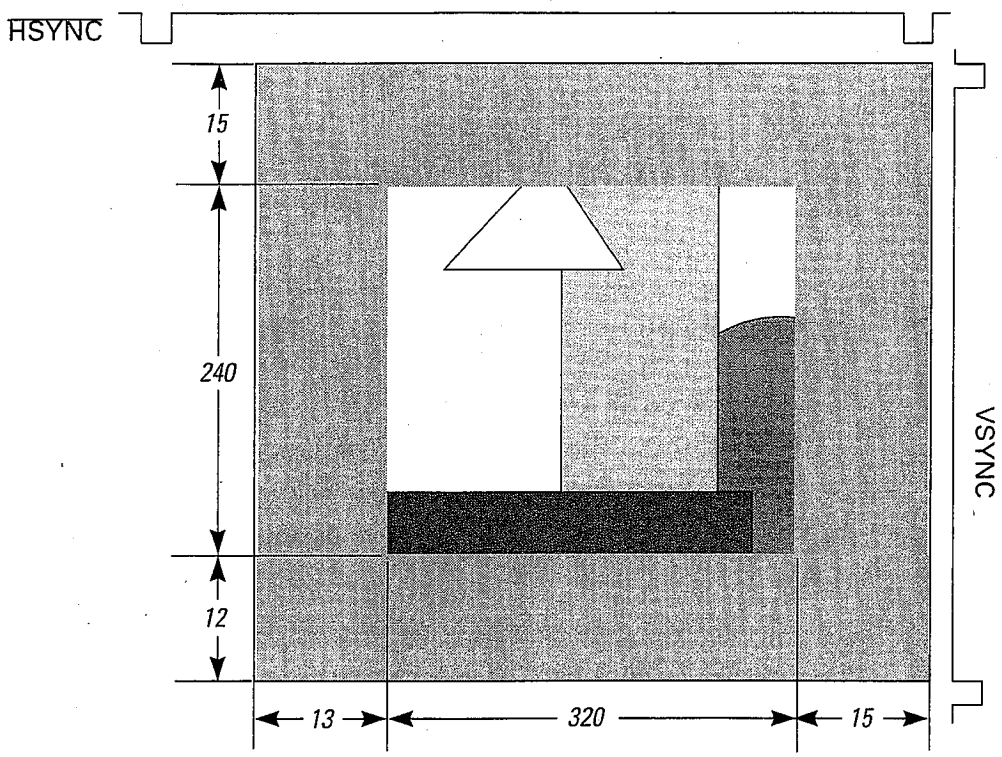


Figure 6-2 Example Video Frame

The display of the first of 15 lines of the top border of Figure 6-2 begins on the rising edge of the vertical synchronization pulse (VSYNC) received from the system video controller as shown in Figure 6-3. The circled numbers of Figure 6-3 refer to the steps below.

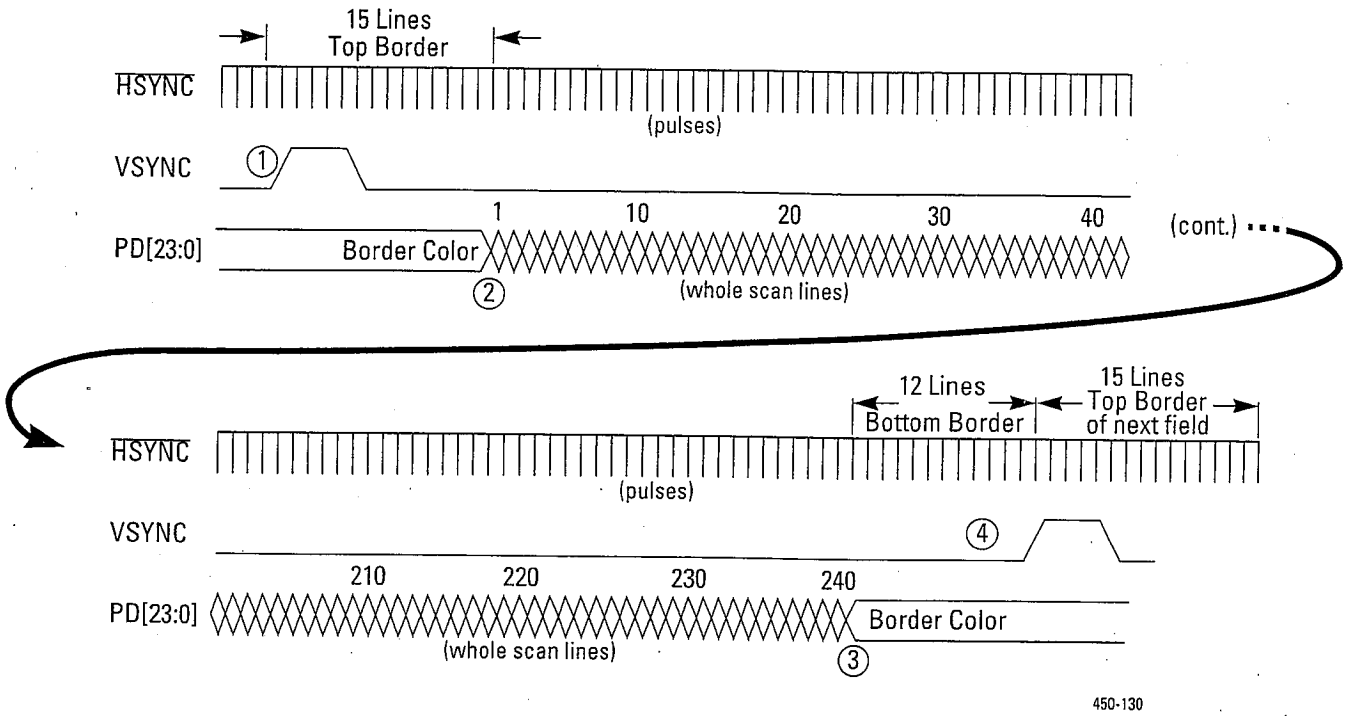


Figure 6-3 Vertical Timing

1. The video field begins when the CL450 receives the first rising edge of $\overline{\text{HSYNC}}$ after the rising edge of the vertical sync (VSYNC) pulse. (VSYNC can occur asynchronously to VCLK.)
2. At the end of the vertical border time, the CL450 starts to output active video lines on the pixel bus: in this example, 240 lines. Video lines start when the rising (inactive) edge of $\overline{\text{HSYNC}}$ is received.
3. The CL450 continues to output video until the last line has been output. At this point, it outputs the border color again until the end of the video field.
4. A rising edge on VSYNC starts a new field.

Figure 6-4 shows the beginning and end of a typical horizontal line. This example line has a border at the left side of the field that is 13 VCLKs wide, 320 active pixels, and a 15-pixel-wide border at the right side of the field. The circled numbers refer to the steps below.

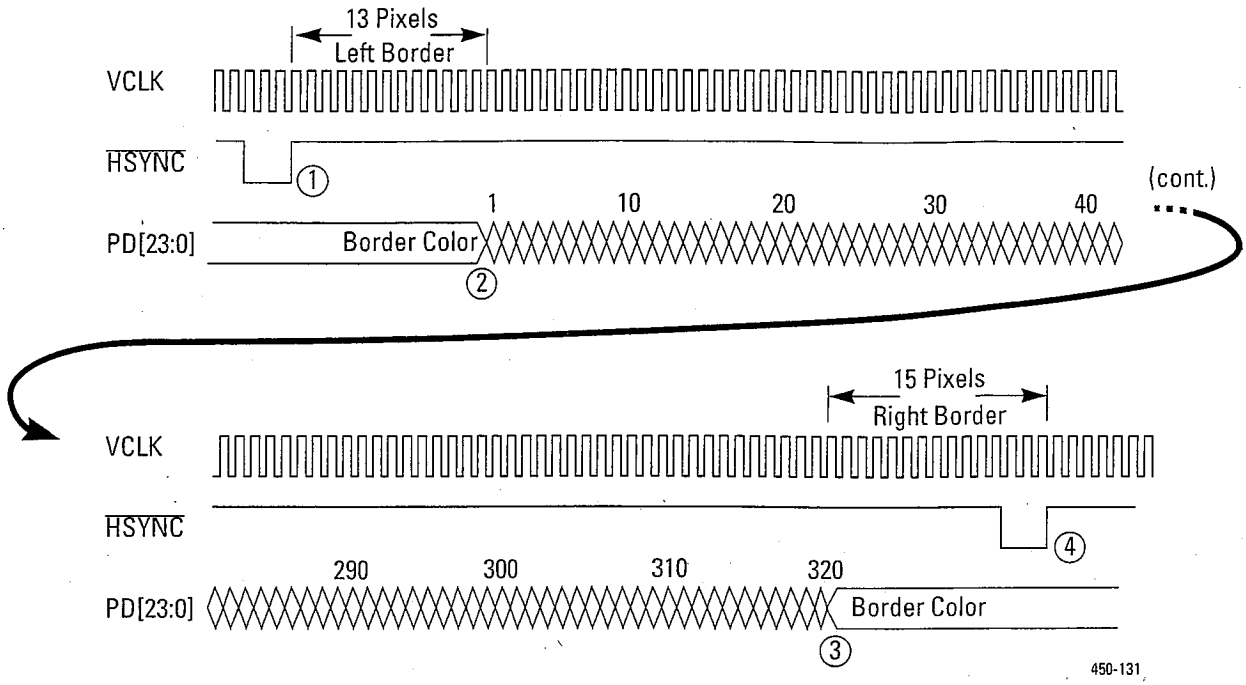


Figure 6-4 Horizontal Timing

1. A horizontal line begins when the CL450 receives an inactive (rising) edge on the horizontal sync signal ($\overline{\text{HSYNC}}$).
2. At the end of the left border time, the CL450 starts to output active video on the pixel bus. In our example, this will continue for 320 VCLK pulse widths.
3. The CL450 continues to output video until the last pixel has been transmitted. At that point, it will output the border color again until the end of the horizontal line.
4. An inactive edge on $\overline{\text{HSYNC}}$ causes a new horizontal line to be displayed.

To be reliably recognized, $\overline{\text{HSYNC}}$ falling edges must occur either less than 2 µs or more than 20 µs after the rising edge of VSYNC. If this restriction is not met, the first $\overline{\text{HSYNC}}$ signal after VSYNC might be ignored, causing the visible video window to randomly shift up and down by one line.

$\overline{\text{HSYNC}}$ edges which occur less than 2 μs after the VSYNC edge are considered to have been received before the VSYNC edge, while

$\overline{\text{HSYNC}}$ edges which occur more than 20 μs after the rising edge of VSYNC are considered as being received after the VSYNC edge.

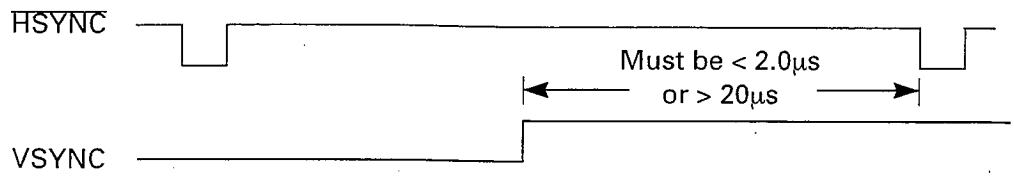


Figure 6-5 VSYNC Timing Restriction

6.4 Video Output

The video display unit outputs pixel values on the 24-bit wide pixel bus, PD[23:0]. The pixels can be output as either YCbCr or RGB values. The display mode is selected by the *RGB* bit in the VID_selmode register as described in Chapter 8.

In YCbCr mode, luminance values (Y) are output on PD[15:8] and chrominance (Cb and Cr) values are output on PD[7:0]. While luminance values are stored for every pixel, chrominance values Cb and Cr are only stored for every alternate pixel as shown in Figure 2-3. The reason for this is that the eye is far less sensitive to changes in color (chrominance) than it is to changes in brightness (luminance). The MPEG standard takes advantage of that fact to increase the data compression factor in this way. Because only a quarter as many Cb and Cr values (each) are stored as Y values, Cb and Cr are output by the CL450 in alternate VCLK periods (as shown in Figure 6-6) and identical CbCr information is output for every pair of scan lines displayed.

Note: The CL450 may begin output with a Cb or a Cr value. The determination as to whether the CL450 begins output with a decoded or interpolated value is, however, somewhat more complicated. That is, if the ***xOffset*** *parameter of the SetWindow() command (see page 11-41) is an even number, the CL450 begins output with decoded (real Y) values; however, if the* ***xOffset*** *parameter is odd, the CL450 begins output with an interpolated value.*

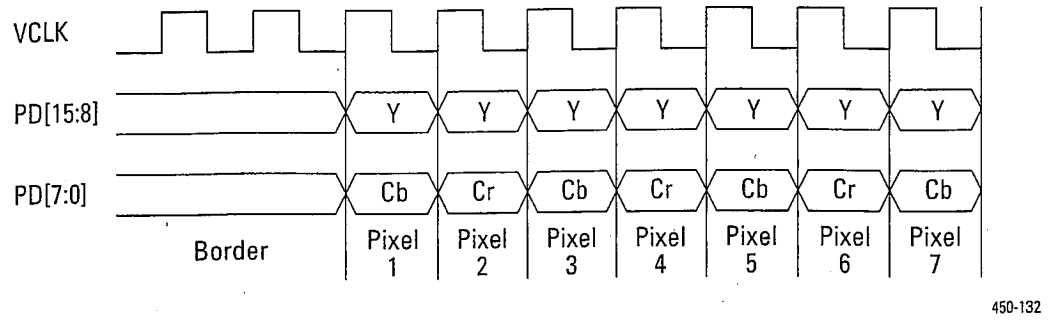


Figure 6-6 YCbCr Display Mode

In YCbCr mode, PD[23:16] outputs the 8-bit value in the VID_selaux register. Since these pins have no predefined use, they can be used as general-purpose programmable outputs.

Decompressed video data is always stored in YCbCr format in the CL450's local DRAM. If RGB outputs are required, the CL450 can be programmed to convert the data to RGB. The equation used to perform the conversion is shown in the description of the VID_sela and VID_-selb registers in Chapter 8. Figure 6-7 shows the PD[23:0] outputs when the CL450 is configured in RGB mode.

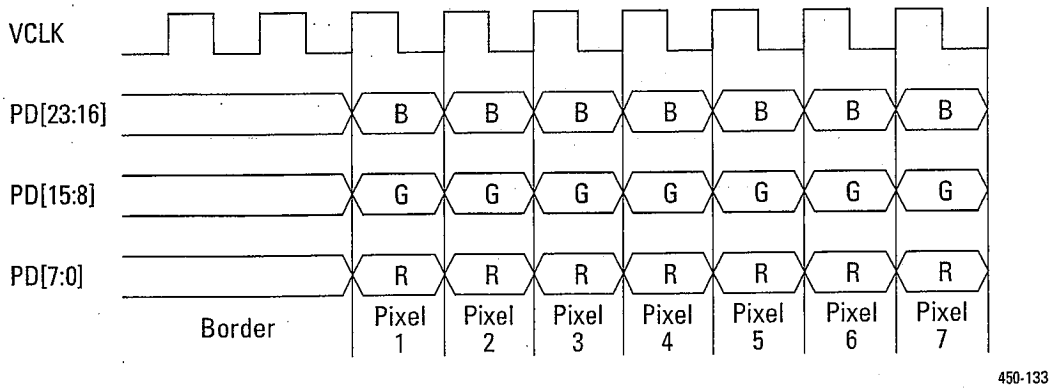


Figure 6-7 RGB Display Mode

Note: CL450 RGB mode is set to be compatible with CD-I players and therefore conforms to an output range of 16-235 as opposed to a range of 0-255 values (see page 6-3).

6.5 Video Output Enable Signal

The $\overline{VOE}$ signal is used to enable or disable the pixel bus outputs. When $\overline{VOE}$ is active (*LOW*), the PD[23:0] signals are driven. When $\overline{VOE}$ is *HIGH*, the outputs are three-stated as shown in Figure 6-8.

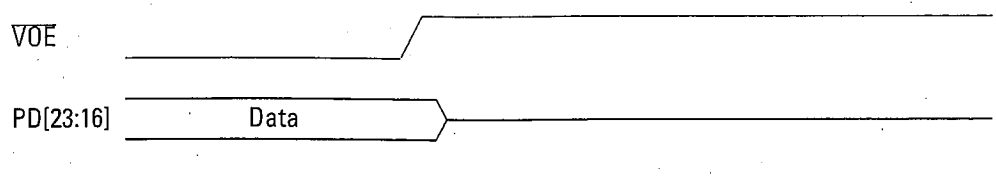


Figure 6-8 Video Output Enable Signal

In a typical video-within-a-window system, PD[23:0] outputs form a three-state bus along with the outputs of the system video controller. This bus is connected to the inputs to the video DACs. The system video controller disables its outputs and enables the CL450 outputs during the time that the CL450 window is being displayed. An example is shown in Figure 6-9.

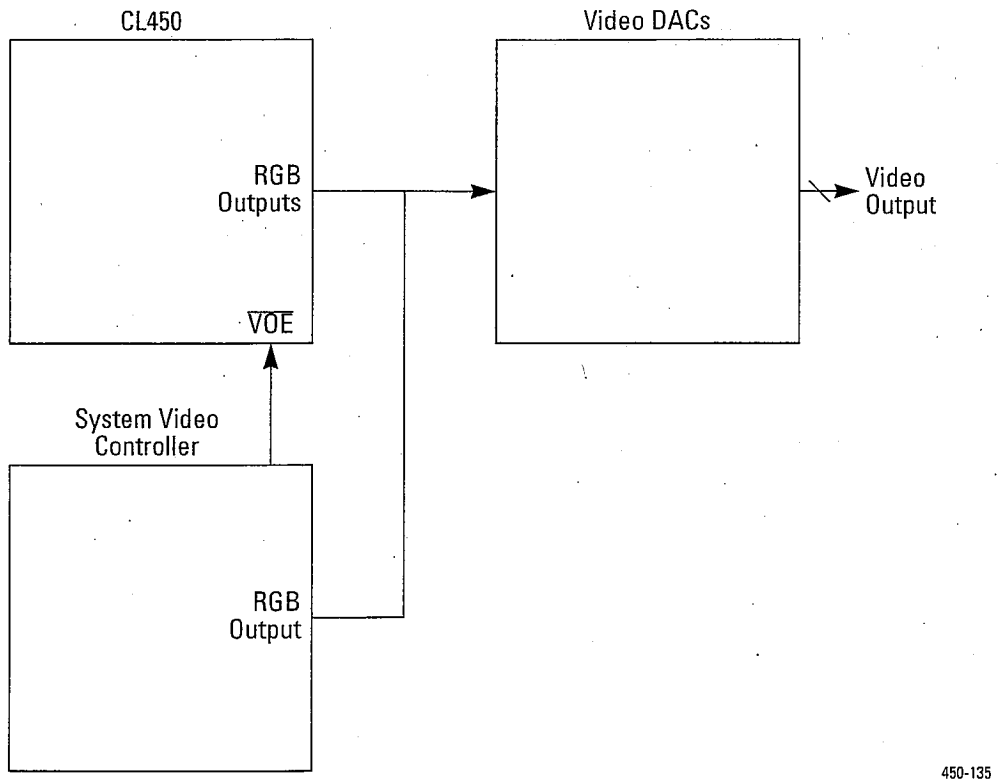


Figure 6-9 Multiplexing Video Outputs Using $\overline{VOE}$

7
Electrical and Physical Specifications

This chapter describes the CL450's electrical and mechanical characteristics. The chapter is divided into three sections:

- 7.1: Operating Conditions
- 7.2: AC Timing Characteristics
- 7.3: Package Specifications

Note: These specifications represent an improvement over previously published specifications.

7.1 Operating Conditions

This section specifies the electrical characteristics of the CL450.

Table 7-1 Absolute Maximum Ratings[1]

Parameter	Value
Supply Voltage (V_{DD})	-0.5 to 6.5 V
Input Voltage	-1.0 to (V_{DD} +1.0)
Output Voltage	-0.5 to (V_{DD} +0.5)
Storage temperature range	-65°C to 150°C
Operating temperature range (case)	0°C to 90°C
Reflow Soldering temperature	240°C for 5 Seconds Maximum

1. Exposure to stresses beyond those listed in this table may result in device unreliability, permanent damage, or both.

Table 7-2 Operating Conditions

Parameters		Commercial		Unit
		Min	Max	
V_{DD}	Supply Voltage	4.75	5.25	V
t_{CASE}	Operating Temperature	0	85	°C

Table 7-3 DC Characteristics

Parameters		Test Conditions	Commercial			Unit
			Min	Typ	Max	
V_{IH}	High-level input voltage [1]	V_{DD} = MAX	2.4			V
V_{IL}	Low-level input voltage [1]	V_{DD} = MIN			0.8	V
V_{OH}	High-level output voltage	V_{DD} = MIN, I_{OH} = -2.0 mA	2.8			V
V_{OL}	Low-level output voltage	V_{DD} = MIN, I_{OL} = 8.0 mA			0.5	V
I_{IH}	High-level input current	V_{DD} = MAX, V_{IN} = V_{DD}			10	µA
I_{IL}	Low-level input current	V_{DD} = MAX, V_{IN} = 0 V	-10			µA
I_{OZ}	Output leakage current	Hi-Z output driven to 0V and 5.25 V	-10		+10	µA
I_{DD0}	Supply Current @ T_j=0°C	V_{DD} = MAX, GCLK = 40MHz V_{IN} = 0 or V_{DD}, C_L=50pF		320	400	mA
I_{DD85}	Supply Current @ T_j=85°C	V_{DD} = MAX, GCLK = 40MHz V_{IN} = 0 or V_{DD}, C_L=50pF		300	350	mA
C_{IN}	Input Capacitance [1]			10		pF
C_{OUT}	Output Capacitance [1]			12		pF
CI/O	Output Capacitance [1]			12		pF

1. Not 100% tested, guaranteed by design characterization

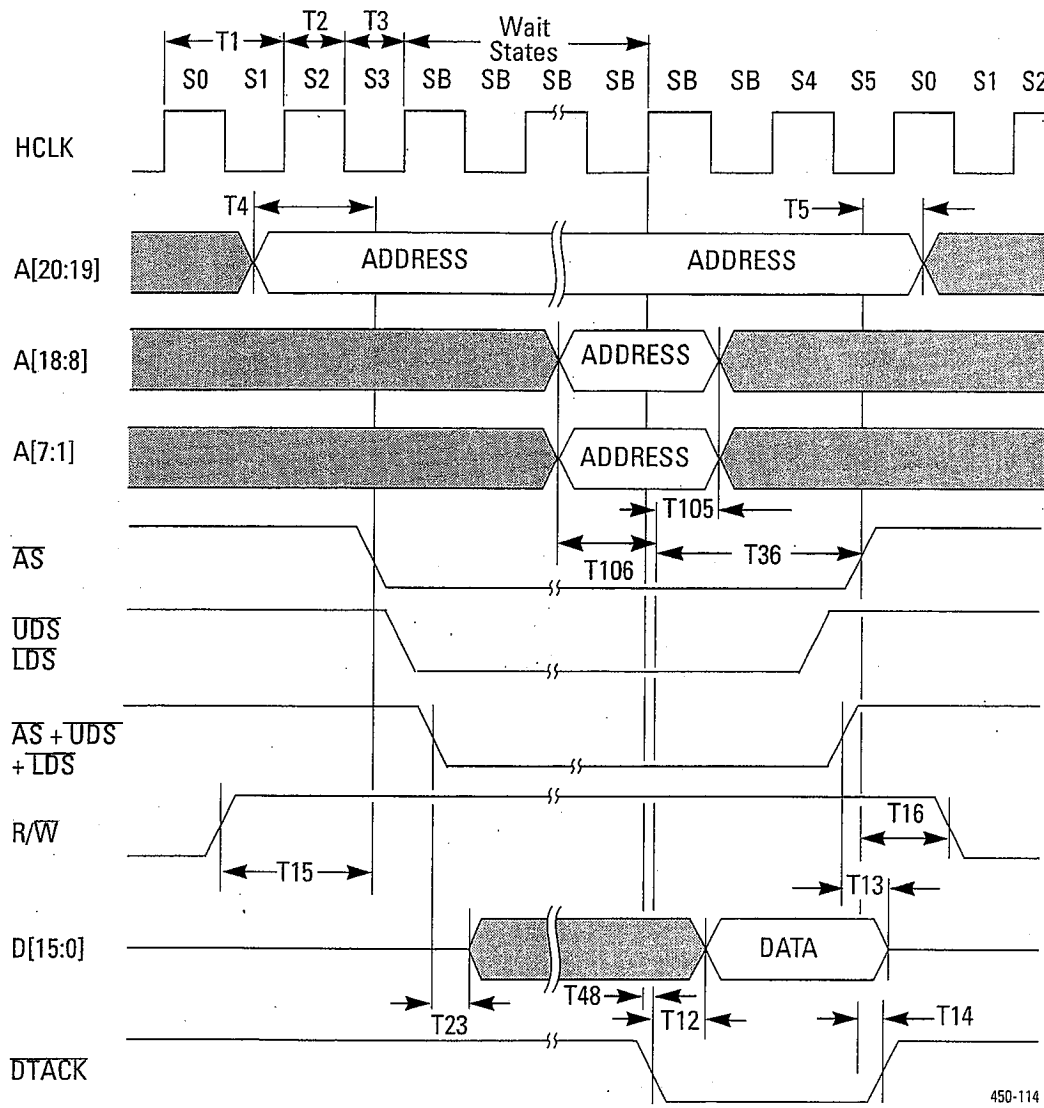
7.2 AC Timing Characteristics

This section describes the AC timing characteristics of the CL450. The timing characteristics are divided into related groups and depicted with one or more timing diagrams and a table of timing values. The groups are:

- Host Bus Memory and Register Timing
  - Figure 7-1, Host Bus Local DRAM Read
  - Figure 7-2, Host Bus Register Read
  - Figure 7-3, Host Bus Local DRAM Write
  - Figure 7-4, Host Bus Register Write
  - Table 7-4, Local DRAM or Register Access
- Host Bus CMEM Timing
  - Figure 7-5, CMEM Write
  - Figure 7-6, CMEM DMA Write
  - Table 7-5, CMEM Access
- Host Bus Vectored Interrupt Cycle Timing
  - Figure 7-7, Vectored Interrupt Cycle with Auto Clear
  - Table 7-6, Vectored Interrupt Cycle with Auto Clear
- Local DRAM Bus Timing
  - Figure 7-8, Local DRAM Bus
  - Figure 7-9, Local DRAM $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ Refresh
  - Table 7-7, Local DRAM Bus
- GCLK, SCLK and $\overline{\text{RESET}}$ Timing
  - Figure 7-10, GCLK
  - Figure 7-11, $\overline{\text{RESET}}$
  - Table 7-8, GCLK and $\overline{\text{RESET}}$
  - Figure 7-12, SCLK Input
  - Table 7-9, SCLK Input
- Video Bus Timing
  - Figure 7-13, Video Bus Inputs
  - Figure 7-14, VCLK
  - Table 7-10, Video Bus Inputs

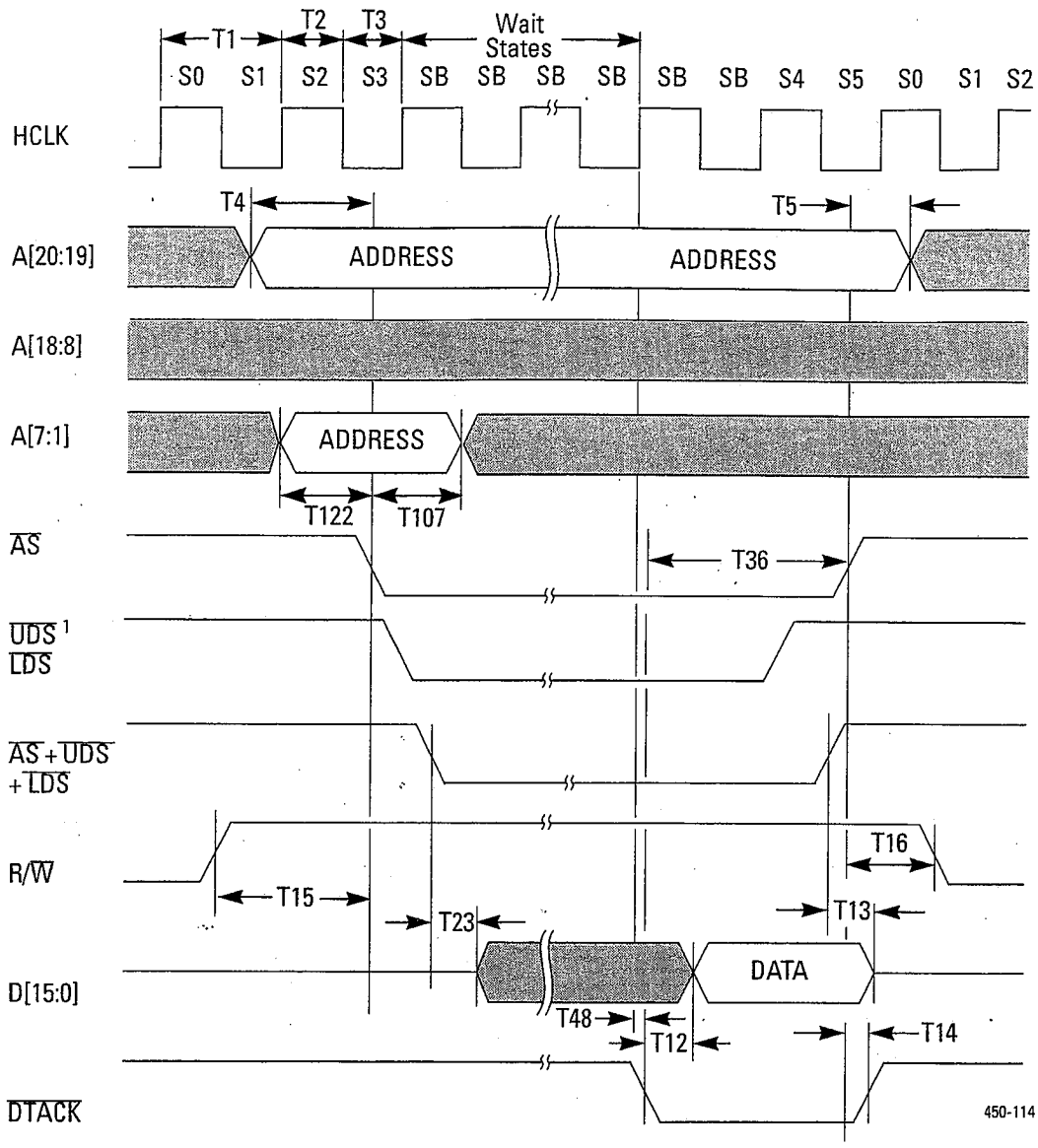
7.2.1 Host Bus Memory and Register Timing

Host requests for access to the CL450's local DRAM and registers are not always honored immediately since higher priority operations within the CL450 may delay access. In such cases, the CL450 inserts wait states by holding off the assertion of $\overline{\text{DTACK}}$. At least one wait state is inserted in every local DRAM or register access. Since register and local DRAM accesses are made primarily during initialization of the CL450, and rarely during operation, the impact of these delays on system performance are minimal.



Note: Please see Motorola's *MC68070 Technical Summary* for an explanation of cycle time naming conventions SB, S0, etc.

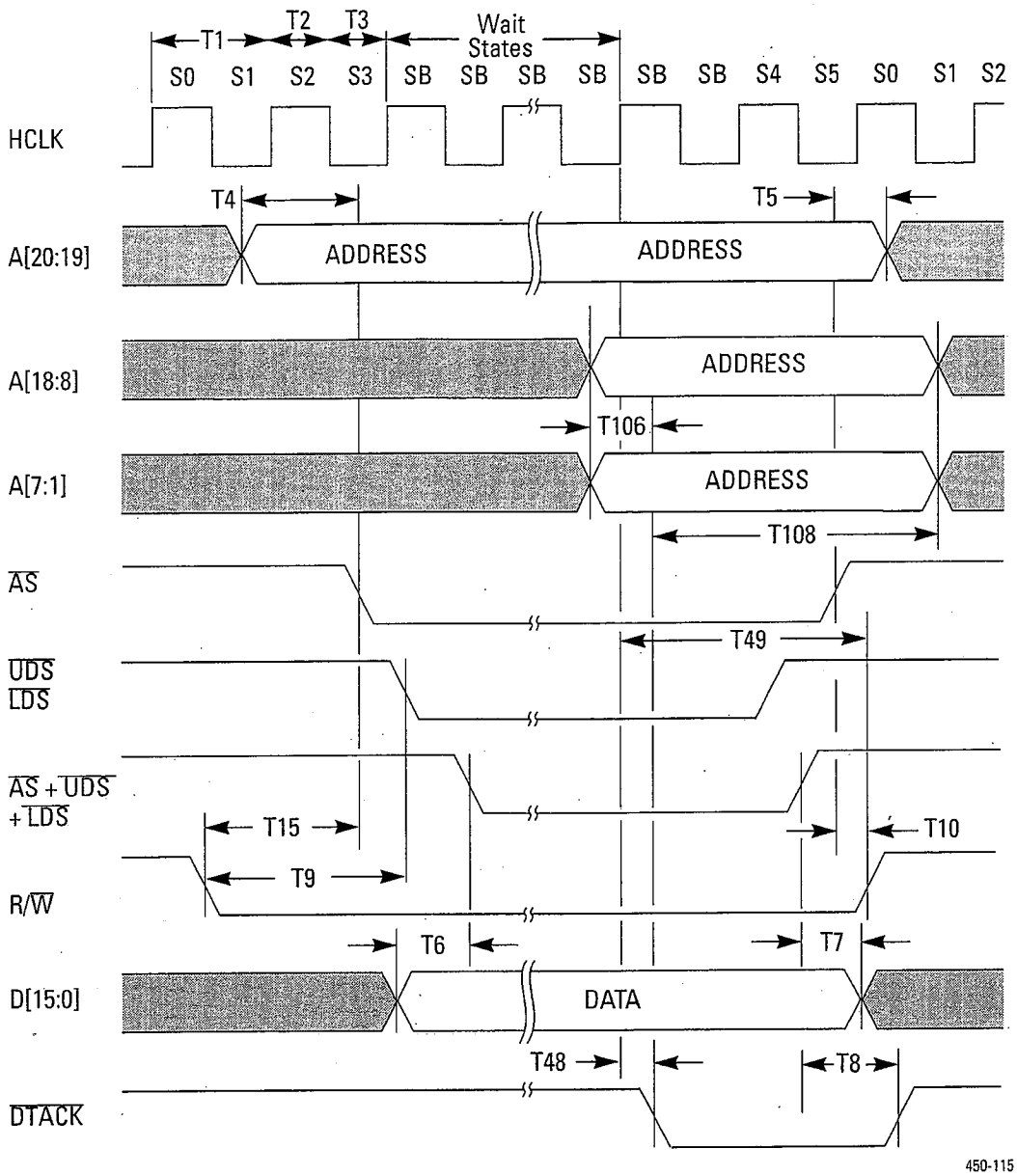
Figure 7-1 Timing Diagram - Host Bus Local DRAM Read



1. CMEM does not support byte-wide writes.

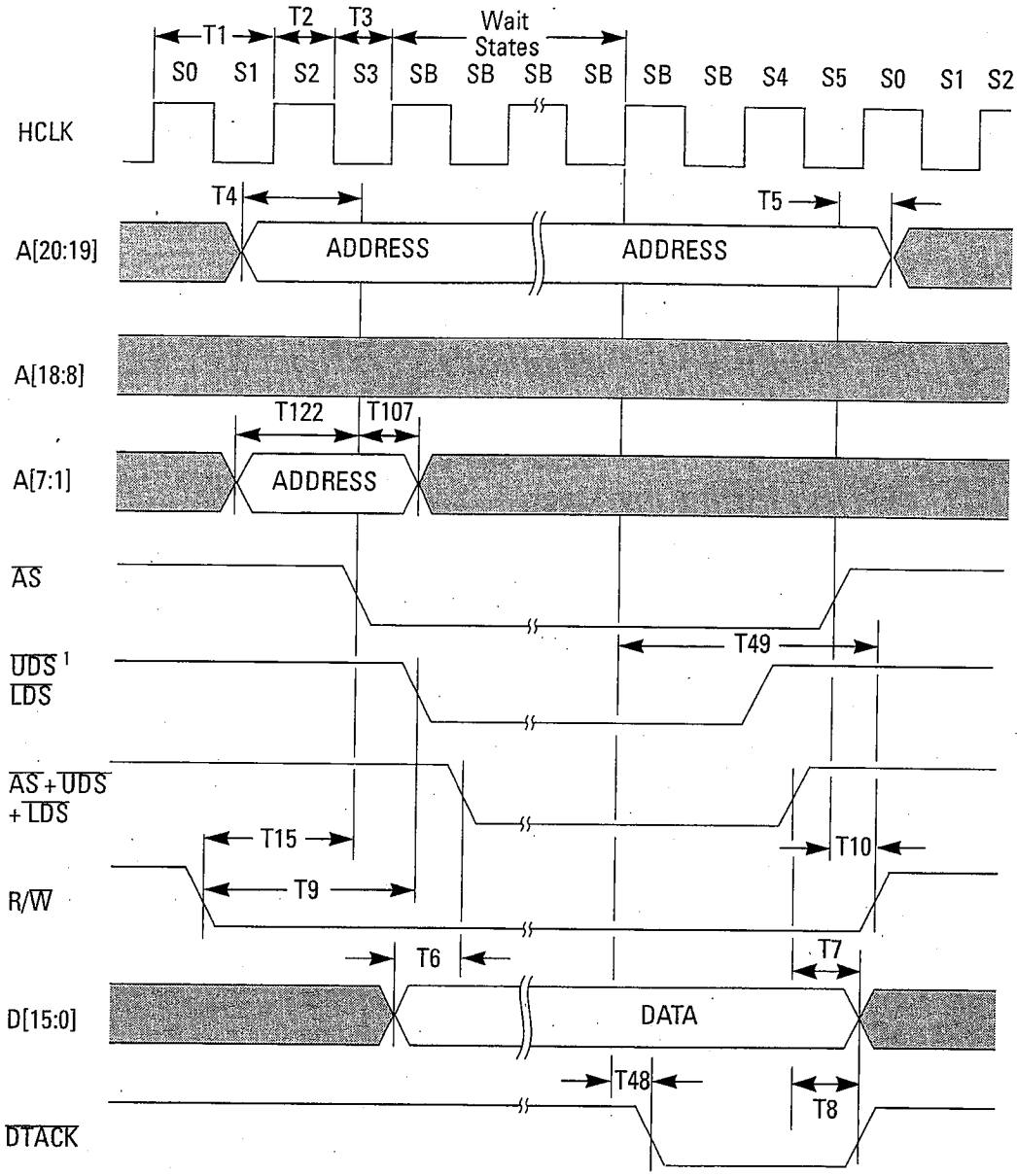
Note: Please see Motorola's *MC68070 Technical Summary* for an explanation of cycle time naming conventions SB, S0, etc.

Figure 7-2 Timing Diagram - Host Bus Register Read



Note: Please see Motorola's *MC68070 Technical Summary* for an explanation of cycle time naming conventions SB, S0, etc.

Figure 7-3 Timing Diagram - Host Bus Local DRAM Write



1. The CL450's registers do not support byte-wide writes.

450-115

Note: Please see Motorola's *MC68070 Technical Summary* for an explanation of cycle time naming conventions SB, S0, etc.

Figure 7-4 Host Bus Register Write

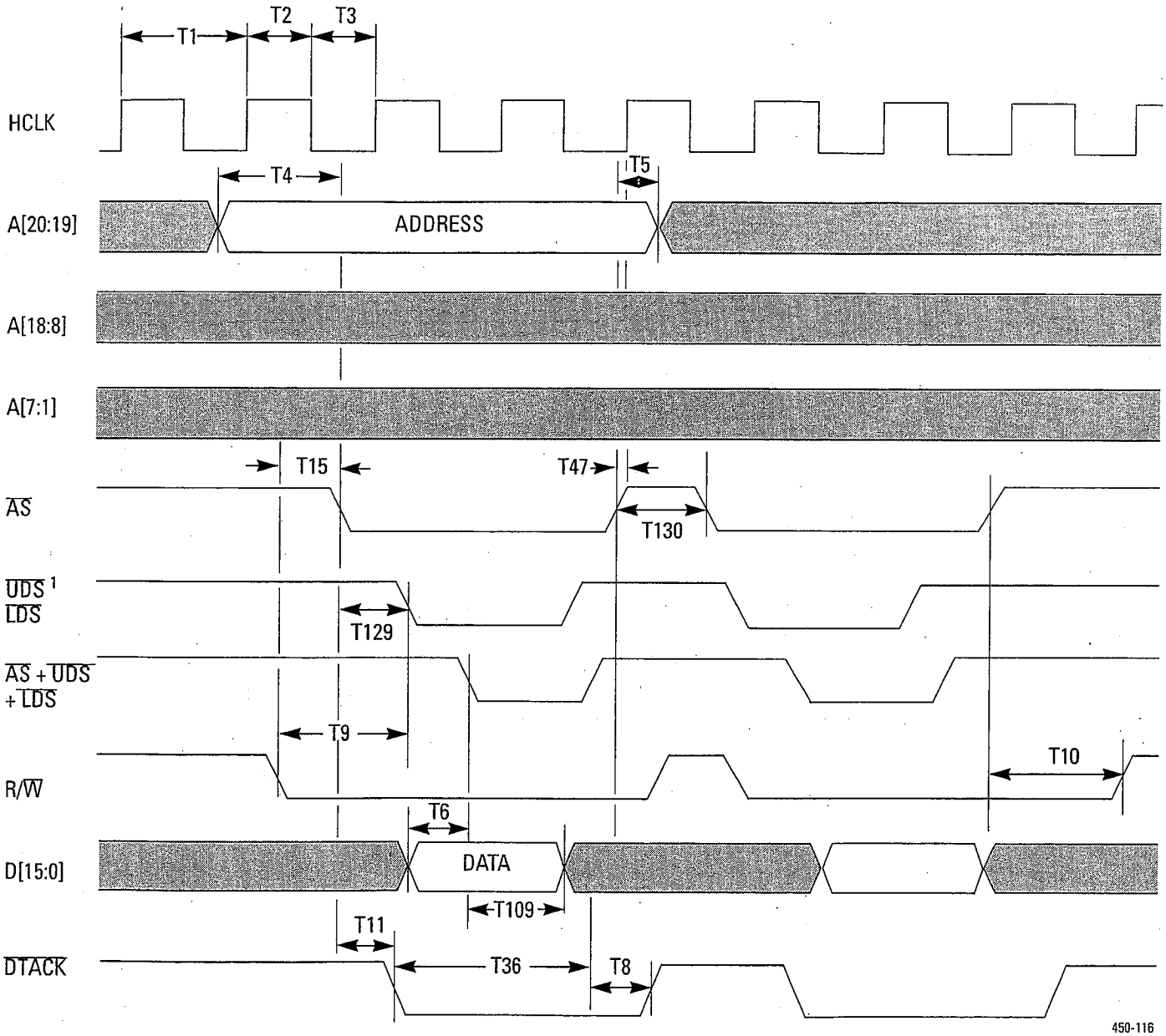
Table 7-4 Timing Characteristics - Local DRAM or Register Access [1]

Time	Description	Min	Max	Units
T1	HCLK Clock Period	2 (T31)		ns
T2	HCLK HIGH Pulse Width [3]	20		ns
T3	HCLK LOW Pulse Width [3]	20		ns
T4	A[20:19] valid setup to $\overline{AS}$ LOW	10		ns
T5	A[20:19] valid hold past $\overline{AS}$ HIGH	0		ns
T6	D[15:0] setup to $\overline{AS}$ + $\overline{UDS}$ + $\overline{LDS}$ LOW (write)	15		ns
T7	D[15:0] hold past $\overline{AS}$ + $\overline{UDS}$ + $\overline{LDS}$ HIGH (write)	0		ns
T8	$\overline{AS}$ + $\overline{UDS}$ + $\overline{LDS}$ HIGH to $\overline{DTACK}$ HIGH (write) [2]		25	ns
T9	R/$\overline{W}$ setup to $\overline{UDS}$, $\overline{LDS}$ LOW (write)	20		ns
T10	R/$\overline{W}$ hold past $\overline{AS}$ HIGH (write)	0		ns
T12	$\overline{DTACK}$ low to D[15:0] valid (read) [3]		15	ns
T13	D[15:0] turn-off time from $\overline{AS}$ + $\overline{UDS}$ + $\overline{LDS}$ HIGH (read) [3,4]	2	15	ns
T14	$\overline{AS}$ HIGH to $\overline{DTACK}$ HIGH (read) [2]			ns
T15	R/$\overline{W}$ setup to $\overline{AS}$ LOW	10		ns
T16	R/$\overline{W}$ hold past $\overline{AS}$ HIGH (read)	10		ns
T23	$\overline{AS}$ + $\overline{UDS}$ + $\overline{LDS}$ LOW to D[15:0] driven	0		ns
T48	HCLK HIGH to $\overline{DTACK}$ LOW		25	ns
T49	R/$\overline{W}$ hold past HCLK HIGH (write)[3]	2 (T1) + 10		ns
T105	A[18:1] hold time from $\overline{DTACK}$ LOW	0		ns
T106	A[18:1] setup time to $\overline{DTACK}$ LOW	TBD		ns
T107	A[7:1] hold time from $\overline{AS}$ LOW	10		ns
T108	A[18:1] hold time from $\overline{DTACK}$ LOW	3 (T31) + 10		ns
T122	A[7:1] setup time to $\overline{AS}$ LOW	TBD		

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V. Output load capacitance = 50 pF.
2. Open-Drain Output. Timing specification assumes an external pull-up resistor with a value of 470 Ohms.
3. Not 100% tested, guaranteed by design characterization.
4. Time at which output achieves an open circuit condition; not referenced to an output voltage level.

7.2.2 Host Bus CMEM Timing

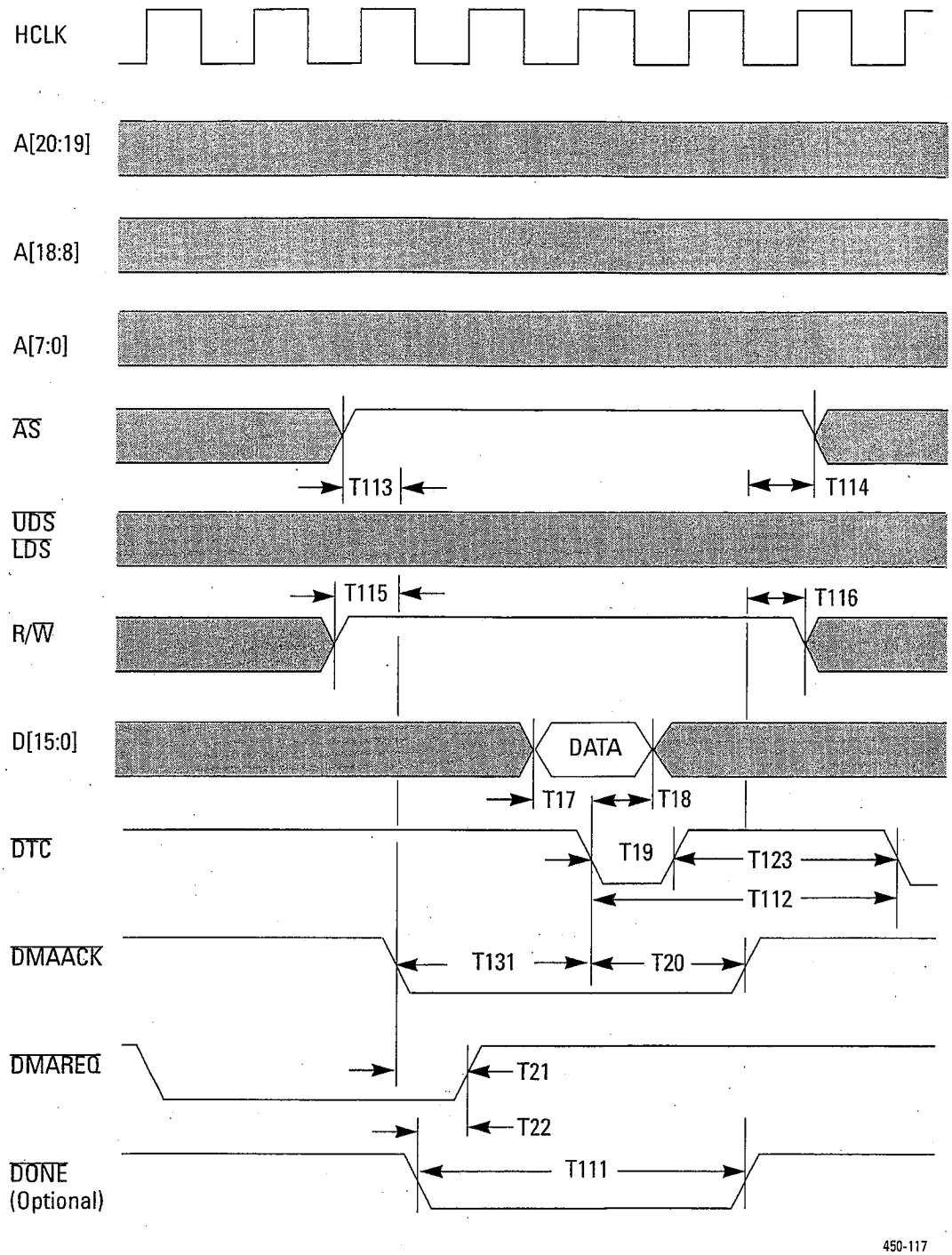
CMEM logic has been optimized to minimize data transfer times. Direct writes and DMA transfers to CMEM are synchronized within the CL450 so no host bus wait states are necessary.



1. CMEM does not support byte-wide writes.

Note: Please see Motorola's *MC68070 Technical Summary* for an explanation of cycle time naming conventions SB, S0, etc.

Figure 7-5 Timing Diagram - CMEM Write



Note: DMAREQ will only be assertd by the CL450 after the DMA Enable bit (*DE*, see page 8-11) has been correctly set by the host. Once set, any number of falling edges may be produced on DMAREQ until the bit is cleared.

Figure 7-6 Timing Diagram - CMEM DMA Write

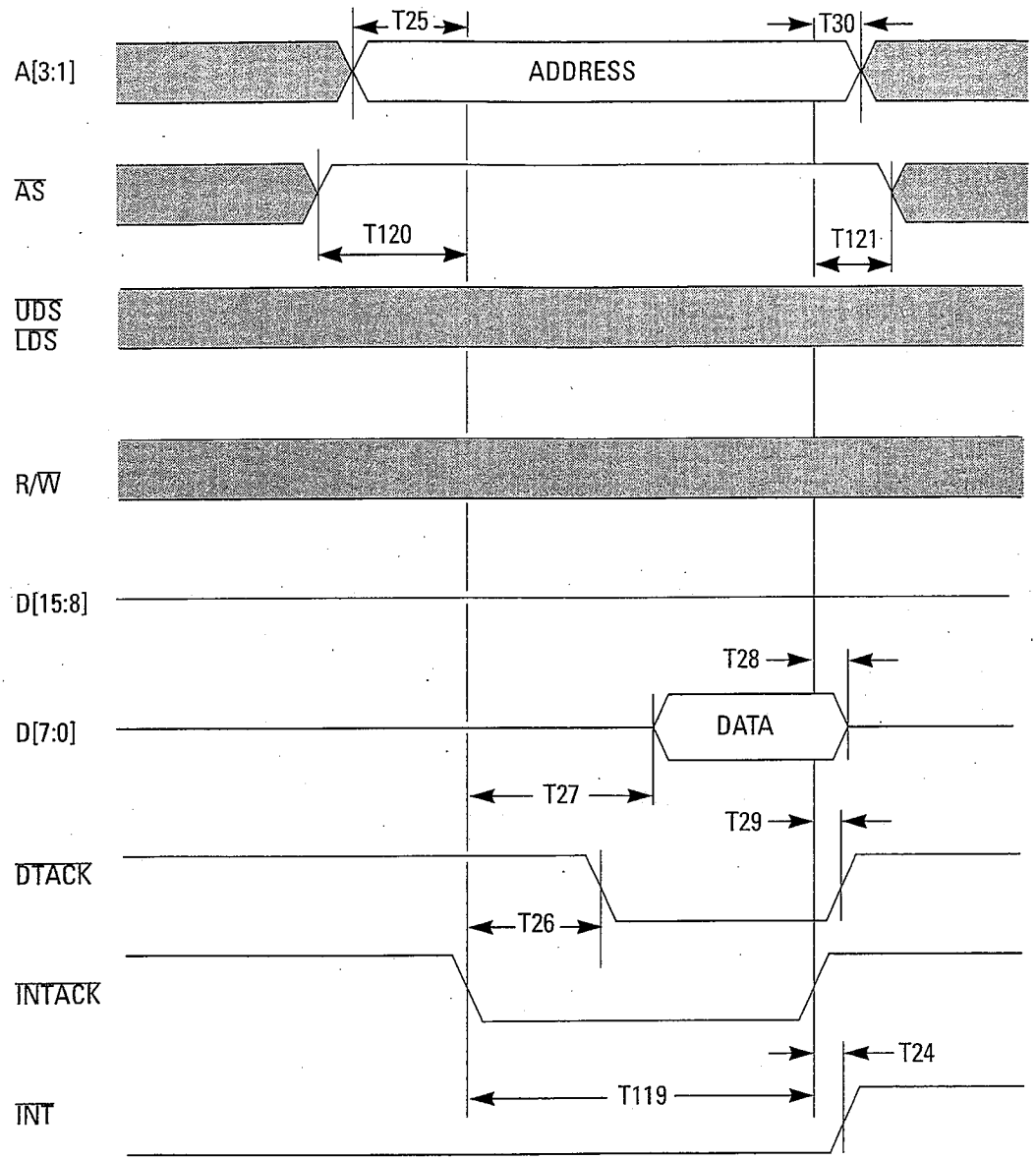
Table 7-5 Timing Characteristics - CMEM Access

Time	Description	Min	Max	Units
T11	$\overline{AS}$ to $\overline{DTACK}$ LOW (CMEM write)	TBD	30	ns
T17	D[15:0] setup to $\overline{DTC}$ LOW	5		ns
T18	D[15:0] hold from $\overline{DTC}$ LOW	10		ns
T19	$\overline{DTC}$ LOW Pulse Width	15		ns
T20	$\overline{DMAACK}$ hold to $\overline{DTC}$ LOW	10		ns
T21	$\overline{DMAACK}$ to $\overline{DMAREQ}$ HIGH (to end request)		20	ns
T22	$\overline{DONE}$ low to $\overline{DMAREQ}$ HIGH[1]		20	ns
T36	$\overline{DTACK}$ LOW to $\overline{AS}$ HIGH	0		ns
T47	$\overline{AS}$ HIGH to HCLK HIGH	10		ns
T109	D[15:0] hold time from $\overline{AS}$ + $\overline{UDS}$ + $\overline{LDS}$ LOW	10		ns
T110	$\overline{DMAACK}$ HIGH time	TBD		ns
T111	$\overline{DONE}$ pulse width	TBD		ns
T112	DMA cycle time	TBD(T31)		ns
T113	$\overline{AS}$ setup to $\overline{DMAACK}$ LOW	TBD		ns
T114	$\overline{AS}$ hold to $\overline{DMAACK}$ HIGH	TBD		ns
T115	R/$\overline{W}$ HIGH to $\overline{DMAACK}$ LOW	TBD		ns
T116	R/W hold to $\overline{DMAACK}$ HIGH	TBD		ns
T129	$\overline{AS}$ setup to $\overline{UDS}$/$\overline{LDS}$ LOW (CMEM write)	TBD		ns
T130	$\overline{AS}$ HIGH pulse width	TBD		ns
T131	$\overline{DMAACK}$ setup to $\overline{DTC}$ LOW	TBD		ns

1. Note: $\overline{DONE}$ causes $\overline{DMAREQ}$ to be come inactive only if CMEM is *not* full when the $\overline{DONE}$ pulse occurs.

7.2.3 Host Bus Vectored Interrupt Cycle Timing

The CL450 supports the 680X0 Vectored Interrupt Cycle. Shown is the Vectored Interrupt Cycle with the CL450's Auto Interrupt Clear enabled (*AIC* in HOST_control equal to 1, see page 8-10).



Note: INTACK should be tied HIGH if not used. When used, INTACK must not be asserted unless INT is asserted.

450-119

Figure 7-7 Timing Diagram - Vectored Interrupt Cycle with Auto Clear

Table 7-6 Timing Characteristics - Vectored Interrupt Cycle with Auto Clear[1]

Time	Description	Min	Max	Units
T24	$\overline{\text{INTACK}}$ HIGH to $\overline{\text{INT}}$ HIGH (release)		30	ns
T25	A[3:1] setup to $\overline{\text{INTACK}}$ LOW	10		ns
T26	$\overline{\text{INTACK}}$ LOW to $\overline{\text{DTACK}}$ LOW (vectored interrupt) [2]		30	ns
T27	$\overline{\text{INTACK}}$ LOW to D[7:0] valid (vectored interrupt) [3]	TBD	15	ns
T28	D[7:0] turn-off from $\overline{\text{INTACK}}$ HIGH (vectored interrupt) [3,4]	TBD	15	ns
T29	$\overline{\text{INTACK}}$ HIGH to $\overline{\text{DTACK}}$ HIGH (vectored interrupt) [2]		25	ns
T30	A[3:1] hold past $\overline{\text{INTACK}}$ HIGH	0		ns
T119	$\overline{\text{INTACK}}$ pulse width	TBD		ns
T120	$\overline{\text{AS}}$ setup time to $\overline{\text{INTACK}}$ LOW	TBD		ns
T121	$\overline{\text{AS}}$ hold time from $\overline{\text{INTACK}}$ HIGH	TBD		ns

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V. Output load capacitance = 50 pF.
2. Open-Drain Output. Timing specification assumes an external pull-up resistor with a value of 470 Ohms.
3. Not 100% tested, guaranteed by design characterization.
4. Time at which output achieves an open circuit condition; not referenced to an output voltage level.

7.2.4 Local DRAM Bus Timing

Local DRAM bus timing is specified differently than the other timing parameters. The DRAM interface times are specified at 0.8 Volts and 2.4 Volts, instead of at 1.5 Volts. This allows the designer to easily cross reference CL450 timing values to DRAM specifications. The Timing Characteristics table also gives the industry-standard DRAM timing parameter names as well as the CL450 parameter names.

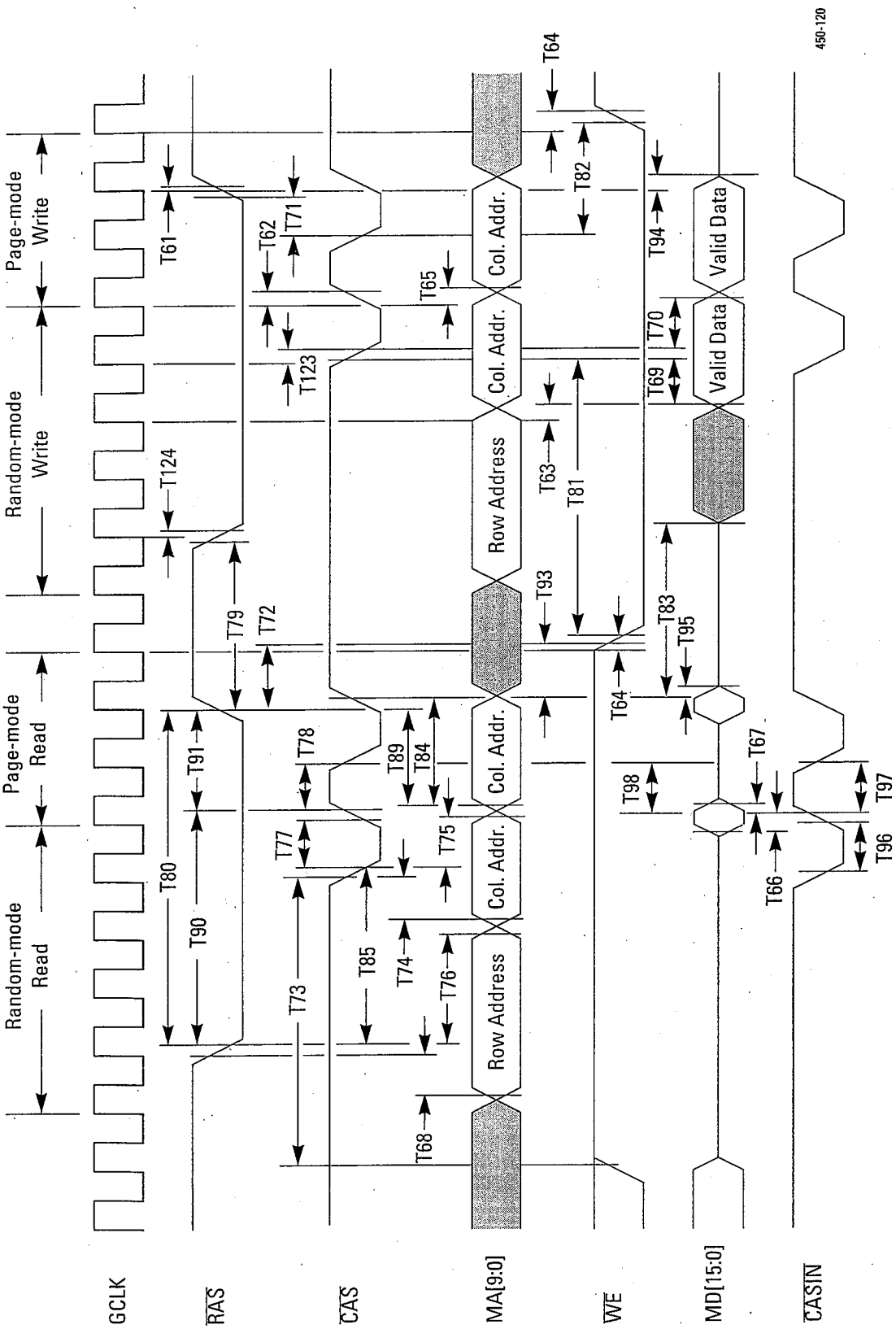


Figure 7-8 Timing Diagram - Local DRAM Bus Timing

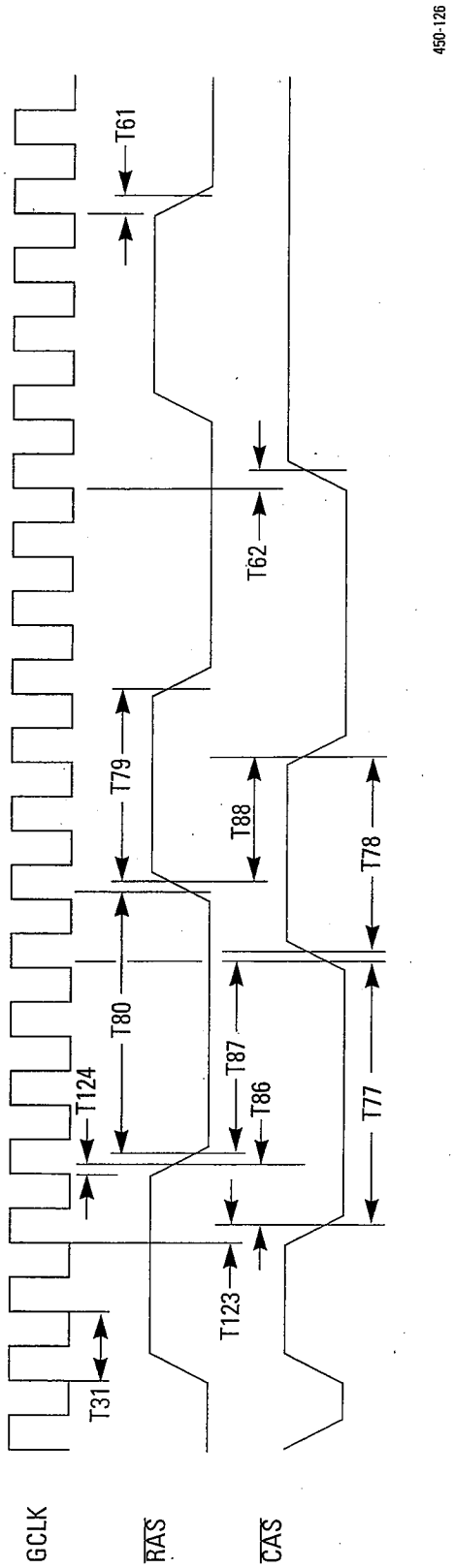


Figure 7-9 Timing Diagram - Local DRAM CAS-before-RAS Refresh

Table 7-7 Timing Characteristics - Local DRAM Bus[1]

Time	Description	Parameter	Min	Max	Units
T31	GCLK Period (tCYC)		25		ns
T61	GCLK HIGH to $\overline{RAS}$ HIGH [2]		TBD	30	ns
T62	GCLK HIGH to $\overline{CAS}$ HIGH [2]		TBD	25	ns
T63	GCLK HIGH to Memory Data HIGH or LOW (DRAM write) [2]		TBD	30	ns
T64	GCLK HIGH to $\overline{WE}$ HIGH or LOW [2]		TBD	22	ns
T65	GCLK HIGH to Memory Address HIGH or LOW [2]		TBD	24	ns
T66	Read Data Setup Time before $\overline{CASIN}$ HIGH		5		ns
T67	Read Data Hold Time after $\overline{CASIN}$ HIGH		5		ns
T68	Row Address Setup Time [2]	t_{ASR}	tCYC - 10		ns
T69	Write Data Setup Time before $\overline{CAS}$ LOW [2]	t_{DS}	tCYC - 15		ns
T70	Write Data Hold Time after $\overline{CAS}$ LOW [2]	t_{DH}	tCYC - 10		ns
T71	$\overline{RAS}$ Hold Time after $\overline{CAS}$ LOW [2]	t_{RSH}	tCYC - 5		ns
T72	Read Command Hold Time from $\overline{CAS}$ [2]	t_{RRH}	tCYC - 15		ns
T73	Read Command Setup Time to $\overline{CAS}$ LOW [2]	t_{RCS}	5*tCYC - 10		ns
T74	Column Address Setup Time to $\overline{CAS}$ LOW [2]	t_{ASC}	tCYC- 15		ns
T75	Column Address Hold Time from $\overline{CAS}$ LOW [2]	t_{CAH}	tCYC - 10		ns
T76	Row Address Hold Time from $\overline{RAS}$ LOW [2]	t_{RAH}	2*tCYC - 15		ns
T77	$\overline{CAS}$ LOW Time [2]	t_{CAS}	tCYC - 5		ns
T78	$\overline{CAS}$ HIGH Time [2]	t_{CP}	tCYC - 10		ns
T79	$\overline{RAS}$ HIGH Time [2]	t_{RP}	3*tCYC - 5		ns
T80	$\overline{RAS}$ LOW Time [2]	t_{RAS}	4*tCYC - 15	TBD	ns
T81	Write Command Setup Time to $\overline{CAS}$ LOW [2]	t_{WCS}	2*tCYC - 15		ns
T82	Write Command Hold Time from $\overline{CAS}$ LOW [2]	t_{WCH}	2*tCYC - 10		ns
T83	$\overline{CAS}$ HIGH to Memory Data Driven [2,3]		3*tCYC - 5		ns
T84	Column Address to $\overline{CAS}$ HIGH [2]	t_{AA} + T66	2*tCYC - 5		ns
T85	$\overline{RAS}$ to $\overline{CAS}$ delay [2]	t_{RCD}	3*tCYC - 15		ns
T86	$\overline{CAS}$ Setup Time to $\overline{RAS}$ (Memory Refresh Cycle) [2]	t_{CSR}	tCYC - 10		ns
T87	$\overline{CAS}$ Hold Time from $\overline{RAS}$ (Memory Refresh Cycle) [2]	t_{CHR}	3*tCYC - 15		ns
T88	$\overline{RAS}$ HIGH to $\overline{CAS}$ LOW delay (Memory Refresh Cycle) [2]	t_{RPC}	2*tCYC - 10		ns
T89	Column Address to $\overline{RAS}$ HIGH [2]	t_{RAL}	2*tCYC - 10		ns
T90	$\overline{RAS}$ LOW to $\overline{CAS}$ HIGH [2]	t_{RAC} + T66	4*tCYC - 15		ns
T91	$\overline{RAS}$ Hold Time from $\overline{CAS}$ Precharge [2]	t_{RHCP}	2*tCYC - 5		ns
T93	Read Command Hold Time from $\overline{RAS}$	t_{RRH},t_{RCH}	tCYC - 15		ns
T94	GCLK to MD three-state (write)		TBD	TBD	ns

1. Inputs switch between 0.0V and 3.5V at 1V/ns and measurements are made at 0.8V and 2.4V. Output load capacitance = 50 pF.
2. Not 100% tested, guaranteed by design characterization.
3. MD[15:0] driven only when next cycle will be a write.

Table 7-7 Timing Characteristics - Local DRAM Bus (cont.)

Time	Description	Parameter	Min	Max	Units
T95	$\overline{CAS}$ HIGH to MD three-state (read)		TBD	TBD	ns
T96	$\overline{CASIN}$ LOW Time		TBD		ns
T97	$\overline{CASIN}$ HIGH TIME		TBD		ns
T98	$\overline{CASIN}$ HIGH to $\overline{CAS}$ LOW		TBD		ns
T123	GCLK HIGH to $\overline{CAS}$ LOW		TBD	TBD	ns
T124	GCLK HIGH to $\overline{RAS}$ LOW		TBD	TBD	ns

7.2.5 GCLK, SCLK, and $\overline{RESET}$ Timing

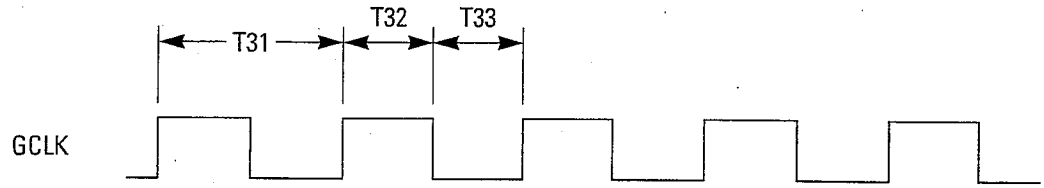


Figure 7-10 GCLK Timing Diagram

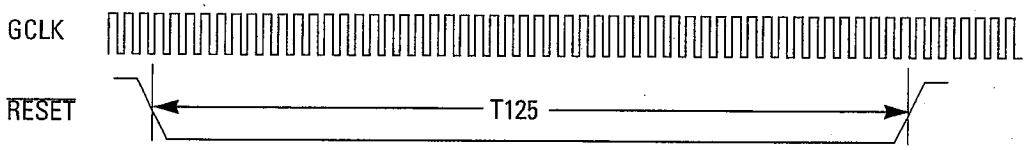


Figure 7-11 $\overline{RESET}$ Timing Diagram

Table 7-8 Timing Characteristics - GCLK and $\overline{RESET}$ [1]

Time	Description	Min	Max	Units
T31	GCLK frequency		40	MHz
T31	GCLK Period (tCYC)	25		ns
T32	GCLK HIGH Pulse Width	10		ns
T33	GCLK LOW Pulse Width	10		ns
T125	$\overline{RESET}$ Pulse Width	50(T31)		ns

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V.

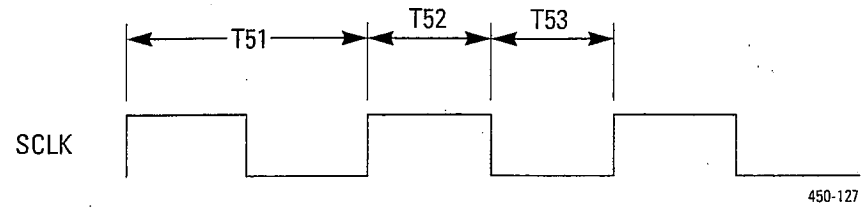


Figure 7-12 Timing Diagram - SCLK Input

Table 7-9 Timing Characteristics - SCLK Input [1]

Time	Description	Min	Max	Units
T51	SCLK Period [2]	2(T31)		ns
T52	SCLK HIGH Pulse Width [2]	T31		ns
T53	SCLK LOW Pulse Width [2]	T31		ns

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V.
2. Not 100% tested, guaranteed by design characterization

7.2.6 Video Bus Timing

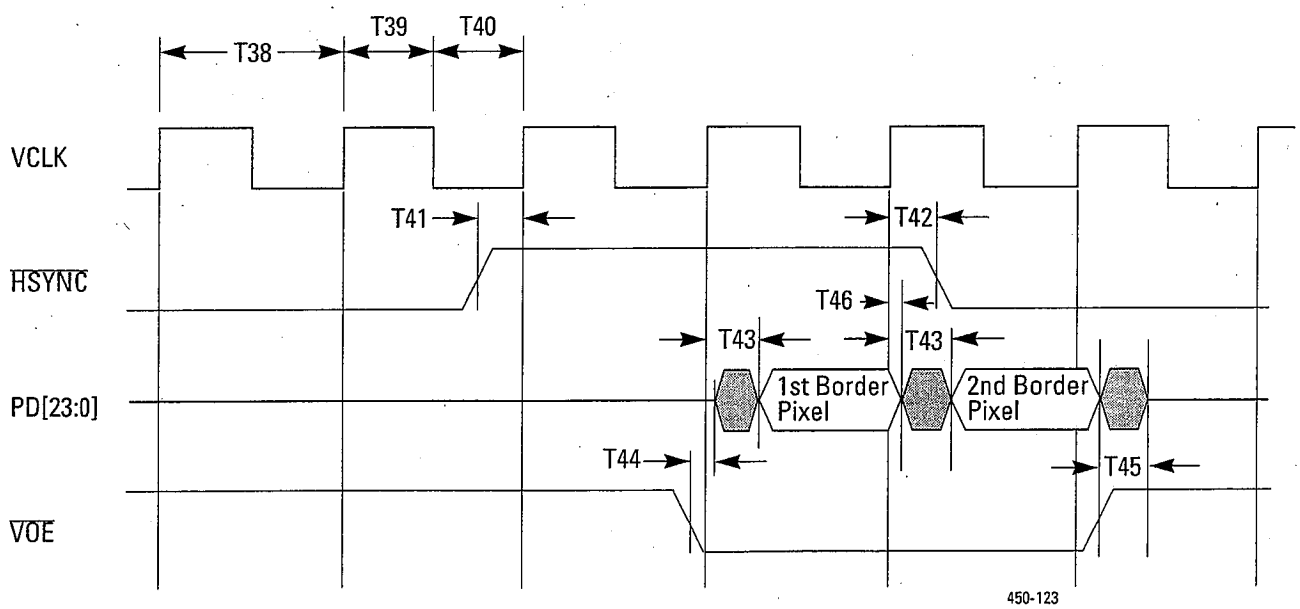


Figure 7-13 Timing Diagram - Video Bus Inputs

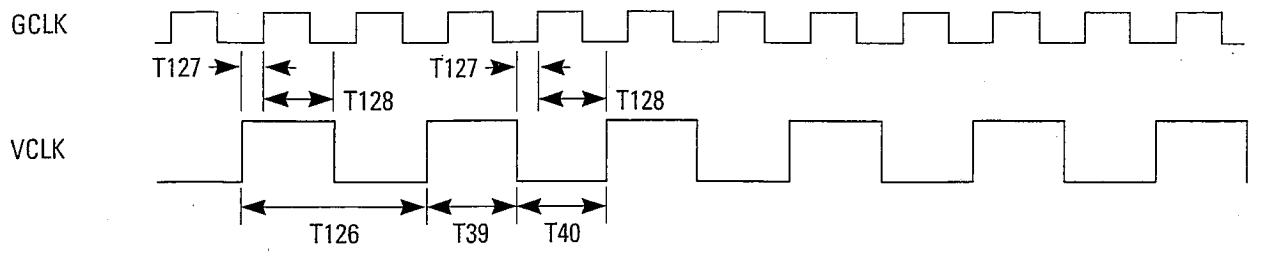


Figure 7-14 VCLK Timing (20 MHz only)

Table 7-10 Timing Characteristics - Video Bus Inputs [1]

Time	Description	Min	Max	Units
T38	VCLK Period[2]	2(T31)+17		ns
T39	VCLK HIGH Pulse Width	20		ns
T40	VCLK LOW Pulse Width	20		ns
T41	$\overline{\text{HSYNC}}$ setup to VCLK HIGH	10		ns
T42	$\overline{\text{HSYNC}}$ hold from VCLK HIGH	0		ns
T43	PD valid from VCLK HIGH		20	ns
T44	$\overline{\text{VOE}}$ LOW to PD[23:0] driven	TBD	20	ns
T45	PD [23:0] turn-off time from $\overline{\text{VOE}}$ HIGH [3,4]	TBD	20	ns
T46	PD hold time from VCLK HIGH	TBD		ns
T126	Alternate VCLK period	2(T31)	2(T31)	ns
T127	VCLK setup to GCLK[5]	TBD		
T128	VCLK hold from GCLK[5]	TBD		

1. Inputs switch between 0.0V and 3.5V at 1V/ns. Measurements are made at 1.5V. Output load capacitance = 50pF.
2. Note: T126 may be substituted for T38 if additional timing constraints in Figure 7-14 (T127 and T128) are met.
3. Not 100% tested, guaranteed by design characterization.
4. Time at which output achieves an open circuit condition; not referenced to an output voltage level.
5. Must be met only when T126 is used in place of T38.

7.3 Package Specifications

The CL450 is packaged in a plastic quad flat pack (PQFP). The package has an internal heat spreader and a copper-alloy lead frame to improve thermal conductivity. This section includes:

- The CL450 Pinout Diagram
- Tables of CL450 pin connections
  - Host Bus Interface Pins
  - DRAM Bus Interface Pins
  - Video Bus Interface Pins
  - Power and Miscellaneous Pins
- Package Physical Dimensions

The CL450 is shipped in a drypack with desiccant and a humidity monitor. Do not use the parts if the humidity indicator indicates that the humidity is greater than 30% at the initial opening of the drypack. To avoid cracking the plastic during soldering, the parts should be soldered within three days after breaking the drypack seal.

Note the following conventions used in this section:

- "NO CONNECT" pins are not connected internally.
- "RESERVED" pins are reserved for future use. They should be pulled either HIGH or LOW.

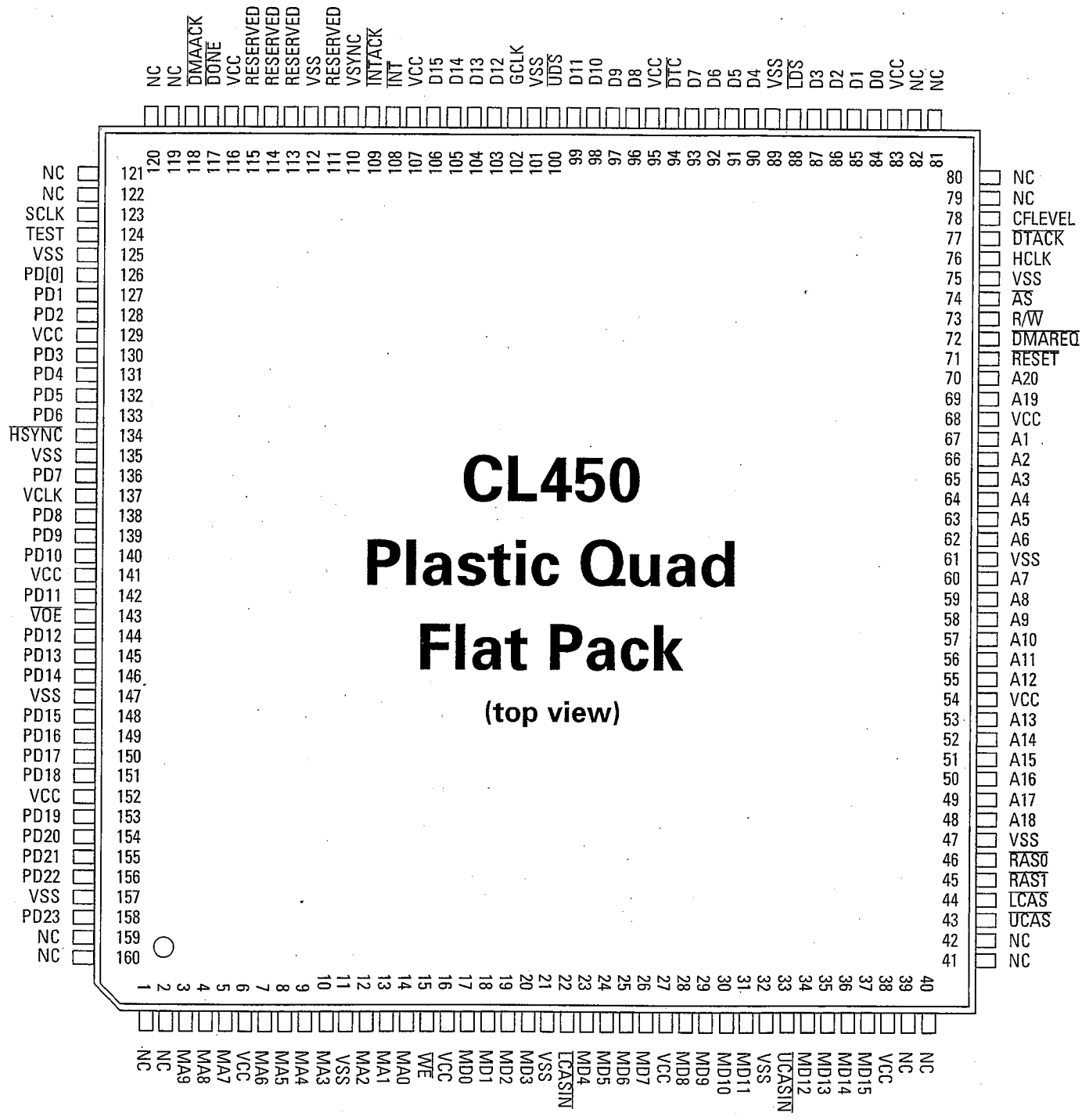


Figure 7-15 CL450 PQFP Pinout Diagram

7.3.7 Pin List

Table 7-11 Host Bus Interface Pins

Function	Group	I/O	Pin #	Function	Group	I/O	Pin #
GCLK	Host	I	102	A2	Host	I	66
HCLK	Host	I	76	A1	Host	I	67
SCLK	Host	I	123	R/$\overline{W}$	Host	I	73
$\overline{INT}$ [1]	Host	O	108	D15	Host	I/O	106
$\overline{INTACK}$	Host	I	109	D14	Host	I/O	105
CFLEVEL	Host	O	78	D[13	Host	I/O	104
$\overline{RESET}$	Host	I	71	D12	Host	I/O	103
TEST	Host	I	124	D11	Host	I/O	99
$\overline{AS}$	Host	I	74	D10	Host	I/O	98
A20	Host	I	70	D9	Host	I/O	97
A19	Host	I	69	D8	Host	I/O	96
A18	Host	I	48	D7	Host	I/O	93
A17	Host	I	49	D6	Host	I/O	92
A16	Host	I	50	D5	Host	I/O	91
A15	Host	I	51	D4	Host	I/O	90
A14	Host	I	52	D3	Host	I/O	87
A13	Host	I	53	D2	Host	I/O	86
A12	Host	I	55	D1	Host	I/O	85
A11	Host	I	56	D0	Host	I/O	84
A10	Host	I	57	$\overline{DTACK}$ [1]	Host	O	77
A9	Host	I	58	$\overline{LDS}$	Host	I	88
A8	Host	I	59	$\overline{UDS}$	Host	I	100
A7	Host	I	60	$\overline{DMAREQ}$	Host	O	72
A6	Host	I	62	$\overline{DMAACK}$	Host	I	118
A5	Host	I	63	$\overline{DTC}$	Host	I	94
A4	Host	I	64	$\overline{DONE}$	Host	I	117
A3	Host	I	65				

1. Open-Drain Output.

Table 7-12 DRAM Bus Interface Pins

Function	Group	I/O	Pin #	Function	Group	I/O	Pin #
MA9	DRAM	O	3	MD8	DRAM	I/O	28
MA8	DRAM	O	4	MD7	DRAM	I/O	26
MA7	DRAM	O	5	MD6	DRAM	I/O	25
MA6	DRAM	O	7	MD5	DRAM	I/O	24
MA5	DRAM	O	8	MD4	DRAM	I/O	23
MA4	DRAM	O	9	MD3	DRAM	I/O	20
MA3	DRAM	O	10	MD2	DRAM	I/O	19
MA2	DRAM	O	12	MD1	DRAM	I/O	18
MA1	DRAM	O	13	MD0	DRAM	I/O	17
MA0	DRAM	O	14	$\overline{\text{RAS}}$[0]	DRAM	O	46
MD15	DRAM	I/O	37	$\overline{\text{RAS}}$[1]	DRAM	O	45
MD14	DRAM	I/O	36	$\overline{\text{LCAS}}$	DRAM	O	44
MD13	DRAM	I/O	35	$\overline{\text{UCAS}}$	DRAM	O	43
MD12	DRAM	I/O	34	$\overline{\text{LCASIN}}$	DRAM	I	22
MD11	DRAM	I/O	31	$\overline{\text{UCASIN}}$	DRAM	I	33
MD10	DRAM	I/O	30	$\overline{\text{WE}}$	DRAM	O	15
MD9	DRAM	I/O	29				

Table 7-13 Video Bus Interface Pins

Function	Group	I/O	Pin #	Function	Group	I/O	Pin #
PD23	Video	O	158	PD9	Video	O	139
PD22	Video	O	156	PD8	Video	O	138
PD21	Video	O	155	PD7	Video	O	136
PD20	Video	O	154	PD6	Video	O	133
PD19	Video	O	153	PD5	Video	O	132
PD18	Video	O	151	PD4	Video	O	131
PD17	Video	O	150	PD3	Video	O	130
PD16	Video	O	149	PD2	Video	O	128
PD15	Video	O	148	PD1	Video	O	127
PD14	Video	O	146	PD0	Video	O	126
PD13	Video	O	145	$\overline{\text{HSYNC}}$	Video	I	134
PD12	Video	O	144	$\overline{\text{VOE}}$	Video	I	143
PD11	Video	O	142	VCLK	Video	I	137
PD10	Video	O	140	VSYNC	Video	I	110

Table 7-14 Power and Miscellaneous Pins

Function	Group	I/O	Pin #	Function	Group	I/O	Pin #
VCC	Power	PWR	6	VSS	Power	GND	32
VCC	Power	PWR	16	VSS	Power	GND	47
VCC	Power	PWR	27	VSS	Power	GND	61
VCC	Power	PWR	38	VSS	Power	GND	75
VCC	Power	PWR	54	VSS	Power	GND	89
VCC	Power	PWR	68	VSS	Power	GND	101
VCC	Power	PWR	83	VSS	Power	GND	112
VCC	Power	PWR	95	VSS	Power	GND	125
VCC	Power	PWR	107	VSS	Power	GND	135
VCC	Power	PWR	116	VSS	Power	GND	147
VCC	Power	PWR	129	VSS	Power	GND	157
VCC	Power	PWR	141	NO CONNECT	Misc	-	1 - 2
VCC	Power	PWR	152	NO CONNECT	Misc	-	39 - 42
RESERVED	Misc	I	111	NO CONNECT	Misc	-	79- 82
RESERVED	Misc	I	113 - 115	NO CONNECT	Misc	-	119 -122
VSS	Power	GND	11	NO CONNECT	Misc	-	159 - 160
VSS	Power	GND	21				

7.3.8 Package Drawings

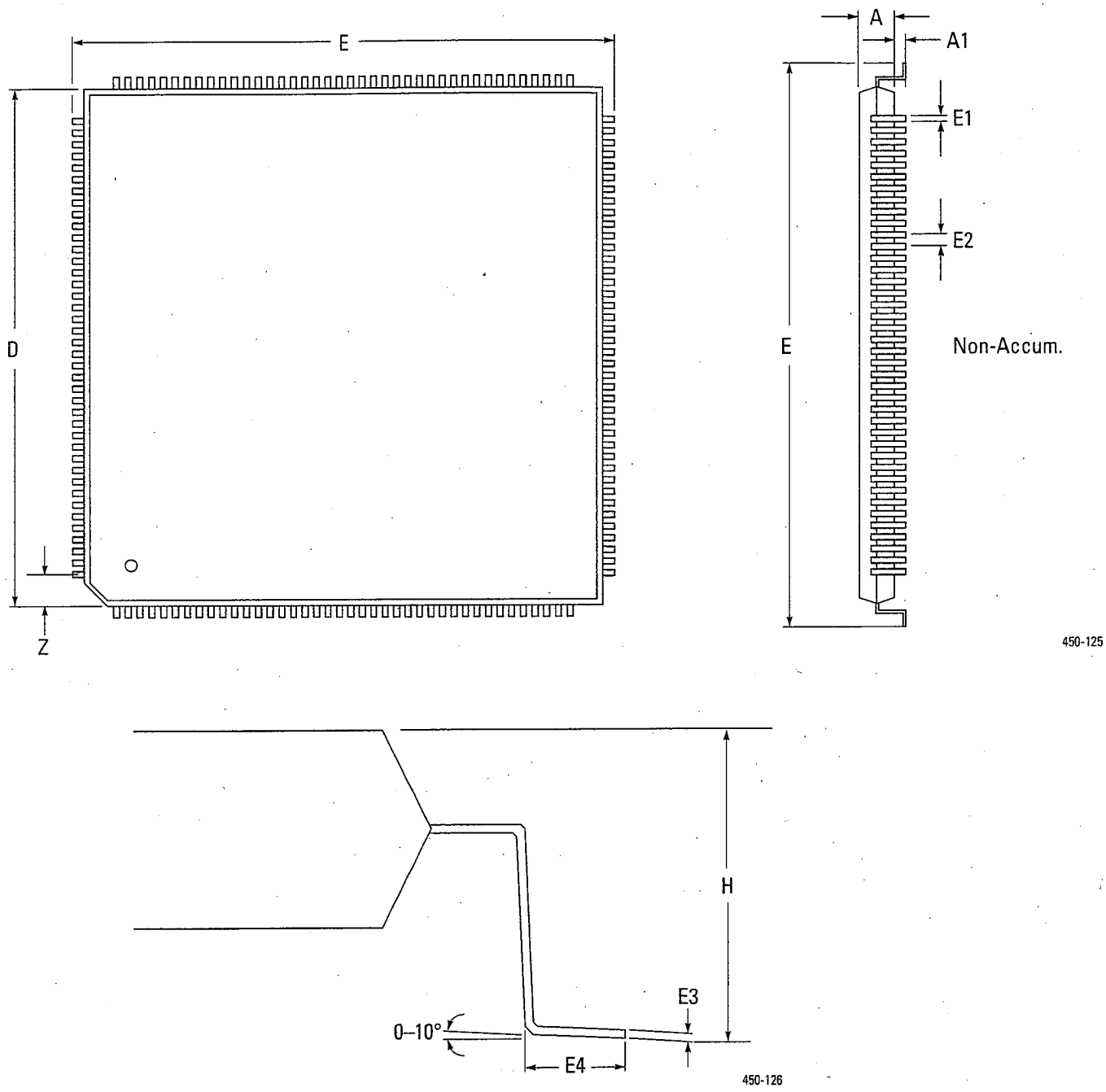


Figure 7-16 Plastic Quad Flat Pack Physical Dimensions

Table 7-15 Plastic Quad Flat Pack Physical Dimensions

SYMBOL	DIMENSIONS	
	INCHES	MM
A	0.146 MAX.	3.7 MAX.
A1	0 MIN.	0 MIN.
D	1.10 ±0.008	28.0 ±0.20
E	1.259 ±0.016	32.0 ±0.40
E1	0.012 TYP.	0.30 TYP.
E2	0.0256 TYP.	0.65 TYP.
E3	0.006 TYP.	0.15 TYP.
E4	0.031 ±0.008	0.80 ±0.20
H	0.157 MAX.	4.0 MAX.
Z	0.052 TYP.	1.33 TYP.

8
Registers

This chapter describes each of the CL450 hardware registers. You should be familiar with the relation of these registers to the CL450's external signals as described in Chapter 3, Signal Descriptions.

The sections in this chapter are:

- 8.1: CL450 Register Categories
- 8.2: CL450 Register Summary
- 8.3: Host Interface Registers
- 8.4: Internal CPU Registers
- 8.5: DRAM Interface Register
- 8.6: Video Interface Registers

8.1 CL450 Register Categories

The CL450 contains 34 registers. Typically, these registers perform low-level operations or act as part of the protocol for communicating with the microapplication executing on the CL450's internal CPU. For purposes of this reference, they are divided into three categories:

- *Normal*: These registers are used routinely to configure/communicate with the CL450 and its microapplication, although not all applications use all of these registers.
- *Diagnostic*: These registers are used *only* when attempting to characterize a problem as discussed in Appendix E, Troubleshooting. They are accessed by the host using an internal data bus which is shared with the CL450's internal CPU. Because host use of this bus may conflict with CL450 operations, diagnostic register accesses should be minimized to prevent degradation of CL450 performance.
- *Internal*: These registers are used exclusively by the CL450's internal CPU. They must not be accessed by the host any time the microapplication is executing; otherwise, indeterminate behavior can occur.
- *Initialization*: These registers should only be accessed by the host prior to the beginning of microapplication execution as part of the chip initialization procedure.

8.1.1 Direct-access Registers

Of the CL450's 34 registers, 26 can be directly accessed using a single transfer across the CL450's host interface and are listed in Table 8-1. To access these registers, perform a host read or write with address bits A[20:19] equal to 10_2 and address bits A[7:1] set to the register address.

Note: Register address values in this chapter are given in byte-wide address space. This is done for the convenience of programmers working with byte-addressable processors.

The least significant bit of each address value corresponds to address bit A[0] (which is not supplied to the CL450), and the remaining register address bits are driven to the CL450 on address lines A[7:1].

Even though register addresses are expressed as byte addresses, the CL450 does not support single-byte access to internal registers; all registers must be accessed 16 bits at a time.

Register address values not listed in Table 8-1 are reserved. Attempting to read or write these register addresses—or to read a write-only register, or to write a read-only register—causes indeterminate results.

Table 8-1 Direct-access Registers (listed by category)

Category	Register Name	Address (byte-wide)	Category	Register Name	Address (byte-wide)
	CMEM_control	0x80		CPU_int	0x54
	CMEM_dmactrl	0x84	Diagnostic	CPU_intenb	0x26
	CPU_control	0x20		CPU_taddr	0x38
	CPU_iaddr	0x3E		CPU_tmem	0x46
	CPU_imem	0x42		CMEM_data	0x02
	CPU_pc	0x22		CMEM_status	0x82
	DRAM_refcnt	0xAC		HOST_scr0	0x92
Normal	HOST_control	0x90	Internal	HOST_scr1	0x94
	HOST_intvecr	0x9C		HOST_scr2	0x96
	HOST_intvecw	0x98		VID_chroma	0x0A
	HOST_newcmd	0x56		VID_y	0x00
	HOST_raddr	0x88			
	HOST_rdata	0x8C			
	VID_control	0xEC			
	VID_regdata	0xEE			

Note that Table 8-1 gives addresses within an 8-bit address space; that is, the values given should be divided by two before being presented to the CL450 using address pins A[7:1]. This restriction applies because CL450 registers must be accessed using 16-bit operations.

8.1.2 Indirect-access (Video) Registers

The remaining eight registers, listed in Table 8-2, are accessed indirectly by specifying the desired register number in bits 4:1 of the VID_control register (the *VRID* field), and then reading or writing the VID_regdata register. Note that the "*VRID* Value" of Table 8-2 replaces the "Address" column of Table 8-1 but that the tables are otherwise identical.

The procedure for accessing an indirect video register is described in Section 8.6.2, Indirect-access Video Registers.

Table 8-2 Indirect-access (Video) Registers (listed by category)

Category	Register Name	*VRID* Value
Normal	VID_selaux	0xC
Internal	VID_sela	0x0
	VID_selactive	0x8
	VID_selb	0x1
	VID_selbor	0x9
	VID_selGB	0xB
	VID_selmode	0x7
	VID_selR	0xA

Note: All register addresses and VRID values not shown in Table 8-1 and Table 8-2 are RESERVED, and attempting to access a register at a reserved location will cause indeterminate results. Also, not all registers may be accessed at all times during CL450 operation. The limitations on when registers may be accessed are discussed in Chapter 10.

8.2 CL450 Register Summary

Table 8-3 and Table 8-4 give a descriptive summary of the same CL450 registers that were listed in Tables 8-1 and 8-2, respectively. Note that the registers in Table 8-3 that begin "CMEM_" are part of the host interface module, and that 8 of the 12 video registers are listed separately in Table 8-4 because they are addressed *indirectly* using the VID_control register as a pointer.

Table 8-3 Direct-access Registers (described)

Register	Addr.	Used For:	R/W	Page #
CMEM_control	0x80	Internal reset, byte swap (normal/initialization)	R/W	8-8
CMEM_data	0x02	CMEM read port (internal)	R	8-9
CMEM_dmactrl	0x84	CMEM status, CFLEVEL assertion, DMA enable (normal)	R/W	8-9
CMEM_status	0x82	CMEM counters (internal)	R	8-11
CPU_control	0x20	Run enable bit (normal)	R/W	8-19
CPU_iaddr	0x3E	IMEM write address (normal/initialization)	R/W	8-21
CPU_imem	0x42	IMEM write data (normal/initialization)	W	8-21
CPU_int	0x54	Internal interrupt status (diagnostic)	R/W	8-19
CPU_intenb	0x26	Internal interrupt enable (diagnostic)	R/W	8-20
CPU_pc	0x22	Program counter (normal/initialization)	R/W	8-19
CPU_taddr	0x38	TMEM address (diagnostic)	R/W	8-22
CPU_tmem	0x46	TMEM data (diagnostic)	R/W	8-22
DRAM_refcnt	0xAC	Refresh clock count (normal)	R/W	8-23
HOST_control	0x90	Interrupt control (normal)	R/W	8-12
HOST_intvecr	0x9C	Read *IVect* and *IPID* (normal)	R	8-13
HOST_intvecw	0x98	Write *IVect* and *IPID* (normal)	W	8-13
HOST_newcmd	0x56	New command bit (normal)	R/W	8-14
HOST_raddr	0x88	Pointer to command/status register (normal)	R/W	8-13
HOST_rdata	0x8C	Data port to command/status register (normal)	R/W	8-14
HOST_scr0	0x92	SCR – lower portion (internal)	R/W	8-17
HOST_scr1	0x94	SCR – middle portion (internal)	R/W	8-16
HOST_scr2	0x96	SCR – upper portion (internal)	R/W	8-16
VID_control	0xEC	Pointer to indirect video register (normal/initialization)	R/W	8-25
VID_regdata	0xEE	Data port for indirect video registers (normal/initialization)	W	8-25
VID_chroma	0x0A	Chrominance data port (internal)	W	8-25
VID_y	0x00	Luminance data port (internal)	W	8-26

Table 8-4 Indirect-access (Video) Registers (described)

Register	*VRID*	Used For:	R/W	Page #
VID_sela	0x0	Conversion coefficients (internal)	W	8-26
VID_selactive	0x8	Width of active region (internal)	W	8-28
VID_selaux	0xC	Pixel data high byte (normal/initialization)	W	8-29
VID_selb	0x1	Conversion coefficients (internal)	W	8-26
VID_selbor	0x9	Left border size (internal)	W	8-28
VID_selGB	0xB	Border green, blue values (internal)	W	8-29
VID_selmode	0x7	RGB or YCbCr mode (internal)	W	8-27
VID_selR	0xA	Border red value (internal)	W	8-28

Figure 8-1 shows a conceptual model of the CL450's four logical register interfaces. This model does not correspond exactly to the physical structure of the device; instead, it has been simplified to show the programmer's interface.

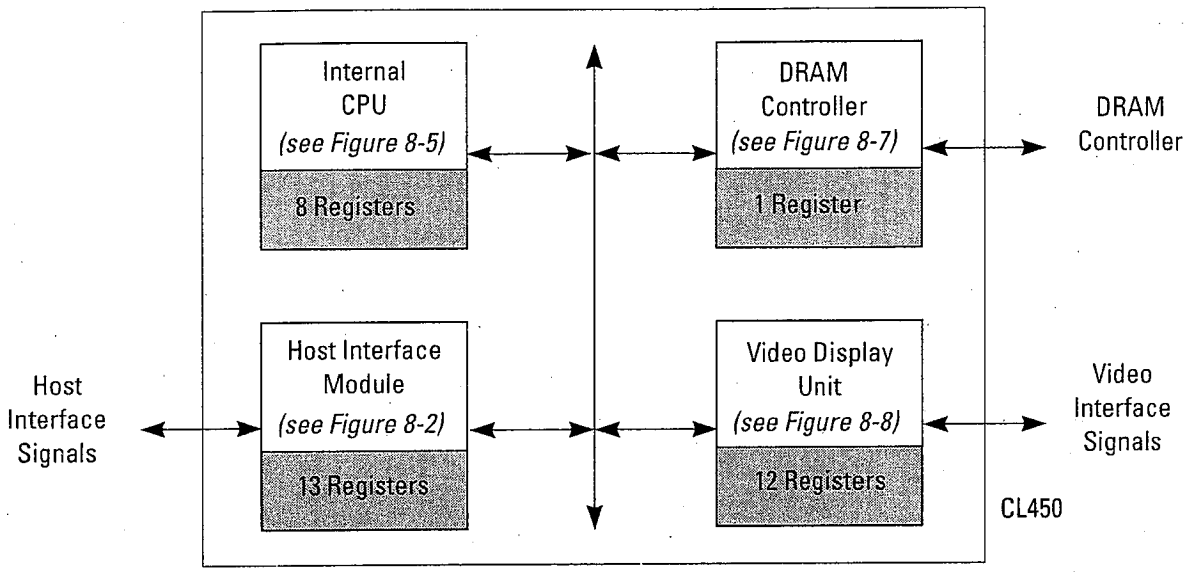


Figure 8-1 Internal Register Structure of the CL450

The sections that follow describe the registers contained in each CL450 interface module. In the detailed definitions given for each register, note that bits marked *Res* are reserved. Reserved bits should be written as zeros. Reading from reserved bits gives undefined data.

8.3 Host Interface Registers

This section describes the registers in the host interface module. Figure 8-2 shows the internal structure of the host interface module.

Note: Registers accessed by the host that are not contained on the host interface module are accessed through the host interface logic that connects across the CL450 internal bus.

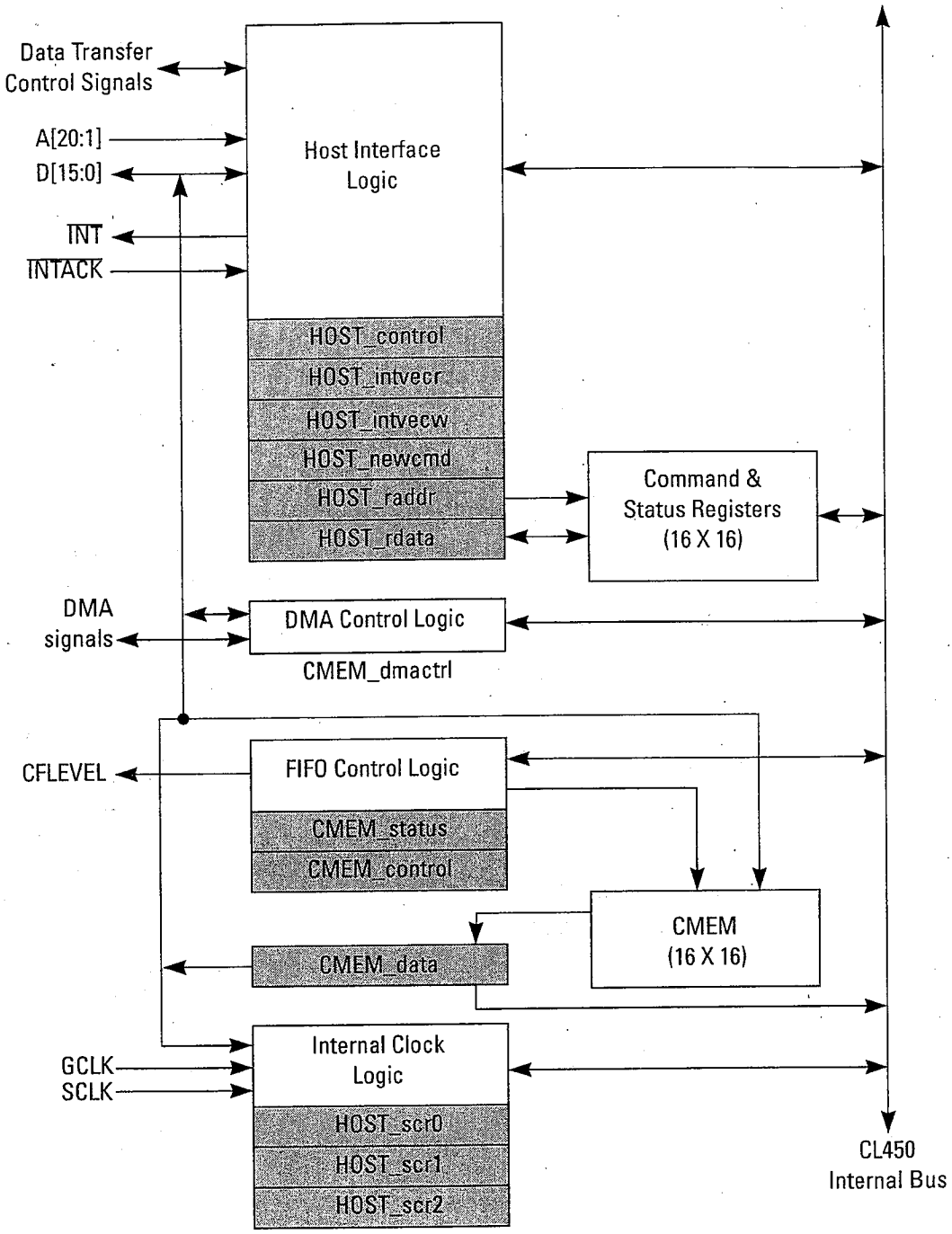


Figure 8-2 Internal Structure of the Host Interface Module

8.3.1 CMEM Registers

The host processor uses these registers to access CMEM for initialization, control, monitoring, and diagnostic purposes.

CMEM_control **(initialization)** **0x80**

15	7	6	5	4	3	2	1	0
Res		*Rst*	*BS*	*Res*		*CR*	*CRE*	*CRst*

Rst **Internal Reset (bit 6)** **R/W**

The host processor writes a 1 to this bit to cause the CL450 to perform a software reset. A software reset initializes the CL450 in the same way as the hardware reset described in Chapter 4, with the exception that the host interface remains active to allow the host processor to clear this bit to 0. The host processor must clear this bit before initiating another software reset; *Rst* is not cleared automatically. After any reset, the CL450 must be re-initialized by the host.

BS **Byte Swap (bit 5)** **R/W**

When *BS* is 1, words of data written to the CL450's CMEM are byte-swapped. (Data written to other internal registers in the CL450 are not swapped regardless of the setting of this bit.)

The byte-swapping capability is provided to ensure compatibility between the CL450, which uses the Motorola style of byte ordering, and host processors that use the Intel style. In the Motorola style, a word address points to the more significant byte (MSB); the less significant byte (LSB) is located at the next higher address. In the Intel style, the byte ordering is reversed. Figure 8-3 shows the memory organization for the two styles.

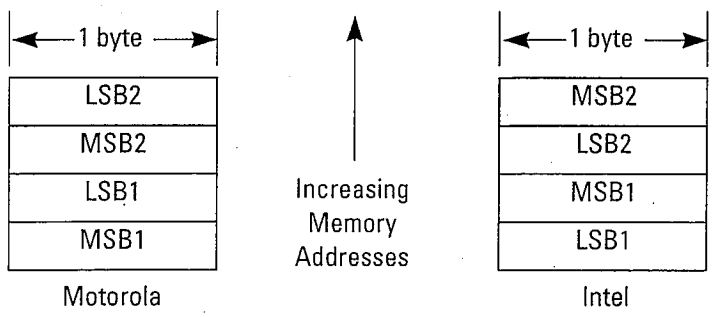


Figure 8-3 Motorola and Intel Byte Ordering

CR **CMEM Request (bit 2)** **R**

When *CRE* (below) is 1, the CL450 sets *CR* to 1 when CMEM contains four or more entries, and clears *CR* to zero when CMEM contains one or no entries. The CL450's internal DRAM controller transfers data from CMEM to the bitstream buffer in the local DRAM when *CR* is 1.

CRE **CMEM Request Enable (bit 1)** **R/W**

When *CRE* is set to 1, CMEM requests a DRAM transfer to the bitstream buffer using *CR* as described above. The microapplication sets this bit to 1 during initialization. When *CRE* is set to 0, the DRAM controller is instructed not to consume any data.

CRst **CMEM Reset (bit 0)** **R/W**

When a 1 is written to *CRst*, the counters in the CMEM_status register are reset to 0, and any data stored in CMEM is deleted. *CRst* must be 0 for normal operation.

(internal) **0x02** **CMEM_data**

15		0
	CData	

CData **CMEM Data (bits 15:0)** **R**

This 16-bit field is the read port from CMEM. The CMEM Read Counter (*CRCnt*) in the CMEM_status register points to the read address in CMEM.

(normal) **0x84** **CMEM_dmactrl**

15	9	8	7	6	5	4	3	2	1	0
Res		*1Q*	*2Q*	*3Q*	*4Q*	*1QE*	*2QE*	*3QE*	*4QE*	*DE*

1Q – 4Q **Empty Status (bits 8:5)** **R**

These four bits indicate the empty status of CMEM as shown in Table 8-5. When using programmed accesses to write data to CMEM, the application program must check this register or the CFLEVEL pin before writing to ensure that CMEM does not overflow. Note that each of bits [8:5] has a corresponding bit in the range [4:1] which is used as described below.

Table 8-5 CMEM Empty Status Bits

1Q	*2Q*	*3Q*	*4Q*	Empty Entries in CMEM	Status
0	0	0	0	0 – 3	Full/nearly full
1	0	0	0	4 – 7	1/4+ empty
1	1	0	0	8 – 11	1/2+ empty
1	1	1	0	12 – 15	3/4+ empty
1	1	1	1	16	Empty

1QE – 4QE **Empty Status Enables (bits 4:1)** **R/W**

These four bits enable the assertion of the CL450's output signal CFLEVEL based on the state of the empty status bits *1Q* through *4Q* described above. Table 8-6 shows how these bits control the assertion of CFLEVEL.

Table 8-6 CFLEVEL Assertion Control

1QE	*2QE*	*3QE*	*4QE*	CFLEVEL Asserted Upon Assertion of Bit:
1	X	X	X	*1Q*
0	1	X	X	*2Q*
0	0	1	X	*3Q*
0	0	0	1	*4Q*[1]
0	0	0	0	CFLEVEL Not Asserted

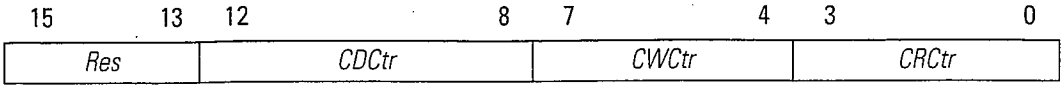
1. Note that this bit will not necessarily become 1 during normal operations.

DE **DMA Enable (bit 0)** **R/W**

When this bit is written with 1 by the host and CMEM is *not* full, the CL450's DMA capability is enabled. Writing a 1 to this bit if CMEM *is* full will not enable DMA. Writing a 0 to this bit always disables DMA, regardless of the CMEM fullness level. The value read from this bit will always be the last value written (or 0 if cleared by the $\overline{\text{DONE}}$ pin), and does not necessarily reflect whether or not DMA is enabled.

When DMA is enabled, the CL450 generates a DMA request by asserting $\overline{\text{DMAREQ}}$ (active low) whenever CMEM has at least one empty space. Note that this bit is automatically cleared to 0 by the CL450 when the $\overline{\text{DONE}}$ pin is asserted by the host and CMEM is *not* full, as described in Section 4.5, CMEM Write Timing.

(internal) **0x82** **CMEM_status**



CDCtr **Difference Counter (bits 12:8)** **R**

This five-bit field contains the count of the current number of words in CMEM (from 0 to 16). *CDCtr* is incremented when a word is written to CMEM (from the host using programmed access or DMA) and decremented when a word is read from CMEM.

CWCtr **Write Counter (bits 7:4)** **R**

This four-bit field contains the CMEM write pointer. *CWCtr* is incremented when a word is written to CMEM from the host using programmed access or DMA.

CRCtr **Read Counter (bits 3:0)** **R**

This four-bit field contains the CMEM read pointer. *CRCtr* is incremented when a word is read from CMEM to the bitstream buffer in the CL450's local DRAM.

8.3.2 Interrupt Control Registers

The HOST_control, HOST_intvecr, and HOST_intvecw registers control the CL450's generation of external interrupts. The HOST_control register determines the interrupt mode and the state of the $\overline{\text{INT}}$ pin, and the other two registers provide access to the interrupt vector used when $\overline{\text{INTACK}}$ is asserted (LOW).

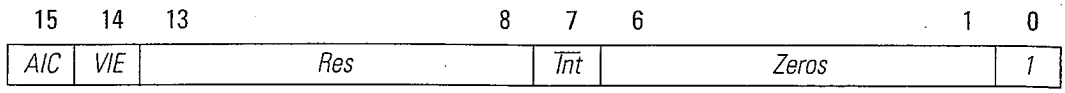
The interrupt modes supported by the CL450 are:

- Non-vectored interrupt
- Vectored interrupt with no auto clear
- Vectored interrupt with auto clear

At device initialization, the host is responsible for ensuring that the *AIC* and *VIE* bits in the HOST_control register have the correct values for the desired operations (see Table 8-7) before the CL450 microapplication attempts to issue the first interrupt to the host. Subsequent writes to the register to reset $\overline{Int}$ must use a read-modify-write operation to preserve the value of the other bits in the HOST_control register.

Note: Whether the CL450 produces interrupts to the host, and what events cause these interrupts, is controlled by the CL450 microapplication. See Chapter 12, Interrupts, for a description of the communications protocol between the host and the microapplication for selecting and servicing interrupts.

HOST_control (normal) 0x90



AIC Auto Interrupt Clear (bit 15) R/W

When this bit is 1, the CL450 sets the *Int* bit to 1, deactivating the interrupt request upon completion of the interrupt acknowledge cycle (i.e., reception of a rising edge on the INTACK pin). If there are no transitions on the INTACK pin, or the *VIE* bit (mentioned next) is 0, the value of this bit has no effect on the operation of the CL450.

VIE Vectored Interrupt Enable (bit 14) R/W

When this bit is 1, the CL450 will respond to the interrupt acknowledge signal, INTACK. If this bit is 0, the CL450 ignores all activity on the INTACK pin and behaves as if INTACK were always inactive (HIGH).

Table 8-7 Interrupt Mode Values

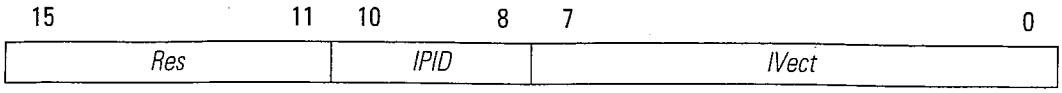
Interrupt Mode	*AIC*	*VIE*
Non-vectored interrupt	0	0
Vectored interrupt without auto clear	0	1
Vectored interrupt with auto clear	1	1

Int Interrupt Status (bit 7) R/W

This bit controls the external interrupt signal INT. When *Int* is 0, the CL450 will assert INT, driving it LOW. When *Int* is 1, the CL450 does not drive INT, which will be pulled HIGH (inactive) by an external pullup resistor.

When auto clear is enabled (*AIC* is 1), *Int* is set to 1 by the CL450 after the host acknowledges the interrupt by asserting INTACK. When auto clear is disabled (*AIC* is 0), the host processor must explicitly write *Int* by a read-modify-write operation.

(normal) **Read, 0x9C** **HOST_intvecr**
(normal) **Write, 0x98** **HOST_intvecw**



HOST_intvecr and HOST_intvecw allow the host to access the interrupt vector and interrupt priority ID stored within the CL450. The host processor writes to HOST_intvecw and reads from HOST_intvecr. The information transferred is used only during the interrupt acknowledge cycle, which occurs when $\overline{\text{INTACK}}$ is asserted and *VIE* in HOST_control is 1.

IPID **Interrupt Priority ID (bits 10:8)**

During the interrupt acknowledge cycle, the CL450 compares bits A[3:1] of the host address bus with this three-bit field. If they match, the CL450 outputs the interrupt vector contained in *IVect* on D[7:0]. *IPID* is forced to the value 0 by hardware reset. The host driver should set *IPID* to match the CL450's interrupt priority in the system hardware. *IPID* should be set once at initialization.

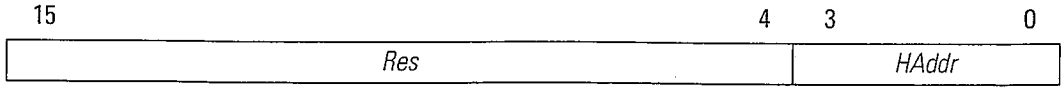
IVect **Interrupt Vector (bits 7:0)**

The CL450 outputs the value in this eight-bit field during the interrupt acknowledge cycle if the interrupt priority ID, *IPID*, matches A[3:1]. *IVect* is forced to the value 0xF by a hardware reset.

8.3.3 Command/Status Registers

These registers—HOST_raddr, HOST_rdata, and HOST_newcmd—allow the host processor to load commands and arguments into the CL450 and to read the status of the microapplication. (See Chapter 11 for information about the CL450 macro commands.)

(normal) **0x88** **HOST_raddr**

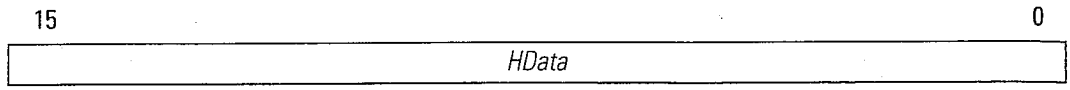


HAddr **Register Address (bits 3:0)** **R/W**

This four-bit field points to one of the 16 locations of HMEM register space. The HMEM location which is selected by this

field may be accessed by the host reading or writing the HOST_rdata register. After a write operation takes place, *HAddr* is automatically incremented by one. For read operations, the host processor must write the address each time before reading HOST_rdata. The arrangement of information in HMEM (under microapplication control) is shown in section 11.1, Writing Macro Commands.

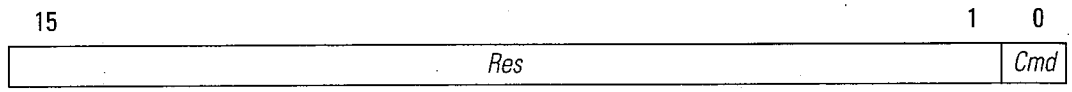
HOST_rdata **(normal)** **0x8C**



HData **Host Data (bits 15:0)** **R/W**

This 16-bit field accesses the location in the register file currently pointed to by *HAddr* in the HOST_raddr register.

HOST_newcmd **(normal)** **0x56**



Cmd **New Command (bit 0)** **R/W**

The host processor sets *Cmd* to 1 after writing the opcode and arguments for a new command to HMEM. The internal CPU resets *Cmd* to 0 after reading the opcode and arguments of the new command. The host processor must not write to the command locations in HMEM if *Cmd* is set to 1.

Command Write Data Flow

Figure 8-4 shows the general flow for a command write. The circled numbers in the figure refer to the steps below. The syntax of the command words and the arguments are described in Chapter 11, Macro Commands.

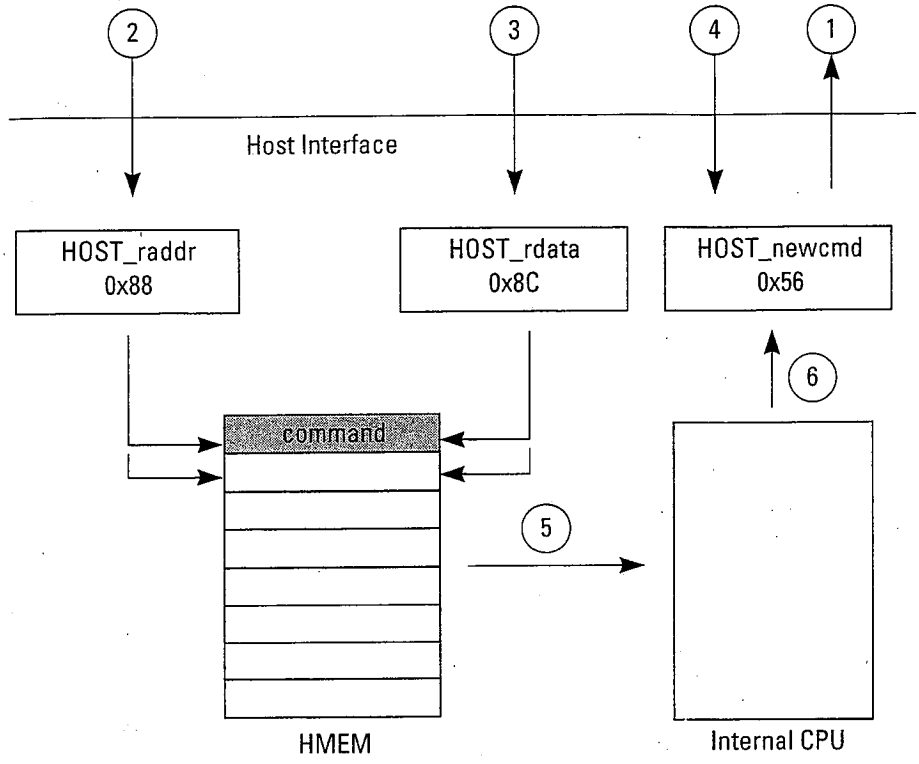


Figure 8-4 Command Write Data Flow

The sequence for writing commands to the CL450 is as follows:

1. The host processor reads the new-command bit, *Cmd,* in HOST_newcmd (address 0x56). If *Cmd* is 1, the CL450 is accessing the command locations in HMEM and the host processor should not access them. When *Cmd* is 0, the host processor can proceed with writing a new command.
2. The host processor sets HOST_raddr (address 0x88) to the HMEM address of the first location used for storing commands.
3. The host processor performs successive write operations to HOST_rdata (address 0x8C) to write the command word and the arguments (if any).
4. After the command word and all arguments have been written, the host processor sets *Cmd* to 1.
5. The CL450's CPU reads the contents of the command registers.
6. The CL450's CPU clears *Cmd* to 0 to indicate that it has completed rendering the command and that the host processor can initiate another command write operation.

8.3.4 System Clock Reference Registers

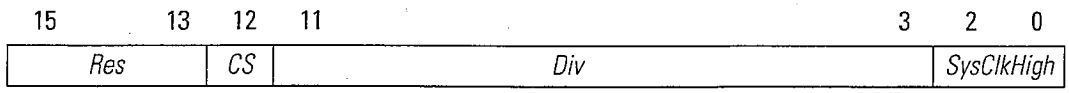
These registers provide access to the CL450's internal system clock counter. The system clock counter is 33-bits wide (MPEG bitstreams use 33-bit clock values) and is used when synchronizing the decompressed video with other data, in particular decompressed audio.

Typically, the host processor extracts system clock values (SCRs or PTSs) from the bitstream and updates the system timer with these values through use of the AccessSCR() macro command. (See page 11-13 for a description of how the AccessSCR() command is used to write or read the CL450's internal counter.) The CL450 then compares the presentation time stamps (PTSs) associated with frames in the bitstream with the system clock counter to decide whether to display, skip, or redisplay each frame.

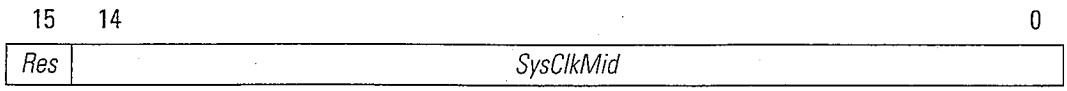
The system clock counter is typically clocked by a 90 kHz signal. This signal can be derived either from GCLK or from an external reference on the SCLK input signal. The source of the 90 kHz signal is programmed by the *CS* bit in the HOST_scr2 register. Regardless of the source, the input signal is divided down by the divisor *(Div)* programmed in HOST_scr2 to create the 90 kHz signal.

> *Note: The host must not access these registers while the microapplication is executing (to prevent access collisions between the host and the microapplication). The microapplication provides facilities for the host to update the system clock counter but forces CS and Div to 1 and 444, respectively.*

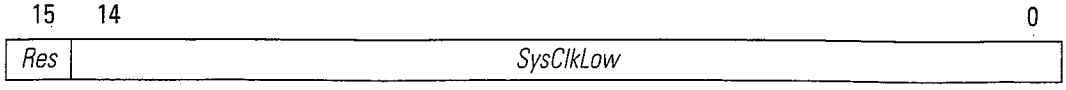
HOST_scr2 **(internal)** **0x96**



HOST_scr1 **(internal)** **0x94**



(internal) **0x92** **HOST_scr0**



CS **Clock Source (bit 12)** **W**

This bit determines the source of the signal which will be used to generate the 90 kHz clock. When *CS* is 1, the clock is derived from GCLK.When *CS* is 0, the 90 kHz signal is derived from an external reference connected to the input signal SCLK.

Div **System Clock Divisor (bits 11:3)** **R/W**

This nine-bit field contains the divisor used to derive the 90 kHz input for the system clock counter from the source selected by *CS*. For example, to derive the system clock from a 40.000 MHz GCLK, the value of *Div* is given by

$$Div = int\left(\frac{40.000 \times 10^6}{90 \times 10^3}\right) = 444$$

When a 90 kHz signal is applied to SCLK and *CS* is set to 0, *Div* should be 1 (divide by 1).

Note: The divisor may only be written to a value other than 444 if the microapplication is not executing; otherwise, the microapplication fixes the divisor at a value of 444.

SysClkHigh	**System Clock – Upper**	**R/W**
SysClkMid	**System Clock – Middle**	**R/W**
SysClkLow	**System Clock – Lower**	**R/W**

These three fields taken together comprise a 33-bit value for the CL450's internal system timer.

When these registers are read, HOST_scr0 must be read first. When the HOST_scr0 register is read, the CL450 loads the higher-order 18 bits into HOST_scr1 and HOST_scr2. The system clock continues to count while the registers are being read.

These registers should be written in order starting with HOST_scr2. The internal timer is loaded with the new value when the last register is written. When the internal timer counts past its maximum value of $2^{33}-1$, the timer rolls over to 0.

8.4 Internal CPU Registers

This next section describes the registers that allow application programs to communicate with the CL450's internal CPU. Figure 8-5 shows the registers in the CPU.

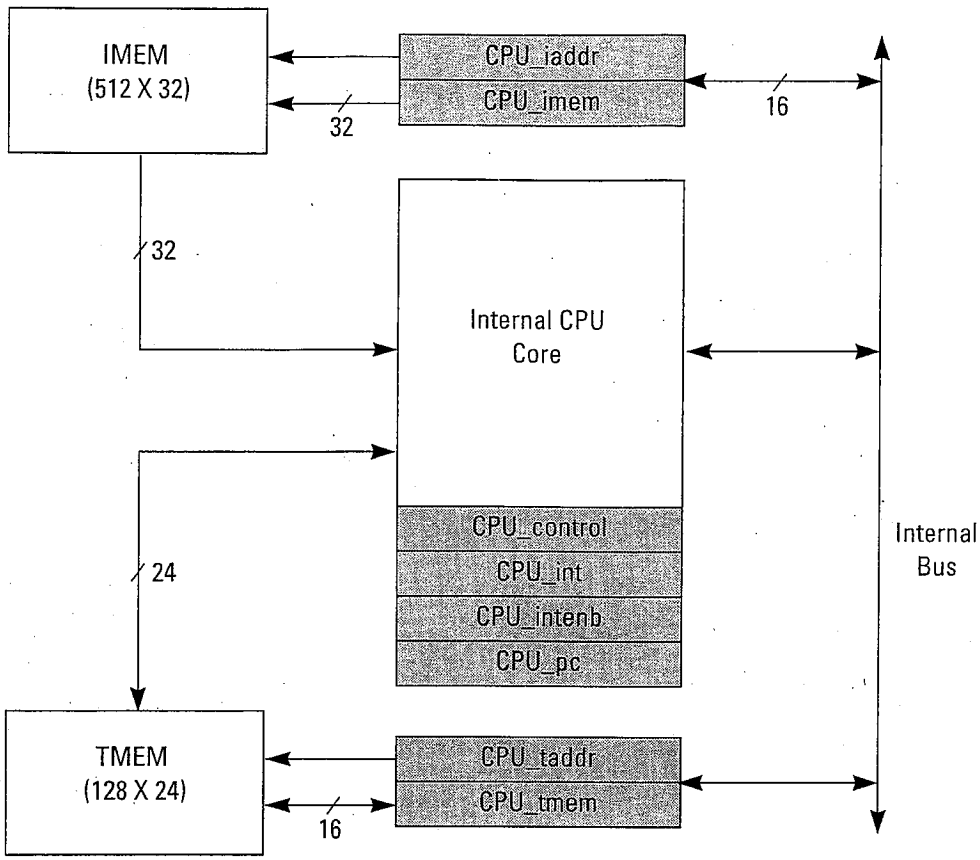


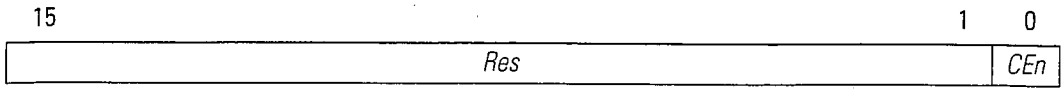
Figure 8-5 Internal CPU Registers

8.4.1 CPU Execution Registers

These registers are used to enable the CPU and to monitor the execution of the internal microapplication.

> *Note: The bits in the CPU_int and CPU_intenb registers and the IE bit in the CPU_pc register control interrupts from CL450 on-chip logic to the CL450 CPU and are unrelated to the interrupts which the CL450 issues to the host.*

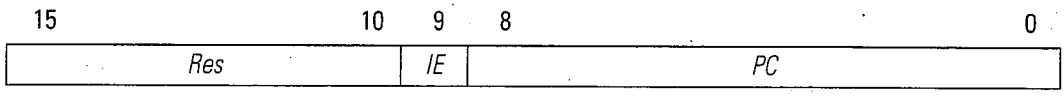
(normal) **0x20** **CPU_control**



CEn **CPU Run Enable (bit 0)** **R/W**

When the host processor sets this bit to 1, the internal CPU is enabled and begins executing instructions. During a software or hardware reset, the CL450 clears *CEn* to 0, halting the CL450's CPU.

(initialization) **0x22** **CPU_pc**



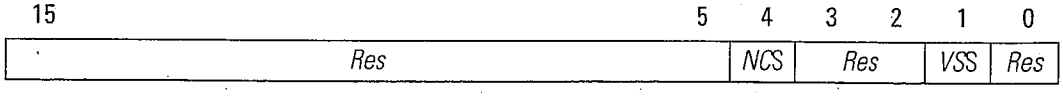
IE **Interrupt Enable (bit 9)** **R/W**

When *IE* is 1, the internal CPU interrupts are enabled. When *IE* is 0, the internal CPU interrupts are disabled. Once the CL450 is initialized and the microapplication is started, the microapplication controls this bit. (Note that interrupts to the CL450's CPU are not related to the interrupts that the microapplication issues to the host.)

PC **Program Counter (bits 8:0)** **R/W**

PC is a nine-bit field that holds the current contents of the program counter used by the internal CPU. When a new value is written to *PC*, the internal CPU executes the instruction at that address at the next instruction cycle.

(diagnostic) **0x54** **CPU_int**



NCS **New Command Interrupt Status (bit 4)** **R**

The internal CPU sets *NCS* to 1 to indicate that the new command interrupt is active. *NCS* is cleared to 0 when the interrupt service routine is completed.

VSS **VSYNC Interrupt Status (bit 1)** **R/W**

The CL450 sets *VSS* to 1 on the rising edge of VSYNC. This action interrupts the internal CPU, which executes an interrupt service routine that clears *VSS*.

CPU_intenb **(diagnostic)** **0x26**

15 … 5	4	3 2	1	0
Res	*NCE*	*Res*	*VSE*	*Res*

NCE **New Command Interrupt Enable (4) R/W**

The microapplication sets *NCE* to 1 to enable the new command interrupt.

VSE **VSYNC Interrupt Enable (bit 1) R/W**

The microapplication sets *VSE* to 1 to enable the vertical sync interrupt.

8.4.2 IMEM Access Registers

The IMEM access registers, CPU_iaddr and CPU_imem, provide a mechanism by which the host processor can write 32-bit microapplication words into the internal instruction memory (IMEM) to initialize the CL450. IMEM can hold up to 512 32-bit microapplication words. The procedure for writing to IMEM is shown in Figure 8-6. The circled numbers refer to the steps that follow.

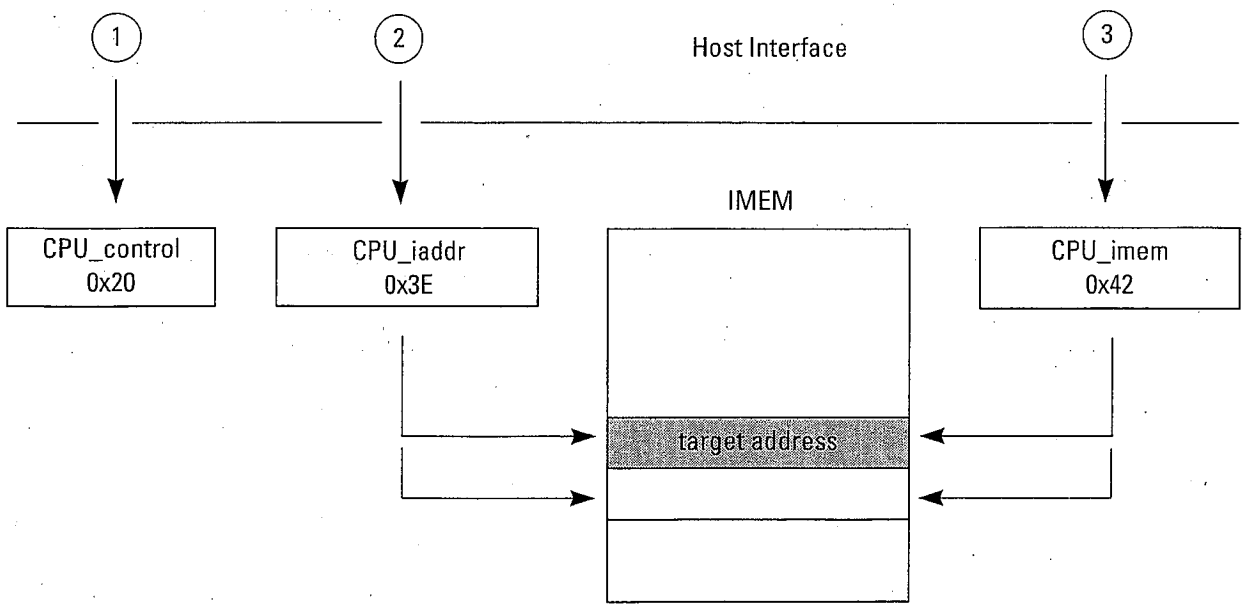
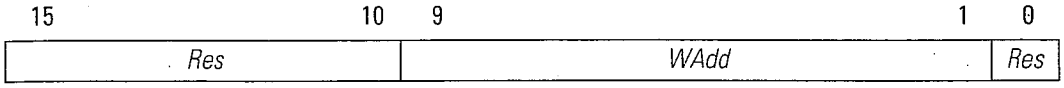


Figure 8-6 IMEM Write Data Flow

The sequence of events for writing to the IMEM is:

1. Ensure that the CPU enable bit, *CEn*, is 0.
2. The host processor loads the IMEM address to be written into CPU_iaddr bits 9:1. Note that this requires the address to be shifted left one bit (multiplied by 2) before the write.
3. The host processor writes successive pairs of 16-bit words to the CPU_imem register. (Each pair of words makes up a single 32-bit instruction.) The CL450 automatically increments CPU_iaddr with every other write to CPU_imem.

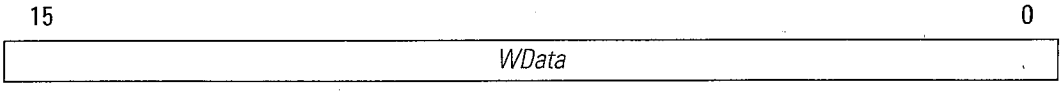
(initialization) **0x3E** **CPU_iaddr**



WAdd **Write Address (bits 9:1)** **R/W**

This nine-bit field contains the address in the IMEM to which the write operation is to be performed. *WAdd* is automatically post-incremented by the CL450 when 32-bit data is written to CPU_imem (two 16-bit writes).

(initialization) **0x42** **CPU_imem**



WData **Write Data (bits 15:0)** **W**

The host processor writes the IMEM data to this 16-bit field. The data written to *WData* is transferred internally to the IMEM address pointed to by CPU_iaddr.

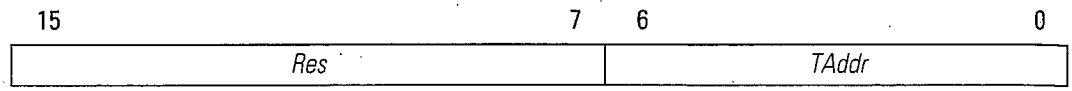
Note: Writes to this register must be done in pairs, with the first word written containing the most significant 16 bits of each instruction. If writes are not performed in pairs, the auto-increment feature of the CPU_iaddr register and the contents of IMEM will be left in an unknown state.

8.4.3 TMEM Access Registers

These two registers, CPU_taddr and CPU_tmem, provide a mechanism by which the host processor can access the CL450's "temporary" memory (TMEM) that is normally used only by the internal CPU. The host processor only accesses these registers for diagnostic purposes.

To avoid interfering with the normal operation of the microapplication, consult with a C-Cube Microsystems technical support specialist before using these registers .

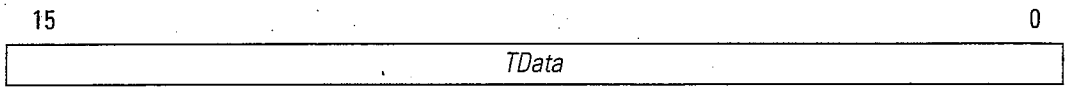
CPU_taddr **(diagnostic)** **0x38**



TAddr **TMEM Address (bits 6:0)** **R/W**

This seven-bit field contains the address in TMEM for read and write operations. After a read or write operation is completed, the value of *TAddr* is incremented by one.

CPU_tmem **(diagnostic)** **0x46**



TData **TMEM Data (bits 15:0)** **R/W**

This 16-bit field is used to read data from and write data to TMEM.

8.5 DRAM Interface Register

This section describes the register used by application programs to set the refresh time for the DRAM controller. Figure 8-7 shows the DRAM interface.

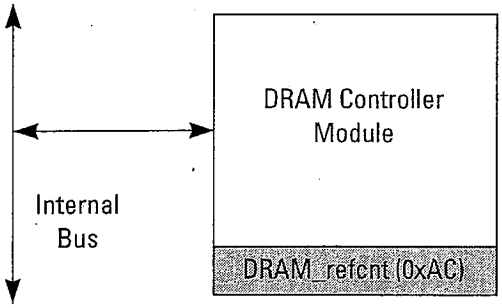
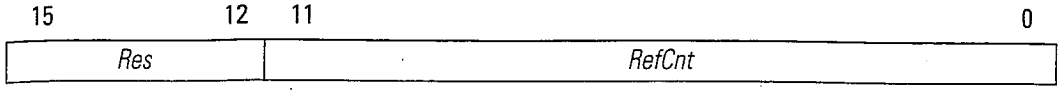


Figure 8-7 DRAM Interface Register

(normal) **0xAC** **DRAM_refcnt**



RefCnt **Refresh Clock Count (bits 11:0)** **W**

This 12-bit field sets the number of GCLK periods between DRAM refresh cycles. The CL450 refreshes a single page at a time rather than refreshing every page of DRAM in a single operation. *RefCnt* should be set to a value that allows the CL450 to refresh all of the pages (typically 512) within the refresh time required by the DRAMs in use (typically 8ms).

RefCnt should only be written once after power-up. The value is calculated from the formula

$$RefCnt = floor\left(\frac{(RP - 512 \times GCLK) / (DR + 1)}{GCLK}\right)$$

where *RP* is the DRAM refresh period (t_{REF}), *GCLK* is the GCLK period, and *DR* is the number of rows in each DRAM.

In this formula, "512 x *GCLK*" represents the maximum time that the CL450 DRAM controller stays in page mode. DRAM refresh cycles do not interrupt page mode cycles, so the refresh period must be shortened to compensate.

"*DR* + 1" represents the number of refresh cycles that must be done in a refresh period because the DRAM controller can delay one refresh request if other requests are pending. When there are two banks of local DRAM, both banks are refreshed at the same time.

For a typical system with a local DRAM of 512 rows with a refresh period of 8ms and a GLCK period of 25ns, the value of *RefCnt* should be 622.

8.6 Video Interface Registers

This section describes the registers in the video interface module. Figure 8-8 shows the internal structure of the video interface module. The registers shown in lightly shaded boxes are addressed directly by the host processor, while the registers shown in darker shaded boxes are addressed indirectly through the VID_control register.

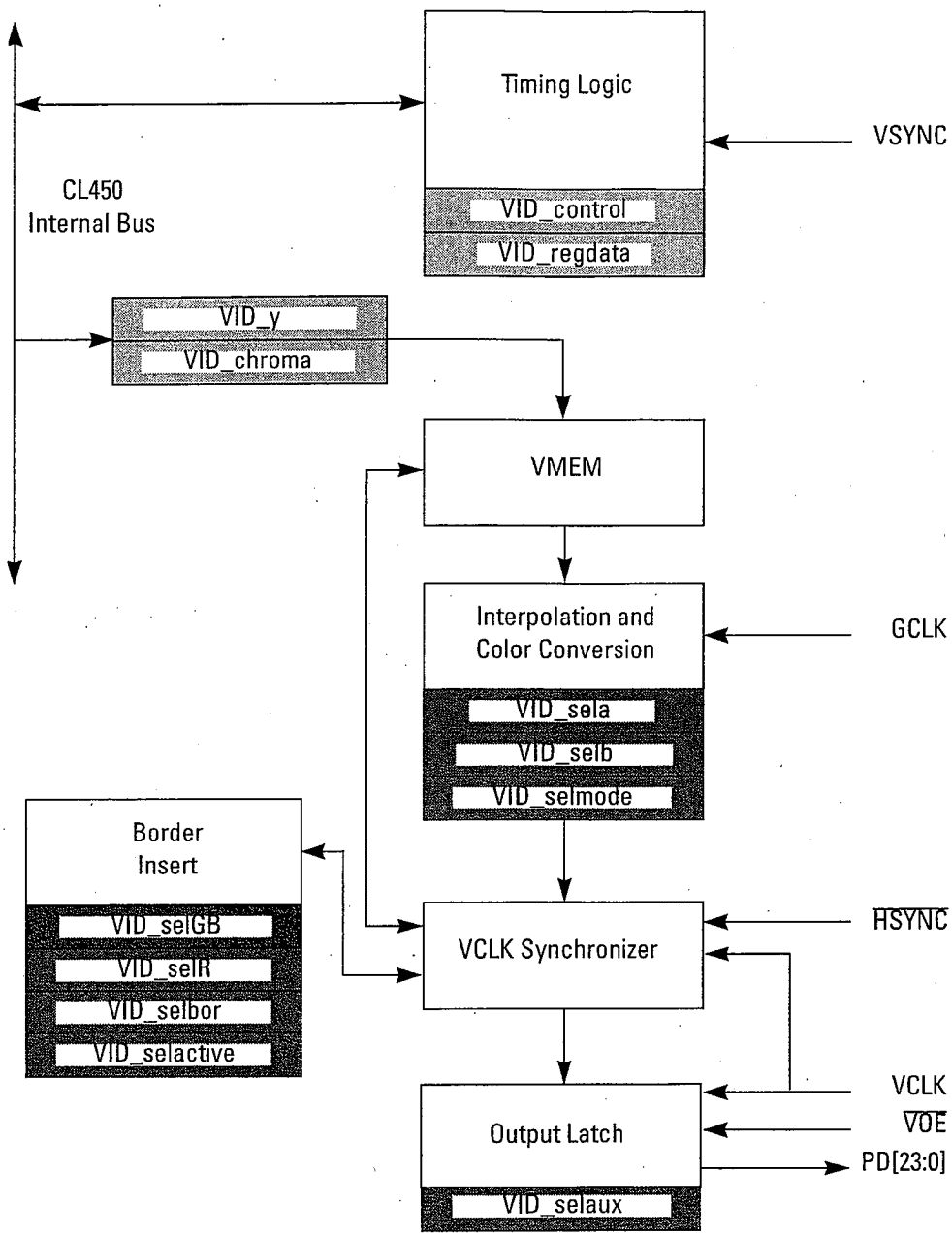
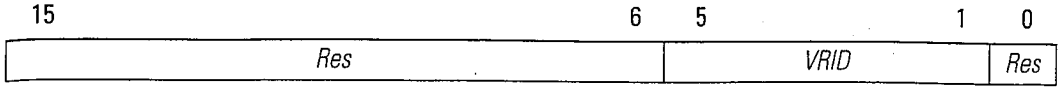


Figure 8-8 Video Interface Registers

8.6.1 Direct-access Video Registers

These registers are directly accessed using a register read or register write operation.

(initialization) **0xEC** **VID_control**



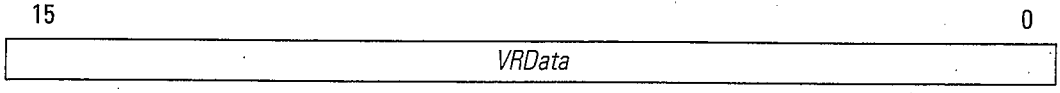
VRID **Video Register ID (bits 4:0)** **R/W**

This five-bit field points to one of the indirect video registers. To access one of the indirect video registers, the host processor loads the video register ID into *VRID*, then performs a read from or a write to the direct register VID_regdata. The video register IDs are listed in numerical order in Table 8-8. *VRID*s not listed are reserved and should not be used.

Table 8-8 Video Register IDs

VRID	Video Register	*VRID*	Video Register
0	VID_sela	9	VID_selbor
1	VID_selb	A	VID_selR
7	VID_selmode	B	VID_selGB
8	VID_selactive	C	VID_selaux

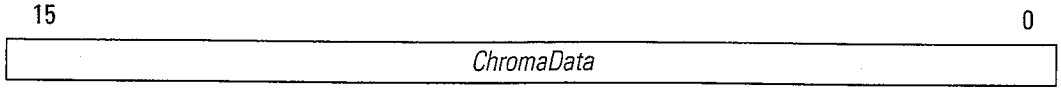
(initialization) **0xEE** **VID_regdata**



VRData **Video Register Data (bits 15:0)** **W**

The host processor uses this 16-bit field to read from and write to the indirect video register pointed to by *VRID* in VID_control.

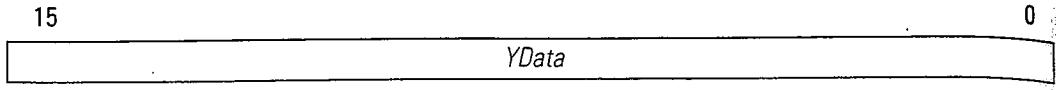
(internal) **0x0A** **VID_chroma**



ChromaData **Chroma Data (bits 15:0)** **W**

Diagnostic programs may use this 16-bit field to write chrominance (CbCr) data to the internal video FIFO (VMEM).

VID_y **(internal)** **0x00**



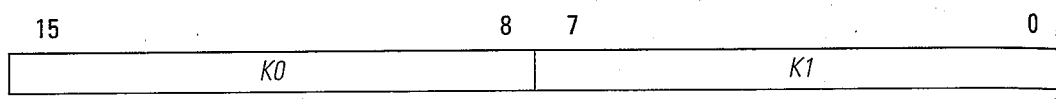
YData **Y Data (bits 15:0)** **W**

Diagnostic programs may use this 16-bit field to write luminance (Y) data to the internal video FIFO (VMEM).

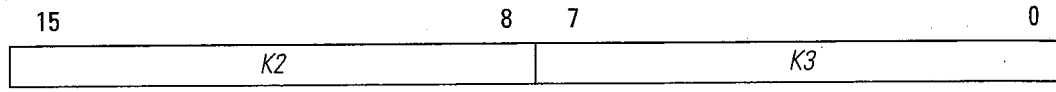
8.6.2 Indirect-access Video Registers

These registers are accessed by first writing the appropriate video register ID (*VRID)* to the VID_control register, then performing the desired operation by reading from or writing to VID_regdata. Application programs should only access these registers when the CL450's CPU is disabled to avoid interfering with the video display.

VID_sela **(internal)** ***VRID* = 0x0 [Indirect]**



VID_selb **(internal)** ***VRID* = 0x1 [Indirect]**



K3 – K0 **Conversion Coefficients** **W**

These four 8-bit fields set the coefficients for the YCbCr-to-RGB conversion performed by the internal color-space converter. The equations for the color-space conversion are:

$$Red = Y + D \times Cr$$

$$Blue = Y + B \times Cb$$

$$Green = Y + A \times Cb + C \times Cr$$

where A, B, C, and D are 10-bit two's-complement coefficients whose binary representations are given in Figure 8-9 (note the location of the binary point).

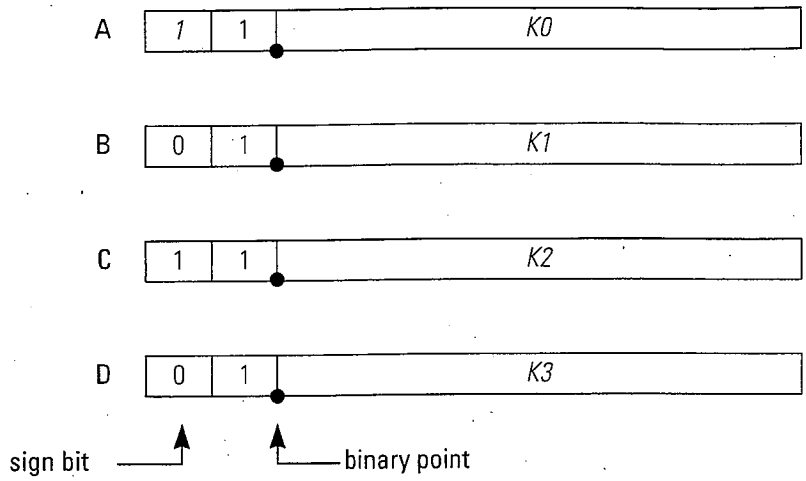
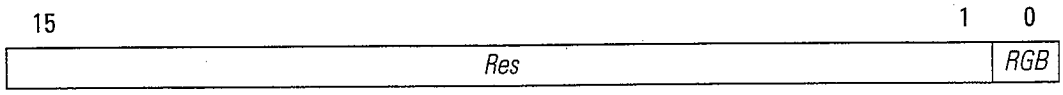


Figure 8-9 Binary Representations of Conversion Coefficients

To set the CL450 for CCIR 601 chromaticity as specified in the MPEG standard, the values of the coefficients should be set as shown below:

K0 = 0xA8
K1 = 0xC6
K2 = 0x49
K3 = 0x67

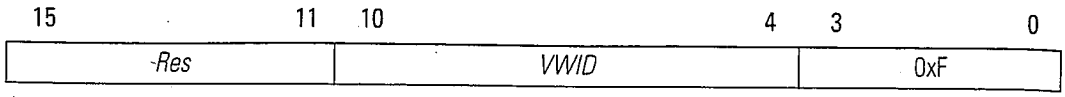
(internal) ***VRID* = 0x7 [Indirect]** **VID_selmode**



RGB **RGB Mode Select (bit 0)** **W**

When this bit is 1, the CL450 operates in RGB mode. When this bit is 0, the CL450 operates in YCbCr mode.

VID_selactive (internal) — *VRID* = 0x8 [Indirect]



VWID **Width of active region (bits 10:4)** **W**

VWID determines the width of the active region. The width in pixels of the active region is set to the following value after horizontal interpolation (i.e., in CCIR 601 resolution):

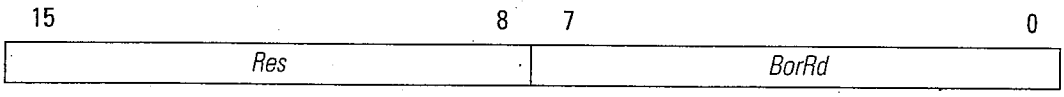
$$width = 8 \times VWID + 1$$

VID_selbor (internal) — *VRID* = 0x9 [Indirect]

BorLeft **Left Border Size (bits 9:0)** **W**

This 10-bit field contains the number of border pixels output on each line from the rising (inactive) edge of $\overline{\text{HSYNC}}$ to the first pixel of the active window. The number of left border pixels must be greater than or equal to 10 to ensure correct operation.

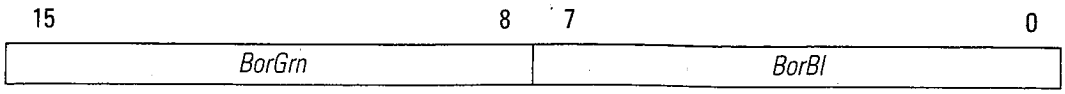
VID_selR (internal) — *VRID* = 0xA [Indirect]



BorRd **Border, Red Component (bits 7:0)** **W**

This eight-bit field sets the red component of the border (in RGB operation) and the Cr component (in YCbCr operation).

(internal) ***VRID* = 0xB [Indirect]** **VID_selGB**



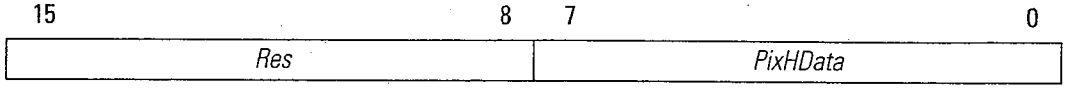
BorGrn **Border, Green Component (bits 15:8)** **W**

This eight-bit field sets the green component of the border (in RGB operation) and the Y component (in YCbCr operation).

BorBl **Border, Blue Component (bits 7:0)** **W**

This eight-bit field sets the blue component of the border (in RGB operation) and the Cb component (in YCbCr operation).

(initialization) ***VRID* = 0xC [Indirect]** **VID_selaux**



PixHData **Pixel Data High Byte (bits 7:0)** **W**

When the CL450 is set for YCbCr operation, the upper eight bits of the output pixel data bus, PD[23:16], do not contain pixel data. In this case, this eight-bit value is output on PD[23:16].

9 Microapplication Overview

The CL450 performs its higher-level functions by executing a microapplication on an internal CPU. The microapplication is supplied by C-Cube Microsystems with the hardware and is considered an integral part of the product.

The microapplication must be loaded into the CL450 by a software driver; it needs to be located in specific sections of the local DRAM and on-chip IMEM. Additional information on loading and executing this code is contained on the distribution disk that is provided with the CL450.

9.1 Process Configurations

The operation of the CL450 microapplication and the host processor can be described by a set of "process configurations"—each a named group of software processes, some executing within the host processor and some executing within the CL450's microapplication. Five different process configurations occur at different times in a CL450 system. These process configurations are shown in Figure 9-1 and Figure 9-2.

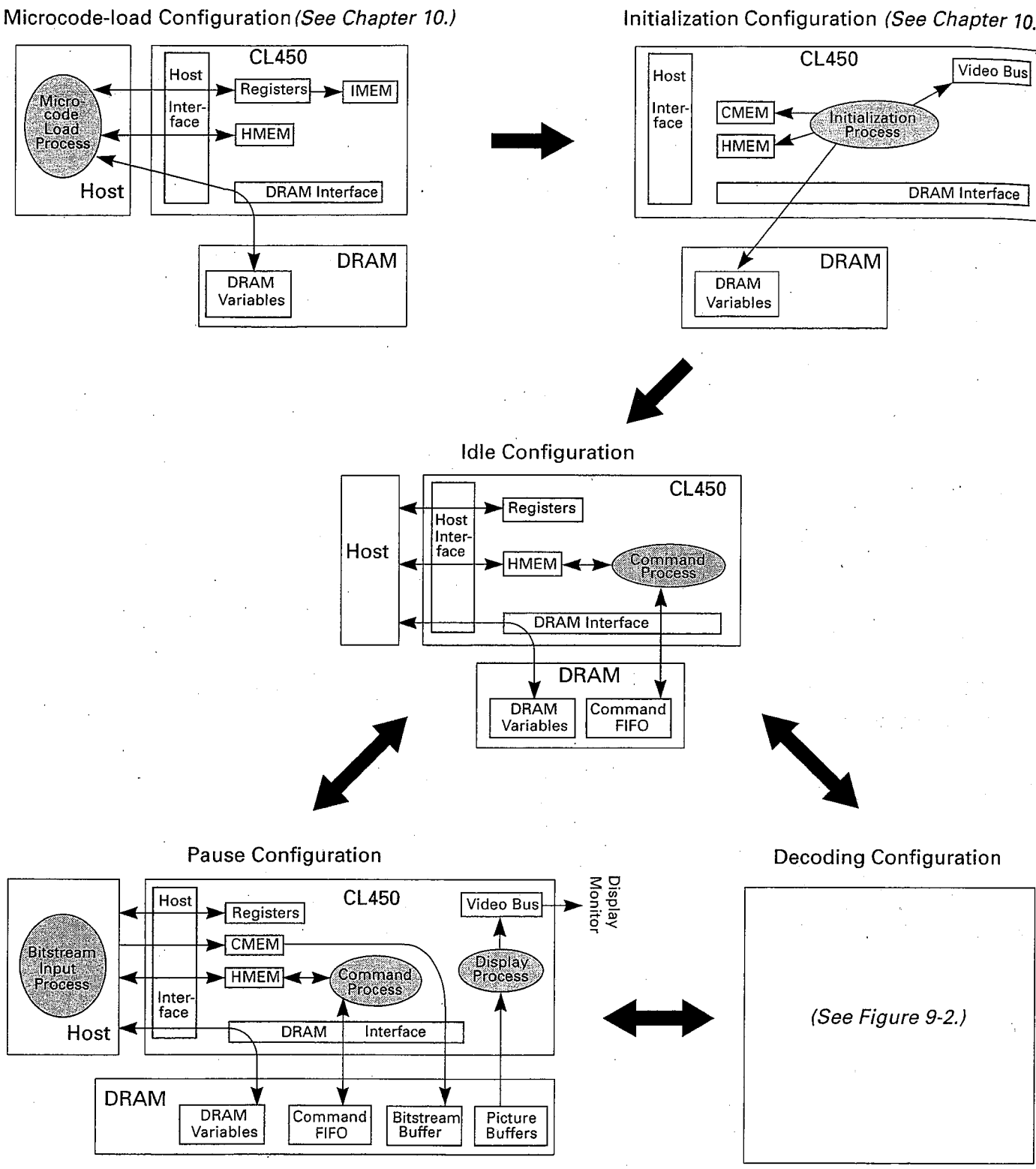


Figure 9-1 CL450 Microapplication Process Configurations

Within each process configuration shown in Figure 9-1 and Figure 9-2, shaded ovals indicate individual but concurrently-running software processes executed by the host or the CL450, rectangles indicate hardware resources, and arrows indicate logical data transfers. (Physical buses used for data transfers are not indicated.) The large arrows pointing to separate process configurations in Figure 9-1 indicate possible microapplication progress over time in a CL450-based system.

A total of six separate software processes are shown (shaded ovals) in the five process configurations of Figure 9-1 and Figure 9-2. They are Microcode-load, Initialization, Command, Bitstream Input, Decode, and Display. The first two of these processes are described in Chapter 10 (Initialization). The remainder of these processes execute simultaneously when the system is in the Decoding process configuration shown in Figure 9-2 and are described in more detail following this figure.

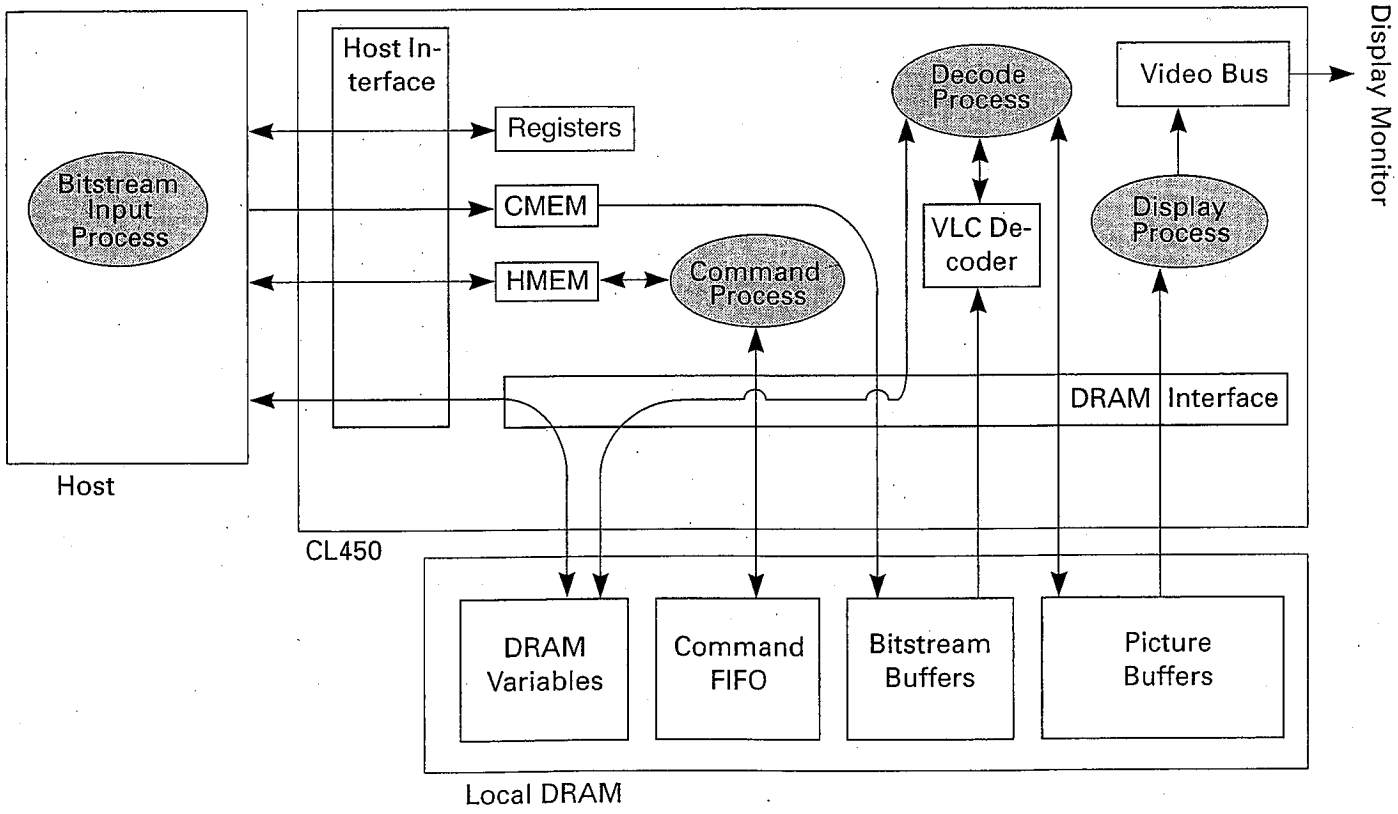


Figure 9-2 CL450 Decoding Process Configuration

9.1.1 Command Process

The primary method by which the microapplication executing on the CL450's CPU receives direction from the host software is by means of *macro commands*, which are function codes and parameter values written into HMEM by the host (see Table 9-1, Macro Command Summary). Once the function code and parameters are written, the microapplication takes these values and either immediately executes the operation corresponding to the function code or stores the complete macro command into the Command FIFO (see Figure 11-3) for later execution.

Table 9-1 Macro Command Summary

Category	Priority	Effect on Command State	Name	Description	Function Code (0x)	Page
Set-type	Low	No	SetBlank()	Blanks/unblanks output window	030f	11-29
			SetBorder()	Sets output window location	0407	11-30
			SetColorMode()	Enables/disables color-space converter	0111	11-34
			SetInterruptMask()	Enables/disables interrupts to host	0104	11-36
			SetThreshold()	Specifies bitstream buffer emptiness	0103	11-38
			SetVideoFormat()	Configures output resolution and timing	0105	11-40
			SetWindow()	Sets output window size and contents	0406	11-42
Play-type	Low	Yes	DisplayStill()	Decodes/displays single still picture[1]	000c	11-14
			Pause()	Keeps last picture on display	000e	11-25
			Play()	Decodes and displays at normal rate	000d	11-26
			Scan()	Decodes and displays next single I-picture	000a	11-28
			SingleStep()	Decodes and stores next single picture	000b	11-45
			SlowMotion()	Decodes and displays at slower rate	0109	11-46
Control	High[2]	No[3]	AccessSCR()	Reads or writes internal SCR counter	8312	11-12
			FlushBitstream()	Discards contents of bitstream buffer	8102	11-16
			InquireBufferFullness()	Measures data in bitstream buffer	8001	11-19
			NewPacket()	Manages bitstream data	0408	11-20
			Reset()	Reinitializes CL450 and its microcode	8000	11-27

1. Allows display of a still image with double vertical resolution, but requires use of a specific subset of the MPEG syntax.
2. Except for NewPacket()
3. Except for Reset()

Note: If a macro command is issued and HMEM[0] contains a value other than one of the values listed in the Function Code column above, indeterminate behavior will occur.

9.1.2 Bitstream Transfer Process

MPEG coded bitstream data is transferred by the host system to the CL450 for decoding. The CL450 accepts MPEG elementary video streams only. System streams and elementary audio streams must be handled by the host prior to data arrival at the CL450.

Data can be transferred from the host to the CL450 using either (1) direct host writes to the input of the CL450's CMEM or (2) DMA. In both cases, the mode is selected by the host writing to the CMEM_dmactrl register, and the behavior of the CL450 (macro commands, interrupts, etc.) is the same for either method chosen.

Once inside the CL450, data is buffered in CMEM and then transferred as a burst to the 47,104-byte *bitstream buffer* located in the local DRAM. (The bitstream buffer is called the *rate buffer* in the MPEG standard.)

If the bitstream buffer is full, the CL450 does not transfer data from CMEM, so the host must ensure that the CMEM does not overflow. For direct host transfers (non-DMA), the host processor must monitor the fullness of CMEM. For DMA transfers, the DMA controller automatically holds off transfers when CMEM is full.

In addition to sending video information to the CL450, the host processor also extracts system clock references (SCRs) and presentation time stamps (PTSs) from the system layer of the MPEG bitstream. It uses these to provide the CL450 with the information required to synchronize the video and audio decoding as described in Section 9.2.

DMA Operation

When DMA is used, the CL450 acts as a DMA slave and, once correctly enabled, asserts the $\overline{\text{DMAREQ}}$ pin any time it can accept another 16-bit DMA write. In this mode, the host does not have to monitor the fullness of CMEM or the bitstream buffer portion of DRAM unless the host needs this information to configure its DMA controller. The hardware protocol and timing used for DMA transfers is given in Chapter 4.

Programmed Access

During programmed access, the host writes 16-bit words of coded data into CMEM by addressing any location in a 256K-word area of the

CL450's address space. These write operations put coded data into CMEM, a temporary holding FIFO with sixteen 16-bit locations between the host and the DRAM-resident bitstream buffer.

When directly writing CMEM, the host must not only configure CMEM_dmactrl correctly during initialization but also continually poll CMEM to prevent overflow, since a high enough instantaneous transfer rate can cause CMEM to fill up even if the bitstream buffer is not full.

> *Note: There is no data rate for which the host can avoid polling CMEM fullness; if the host does not ensure that CMEM can accept data before the data is written, CMEM overflow and data corruption will result. Because of this, the host must poll the fullness of CMEM while performing a data transfer.*

Pseudocode for a typical transfer loop is shown in Figure 9-3. Note that only the coded data transfer operation is shown. Other non-transfer operations (not shown) need to be performed for most applications.

```
#define      BLOCK    < size of burst>        /* typically 3/4 empty */

  void SendDataToCL450(void)
  {
     int c;

     initialize CMEM_dmactrl;                /* see section 3.1 for the
                                              * proper timing of this write
                                              * with respect to other CL450
                                              * initialization */

     while (still more data to transfer){
        get CMEM fullness;                   /* see below */
        if (room in CMEM for BLOCK words)
           for (c=0;c < BLOCK;c++)
              write word of data to CMEM;
     }
  }
```

Figure 9-3 Programmed Transfer of Coded Data to the CL450

The fullness of CMEM may be determined by one of the following two methods:

- *Using the CFLEVEL pin*: This pin can be programmed by the CMEM_dmactrl register to go active at any one of four different CMEM fullness levels. This pin can either be polled by the host (if it is accessible through a host I/O port) or wired directly to a device which is providing data to the CL450 instead of the host procesor. The "entirely empty" selection in CMEM_dmactrl (bit 1) should not be used because of the existence of operating modes in which CMEM never becomes entirely empty.
- *Performing a register read of CMEM_dmactrl*: Note that while CMEM_status contains information similar to CMEM_dmactrl, CMEM_status is an internal register and should therefore not be used to determine CMEM fullness while the microapplication is running.

9.1.3 Decode Process

The decode process is the process by which the CL450 decompresses the input bitstream using the MPEG decoding algorithm and places the decompressed frames in the picture buffers in the CL450's local DRAM.

The decoding process reconstructs spatial frequency coefficients from the variable-length Huffman codes, dequantizes them, and applies the inverse discrete-cosine transform. It uses the motion vectors in the bitstream to select a predictor from the previously decompressed frames stored in the local DRAM.

The decoding process pauses if the bitstream buffer is empty or if no space is available for writing the decoded frame. Such a condition could occur if the display of the previous frame was not completed before the decoding process was ready to begin writing to its frame buffer.

9.1.4 Display Process

The display process in the CL450's microapplication handles the transfer of decoded pictures from the local DRAM picture buffers through the video bus from where data is passed to the display monitor. The CL450 supports clipping and positioning of the decoded video data anywhere on the display.

Images coded at 24, 25 or 30 Hz can be displayed at field rates of 50 or 60 Hz.

I- and P-pictures are fully buffered before being displayed. B-pictures are written to and displayed from the frame buffer as they are being decoded, provided the presentation time stamp requirements are met as described in Chapter 13, Audio/Video Synchronization.

9.2 Synchronization

The CL450 can synchronize its decoding operations with an external time reference such as an audio decoder. Synchronization is performed using the MPEG constructs of *system clock references (SCRs)* and *presentation time stamps (PTSs)*—both measured in terms of a 90-kHz time reference. When performing synchronization, the CL450 compares the PTS value associated with each picture—either supplied by the host via the NewPacket() macro command or synthesized by the microapplication—with the current value of the CL450's internal SCR counter.

If the presentation time stamp differs from the internal clock by more than the allowed jitter tolerance of 3000 clock periods (1/30th of a second), the display rate is adjusted as follows:

- If the presentation time stamp for a B-picture is less than the SCR by more than 3000, the picture is skipped.
- If the presentation time stamp for a picture is greater than the SCR by more than 3000, the picture is repeatedly displayed.

9.3 Interrupts

The CL450 provides an interrupt output pin ($\overline{\text{INT}}$) to the host processor which allows the microapplication to alert the host when certain events occur. The host determines the hardware interrupt protocol by writing the HOST_control register. The host then selects the interrupt events of which it wishes to be informed by using the SetInterruptMask() macro command (see page 11-36) to assign mask bits as shown in Figure 9-4.

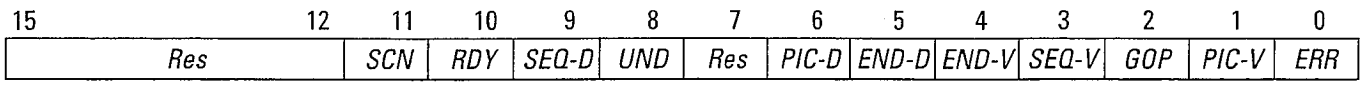


Figure 9-4 Mask Bit Allocation

There are 11 logical interrupts which can be produced by the CL450 microapplication. Each belongs to one of three categories (based on when they are reported to the host) and is referenced by name and the event which causes it in Table 9-2. The interrupts and their categories are described in Chapter 12, Interrupts.

Table 9-2 CL450 Interrupt Summary

Category	Interrupt Name	Event	Mask Bit	Page
Decode-time	END-D	`sequence_end_code` found	5	12-15
	ERR	Bitstream data error	0	12-17
	PIC-D	New picture decoded	6	12-20
	SEQ-D	`sequence_header_code` found	9	12-25
	SCN	Picture decode complete in Scan()	11	12-24
	UND	Bitstream buffer underflow error	8	12-27
VSYNC	END-V	Last picture display before `sequence_end_code`	4	12-16
	GOP	First I-picture display after `group_start_code`	2	12-19
	PIC-V	New picture display	1	12-21
	SEQ-V	First I-picture display after `sequence_header_code`	3	12-26
Display-time	RDY	Ready for data	10	12-22

Note: The CL450's internal CPU can also receive interrupts from on-chip sources. The host has no control over these interrupts and they should not be confused with the host interrupts generated by the CL450.

10
Initialization

The host processor executes an initialization sequence for the CL450. During the initialization sequence, the host:

1. Loads the microapplication into the CL450 DRAM.
2. Loads bootstrap code into the CL450's instruction memory (IMEM).
3. Sets the program counter in the CPU_pc register to point to the starting address of the bootstrap code.
4. Enables the CL450's CPU by setting the run enable bit, *CEn* (CPU_control[0]), to 1.

Once enabled, the CL450's CPU bootstraps itself by accessing internal instruction memory (IMEM) and executing code that was written by the host processor; it acts essentially as a microcontroller with on-chip peripherals used to aid MPEG decoding.

Note: Enabling the CPU is different than sending the Play() command. The enable bit, CEn, turns on the CL450's internal processor, while the Play() command is a specific macro command that runs a portion of code in the DRAM to read and decompress an MPEG bitstream. The Play() command cannot be executed before the microapplication is running; the microapplication, once enabled, will not decode bitstreams until the Play() command is executed. Other macro commands include those that change the border color and display size, and all are described in Chapter 11.

10.1 Registers

When the CL450's microapplication is first started, the contents of several registers are written with default parameters. Similarly, when the Reset() macro command is issued, a subset of these registers is re-initialized. Because of these automatic defaults, some registers cannot be effectively written before microapplication initialization.

10.1.1 Default Settings

Table 10-1 summarizes the registers that are modified when microapplication is loaded and when the Reset() command is executed (uninitialized registers are not included). Default values are given in binary, with "1" and "0" representing bits which are set and cleared, "-" representing bits which are unmodified, "?" representing bits which are given indeterminate but acceptable values, and "R" indicating RESERVED bits. The space between each group of four bits is for ease of reading only—these groupings do not indicate register fields.

Note: If the host performs read-modify-write operations on registers in Table 10-1, the host must ensure that RESERVED bits are written with the values specified in Chapter 8, Registers, regardless of the value read for these bits.

Table 10-1 Automatic Register Defaults

Register Name	Use	Initialization[1]	Reset()
HOST_newcmd	Normal	RRRR RRRR RRRR RRR0	RRRR RRRR RRRR RRR0
CMEM_control	Normal	RRRR RRRR R--R RR10	RRRR RRRR R--R RR10
HOST_control	Normal	00RR RRRR 1RRR RRR1	00RR RRRR 1RRR RRR1
VID_control	Normal	RRRR RRRR RR?? ???R	No effect
VID_sela	Normal	1010 1000 1100 0110	No effect
VID_selb	Normal	0100 1001 0110 0111	No effect
VID_selmode	Internal	RRRR RRRR RRRR RRR1	No effect
VID_selactive	Internal	RRRR R101 0111 ????	No effect
VID_selbor	Internal	RRRR RR00 0011 1100	No effect
VID_selR	Internal	RRRR RRRR 0000 0000	No effect
VID_selGB	Internal	0000 0000 0000 0000	No effect
HOST_scr2	Internal	RRR1 1101 1110 0000	No effect
HOST_scr1	Internal	R000 0000 0000 0000	No effect
HOST_scr0	Internal	R000 0000 0000 0000	No effect

1. "R" indicates RESERVED bits, "-" indicates bits which are unmodified by the microapplication, "?" indicates bits written with variable values, and "1" and "0" indicate fixed values written by the microapplication.

In addition to initialization of the above registers, the following operations are performed on initialization or Reset():

- All 16 locations of HMEM are cleared to 0.
- CMEM is reset via the *CRst* bit of CMEM_control.
- MPEG default parameters are restored.

10.1.2 Loading Sequence

In general, not all CL450 registers will be used by the host in all applications. Some of the registers can be accessed by the host at any time, while some may only be accessed when microapplication is *not* executing. In addition, there are some registers which the microapplication automatically writes with default values upon microapplication startup (shown in Table 10-1), negating any writes by the host which occurred previously. Because of these factors, the sequence with which registers are written is critical to the proper operation of the CL450.

Specifically, the following four registers are the only registers that may be accessed at any time by the CPU without any interference from the microapplication:

- CMEM_dmactrl
- HOST_intvecw
- HOST_intvecr
- DRAM_refcnt

In general, the host can access most Normal registers at any time (although there may be undesirable side effects if an Internal register is accessed). However, there are three classes of exceptions:

- *Indirect video registers* (see Table 8-4 on page 8-6): These registers can only be accessed by the host after a selection has been made using VID_control. In addition, the CL450 microapplication *also* uses VID_control to access these registers. Therefore, host access (read or write) to an indirect video register is prohibited while microapplication is executing, and any access attempts will produce indeterminate results.
- *Registers for accessing HMEM* (HOST_raddr and HOST_rdata): These registers are reserved for the exclusive use of the host. Because the CL450 microapplication does not use these registers, the host can use them at any time, making HMEM unlike the other on-chip memories. However, while HMEM may be read at any time, HMEM writes performed while the microapplication is executing *must* occur in accordance with the protocols given in Section 11.2 , Processing Macro Commands; and Section 12.2, Interrupt Status Location.
- *Registers for accessing other on-chip memories* (IMEM, TMEM, etc.): These registers are shared by the CL450 and by the host. For this reason, indeterminate results will occur if on-chip memories other than HMEM are accessed while the microapplication is executing.

The following text outlines the sequence in which registers should be written by the host relative to CL450 start-up *if* they are to be written at all:

1. The VID_selaux register *must* be written prior to starting the execution of the microapplication because it will be inaccessible afterwards.

2. The CMEM_control register *may* be written prior to starting the execution of the microapplication, although its value will be modified by microapplication initialization.
3. The microapplication should be loaded and started by following the procedure given in Section 10.2.2. The following registers should be used *only* as part of the microapplication loading process:
   - CPU_control
   - CPU_pc
   - CPU_iaddr, CPU_imem
4. Once the microapplication has started execution, the following registers may be used by the host:
   - HOST_newcmd, as specified by Section 11.1, Writing Macro Commands
   - CMEM_control
   - HOST_raddr, HOST_rdata
   - HOST_control, as specified by Section 12.3, Handshaking Protocol
5. Finally, several macro commands can be used to implicitly load values into registers. These commands and their corresponding registers are listed in Table 10-2 below.

Table 10-2 Macro Commands for Loading Registers

Macro Command Name(s)	Register(s) Loaded
SetColorMode()	VID_selmode
SetBorder(), SetWindow()	VID_selactive, VID_selbor[1]
SetBorder()	VID_selR, VID_selGB

1.Note that these registers may also be modified in response to new MPEG sequence-layer information. See Section 10.2.1, Default Settings; Section 14.2.2, Writing DRAM-resident Variables; and the descriptions of the SetBorder() and SetWindow() macro commands contained in Chapter 11.

10.2 Microapplication

The CL450 microapplication is logically contained in a series of "segments," each of which is a block of numeric constants (constant data or microapplication) that the host must load into either the internal IMEM (instruction memory) *or* the external DRAM.

All the segments of the CL450 microapplication are distributed in a single microcode executable file. (The format of this file is defined in Appendix B.) In addition to microapplication segments, the executable file also contains an information header which includes the initial program counter value for the CL450's CPU.

10.2.1 Default Settings

All of the programmable parameters used by the CL450 microapplication have defaults. This section lists these defaults, their functional relationship and the event(s) which causes the CL450 to change from the default setting or return to it.

Table 10-3 lists the macro commands which can be used to change settings within the CL450 and the default values for these settings. Note that these defaults are set on microapplication initialization only and are *not* restored by execution of the Reset() macro command.

Table 10-3 Macro Command Defaults

Macro Command Name	Default Setting
SetThreshold()	**threshold**= 4096 bytes
SetInterruptMask()	**mask**= 0; no interrupts enabled
SetVideoFormat()	**format**= 4 (NTSC): □ 60Hz nominal VSYNC frequency □ 30Hz nominal picture rate □ 352 pixel (SIF) horizontal active display □ 240 pixel (SIF) vertical active display
SetWindow()	**xOffset** = **yOffset** = 0 **width** & **height** from sequence-layer parameters (`horizontal_size` and `vertical_size`)
SetBorder()	**leftBorder**= **topBorder**=auto-center based on sequence layer parameters border color is "sub-black" (R=G=B=Y=0)
SetBlank()	**state**= 0; video display not blanked
SetColorMode()	**mode**= 1 (RGB output)

In addition to the default parameters which can be specified by macro commands, the CL450 contains default values for MPEG bitstream parameters as shown in Table 10-4 and Table 10-5. The decoder is restored to these defaults following microapplication initialization, the execution of the Reset() command, and detection of an MPEG `sequence_end_code` in the bitstream. These default values are used for decoding unless replaced by the host (see Section 14.2.2, Writing DRAM-resident Variables) or decoded from the bitstream.

In general, the default values are not used for decoding if the decode operation starts at the beginning of a legal MPEG bitstream. However, when performing random-access within a bitstream, the defaults are used if no other values are provided. Note that only parameters which are required for decoding have default values.

Table 10-4 MPEG Decoding Defaults, Sequence Layer

Bitstream Parameter	Default Setting
horizontal_size	352 pixels
vertical_size	240 pixels
picture_rate	4 (30Hz)
load_intra_quantizer_matrix	0 (use default from MPEG standard)
load_non_intra_quantizer_matrix	0 (use default from MPEG standard)

Table 10-5 MPEG Decoding Defaults, GOP Layer

Bitstream Parameter	Default Setting[1]
closed_gop	0
broken_link	1

1. The CL450 makes no assumptions about whether the current GOP follows the last GOP and waits for an I-picture before beginning to decode.

Table 10-6 summarizes the bits in the CMEM_control, HOST_control and CMEM_dmactrl registers which control communication between the host and the CL450, plus their reset values, if any. For a complete list of all register bits which have default values, see Table 10-1.

Note: Some of the register bits which control interaction with the host have no defaults.

Table 10-6 Host/CL450 Interface Defaults

Register Name	Bit(s)	Bit Name	Default	Description
CMEM_control	5	*BS*	unknown[1]	Determines whether or not bytes are swapped when 16-bit words enter CMEM.
CMEM_dmactrl	4:1	*1QE* to *4QE*	unknown	These bits determine when the CL450 will assert the CFLEVEL pin.
	0	*DE*	unknown	This bit (DMA Enable) is used to select whether bitstream data is provided to CMEM using DMA or programmed access (see Section 9.3, Bitstream Transfer Process). Note that this bit is cleared automatically if the $\overline{\text{DONE}}$ pin is asserted.
HOST_control	15	*AIC*	unknown	Auto Interrupt Clear; determines whether bit 7 (*Int*) is set ($\overline{\text{INT}}$ pin inactive) automatically when the $\overline{\text{INTACK}}$ pin is asserted.
	14	*VIE*	0	Vectored Interrupt Enabled; determines if the current interrupt vector (written by the host through the HOST_intvecw register) is driven onto the data bus when the $\overline{\text{INTACK}}$ pin is asserted.
	7	*Int*	0	$\overline{\text{INT}}$ pin inactive.

1. The host should initialize these bits following hardware reset; they are unmodified by microapplication initialization or execution of the Reset() macro command.

Finally, several of the DRAM-resident variables also have default values (see Chapter 14). Those variables which have defined defaults are listed in Table 10-7 below. DRAM-resident variables not listed in the table do not have defaults. The DRAM-resident variables are restored to their defaults on both microapplication initialization and Reset().

Table 10-7 DRAM-Resident Variable Defaults

Group	Name	Value
sequence group	SEQ_SEM	0
	SEQ_CONTROL	0
picture group	PIC_SEM	0

10.2.2 Loading Sequence

The microapplication loading sequence is as follows:

1. Typically, microapplication loading starts with the host parsing the microcode executable file and writing the appropriate segments into DRAM.
2. Some segments are also written to IMEM using the CPU_iaddr and CPU_imem registers.

   Note: The host must ensure that segment values are written to IMEM in an order such that the last value written goes to an IMEM address other than the initial program counter value. The segments in the executable file are arranged such that, if the host writes IMEM locations in the same order the data appears in the file, this restriction will always be met.

3. Once both DRAM and IMEM have been completely loaded, the initial program counter value should be written to the CPU_pc register.

   Note: When the initial value is written to CPU_pc, the IE bit (bit 9) must be 0. This bit is the internal master interrupt enable for the CL450's CPU and is not related to interrupts from the CL450 to the host.

4. After the CPU_pc register has been initialized, the host should write the value 1 to the CPU_control register to enable the CL450's CPU.

After the on-chip CPU is enabled, the host must wait until the internal initialization is complete before issuing the first macro command.

One way to determine if initialization is complete is to write a non-zero value to HMEM location 15 prior to enabling the CPU. Then the contents of this HMEM location can be polled. Once this location becomes 0, the CL450 has completed its internal initialization and is ready to accept macro commands. This method can also be used to determine when the execution of the Reset() macro command is complete (see Reset() on page 11-28).

10.2.3 Halting

Microapplication execution may be halted from the host either by:

- Resetting the CL450
- Writing a 0 to the CPU_control register

These two alternatives will produce different results on the display depending on whether or not the CL450 was displaying a picture at the time the CPU was halted.

If the CL450 is reset, then the last-specified border color will be displayed continually until the microapplication is reloaded and bitstream decoding is resumed.

If the host writes a 0 to CPU_control, then the display depends on what the CL450 was doing when halted. If the output window was blanked or the CL450 was in vertical border time, the visual result will be the same as if the CL450 was reset. If CPU_control was cleared while the CL450 was displaying an active scan line, then that scan line will be repeatedly output (regardless of vertical timing) until the CL450 is reset or the microapplication re-loaded.

> *Note: Once halted, it is not possible to resume microapplication execution without performing the entire microapplication loading sequence described above.*

When the microapplication is loaded, there are four DRAM locations (see Table A-6 on page A-3) which contain information describing the microapplication version. This information is duplicated in the microapplication executable file header but is loaded into DRAM to ensure that the microapplication version present in a running system can be determined.

11
Macro Commands

The host software and the CL450's microapplication use 18 macro commands as their primary method of communication. Macro commands are function codes and parameter values written into the dual-ported HMEM by the host. Each command has a separate function code and may have 0 to 4 parameters.

Once the function code and parameters are written, the microapplication acts on them according to their priority:

- *High-priority*: These commands start execution as soon as the CL450's microapplication detects their presence and complete execution *before* the HOST_newcmd semaphore is cleared.
- *Low-priority*: These commands are stored by the CL450 in the Command FIFO (see Section 11.4, Command Latency) and executed by the CL450 in the order that they were received (see Section 11.4, Command Latency).

11.1 Writing Macro Commands

Macro commands are written into HMEM by the host. HMEM is a 16-word-by-16-bit memory within the CL450. It is accessed indirectly by writing the desired HMEM address (0 through 0xf) to HOST_raddr and then reading or writing the HOST_rdata register.

When HOST_rdata is read, the contents of the currently selected HMEM location are returned to the host. When HOST_rdata is written by the host, the currently selected HMEM location is written with the same data.

Note: Indeterminate behavior occurs if the host reads HOST_rdata more than once without re-writing HOST_raddr.

HMEM locations are allocated for (1) macro command use and (2) CL450 status information. Table 11-1 indicates the standard usage for each HMEM location.

Table 11-1 HMEM Address Allocation

HOST_raddr (0x)	Usage
0	Macro Command, Function Code
1	Macro Command, Argument 1
2	Macro Command, Argument 2
3	Macro Command, Argument 3
4	Macro Command, Argument 4
5	RESERVED
6	RESERVED
7	RESERVED
8	RESERVED
9	RESERVED
a	Interrupt Status
b	Buffer Fullness Status
c	RESERVED
d	RESERVED
e	RESERVED
f	RESERVED

The host should not use the reserved HMEM locations for scratch storage because the CL450 may write to them at any time. Also, the host

should generally avoid writing to reserved locations within HMEM to facilitate upgrading to future microapplication versions.

Because HMEM is dual-ported and contains quantities other than the current macro command (see InquireBufferFullness() on page 11-19 and Section 12.2, Interrupt Handshaking), it uses a semaphore to assure that only one of either the host or the CL450's CPU accesses the HMEM locations used for macro commands at any one time. This semaphore is located in HOST_newcmd[0].

Any time the semaphore is 0, the host may read and write the macro command locations of HMEM (HMEM addresses 0 through 4). Once an entire new macro command has been written, the host should write a 1 to the semaphore. Once this has occurred, the host may not write to HMEM[4-0] or HOST_newcmd again until the CL450 has cleared the semaphore (i.e., HOST_newcmd[0] = 0).

To ensure that a deadlock does not occur if the CL450 fails to acknowledge a command, the host may use a time-out counter when polling HOST_newcmd.

The mechanism for issuing a macro command is shown in the pseudo-code example of Figure 11-1.

```
ERROR_RETURN SendNewMacroCommand(arguments)
{
   write function code to HMEM[0];
   write arguments (if any) to HMEM[4-1], as appropriate;
   HOST_newcmd= 1;                              /* tell microcode new macro
                                                 * command is ready */
   while (     (HOST_newcmd&1 == 1)         &&
               (haven't timed-out yet)       )
      ;                                         /* wait for command to be
                                                 * accepted */
   if (timed-out)
      return ERROR;
   else
      return NO_ERROR;
}
```

Figure 11-1 Issuing a Macro Command to the CL450

In Figure 11-1, note that the time-out processing could also be moved to the beginning of the function without violating the semaphore protocol: the host must *not* write to HMEM[4-0] when HOST_newcmd[0] is 1. In general, the host software will have better performance if the polling loop is moved up because the microapplication's acceptance of the new macro command will occur in parallel with the host's next processing task.

The function shown provides advantages in the diagnostic process because it returns an error immediately upon determining that the CL450 has stopped responding to commands. If the CL450 fails to accept a macro command, it indicates an incorrect microapplication loading procedure or an unrecoverable failure during operation. In either case, the microapplication must be reloaded.

11.2 Command States

The CL450 operates in one of eight possible internal command processing states which affect how future commands are interpreted and how the decode and display processes operate.

Several command states correspond to a macro command which can cause a command state transition. The macro commands which can explicitly change the command state are the Reset() command and all of the Play-type commands. In addition to macro commands, several internal operations can also cause a command state transition.

The command states and their relationships are shown in Figure 11-2. Within this figure, the eight command states are inscribed within ovals. The solid arrows indicate state transitions which occur due to the completion of internal processing, while shaded arrows indicate transitions caused by the execution of a macro command.

The behavior of the IDLE and PLAY-SETUP states are described next, while the remainder of the command states are described with their corresponding macro commands listed in the second half of this chapter.

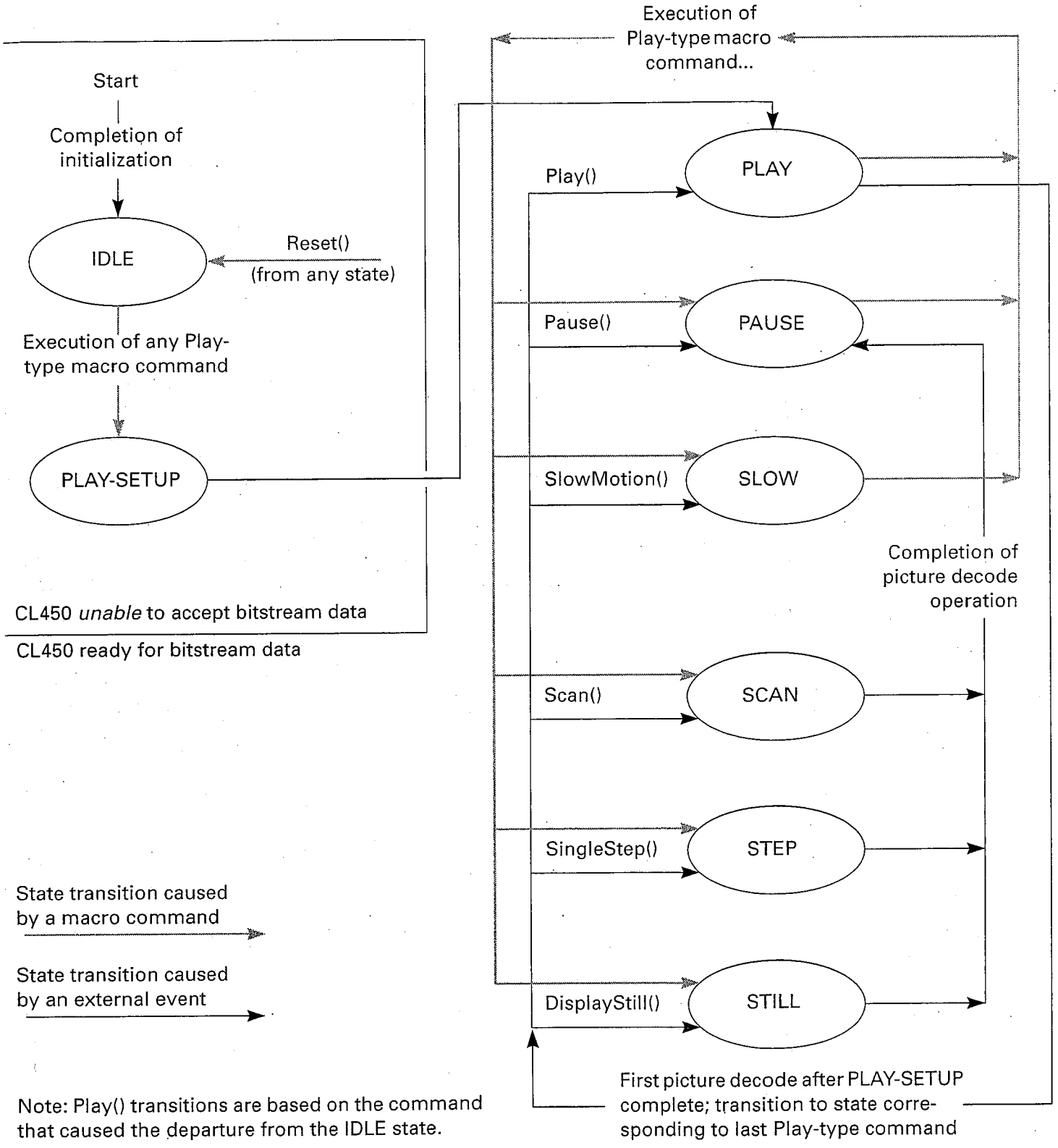


Figure 11-2 Command State Transition Diagram

11.2.1 IDLE State

The CL450 microapplication is in the IDLE state immediately following initialization or execution of the Reset() command and stays in the IDLE state until a Play-type macro command is executed. While in the IDLE state, the CL450's output window is blanked (the screen is filled with the current border color) and the CL450 does not accept bitstream data. Because of this, Display-time interrupts are not produced (see Section 12.1.1, Display-time Interrupt), and the host must *not* send coded data to the CL450. However, the host may issue any macro command while the CL450 is in the IDLE state, although some arguments to the Set-type commands will not have an immediate effect.

11.2.2 PLAY-SETUP State

The PLAY-SETUP state is a transition state between IDLE and the remaining states. The CL450 microapplication spends only a short time in this state, during which it performs internal housekeeping to set up for reception of bitstream information.

Once the operations associated with the PLAY-SETUP state are complete, the microapplication *immediately* and *always* transitions to the PLAY command state, regardless of which Play-type macro command caused the transition from IDLE. Because of this, the first decoded picture after the microapplication is initialized will always be decoded in the PLAY state. Once this picture is decoded, the microapplication effectively "re-executes" the Play-type command and then transitions to the state corresponding to the macro command which caused the original transition from IDLE to PLAY-SETUP. This occurs even when the first Play-type command is Pause(), SingleStep(), Scan(), DisplayStill(), etc. In addition to causing an "extra" picture to be decoded before entering the desired state, this also causes the execution of subsequent low-priority macro commands to be deferred one extra picture decoding time. Because all Play-type commands executed in the IDLE state are interpreted in the same way that a two-command sequence beginning with the Play() command would be, two picture decodes must complete (one for the implicit Play() command and one for the actual command issued) before other commands from the Command FIFO are read and executed.

Once the microapplication has left the PLAY-SETUP state, the CL450 begins decoding incoming bitstream data as it is received. The output window is subsequently unblanked once enough of the bitstream has been decoded for display to begin.

> *Note: Unblanking of the output window only occurs if the display has not been explicitly blanked by the host (see the SetBlank() macro command). If the display has been blanked by the host, then the output window is unblanked only when another SetBlank() is executed with the appropriate argument.*

11.3 Command FIFO

The CL450 microapplication maintains a Command FIFO in DRAM. The Command FIFO occupies 630 words of DRAM and is used to store all low-priority (HMEM[0][15] == 0) macro commands issued by the host. Each macro command occupies five words of storage (loaded from HMEM[4-0]), so that the Command FIFO can accommodate up to 126 entries.

Typically, the number of pending commands in the Command FIFO is significantly less than 126. However, NewPacket() commands remain in the Command FIFO until sometime after the corresponding portion of the bitstream has been decoded, regardless of whether the other macro commands ahead of and behind them have been executed, which increases the number of entries required.

> *Note: The CL450 microapplication cannot prevent the Command FIFO from overflowing. The host must ensure that the maximum number of possible entries in the Command FIFO is not exceeded.*

The Command FIFO is maintained as a circular buffer with one write pointer and two read pointers ("command read" and "packet read") as shown in Figure 11-3. Each time a new low-priority macro command is issued by the host, it is written into the circular buffer and the write pointer is incremented (wrapping if necessary). Each time a macro command *other than* NewPacket() is extracted and executed, the "command read pointer" is incremented. In addition, if a NewPacket() command is encountered when looking for another macro command, the command read pointer is incremented *past* the NewPacket(), effectively skipping it.

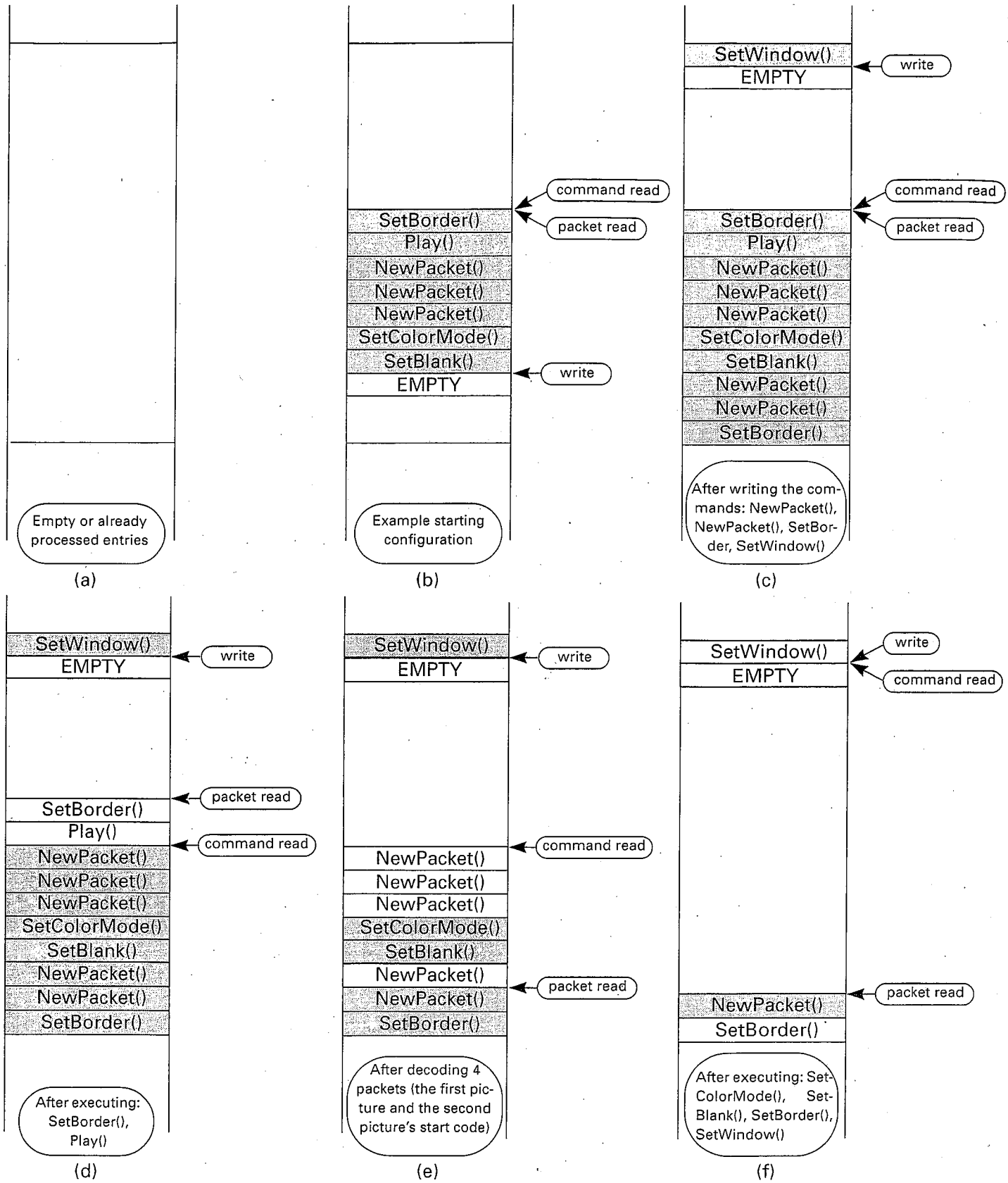


Figure 11-3 CL450 Command FIFO Example

Each time a new `picture_start_code` is detected during decode, the microapplication searches through the Command FIFO using the "packet read" pointer and looks for the NewPacket() command corresponding to the packet which contains the start code of the picture which is about to be decoded. While this is done, the packet read pointer is incremented to skip macro commands other than NewPacket().

11.4 Command Latency

All low-priority macro commands are placed into the Command FIFO before the CL450 clears HOST_newcmd[0]. The delay before macro command execution is determined by the current command processing state. The microapplication operates according to the following rules:

- NewPacket() commands have no impact on when other low-priority macro commands are executed.
- When in the IDLE or PAUSE command states (see Section 11.4, Command Latency, and the Pause() command on page 11-25), macro commands are executed as they are received unless the host issues commands faster than they can be processed, in which case they accumulate in the Command FIFO. Command execution continues uninterrupted *until* a Play-type macro command is executed, which changes the command state from IDLE or PAUSE to PLAY-SETUP or a decoding state, respectively.
- When not in the IDLE or PAUSE command states, macro commands which have accumulated in the Command FIFO are executed each time the CL450 completes decoding a picture. Note that this rate is dependent on the output frame rate, the `picture_rate` sequence parameter, audio/video synchronization, and the last **speed** parameter if in the SLOW state. If a Play-type command is *not* encountered, the microapplication continues to execute commands until all commands in the Command FIFO have been consumed or until it is time to begin decoding the next picture, whichever is *later*.

The host can issue more macro commands than the CL450 is able to process without falling behind in picture decoding. In general, the host:

- Must ensure that the Command FIFO does not overflow.
- May issue NewPacket() and any high-priority commands as frequently as desired.
- May issue one SetBorder() and one SetWindow() command per frame period (nominally every other VSYNC).

Other low-priority commands should be issued only when necessary.

11.5 Macro Command Groups

Note that CL450 macro commands are divided into three functional categories:

- Set-type
- Play-type
- Control

Each command group has distinct properties, which are described below, in Table 11-2, and in Section 11.4.

11.5.1 Set-type Commands

The CL450 has seven Set-type macro commands, all of which:

- Are low priority
- Never affect the command state
- May be issued regardless of the current command state
- Have no effect on the decoding process, except for SetVideoFormat(), which helps determine if video sequences are transcoded (see SetVideoFormat on page 11-40)

11.5.2 Play-type Commands

The CL450 has six Play-type macro commands, each of which causes the:

- Current command state to change
- Macro command processing to be suspended
- Microcode to transition into the PLAY-SETUP state and the CL450 to be configured to accept bitstream data (when issued in the IDLE state)

11.5.3 Control Commands

The CL450 has five Control macro commands (the least homogeneous of the three groups), each of which:

- Can be issued regardless of the current command state
- Is high priority, except for NewPacket()
- Has no effect on the command state, except for Reset()

Table 11-2 Macro Command Summary

Category	Priority	Effect on Command State	Name	Description	Function Code (0x)	Page
Set-type	Low	No	SetBlank()	Blanks/unblanks output window	030f	11-29
			SetBorder()	Sets output window location	0407	11-30
			SetColorMode()	Enables/disables color-space converter	0111	11-34
			SetInterruptMask()	Enables/disables interrupts to host	0104	11-36
			SetThreshold()	Specifies bitstream buffer emptiness	0103	11-38
			SetVideoFormat()	Configures output resolution and timing	0105	11-40
			SetWindow()	Sets output window size and contents	0406	11-42
Play-type	Low	Yes	DisplayStill()	Decodes/displays single still picture	000c	11-14
			Pause()	Keeps last picture on display	000e	11-25
			Play()	Decodes and displays pictures	000d	11-26
			Scan()	Decodes and displays next single I-picture	000a	11-28
			SingleStep()	Decodes and stores next single picture	000b	11-45
			SlowMotion()	Decodes and displays at slower rate	0109	11-46
Control	High[1]	No[2]	AccessSCR()	Reads or writes internal SCR counter	8312	11-12
			FlushBitstream()	Discards contents of bitstream buffer	8102	11-16
			InquireBufferFullness()	Calculates fullness of bitstream buffer	8001	11-19
			Newpacket()	Manages bitstream data	0408	11-20
			Reset()	Reinitializes CL450 and its microcode	8000	11-27

1. Except for NewPacket()
2. Except for Reset()

Note: If a macro command is issued and HMEM[0] contains a value other than one of the values listed in the Function Code column above, indeterminate behavior occurs.

11.6 Macro Command Reference

All CL450 macro commands are listed alphabetically in the pages that follow.

AccessSCR()

Format: AccessSCR (**timeStamp2, timeStamp1, timeStamp0**)
Priority: High
Category: Control

Syntax:

HMEM[0]	AccessSCR	0x8312
HMEM[1]	**timeStamp2**	bit[15] = R/W flag bits[14:3] = 0 bits[2:0] = SCR bits 32:30
HMEM[2]	**timeStamp1**	bit[15] = 0 bits[14:0] = SCR bits 29:15
HMEM[3]	**timeStamp0**	bit[15] = 0 bits[14:0] = SCR bits 14:0
HMEM[4]		0x0000

The AccessSCR() macro command is issued by the host to write or read the current value of the CL450's internal SCR (System Clock Reference) counter. The microapplication uses the contents of the SCR counter when performing audio/video synchronization. (For a full description of the CL450's synchronization mechanism, see Chapter 13, Audio/video Synchronization.)

The R/W flag bit (HMEM[1][15]) determines whether the command performs a read or a write, as follows:

- *R/W flag bit =1*: The microapplication reads the SCR counter and places the results in HMEM[1], HMEM[2] and HMEM[3]. When a read operation is performed, the read completes before the microapplication clears HOST_newcmd[0].

  The current counter value is written into HMEM in the format given above. However, the value written to the R/W flag bit and the bits to which the host is required to write 0's is random, and the host should mask the values read back from HMEM before using them.

- *R/W flag = 0*:When the host issues this command, the microapplication writes the new SCR value from HMEM into the CL450's counter. Note that the contents of the SCR counter have no effect on the CL450's behavior unless audio/video synchronization is being performed.

As the host processor extracts system clock references from the system layer of a bitstream, it updates the SCR counter by using an AccessSCR() macro command. The microapplication then updates the counter through the three HOST_scr registers shown in Figure 11-4: HOST_scr2, HOST_scr1, and HOST_scr0.

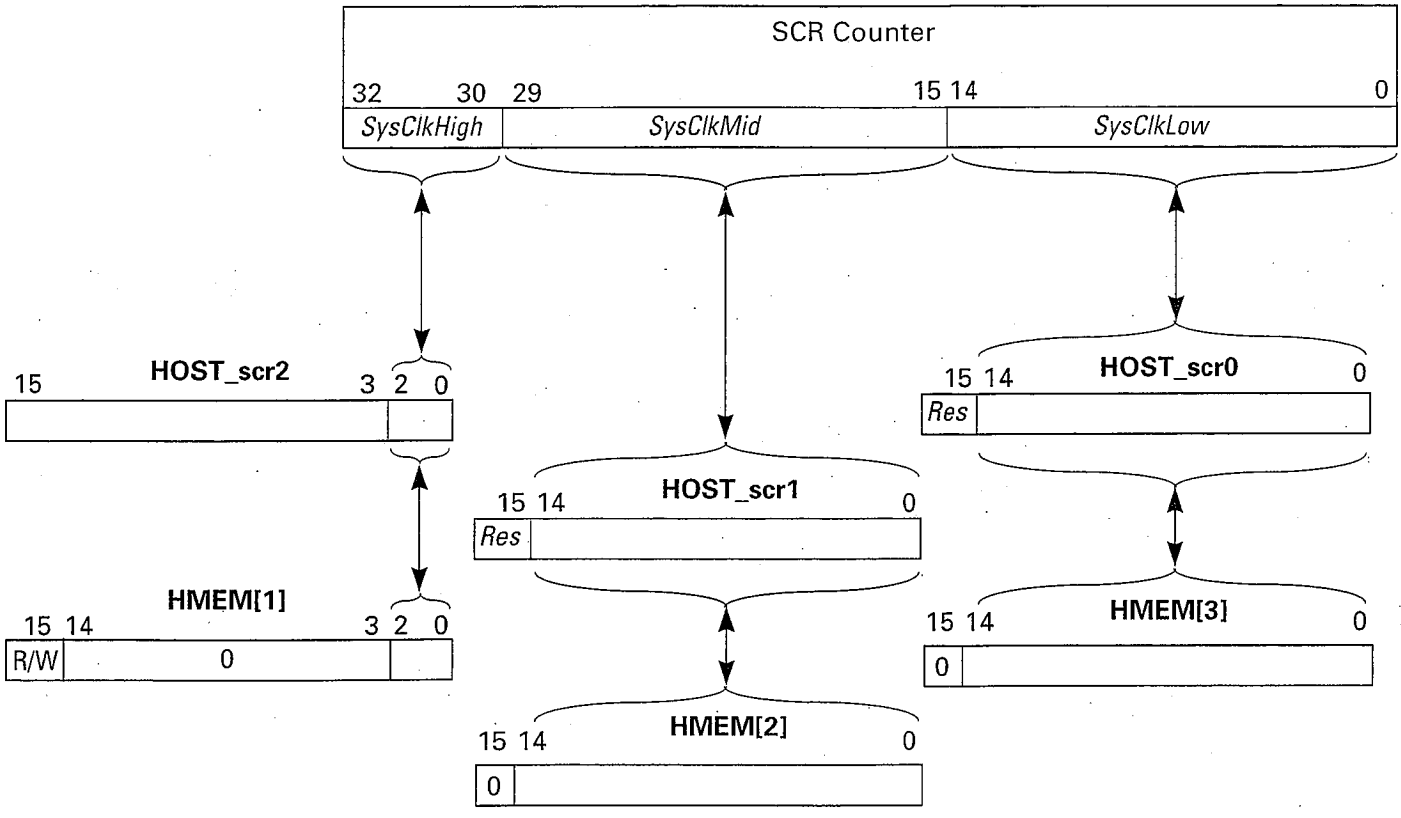


Figure 11-4 AccessSCR() Arguments Block Diagram

DisplayStill()

Format: DisplayStill()
Priority: Low
Category: Play-type

Syntax:

HMEM[0]	DisplayStill	0x000c
HMEM[1]		0x0000
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

The DisplayStill() macro command is used to decode and display a high-vertical-resolution still picture. When this command is executed, the microapplication enters the STILL state, possibly passing through the PLAY-SETUP and PLAY states.

While in the STILL state, a single still picture from the incoming bit-stream is decoded and posted for display.

> *Note: Audio/video synchronization has no effect on the timing of picture decoding which occurs in the STILL state regardless of whether or not NewPacket() commands are being used.*

Once decoding is complete, the microapplication automatically changes to the PAUSE state (see the description of the Pause() command, page 11-25).

A still picture must be constructed as follows:

```
sequence_header()              /* optional */
group_of_pictures()            /* optional */
   picture()                   /* must be a single I-picture, supplies
                                * image for upper field display */
   picture()                   /* must be a single I-picture, supplies
                                * image for lower field display */
sequence_end_code              /* optional */
```

Figure 11-5 Still Picture Bitstream Format

The header and picture information within a high-vertical-resolution still picture is decoded normally as two separate images, one containing all of the scan lines for the upper display field and the other containing the lower scan lines. These two images are then displayed during the even and odd display fields (as appropriate) while the microapplication remains in the PAUSE state.

Typically, both images in a still picture contain as many scan lines as a SIF image. Note that the **yOffset** argument to the SetWindow() macro command (see page 11-42) is interpreted as an offset both from the beginning of the even field to the first even scan line to be displayed *and* from the beginning of the odd field to the first odd scan line displayed.

Note that the microapplication clears the CL450's internal SCR counter to 0 each time a picture pair is decoded in the STILL state (see Section 13.3.2, Automatic SCR Modifications).

FlushBitstream()

Format:	FlushBitstream(**filter**)		
Priority:	High		
Category:	Control		
Syntax:	HMEM[0]	FlushBitstream	0x8102
	HMEM[1]	**filter**	bits[15:3] = 0 bit[2] = restore default sequence parameters flag bits[1:0] = filter parameter
	HMEM[2]		0x0000
	HMEM[3]		0x0000
	HMEM[4]		0x0000

The FlushBitstream() macro command causes the CL450 to:

- Discard the current contents of the bitstream buffer (including any data currently in CMEM or the VLC Decoder) and any unprocessed NewPacket() commands in the Command FIFO (see page 11-9).
- Allow the host to specify the conditions under which picture decoding and display should resume when new bitstream information is received by the CL450.

In all cases, the CL450 continues to display the most-recently decoded picture after FlushBitstream() has been issued.

When using this command, the host must ensure that the bitstream data which is sent *following* the command has the correct relationship with subsequent NewPacket() commands, if there are any. In particular, the following facts must be considered:

- When FlushBitstream() is executed, *all* unprocessed NewPacket() commands are discarded, even if the CL450 has not yet received the bitstream data that corresponds to those packets.
- The FlushBitstream() command takes a finite amount of time to execute. If bitstream data transmission to the CL450 is not halted

prior to issuing this command, it is impossible to determine what data will be flushed and what data will be considered to have arrived *after* the command executed.

- Because FlushBitstream() is a high-priority command, all operations required to clear the bitstream buffer are completed *before* the CL450 clears HOST_newcmd[0].
- Because FlushBitstream() is a high-priority command, it may be executed while in the middle of a picture decode operation. If this occurs, the CL450 may spuriously report a bitstream underflow (see page 12-26 for a discussion of the buffer underflow interrupt, UND). Also, because the on-going picture-decode operation must be terminated when the bitstream is flushed, the video display may tear if the picture being decoded is a B-picture.
- When bitstream transmission resumes, the host must issue the NewPacket() command(s) which correspond to the beginning of the new bitstream data *before* the CL450 begins to receive it. This requirement is essentially the same as the requirement for beginning to send a bitstream to the CL450 following initialization or Reset().

After the bitstream and pending NewPacket() commands have been flushed and the CL450 has started to receive new bitstream data, the decoding process resumes. How the decoding process behaves is controlled by the **filter** argument of the FlushBitstream() command. The **filter** argument contains two fields:

- *Bit [2]* - This bit determines whether or not the sequence-layer parameters should be restored to their defaults prior to resuming decode. (See Table 10-4 for a definition of the sequence defaults.) Typically, this bit would be set to 1 (causing defaults to be restored) if decoding is to resume in a new video bitstream or in a different sequence within the current bitstream.

  If FlushBitstream() is being used to skip a section within a bitstream, **filter**[2] is usually set to 0. Note however that the quantization matrices in the sequence layer, if present, may change each time a sequence header is incorporated into the bitstream.

  When a bitstream is being accessed randomly, either following execution of FlushBitstream() or immediately following initialization or Reset(), the host must ensure that the correct quantization matrices are provided to the CL450 if the portion of the bitstream containing them is not decoded (see Section 14.2.2, Writing

DRAM-Resident Variables). Note that the GOP-layer defaults (Table 2-4) are *always* restored when FlushBitstream() is issued.

- *Bits[1:0]* - This field is used to specify the point in the bitstream at which the CL450 should resume decoding. The field is encoded as shown in Table 11-3.

Table 11-3 Filter Argument Encoding

filter[1:0]	**Meaning**
00_2	Resume decode/display with next I-picture
01_2	Resume decode/display with first I-picture after the next GOP header
10_2	Resume decode/display only after the next sequence header
11_2	Resume decode/display at the earliest possible time

If a **filter**[1:0] value of 0 or 1 is selected, the host must provide the appropriate sequence-layer parameters (either by setting **filter**[2] or as described in Section 14.2.2, Writing DRAM-Resident Variables).

Alternately, if a **filter**[1:0] value of 2 is selected, the value of **filter**[2] is irrelevant because picture decoding will not resume until a sequence header is found within the bitstream, typically replacing the previous sequence parameters.

InquireBufferFullness()

Format: InquireBufferFullness()
Priority: High
Category: Control

Syntax:

HMEM[0]	InquireBufferFullness	0x8001
HMEM[1]		0x0000
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

When the host issues the InquireBufferFullness() macro command, the CL450 calculates the amount of data currently stored in the bitstream buffer and updates the Buffer Fullness Status location, HMEM[0xb].

Because this is a high-priority command, the microapplication completes updating HMEM *before* clearing HOST_newcmd[0]. Therefore, the host may read the updated data as soon as the microapplication is capable of accepting a new command. Note that HMEM[0xb] will also be periodically updated by the CL450 approximately every eight video lines (see Section 12.1.1, Display-time Interrupt, for exact timing).

The value placed in HMEM[0xb] should be treated as accurate to ±32 words. The uncertainty implicit in this range of values is due to the CL450 hardware moving blocks of data between DRAM, CMEM and the VLC Decoder's working buffer without intervention from the CL450's CPU. Values computed by the microapplication are based on an internal snapshot that may be slightly out of date before the buffer fullness computation is complete.

NewPacket()

Format:	NewPacket (**length**, **timeStamp2**, **timeStamp1**, **timeStamp0**)		
Priority:	Low		
Category:	Control		
Syntax:	HMEM[0]	NewPacket	0x0408
	HMEM[1]	**length**	bits[15:0] = packet length, in bytes (bit 0 always 0)
	HMEM[2]	**timeStamp2**	bit[15] = *Vld* flag bits[14:3] = 0 bits[2:0] = *PTS*[32:30] (or "don't care" if *Vld* is 0)
	HMEM[3]	**timeStamp1**	bit[15] = 0 bits[14:0] = *PTS*[29:15] (or "don't care" if *Vld* is 0)
	HMEM[4]	**timeStamp0**	bit[15] = 0 bits[14:0] = *PTS*[14:0] (or "don't care" if *Vld* is 0)

The NewPacket() command is used by the host to:

- Supply MPEG system-layer bitstream information to the CL450.
- Allow the CL450 microapplication to associate this information with the correct portions of the elementary video bitstream it is receiving as they pass through the circular bitstream buffer located in DRAM.

NewPacket() commands do not "execute" in the same sense as other macro commands; instead, they are queued until the corresponding packet data has been decoded. That is, if a NewPacket() command is given immediately preceding a SetBorder() command, the SetBorder() command does not have to wait until the NewPacket() command "executes."

The microapplication assumes that the use of NewPacket() is *modal* when not in the IDLE command state. NewPacket() is modal because the host must either (1) issue no NewPacket() commands at all or (2) issue NewPacket() commands corresponding to *all* of the bitstream data.

Following execution of the Reset() macro command or CL450 initialization, the host may choose whether or not NewPacket() commands are used by either issuing or not issuing a NewPacket() command prior to transmitting its first word of data to the CL450 as shown in Figure 11-6.

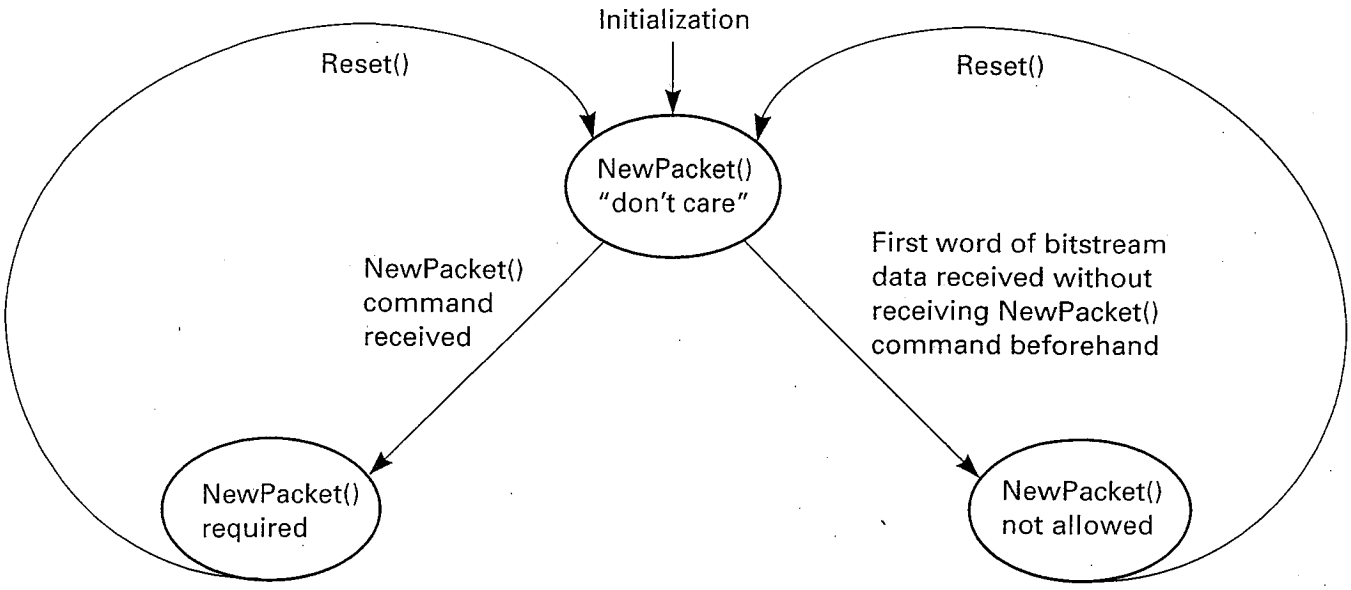


Figure 11-6 CL450's State with Regard to Host NewPacket() Commands

NewPacket() is considered an *optional* command from the point of view of the host. However, the following features are unavailable if NewPacket() is *not* used:

- *The RDY (ready for data) interrupt* : If RDY is not disabled, the microapplication may still produce interrupts to the host, but with indeterminate timing. (See page 12-22.)
- *Audio/video synchronization*: See Chapter 13, Audio/Video Syncronization.

If the NewPacket() macro command *is* used, then the following items are true:

- Any number of NewPacket() commands may be issued *ahead* of the corresponding data arriving at the CL450, as long as the Command FIFO does not overflow (see Section 11.4, Command Latency).
- The NewPacket() command which corresponds to a packet of data must be accepted by the CL450 (HOST_newcmd[0] returned to 0) before the first word of data from that packet enters CMEM *for every packet*.
- The SCR counter must be initialized with the correct value *before* the first NewPacket() command that has HMEM[2][*Vld*] equal to 1 is issued.

11.6.1 length Argument

Each NewPacket() command issued corresponds to a specific portion of the elementary video bitstream, the size of which is specified by the **length** argument. Note that this value is in units of bytes and must be an even number because transfers from the host to CMEM must always be performed one word (two bytes) at a time.

If the NewPacket() command is used, bitstream data received by the CL450 must be preceded by the corresponding NewPacket() command. This means that the sum of the **length** arguments of all NewPacket() commands which have been issued since the command state last changed from IDLE must be greater than or equal to the number of bytes of bitstream data that the CL450 has received. If at any time this condition is not met, the behavior of the CL450 is indeterminate until the command state next becomes IDLE.

While it is typically most convenient for the host to create NewPacket() macro commands using information from the packets in the system-level portion of an MPEG bitstream, the host can also add or remove packet boundaries. In particular, if a host is playing an MPEG elementary video stream and wishes to use the RDY interrupt, the host will have to create NewPacket() commands. In any case, if the host constructs its own packets, the following criteria must *still* be met:

- Packets sent to the CL450 must always have an even **length**.
- Packet **length** arguments must be less than 64K bytes (because the **length** argument is only 16 bits).
- PTSs (if provided) must be encoded consistent with the MPEG requirements for system-level packets and satisfy the buffering constraints of a System Target Decoder (STD).

11.6.2 timeStamp Arguments

The remaining three arguments to NewPacket() make up the PTS associated with the data corresponding to the NewPacket() command. Note that the PTS is *optional* for each particular NewPacket() command which is issued. Although, if the CL450 is to perform audio/video synchronization (see Chapter 13), at least one PTS must be supplied. The host indicates that a valid PTS is provided with a NewPacket() command by setting the *Vld* bit (HMEM[2][15]) to 1.

Note that, as in the MPEG standard, the PTS is broken into three fields, the first two being 15 bits long, the third field being three bits long. For more information on how the CL450 uses PTSs, see Chapter 13.

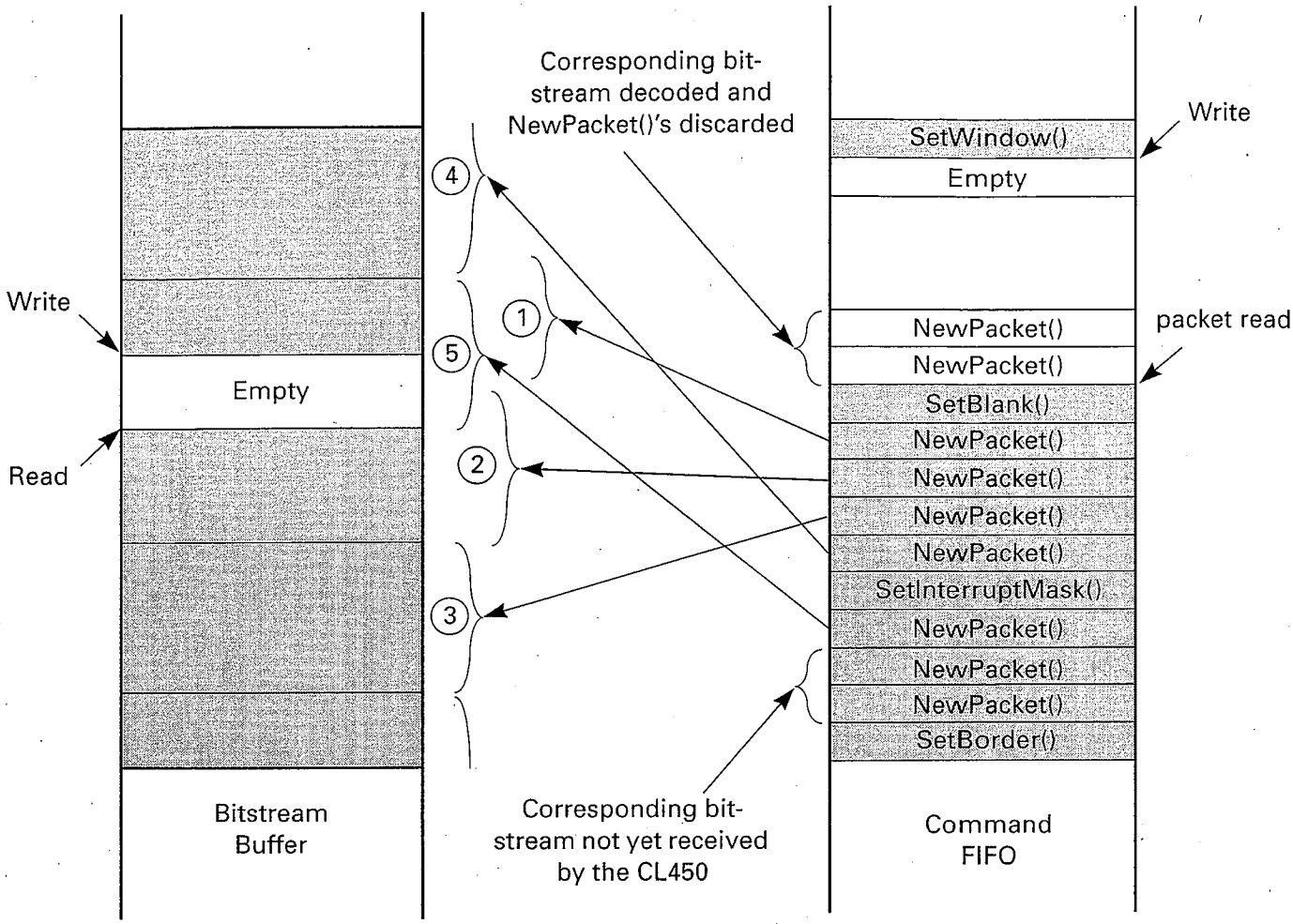
11.6.3 Loading and Removing NewPacket Commands

NewPacket() commands are placed in the Command FIFO as they are issued by the host. The CL450 buffers the commands in the Command FIFO and then applies the information found in each NewPacket() command to the number of coded data bytes specified by the **length** portion of the command.

NewPacket() commands are removed from the Command FIFO in groups each time a `picture_start_code` is encountered while decoding the bitstream.

An example of the relationship between NewPacket() commands stored in the Command FIFO and the corresponding data in the bitstream buffer is shown in Figure 11-7.

NewPacket()



(1) Bitstream decoded but NewPacket() not yet discarded
(2) Bitstream partially decoded
(3) and (4) Bitstream resident in buffer; waiting for decode
(5) Packet has not been entirely received by the CL450

Figure 11-7 Packet Representation in Command FIFO vs. Bitstream Buffer

Pause()

Format:	Pause()		
Priority:	Low		
Category:	Play-type		
Syntax:	HMEM[0]	Pause	0x000e
	HMEM[1]		0x0000
	HMEM[2]		0x0000
	HMEM[3]		0x0000
	HMEM[4]		0x0000

When the Pause() macro command is executed, the microapplication enters the PAUSE state, passing through the PLAY-SETUP and PLAY states if necessary. In the PAUSE state, the last picture is kept on the display, and the CL450 continues to process macro commands as if it were in the IDLE state (i.e., without waiting for a picture decode to complete).

While in the PAUSE state, no bitstream decoding is performed. Because of this, the host must ensure that the bitstream buffer does not overflow either by (1) ending the transmission of coded data to the CL450 or by (2) periodically executing the FlushBitstream() macro command (see page 11-16). Upon execution of another Play-type macro command, decoding will resume with the next unused bit from the bitstream buffer.

Pictures are displayed while in the PAUSE state in the manner appropriate to how they were decoded. Still pictures decoded while in the STILL state are displayed with data from two separate DRAM picture buffers in the even and odd display fields, while pictures decoded in other states are displayed with data from one DRAM picture buffer displayed in both the even and odd display fields.

Play()

Format: Play()
Priority: Low
Category: Play-type

Syntax:

HMEM[0]	Play	0x000d
HMEM[1]		0x0000
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

When the Play() macro command is executed, the microapplication enters the PLAY state (passing through PLAY-SETUP if necessary). In this state, the CL450 decodes and displays pictures at the rate prescribed by the `picture_rate` parameter contained within the bitstream, and uses the time stamp information transmitted by the NewPacket() command, if any, to synchronize the video display to the SCR.

While the microapplication is in the PLAY state, bitstream data is consumed at the normal rate. For systems in which the CL450 is supplied with bitstream data at a fixed rate, the host does not need to allow for bitstream buffer overflow, assuming that the bitstream is being transferred at the rate for which it was encoded. Note that the host must still prevent CMEM overflow if programmed access (rather than DMA) is being used (see Section 9.2, Bitstream Transfer Process).

All audio/video synchronization options may be used with the Play() command (see Chapter 13).

The Play() macro command should only be executed when the microapplication is not in the PLAY command state.

Reset()

Format: Reset()
Priority: High
Category: Control

Syntax:

HMEM[0]	Reset	0x8000
HMEM[1]		0x0000
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

Reset() is a high-priority macro command which is used to re-initialize the CL450 and its microapplication. When this command is executed:

- The contents of the bitstream buffer, the Command FIFO, and the picture buffers are lost.
- The video display process is re-initialized and the output window blanked (screen is filled with current border color).
- The default settings given in Tables 10-1 and 10-3 are restored.

This is the only high-priority macro command which has an effect on the command processing state.

The Reset() macro command is typically used only to recover from error conditions. When suspending and resuming decode operations, or when changing from decoding one bitstream to another, some combination of the FlushBitstream() (page 11-16), Pause() (page 11-25), and SetBlank() (page 11-29) commands should be used. Unlike the Reset() command, these commands allow the CL450 to continue to receive bitstream data and to continue to display the last picture decoded.

Note that, unlike the other high-priority commands, the Reset() macro command has *not* been completely executed when HOST_newcmd[0] is cleared (see Section 10.2.2, Loading Sequence). Also, the Reset() command must not be executed while the host has a DRAM semaphore (SEQ_SEM or PIC_SEM) allocated.

Scan()

Format:	Scan()		
Priority:	Low		
Category:	Play-type		
Syntax:	HMEM[0]	Scan	0x000a
	HMEM[1]		0x0000
	HMEM[2]		0x0000
	HMEM[3]		0x0000
	HMEM[4]		0x0000

When the Scan() macro command is executed, the microapplication enters the SCAN state, possibly passing through the PLAY-SETUP and PLAY states. While in the SCAN state, a single I-picture from the incoming bitstream is decoded and stored in the CL450's DRAM.

> *Note: Whether or not NewPacket() commands are being used, audio/video synchronization has no effect on the timing of picture decoding which occurs in the SCAN state.*

Once decoding is complete, the microapplication will automatically transition to the PAUSE state (see the Pause() command, page 11-27) and issue the SCN interrupt, if enabled (see page 12-24).

Coded data passed to the CL450 before an I-picture is either decoded normally (sequence and GOP header information) or discarded (P- and B-pictures). Once the first I-picture has been decoded, the microapplication stops processing the bitstream until some other macro command causes the microapplication to leave the PAUSE state.

The decoded I-picture is stored in one of the picture buffers in the CL450's DRAM and is immediately posted for display; it remains in DRAM until the next Play-type macro command is executed.

SetBlank()

Format:	SetBlank(**state**)		
Priority:	Low		
Category:	Set-type		
Syntax:	HMEM[0]	SetBlank	0x030f
	HMEM[1]	**state**	bits[15:1] = 0 bit[0] = 1 (output window blanked) or 0 (output window not blanked)
	HMEM[2]		0x0000
	HMEM[3]		0x0000
	HMEM[4]		0x0000

The SetBlank() macro command is used to blank and unblank the output window. The **state** argument determines whether the output window should be blanked (1) or unblanked (0). If the output window is unblanked, then the microapplication resumes displaying decoded pictures. If the output window is to be blanked, the output window is filled with the border color.

Note that blanking the output window has no effect on the border color, and the decoding process continues as though the output window were not blanked.

Once a SetBlank(1) has been executed, the display is blanked after the first VSYNC following execution of the command. After a SetBlank(0) has been executed, the display is unblanked after the first VSYNC after the command is executed.

SetBorder()

Format: SetBorder(**leftBorder, topBorder, rBorder, gbBorder**)
Priority: Low
Category: Set-type

Syntax:			
	HMEM[0]	SetBorder	0x0407
	HMEM[1]	**leftBorder**	bits[15:10] = 0 bits[9:0] = distance from active $\overline{\text{HSYNC}}$ to left edge of output window, in VCLK periods (min. 10)
	HMEM[2]	**topBorder**	bit[15] = 0 bits[14:0] = distance from active VSYNC to top edge of output window, in $\overline{\text{HSYNC}}$ periods
	HMEM[3]	**rBorder**	bits[15:8] = 0 bits[7:0] = Red or Cr (=Cb) component of border color
	HMEM[4]	**gbBorder**	bits[15:8] = Green or Y component of border color bits[7:0] = Blue or Cb (=Cr) component of border color

The SetBorder() macro command is used to set:

- The color of the border
- The size of the top and left borders around the output window

Together with the SetWindow() command and the $\overline{\text{HSYNC}}$ and VSYNC input signals, the SetBorder() command is also used to determine the right and bottom borders as well as the size of the active window. The new values for both commands take effect on the active edge of VSYNC following command execution.

Figure 11-8 shows the relationship of the SetBorder() command's four arguments to the arguments of the SetWindow() macro command (**xOffset**, **yOffset**, **width**, **height**) and the configuration of the CL450's display output.

rBorder and **gbBorder** are 16-bit integers that determine the border color. The border color specified is encoded either in the RGB color space (if the CL450's color space converter is enabled; see page 11-34)

or the YCbCr color space. Note that both chrominance values must be the same (Cb = Cr) if the color space converter is not used.

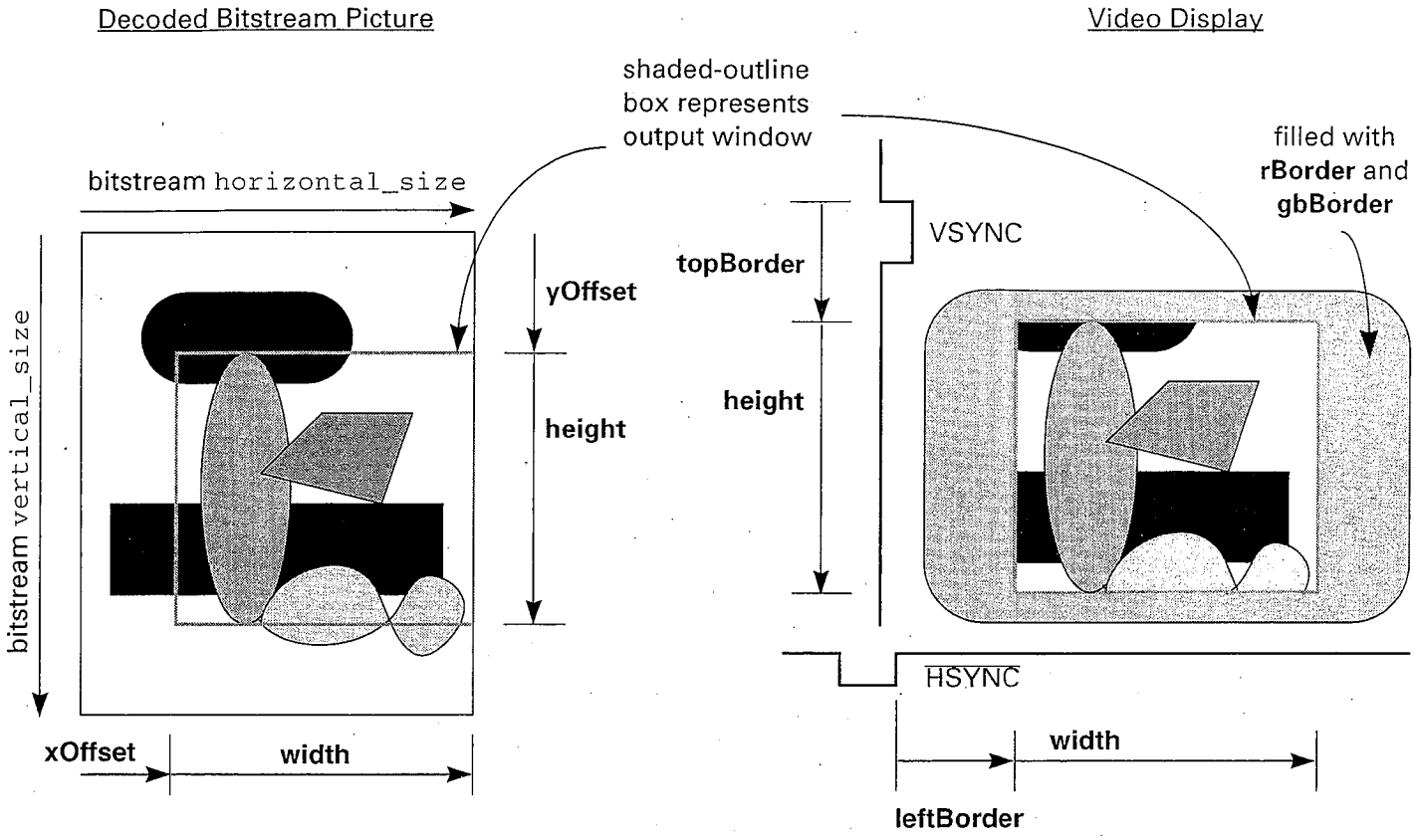


Figure 11-8 Video Display Configuration

The **leftBorder** and **topBorder** arguments are encoded specially, so that a value of 0 in either causes the CL450 to automatically set the corresponding parameter such that the output window is centered within the video display.

Generally, auto-centering should not be used if one extent of the decoded picture is greater than the maximum output window size. If it is, then auto-centering causes the output window to be expanded to its maximum size in the corresponding direction and the corresponding offset (**xOffset** or **yOffset**) to be set by the SetWindow() command. **xOffset** will be computed so the horizontal center of the decoded image is displayed, and **yOffset** will be forced to 0.

SetBorder()

Auto-centering is performed using the current display resolution set by the SetVideoFormat() macro command (see page 11-40). The microapplication also assumes that the correct **leftBorder** value to center a maximum-width output window is 60, and that the correct **topBorder** value to center a maximum-height output window is 18. If these values are not correct for the VSYNC and $\overline{\text{HSYNC}}$ timing used by the video display, auto-centering cannot be used.

In general, the host must ensure that the dimensions of the display window do not exceed the dimensions of the decoded pictures, and that the display window is not positioned on the screen in such a way that the CL450 does not attempt to complete a B-picture display before B-picture decoding is guaranteed to complete (see Section 15.3.2.)

The units and minimum/maximum values of the **leftBorder** and **topBorder** arguments are shown in Table 11-4.

Table 11-4 SetBorder() Argument Restrictions: leftBorder and topBorder

		Value		
Argument	**Units**	**Minimum**	**Maximum**	**Equal to 0 Means:**
leftBorder	1 display pixel	10	0x3ff	Enable horizontal auto-centering
topBorder	1 source pixel	6	0x7fff	Enable vertical auto-centering

The size of the bottom and right border areas are determined by the corresponding video timing signal, opposite border size, and corresponding output window size as shown in Table 11-5 below. Note that the video "Borders" defined in the table are the sum of the border and blanking times in the timing of the CL450's output. The actual size of the border seen on the video display varies depending on the relationship between the individual monitor's timing and display area.

Table 11-5 Video Border Sizes

Border	**Size**
Left Border	**leftBorder**
Right Border	$\overline{\text{HSYNC}}$ period - (**leftBorder** + **width**)
Top Border	**topBorder**
Bottom Border	VSYNC period - (**topBorder** + **height**)

There are circumstances in which the CL450 does not decode all of the picture area specified by the `picture_height` and `picture_width` parameters; however, the following rules always apply:

- The full width of the coded picture is always decoded.
- Only the first 330 or 396 macroblocks (for NTSC or PAL video output formats, respectively) of any picture are decoded, regardless of the picture size.

Note that when vertical auto-centering is used, the limited decoded picture height is accounted for, but not necessarily the picture decoding time requirements.

For additional information on the timing of the SetBorder() macro command, see the description of the SetWindow() macro command on page 11-42 .

SetColorMode()

Format: SetColorMode(**mode**)
Priority: Low
Category: Set-type

Syntax:

HMEM[0]	SetColorMode	0x0111
HMEM[1]	**mode**	bits[15:1] = 0 bit[0] = 1 (RGB) or 0 (YCbCr)
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

The SetColorMode() macro command is used to enable and disable the CL450's color-space converter; it configures the CL450 for either RGB or YCbCr video output. (The color-space converter operates using the method and coefficients included in the description of the VID_sela and VID_selb registers in Section 8.5.2, Indirect Video Registers.)

The **mode** argument to SetColorMode() is a 16-bit parameter whose only allowed values are as follows:

- ***mode 0:*** Puts the CL450 into YCbCr mode by disabling the color-space converter. Decoded pixel values are output with a Y component value output on PD[15:8] every VCLK, Cb and Cr are alternately output on PD[7:0], and the contents of VID_selaux are output on PD[23:16].
- ***mode 1:*** Puts the CL450 into RGB mode by enabling the color-space converter. Converts each decoded pixel into R, G, and B components, which are output on the PD[7:0], PD[15:8], and PD[23:16] pins, respectively.

This command affects the encoding of the border color selected using the SetBorder() macro command (see page 11-30); the Cr and Cb components of the border color must be equal if the color-space converter is disabled.

SetColorMode()

The SetColorMode() command *must* be used to enable and disable the CL450's color-space converter because the microapplication initializes the VID_selmode, VID_sela, and VID_selb registers when execution begins, and because these registers cannot be accessed by the host while the microapplication is executing.

SetInterruptMask()

Format: SetInterruptMask(**mask**)
Priority: Low
Category: Set-type

Syntax:			
	HMEM[0]	SetInterruptMask	0x0104
	HMEM[1]	**mask**	bits[15:12] = 0 bits[11:0] = interrupt enable bits
	HMEM[2]		0x0000
	HMEM[3]		0x0000
	HMEM[4]		0x0000

The SetInterruptMask() macro command is used to enable and disable CL450 interrupts to the host. Each bit in the argument to this command is either reserved or corresponds to a logical interrupt which the CL450 can produce. The assignment of **mask** bits is given in Table 11-6.

Table 11-6 Mask Bit Assignments

Mask Bit[1]	Interrupt Name	Category	Event	Page
10	RDY	Display-time	Ready for data	12-22
5	END-D		`sequence_end_code` found	12-15
0	ERR		Bitstream data error	12-17
6	PIC-D	Decode-time	New picture decoded	12-20
9	SEQ-D		`sequence_header_code` found	12-25
11	SCN		Picture decode complete in SCAN state	12-24
8	UND		Bitstream buffer underflow error	12-27
4	END-V		Last picture display before `sequence_end_code`	12-16
2	GOP	VSYNC	First I-picture display after `group_start_code`	12-19
1	PIC-V		New picture display	12-21
3	SEQ-V		First I-picture display after `sequence_header_code`	12-26

1. Interrupt bits 7 and 12-15 are reserved and must be written with zero only.

Argument bits which are reserved must always be 0 when the SetInterruptMask() command is issued or indeterminate behavior will result.

A new **mask** value does not take effect until the SetInterruptMask() command is executed (*not* accepted), and controls events *only* at the time the microapplication internally recognizes interrupt events. Several interrupts are delayed between the time when the interrupt event is recognized and the time when the interrupt is issued to the host. (See the rules governing command latency in Section 11.4.)

In the case in which the host is enabling new interrupts with SetInterruptMask(), interrupt events which occur between the time the command is accepted and the time it is executed are not recognized, and therefore no corresponding interrupts are produced.

In the case in which the host disables a previously enabled interrupt, the microapplication issues interrupts for any events which are recognized before SetInterruptMask() is executed, even those which are not issued immediately after being recognized. This case is discussed more fully in Chapter 12, Interrupts.

SetThreshold()

Format: SetThreshold(**level**)
Priority: Low
Category: Set-type

Syntax:

HMEM[0]	SetThreshold	0x0103
HMEM[1]	**level**	bits[15:0] = bitstream buffer emptiness, in bytes
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

The SetThreshold() macro command is used to specify the minimum number of bytes that must be empty in the DRAM bitstream buffer to cause the CL450 to issue the RDY interrupt (see page 12-22). The bitstream buffer threshold is checked when a new value for Buffer Fullness Status (HMEM[0xb]) is *automatically* computed.

The **level** argument is a 16-bit integer that specifies the number of empty bytes that must be exceeded to cause the generation of the RDY interrupt. **Level** must be a value (in bytes) less than the size of the bitstream buffer. The default value of **level** is 4096 bytes, so that the CL450 will produce RDY interrupts as long as there are more than 4096 empty bytes in the bitstream buffer and the other conditions necessary for the production of the RDY interrupt remain true.

The computation of the contents of the bitstream buffer which is compared against **level** is accurate to ±32 bytes (see page 11-19). However, the fullness of the bitstream buffer is only checked following approximately every eight active scan lines displayed, so the bitstream buffer may become considerably more "empty" before an interrupt is produced. (For a more detailed description of the timing with which buffer fullness is recomputed, see Section 12.1.1, Display-time Interrupt.)

Typically, a **level** value which is slightly less than the intended burst transfer size (possibly a multiple of the nominal average packet size) would be used. Such values range from below 2048 to below 10240, but **level** must be less than the size of the bitstream buffer and greater than 255.

Note that the internal variable which is used to generate the RDY interrupt is changed when this command is *executed*, not issued. If **level** is changed dynamically, the host must be aware of the latency between the CL450 accepting a SetThreshold() command and the time at which it is executed. Also, once the command is executed, it affects the *future* generation of the RDY interrupt. If an RDY interrupt is already pending when this command is issued, it will not be cleared by the CL450.

SetVideoFormat()

Format: SetVideoFormat (**format**)
Priority: Low
Category: Set-type

Syntax:

HMEM[0]	SetVideoFormat	0x0105
HMEM[1]	**format**	bits[15:3] = 0 bits[2:0] = 3 (PAL) or 4 (NTSC) *only*
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

The SetVideoFormat() macro command is used to configure the output resolution and timing of the CL450's video bus. Values 3 and 4 of the **format** argument correspond to PAL and NTSC resolution and timing values, respectively. The quantities which are set by this command are given in Table 11-7.

Table 11-7 Video Format Summary

Parameter[1]	3 (PAL) Value	4 (NTSC) Value
Maximum Horizontal Resolution	352 pixels	352 pixels
Maximum Vertical Resolution	288 pixels	240 pixels
Nominal VSYNC Frequency	50 Hz	60 Hz

1. All resolutions in this table are given in MPEG source pixels, *not* the unsampled (interpolated) output pixels.

The resolution values are used by the CL450 when SetWindow() and SetBorder() commands are executed (see page 11-42 and page 11-30, respectively) or when new sequence-layer parameters are processed (see Section 14.1.1, Sequence Variable Group). The VSYNC frequency is used in conjunction with the `picture_rate` sequence-layer parameter to determine if frame rate conversion must be performed (see Section 15.1, Frame Rate Conversion).

The new VSYNC frequency takes effect immediately after the command is executed. However, if auto-sizing (see SetWindow() on page 11-42) or auto-centering (see SetBorder() on page 11-30) are being used, then SetBorder() and/or SetWindow() must be explicitly executed by the host, or new sequence header information processed, for the new display resolution to take effect.

SetWindow()

Format: SetWindow (**xOffset, yOffset, width, height**)
Priority: Low
Category: Set-type

Syntax:

HMEM[0]	SetWindow	0x0406
HMEM[1]	**xOffset**	bit[15] = 0 bits[14:0] = horizontal offset of output window start from picture start[1]
HMEM[2]	**yOffset**	bit[15] = 0 bits[14:0] = vertical offset of output window start from picture start
HMEM[3]	**width**	bit[15] = 0 bits[14:0] = horizontal size of output window
HMEM[4]	**height**	bit[15] = 0 bits[14:0] = vertical size of output window

1.See Table 11-8 for argument units.

The SetWindow() macro command is used in conjunction with SetBorder() (see page 11-30) to determine the portion of the decoded image that the CL450 displays in the video window.

Figure 11-9 shows the relationship of this command's four arguments to the configuration of the CL450's display output. Note that the quantities **leftBorder**, **topBorder**, **rBorder**, and **gbBorder** shown in the figure are the arguments of the SetBorder() command.

The **width** and **height** arguments are encoded specially, so that a value of 0 in either causes the CL450 to automatically set the corresponding parameter from the `_size` fields in the sequence-layer parameters (i.e., the output window becomes the full size of the decompressed picture).

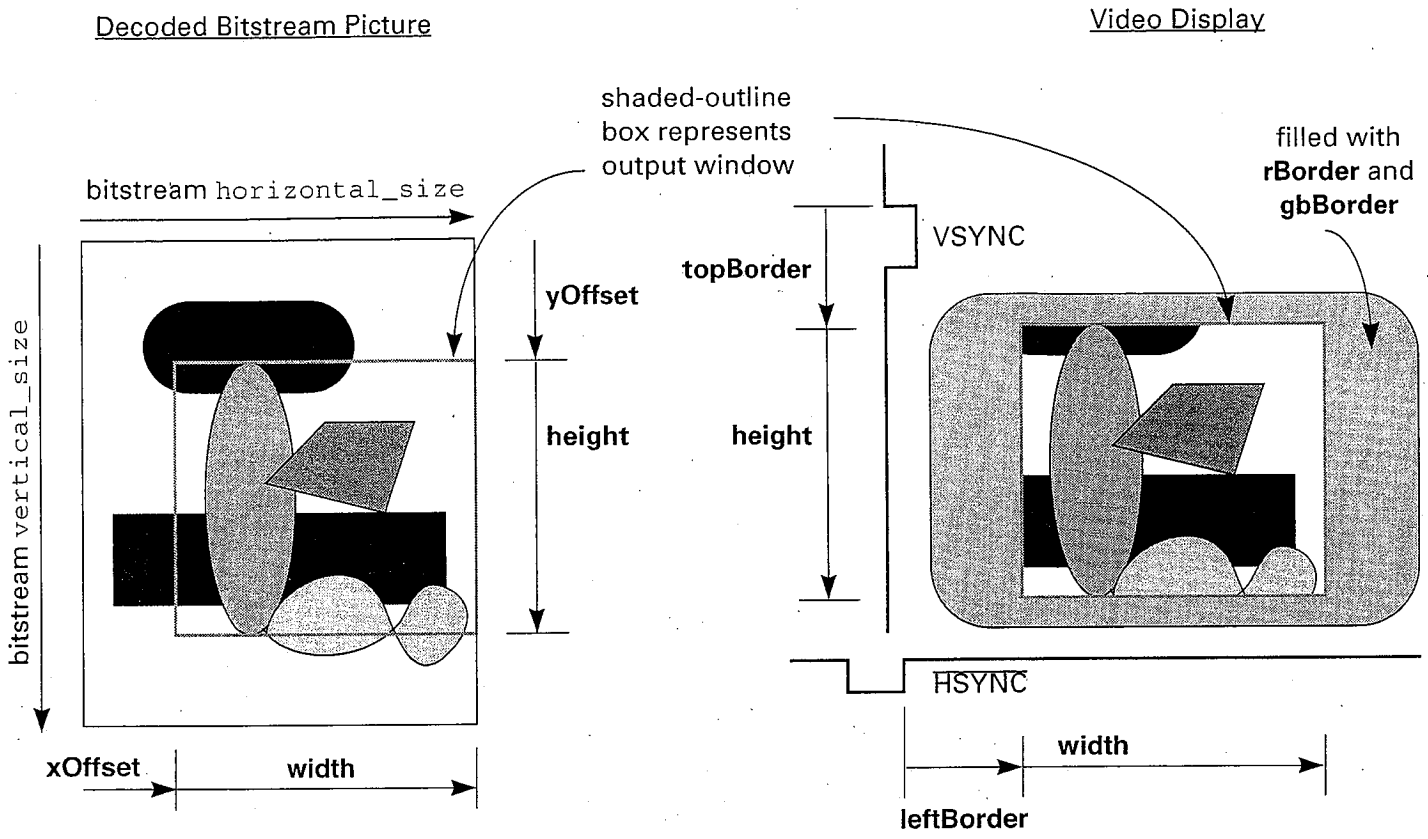


Figure 11-9 Video Display Configuration - SetWindow()

The host must ensure that the combination of the arguments to this command does not cause areas outside of the decoded picture to be displayed. If the host provides arguments which specify that part of the output window falls outside of the decoded picture, then indeterminate data will appear in that portion of the output window in the video display. To avoid this occurrence, use argument values less than or equal to the maximums given in Table 11-8.

The arguments to SetWindow() take effect upon the first active VSYNC edge after the command has executed.

SetWindow()

Table 11-8 SetWindow() Argument Restrictions

Argument	Unit	Recommended Maximum Value
xOffset	1 source pixel	`horizontal_size` -1
yOffset	1 source pixel	`vertical_size` -1
width	1 display pixel	Smallest of: □ 2 * (`horizontal_size` – **xOffset**) □ 704
height	1 source pixel	Smallest of: □ `vertical_size` – **yOffset** □ Maximum Vertical Resolution, see Table 11-7

Note that the arguments to the SetWindow() command use units of one *source* pixel for offsets within the *decoded picture* and for all vertical units, and one *display* pixel (half a source pixel) for the horizontal dimension of the *video display*, as shown in Table 11-8. Pixel dimensioning is done this way because each pixel in the decoded picture (source pixel) is horizontally interpolated upon display to provide two display pixels. Thus, the **xOffset** within the picture is in source pixels (pre-interpolation), while the left display border and the width of the output window are in display pixels (post-interpolation).

SingleStep()

Format: SingleStep()
Priority: Low
Category: Play-type

Syntax:

HMEM[0]	SingleStep	0x000b
HMEM[1]		0x0000
HMEM[2]		0x0000
HMEM[3]		0x0000
HMEM[4]		0x0000

When the SingleStep() macro command is executed, the microapplication enters the STEP state, possibly passing through the PLAY-SETUP and PLAY states. While in the STEP state, a single picture from the incoming bitstream is decoded and posted for display.

> *Note: Audio/video synchronization has no effect on the timing of picture decoding which occurs in the STEP state, regardless of whether or not NewPacket() commands are being used.*

Once decoding is complete, the microapplication automatically transitions to the PAUSE state (see the Pause() command, page 11-25).

Note that the picture most recently decoded is not necessarily the one which is displayed while in the PAUSE state. Because of the picture reordering which is performed after I- and P-pictures are posted for display, the picture which the SingleStep() command caused to be decoded can be stored within the CL450's DRAM, and another picture (earlier in the display order) can be displayed instead. This situation is particularly apparent when SingleStep() is issued while in the IDLE command state. In this case (for most bitstreams), two pictures must be decoded before the output window is unblanked and the first picture decoded is displayed. Typically, one picture will be decoded in the PLAY state, and the other one in the STILL state.

SlowMotion()

Format:	SlowMotion(**speed**)		
Priority:	Low		
Category:	Play-type		
Syntax:	HMEM[0]	SlowMotion	0x0109
	HMEM[1]	**speed**	bits[15:4] = 0 bits[3:0] = 2 through 8, *only*
	HMEM[2]		0x0000
	HMEM[3]		0x0000
	HMEM[4]		0x0000

When the SlowMotion() macro command is executed, the microapplication enters the SLOW state, possibly passing through the PLAY-SET-UP and PLAY states. In the SLOW state, pictures are decoded and displayed at a slower rate than prescribed by the `picture_rate` parameter contained within the bitstream. The decode and display rate is:

$$\texttt{picture\_rate} \times \frac{1}{\textbf{speed}}$$

where the **speed** argument to the SlowMotion() command ranges from 2 through 8. This command may be issued to change the display rate as frequently as desired.

Although the host must always ensure that neither the bitstream buffer nor its own buffers overflow, special attention must be given here since *the CL450's rate of coded data consumption drops while decoding is performed in the SLOW state*. Audio/video synchronization has no effect on the rate of picture decoding while in the SLOW state.

If synchronization has been performed prior to the execution of the SlowMotion() command (see Chapter 13, Audio/video Synchronization), synchronization may be terminated using the Reset() macro command (see page 11-27) prior to SlowMotion(). Alternately, the

AccessSCR() macro command and/or the FlushBitstream() macro command may be used to correct the value of the SCR counter before transitioning back to the PLAY state.

When in the SLOW state, every decoded picture is displayed for two times the **speed** argument field times. This function overrides both the microapplication frame rate transcoding mechanism and audio/video synchronization.

Normally, the number of fields for which each decoded picture is displayed varies from 1 to 3 and is determined dynamically by the ratio of the `picture_rate` coded in the bitstream and the current video output format, controlled by the SetVideoFormat() macro command. While in the SLOW state, this comparison is disregarded, and the nominal value of 2 field periods per picture for normal rate display is assumed. Because of this, the actual play rate while in the SLOW state is derived from multiplying the **speed** argument times one of the values from Table 11-9.

Table 11-9 Picture Decode Play Rate while in SLOW State

Format	`picture_rate`	Scale Factor
NTSC (30 Hz)	30Hz / 29.97Hz	1.00
NTSC (30 Hz)	25 Hz	.83
NTSC (30 Hz)	24 Hz / 23.97 Hz	0.80
PAL (25 Hz)	30 Hz / 29.97 Hz	1.20
PAL (25 Hz)	25 Hz	1.00
PAL (25 Hz)	24 Hz / 23.97 Hz	0.96

For example, requesting a speed of 4 using SlowMotion() when displaying a bitstream coded at 25 Hz on a 30 Hz display will actually play at 0.83 x 4, or 3.32 times slower than normal, rather than 4.00 times slower.

12
Interrupts

The 11 logical host interrupts which the CL450 microapplication generates are produced within the microapplication by polling internal conditions while the decoding and display processes execute. Because of this, the times at which interrupts can be produced (the domain for interrupts) is restricted. For example, the CL450 microapplication will *not* produce new interrupts in any of the following circumstances:

- The microapplication is not executing (CPU_control[0] == 0).
- The microapplication is in the IDLE state.
- The SetInterruptMask() macro command has executed with a **mask** argument of 0.
- The Interrupt Status location of HMEM is non-zero.

Note: Although new interrupts ($\overline{INT}$ pin transitioning to low) cannot be generated during any of these conditions, it is possible for the $\overline{INT}$ pin to remain active during these conditions. Once the $\overline{INT}$ pin is active, it will remain active until forced inactive by the host (see Section 12.2, Interrupt Handshaking).

SetInterruptMask(), used to enable and disable CL450 interrupts to the host, is a low-priority macro command. Rules given in Chapter 11, Macro Commands, explain how to determine the timing delay before a low-priority command is executed. Note that new **mask** bits won't be effective until SetInterruptMask() is *executed*, not just *accepted*.

12.1 Interrupt Types

Interrupts are divided into the following three separate classes based on when they are reported to the host:

- Display-time Interrupt
- Decode-time Interrupts
- VSYNC Interrupts

Each interrupt is referenced by the event which causes it in Table 12-1 below, and is described in detail in Section 12.3.

Table 12-1 CL450 Interrupt Summary

Category	Interrupt Name	Event	Mask Bit[1]	Page
Display-time	RDY	Ready for data	10	12-22
Decode-time	END-D	`sequence_end_code` found	5	12-15
	ERR	Bitstream data error	0	12-17
	PIC-D	New picture decoded	6	12-20
	SEQ-D	`sequence_header_code` found	9	12-25
	SCN	Picture decode complete in SCAN state	11	12-24
	UND	Bitstream buffer underflow error	8	12-27
VSYNC	END-V	Last picture display before `sequence_end_code`	4	12-16
	GOP	First I-picture display after `group_start_code`	2	12-19
	PIC-V	New picture display	1	12-21
	SEQ-V	First I-picture display after `sequence_header_code`	3	12-26

1. Interrupt bits 7 and 12-15 are undefined and must be set to zero only.

Note: The CL450's internal CPU can also receive interrupts from on-chip sources. The host has no control over these interrupts, and they should not be confused with the host interrupts generated by the CL450.

12.1.1 Display-time Interrupt

The single Display-time interrupt may be issued any time the microapplication is in a command processing state other than IDLE and PLAY-SETUP. In all cases, the event which causes a Display-time interrupt to be issued is the active (falling) edge of the $\overline{\text{HSYNC}}$ input to the CL450. Which $\overline{\text{HSYNC}}$s can produce Display-time interrupts is determined differently in each of the three different sections of the field display (VSYNC) period: the top border, vertical active region, and bottom border. Figure 12-1 shows these three sections in typical NTSC vs. PAL display environments.

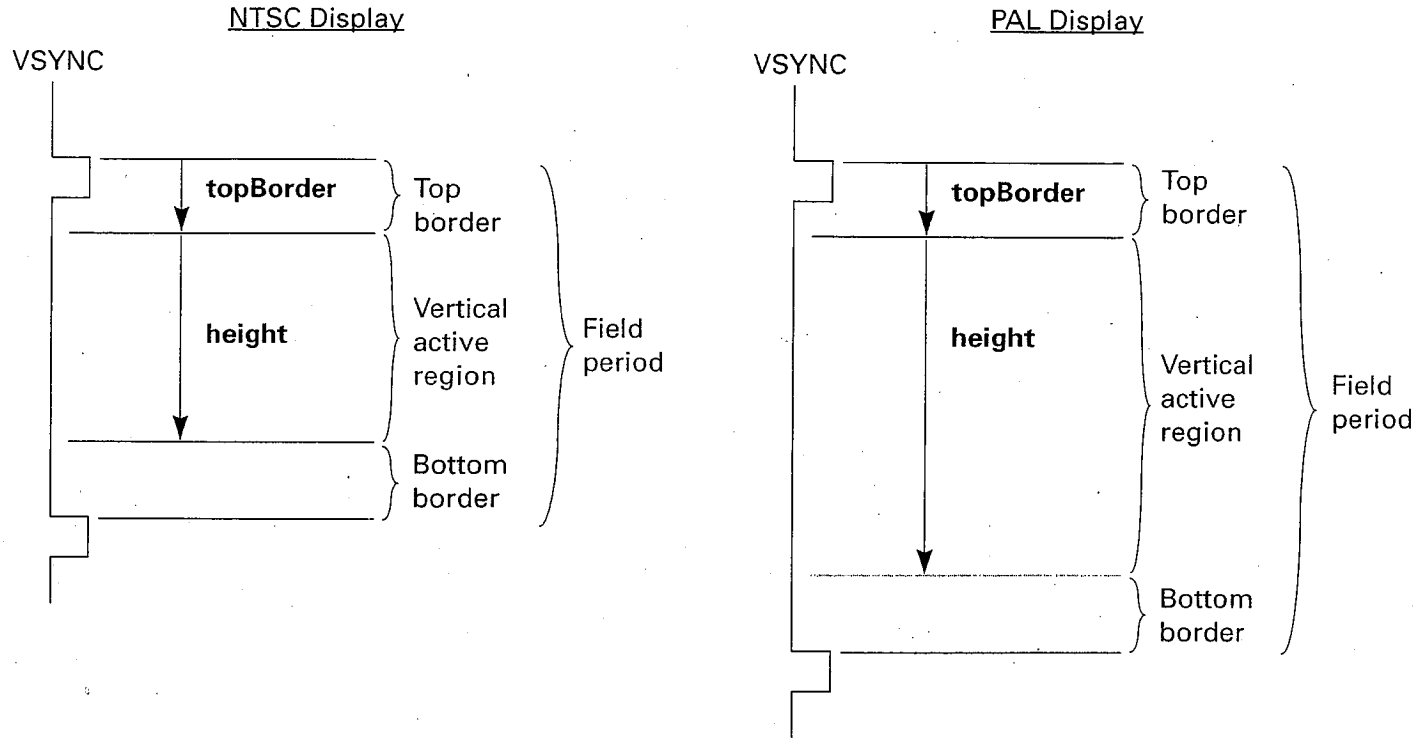


Figure 12-1 VSYNC Field Period Display for NTSC vs. PAL Environments

Within the top border, Display-time interrupts *may* be generated roughly every eighth $\overline{\text{HSYNC}}$. Within the vertical active region, Display-time interrupts *may* be generated at the end of each block row displayed, as shown in Figure 12-2. That is, after the display of the last scan line of a decoded block row, the microapplication checks to see if the Display-time conditions are true. This check is done on block row boundaries, which means after the last (eighth, or seventh if counting from 0) scan line of one block row and before the first scan line of the next within the decoded picture.

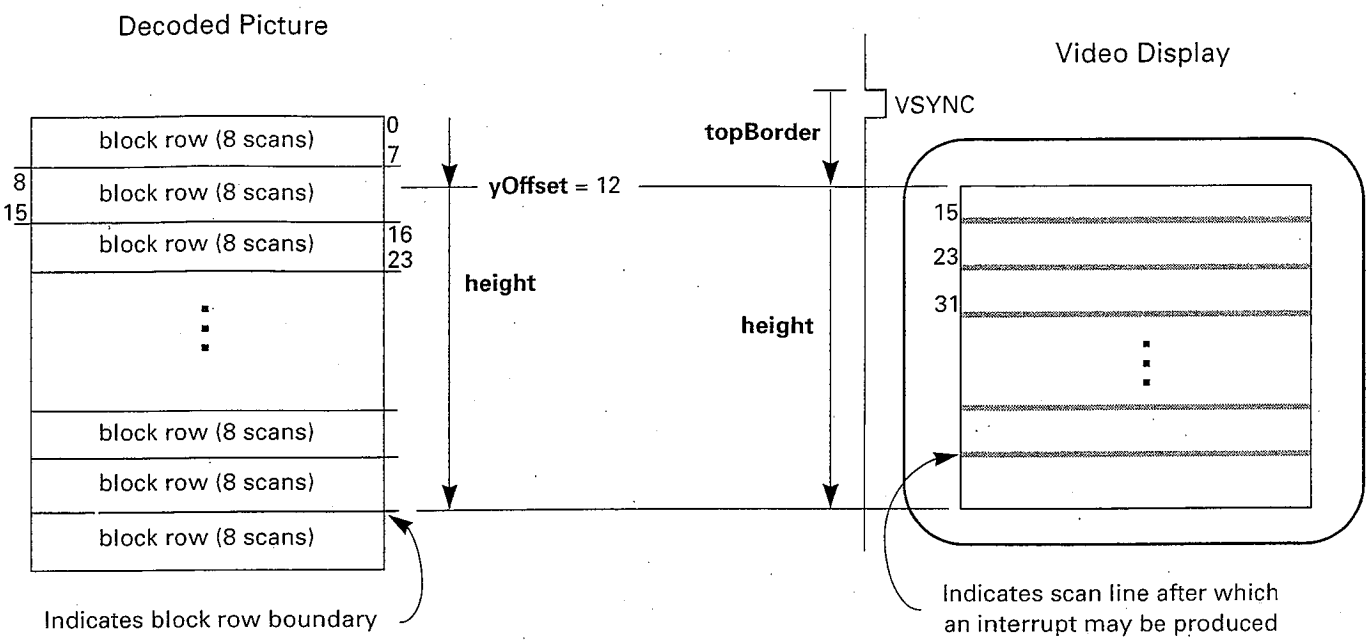


Figure 12-2 Display-time Interrupt Generation

Top Border

The internal counter used to determine when Display-time interrupts are to be produced is initialized at the active edge of VSYNC with the **topBorder** value. Subsequently, each time an active $\overline{\text{HSYNC}}$ occurs, the counter is decremented, and if the least-significant three bits of the result are 0, then a Display-time interrupt may be produced. Thus the first possible time a Display-time interrupt can be produced in the top border will be

(**topBorder** & 7) + 1

active $\overline{\text{HSYNC}}$ edges after active VSYNC. The last possible Display-time interrupt in the top border will always occur on the $\overline{\text{HSYNC}}$ edge which marks the end of the last top border scan line and the beginning of the first scan line of the vertical active region.

The block row boundary within the decoded picture which causes the first possible interrupt is dependent only on **yOffset** of the SetWindow() macro command.

Because of selected **yOffset**, the block row boundaries of the decoded picture are not necessarily the same as scan line numbers within the video display. In fact, the output window could begin in the middle of a decoded picture block row as in the case of **yOffset**=12 of Figure 12-2, which causes the first possible interrupt to be delayed from the top of the output window by "8-(**yOffset**%8)" (4) scan lines. (Note that scan line numbers start at 0.)

Bottom Border

Within the bottom border, the number and location of opportunities for issuing Display-time interrupts depends on the **height** of the active display region and the current video display **mode**. As in the top border region, an internal counter is used to determine when and how many Display-time interrupts may be issued during the bottom border. This counter is initalized on the active $\overline{\text{HSYNC}}$ edge which separates the last scan line of the vertical active period from the first bottom border scan line, with the value:

255 - (**topBorder** + **height**)

If the initial value is less than or equal to 0, then no Display-time interrupts will be generated during bottom border. Otherwise, the counter is decremented on subsequent active $\overline{\text{HSYNC}}$ edges, and interrupts may be produced each time the least significant three bits are 0. The last Display-time interrupt during bottom border will occur when the counter decrements to 0, after which no Display-time interrupts will be generated until an active VSYNC edge is detected and the sequence begins again.

> *Note: For proper operation of the video display, the active VSYNC edge must occur after this counter decrements to 0. This creates an effective minimum field period requirement.*

When the video display is blanked, either because the first picture after leaving the IDLE state has not yet been displayed or due to the SetBlank() macro command, the entire field period is treated as if it were the bottom border and the sum of the **height** and **topBorder** parameters was 15. In this case, there will be exactly 30 possibilities for Display-time interrupts to be issued (assuming a normal field period is used), and

the first opportunity will occur 8 active $\overline{\text{HSYNC}}$ edges after each active VSYNC edge.

12.1.2 Decode-time Interrupts

Unlike the Display-time interrupt, which the microapplication periodically polls for, Decode-time interrupts are issued every time the appropriate bitstream element is decoded, *if* the interrupt is enabled. Thus, Decode-time interrupts are synchronous with picture decoding (which occurs in advance of picture display) and they may be issued as soon as the microapplication leaves the PLAY-SETUP state, even if the output window is not yet unblanked.

> *Note: There may be a delay between the time that the bitstream construct which could cause a Decode-time interrupt event enters the CL450's host bus and the time when the interrupt occurs. This delay is due to the bitstream being stored in the CL450's internal bitstream buffer before decoding.*

12.1.3 VSYNC Interrupts

VSYNC interrupts are issued following the active (rising) edge of VSYNC which begins the top border time of a *new* display field. VSYNC interrupts are *only* issued at the beginning of those fields which are the first display of a newly-decoded picture. VSYNC interrupts are *not* issued at the start of field displays for pictures which have already been displayed once, including the second field display of a still picture, even though the pixel values being displayed have not been displayed before.

Figure 12-3 shows an example of the relationship between the VSYNC pulses which divide field periods, the pictures which are displayed, and the times when VSYNC interrupts may be produced. Each box shown represents the display of a video field. The boxes are labeled using the standard MPEG nomenclature in which "B_{16}" indicates that the picture is type B (of I, P, and B) and number 16 (in display order) in a bitstream. In addition, each box also contains a label of the form "f_n" in which *n* indicates the number of times that a field containing that picture has already been displayed.

Figure 12-3 shows a bitstream with a `picture_rate` of 25 Hz being displayed at a nominal picture rate of 30 Hz (field rate 60 Hz), with picture B_{16} therefore displayed for three field periods due to pull-down.

VSYNC interrupts are not produced while the output window is blanked because newly decoded pictures are not being displayed.

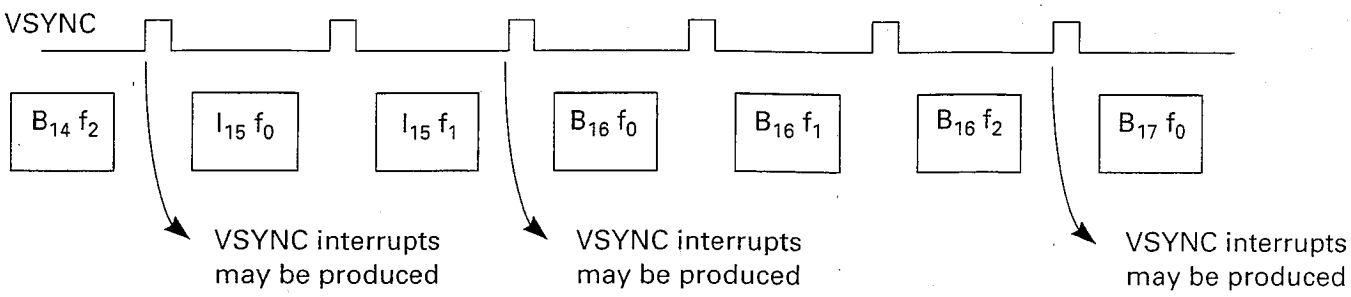


Figure 12-3 VSYNC Interrupt Generation

12.2 Interrupt Handshaking

All interrupts cause the $\overline{INT}$ pin to be asserted. To determine which interrupt(s) has occurred, the Interrupt Status location in HMEM is written by the CL450 at the same time the $\overline{INT}$ pin is asserted.

Each bit in Interrupt Status corresponds to a different logical interrupt source. These bits are allocated in the same order used for the **mask** argument to the SetInterruptMask() macro command, which is given in Table 12-1, CL450 Interrupt Summary.

The host may read the Interrupt Status location at any time to determine which interrupts, if any, are pending. The host may also write to this location to clear pending interrupts. However, writes to Interrupt Status must *only* occur as specified in this section.

The same handshaking protocol is used for all CL450 interrupts, regardless of the logical interrupt source. When the CL450 issues an interrupt to the host, it makes the $\overline{INT}$ pin active (low) and sets one or more bits in the Interrupt Status location of HMEM.

Because there is no semaphore to protect Interrupt Status from simultaneous access by both the host and the microapplication, the protocol for issuing interrupts is structured such that simultaneous access cannot occur for the following two reasons:

- The microapplication only writes Interrupt Status when it is 0.
- The host is only allowed to write Interrupt Status when it is not 0.

Once the host writes a 0 to Interrupt Status, the host cannot write to Interrupt Status again (even a 0) until the microapplication sets one or

more bits. If the host violates this protocol, then interrupts may be lost or spuriously created.

Excessive latency by the host in clearing the active bit(s) within Interrupt Status could also prevent the microapplication from issuing subsequent interrupts as distinct events.

12.2.1 Microapplication Behavior

When an interrupt *can* be produced, the microapplication polls Interrupt Status. If Interrupt Status is 0, then all pending interrupts, if any, are posted to Interrupt Status and the $\overline{\text{INT}}$ pin is activated, if appropriate. If Interrupt Status is non-zero, then the new interrupts, if any, will be ORed into the microapplication's internal variable for storing pending interrupts. However, the microapplication will not re-check Interrupt Status until the next occasion in which it might post an interrupt, even if interrupts have been queued internally.

Interrupt Status is checked and queued interrupts potentially issued every time the timing for Display-time, VSYNC, or Decode-time interrupts is satisfied.

> *Note: If HMEM[Interrupt Status] is non-zero for an extended period, then multiple interrupt events of the same type may be ORed into the microapplication's internal variable for storing new interrupts, causing events to be lost.*

For example, assume that when the microapplication is in the PLAY state and in the midst of decoding and displaying a video sequence, the microapplication then decodes a new `sequence_header_code`. When this occurs, the current interrupt **mask** is checked to determine if the corresponding interrupt (SEQ-D) is enabled. If it is, the microapplication checks the current state of Interrupt Status. If Interrupt Status is 0, then the SEQ-D interrupt and *any other* already queued interrupts will be posted to Interrupt Status. If Interrupt Status is non-zero, then the SEQ-D interrupt is queued internally, and the next occasion when the microapplication checks Interrupt Status will depend on the other interrupt sources. If SEQ-D was disabled in **mask**, but there were queued interrupts, the queued interrupts would still be posted.

The microapplication's half of the interrupt handshaking protocol is shown in the pseudocode example of the "MaybeIssueInt" function of Figure 12-4.

```
void MaybeIssueInt(unsigned short newInts)
{
   static unsigned short pendingInts= 0;

   pendingInts|= newInts;
   if (HMEM[Interrupt Status] == 0){
      HMEM[Interrupt Status]= pendingInts;
      INT pin= LOW;
      pendingInts= 0;
   }
}
```

Figure 12-4 CL450 Interrupt Posting Procedure (pseudocode

The decision flow of the CL450 posting procedure of Figure 12-4 is shown from a block diagram viewpoint in Figure 12-5. Every time the microapplication reaches a point where it *could* issue a Display-time, Decode-time, or VSYNC interrupt, it performs equivalent operations.

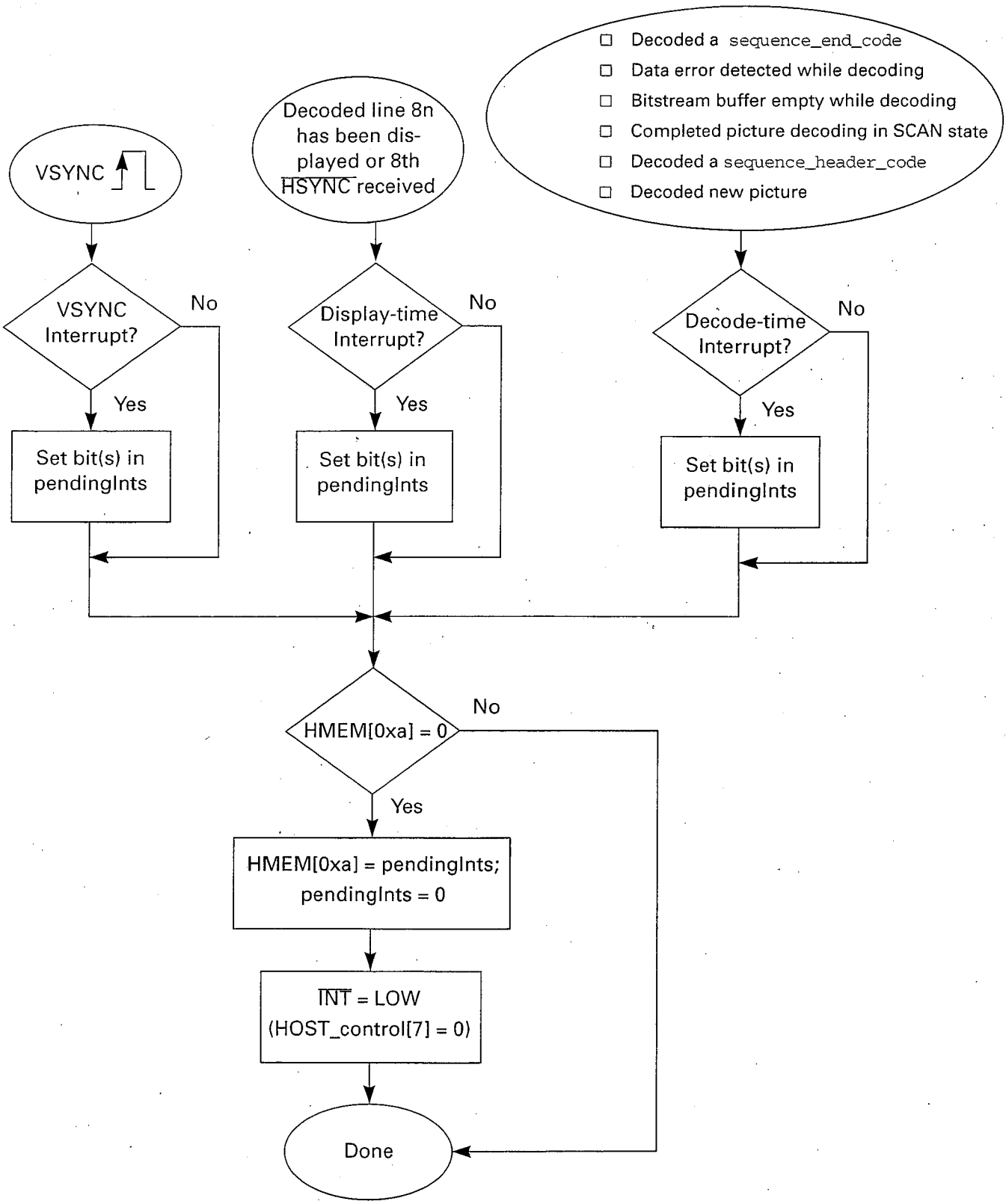


Figure 12-5 CL450 Interrupt Posting Procedure (block diagram)

12.2.2 Host Behavior

Each time the microapplication issues one or more new interrupts to the host, it always sets the corresponding bit(s) in Interrupt Status *and* makes the $\overline{\text{INT}}$ pin active. The microapplication performs these actions inseparably. However, the times at which the host may clear the Interrupt Status location of HMEM and make the $\overline{\text{INT}}$ pin inactive may vary greatly from each other.

For example, in a system in which the automatic interrupt clear feature is used (enabled by HOST_control[15]), the $\overline{\text{INT}}$ pin will become inactive as soon as the hardware interrupt acknowledge cycle occurs. But even though the hardware interrupt acknowledge has occurred, the microapplication will not issue another interrupt until the host clears Interrupt Status.

Alternately, since many hardware interrupt controllers cannot be reconfigured to receive a new interrupt until the interrupt signal has become inactive, the host may clear Interrupt Status (telling the CL450 that a new interrupt can be produced) but not deactivate $\overline{\text{INT}}$ or re-initialize the host's interrupt controller. In this case, the microapplication might issue another interrupt which would not be recognized by the host processor, possibly resulting in lost interrupts or a deadlock in which it appears that no interrupts are being generated.

Because the host typically needs to (1) clear Interrupt Status as soon as possible to minimize the interrupt latency added to the system by the host and (2) service multiple logical interrupts received with the same physical interrupt, it follows an interrupt handling procedure somewhat more complicated than that of the microapplication.

Figure 12-6, for example, shows sample pseudocode for a host interrupt service routine for the CL450, which minimizes the amount of elapsed time between an interrupt being issued by the microapplication and Interrupt Status being cleared. However, once entered, this routine holds the processor and does not allow any other interrupts to be serviced or any non-interrupt code to execute until *all* CL450 interrupts, even those issued while earlier interrupts are being processed, have been processed.

```
INTERRUPT_SERVICE_ROUTINE_TYPE isr(void)
{
   static unsigned int unservicedInts= 0;
   unsigned int newInts;
   unsigned short temp;
   do {

      if (INT pin == LOW){                    /* is new int pending? Or are
                                               * we servicing another logical
                                               * source from last physical int?
                                               */
         temp= HOST_control;                  /* INT pin = HIGH */
         temp&= 0xC001;                       /* RESERVED bits must be written
         temp|= 0x80;                          * with 0 regardless of value read
         HOST_control=  temp;                  */

         newInts= HMEM[Interrupt Status];
         HMEM[Interrupt Status]= 0;
      }
                                              /* At this point the microcode is
                                               * able to issue another interrupt
                                               */

      if (newInts & unservicedInts)
         handle error condition;              /* we've received a new int and
                                               * still haven't serviced the last
                                               * one of the same type
                                               */
      unservicedInts|= newInts;
      switch (unservicedInts){
         /* might be a series of if-else's instead. Arrange cases so
          * that highest priority interrupt is checked first and
          * process one interrupt only for each pass through the switch.
          */
         /* For each pass: */
         perform interrupt-specific processing;
         clear corresponding bit in unservicedInts;
      }
   } while (unservicedInts != 0 ||            /* got multiple logical ints */
            INT pin == LOW);                  /* CL450 issued new physical int
                                               * while processing last int */

   /* no more interrupts to process */
   configure host's hardware interrupt controller to receive new ints;
}
```

Figure 12-6 Host Interrupt Handler Example 1

If the behavior shown in Figure 12-6 is undesirable (perhaps because there are higher-priority interrupt sources in the system), then an approach which takes advantage of the CL450's internal interrupt queuing can be used, as shown in Figure 12-7. This routine processes only one logical interrupt each time it is entered and uses Interrupt Status to hold unprocessed logical interrupts.

> *Note: Figure 12-6 and Figure 12-7 should be treated as examples only; the optimal interrupt handling procedure for a given CL450-based system will probably be different from either of these.*

In addition to initializing the interrupt controller so a new CL450 interrupt can be received (if the interrupt controller is edge sensitive), the host must use the HOST_control register to cause an inactive pulse on the CL450's $\overline{\text{INT}}$ pin (simulating a new physical interrupt) immediately before leaving the routine. However, with this method, it cannot be determined if more than one of each logical interrupt has occurred before being serviced.

In general, the host must ensure that all interrupts produced by the CL450 are produced when the rest of the system is ready to handle them. This responsibility not only includes configuration of the host interrupt controller, but writing proper values to HOST_control[15:14] and HOST_intvecw.

> *Note: It is impossible to tell which values will be used if the host writes to one of these registers "simultaneously" with the microapplication issuing a new interrupt. In general, these registers should either be configured correctly as part of system initialization, or SetInterruptMask() should be used to disable all CL450 interrupts while the registers are being modified.*

```
INTERRUPT_SERVICE_ROUTINE_TYPE isr(void)
{
   unsigned int newInts;
   unsigned short temp;
   newInts= HMEM[Interrupt Status];
   switch (newInts){
        /* might be a series of if-else's instead. Arrange cases so
         * that highest priority interrupt is checked first and
         * process one interrupt only for each pass through the switch.
         */
        /* For each pass: */
        perform interrupt-specific processing;
        clear corresponding bit in newInts;
   }
   HMEM[Interrupt Status]= newInts;          /* put back ones we didn't process
                                              */
   configure host's hardware interrupt controller to receive new ints;
   if (!newInts){
      temp= HOST_control;                    /* INT pin = HIGH */
      temp&= 0xC001;
      temp|= 0x80;
      HOST_control= temp;
   else
      maybe create INT pulse;                /* see text */
}
```

Figure 12-7 Host Interrupt Handler Example 2

12.3 Interrupt Listing

This section gives a detailed description of the CL450 Display-time, Decode-time, and VSYNC interrupts listed together in alphabetical order.

Event: sequence_end_code found
Category: Decode-time
Mask Bit: 5

END-D

An END-D interrupt event occurs if both of the following are true:

- A sequence_end_code is detected in the coded data stream while decoding is being performed.
- The END-D interrupt is enabled (**mask**[5] of the last SetInterrupt-Mask() macro command executed equals 1).

END-D interrupts are issued to the host immediately after the sequence_end_code is *decoded*.

When an END-D event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an END-D interrupt or places an END-D interrupt in the pending interrupt queue.

This interrupt can be used to warn the host of (1) a transition between video sequences, or (2) the end of a video stream (if the host knows how many sequences are in the stream). In particular, each time a sequence_end_code is decoded, the microapplication returns the sequence parameters in DRAM to the default values unless *SeqNoDef* is 1 (see Section 14.1.1, Sequence Variable Group).

> *Note: The END-V interrupt is also caused by detection of a* sequence_end_code *but is not issued until the VSYNC prior to the display of the last picture (in display order) found in the bitstream before the* sequence_end_code. *(See Section 12.1.3, VSYNC Interrupts, and END-V on page 12-16.)*

END-V

Event: Last picture display before `sequence_end_code`
Category: VSYNC
Mask Bit: 4

The END-V interrupt event occurs at the rising edge of VSYNC prior to the first display field of the last picture (in display order) decoded prior to the detection of a `sequence_end_code`. Each time an END-V event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an END-V interrupt or places an END-V interrupt in the pending interrupt queue.

An END-V interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- A `sequence_end_code` has been decoded.
- The picture which will be displayed in the following field time is the last picture (in display order) in the sequence terminated by the `sequence_end_code`.
- The END-V interrupt is enabled (**mask**[4] of the last SetInterruptMask() macro command executed equals 1).

Note: The END-D interrupt is also caused by detection of a `sequence_end_code`*, but is issued as soon as the decode process detects it. (See Section 12.1.2, Decode-time Interrupts, and END-D on page 12-15.)*

Event: Bitstream Data Error
Category: Decode-time
Mask Bit: 0

ERR

The ERR interrupt (bitstream data error) event occurs each time the decoding process detects one of the following errors in the coded data:

- An MPEG `sequence_error_code`
- Invalid variable-length codes
- `marker_bit`'s with 0 values
- Header fields with illegal values

The decoding process does not detect dropped or incorrect bits unless they cause one of these detectable conditions, in which case this interrupt may be used by the host to take the appropriate action (possibly applying high-level error concealment techniques), if any.

When an ERR event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an ERR interrupt or places an ERR interrupt in the pending interrupt queue.

An ERR interrupt event occurs if both of the following are true:

- An error is detected in the coded data stream while decoding is being performed.
- The ERR interrupt is enabled (**mask**[0] of the last SetInterruptMask() macro command executed equals 1).

Not all data errors can be detected, and the portion of the bitstream being decoded when an error is detected will not necessarily be the portion containing the error. For example, a dropped bit may cause an erroneous VLC (variable-length code) to be detected considerably later in the bitstream.

Typically, the microapplication's internal error concealment operates by continuing to display the nearest (in display order) correctly-decoded reference frame until it finds a portion of the bitstream that it can decode. When this is occurring, undecodable portions of the bitstream (typically B-pictures and sometimes P-pictures) are discarded at a rapid rate. Because of this, the bitstream buffer can underflow temporarily, which may cause an additional interrupt.

In a system using audio/video synchronization, the bitstream buffer eventually returns to the correct fullness level because the synchronization mechanism causes the microapplication to wait until the proper system time before decoding the next picture. Similarly, in a fixed bit-

rate system, the CL450's decode process will underflow until the data source provides valid data, at which point the CL450 will become re-synchronized.

If pictures are discarded due to error recovery in a variable bit-rate system in which synchronization is *not* being performed, the microapplication resumes decoding and display as soon as a decodable picture is found, which effectively reduces the playing time of the current video sequence by the number of pictures discarded. If this reduction is undesirable, the host should use the ERR interrupt and generate appropriate Pause() commands.

Name: First I-picture display after `group_start_code`
Category: VSYNC
Mask Bit: 2

GOP

The GOP interrupt event occurs at the rising edge of VSYNC prior to the first display field of the first I-picture decoded following the detection of a `group_start_code`. Each time a GOP event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a GOP interrupt or places a GOP interrupt in the pending interrupt queue.

The microapplication uses an internal flag to produce the GOP interrupt. This flag is cleared each time the microapplication reaches the IDLE state, the FlushBitstream() command is executed, or a GOP interrupt event occurs. The flag is set each time a `group_start_code` is decoded.

A GOP interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- The picture to be displayed in the following field time is an I-picture which has not been previously displayed.
- The GOP interrupt flag is set.
- The GOP interrupt is enabled (**mask**[2] of the last SetInterruptMask() macro command executed equals 1).

A group of pictures can be decoded and displayed without a GOP interrupt event occurring (if it contains only P- and B-pictures), but GOP interrupt events cannot occur more often than `group_start_codes`.

PIC-D

Event: New Picture Decoded
Category: Decode-time
Mask Bit: 6

The PIC-D interrupt (new picture decoded) event occurs each time all of the following are true:

- A coded picture is detected in the coded data stream and completely decoded without error.
- The PIC-D interrupt is enabled (**mask**[6] of the last SetInterruptMask() macro command executed equals 1).

Each time a PIC-D event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a PIC-D interrupt or places a PIC-D interrupt in the pending interrupt queue.

The PIC-D interrupt is closely related to the PIC-V interrupt because both are produced by the processing of newly coded pictures. However, while the PIC-D and other Decode-time interrupts are issued to the host immediately after the *entire* picture is decoded, the PIC-V interrupt is produced by the detection of a `picture_start_code` and is not issued until the VSYNC prior to the display of the picture found in the bitstream after the `picture_start_code`. (See Section 12.1.3, VSYNC Interrupts, and PIC-V on page 12-21.)

The order in which PIC-D and PIC-V interrupts are issued depends on the type of picture being decoded. If an I- or P-picture is decoded, then the entire picture is completely decoded before display starts, and PIC-D precedes PIC-V. If a B-picture is being decoded, then display will begin after only half of the picture is decoded, and PIC-V precedes PIC-D by approximately one field time. In addition, bitstream errors may cause PIC-V interrupts to be produced without a corresponding PIC-D (if an error occurs while decoding a B-picture) or vice versa (if an error occurs which causes a completely decoded reference frame never to be displayed).

Event: New picture display
Category: VSYNC
Mask Bit: 1

PIC-V

The PIC-V interrupt (new picture display) event occurs at the rising edge of VSYNC prior to the first display field of a newly-decoded picture. Each time a PIC-V event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a PIC-V interrupt or places a PIC-V interrupt in the pending interrupt queue.

A PIC-V interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- The picture to be displayed in the following field time has not been previously displayed.
- The PIC-V interrupt is enabled (**mask**[1] of the last SetInterrupt-Mask() macro command executed equals 1).

Note that the second field of a picture decoded using the DisplayStill() command (page 11-14) is not considered a "newly-decoded picture," even though the data from that field has not been previously displayed. In other words, only one PIC-V interrupt will be produced per Display-Still() command.

RDY

Event: Ready for data
Category: Display-time
Mask Bit: 10

The RDY interrupt (ready for data) event occurs when the number of empty bytes in the bitstream buffer in DRAM is greater than a given **threshold** value. It can be used by the host:

- To indicate that more data needs to be sent to the CL450.
- With a larger **threshold**, as a warning that bitstream buffer underflow is about to occur.

Each time a RDY event occurs, the microapplication either issues a RDY interrupt or places a RDY interrupt in the pending interrupt queue (See Figure 12-4).

A RDY interrupt event occurs when the following conditions are true:

- The timing for the Display-time interrupt is satisfied (see Section 12.1.1, Display-time Interrupt).
- The number of free bytes in the bitstream buffer meets or exceeds the **threshold** argument to the most recently executed SetThreshold() macro command.
- The number of data bytes which have been delivered to the CL450 since the last occurrence of the IDLE state or execution of the FlushBitstream() command is greater than or equal to the sum of all the **length** arguments to all the NewPacket() commands which have been accepted in the same interval *minus 6*. That is:

$$\text{total bytes received} \geq \left(\sum \textbf{length} \right) - 6$$

- At least one NewPacket() macro command has been issued.
- The RDY interrupt is enabled (**mask**[10] of the last SetInterruptMask() macro command is set equal to 1).

The interlock between NewPacket() commands and the production of RDY interrupts is provided to minimize the interrupt traffic to the host. The microapplication assumes that if the host has issued a NewPacket() command, it knows that the corresponding data must be sent to the CL450 and the host does not have to be interrupted. The *-6* bias is provided so that the interrupt will be produced slightly before the current data transfer completes.

Note: For the second condition listed, the RDY interrupt is sensitive to the buffer emptiness being above ***threshold*** *(level*

sensitive), not becoming above ***threshold*** *(not edge sensitive). This means that, once an RDY interrupt is produced, new RDY interrupts are produced each time a Display-time interrupt can be issued until the host takes steps to make one of the other required conditions false.*

Typical host systems which use the RDY interrupt issue one or more NewPacket() macro commands within the interrupt handler that responds to the RDY interrupt. This procedure prevents redundant RDY interrupts from being produced but allows the actual transfer of data (if DMA is not used) to occur outside the interrupt handler, improving the overall interrupt latency of the system.

If the RDY interrupt is to be used by the host to control data transfers to the CL450, the host must ensure that the latency between the times when the microapplication polls to determine if an RDY interrupt should be produced does not cause bitstream buffer underflow.

SCN

Event: Picture decode complete in SCAN state
Category: Decode-time
Mask Bit: 11

The SCN interrupt (picture decode complete in SCAN state, see page 11-28) event occurs each time the decoding process finishes decoding an I-picture in the SCAN state. This interrupt event also marks the automatic transition between the SCAN and PAUSE states.

Each time an SCN event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an SCN interrupt or places an SCN interrupt in the pending interrupt queue.

An SCN interrupt event occurs if all of the following are true:

- The microapplication is in the SCAN state.
- Decoding of an I-picture has completed.
- The SCN interrupt is enabled (**mask**[11] of the last SetInterruptMask() macro command executed equals 1).

This interrupt can be used to inform the host when a new picture is available to be displayed.

Event: sequence_header_code found
Category: Decode-time
Mask Bit: 9

SEQ-D

The SEQ-D interrupt (sequence_header_code found) event occurs each time the decoding process detects a sequence_header_code in the coded data. Each time a SEQ-D event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an SEQ-D interrupt or places an SEQ-D interrupt in the pending interrupt queue. SEQ-D interrupts are issued to the host immediately after the code is *decoded*.

An SEQ-D interrupt event occurs if all of the following are true:

- A sequence_header_code is detected in the coded data stream while decoding is being performed.
- The SEQ-D interrupt is enabled (**mask**[9] of the last SetInterruptMask() macro command executed equals 1).

This interrupt can be used:

- To warn the host of a transition into a new video sequence.
- At the beginning of decode of the first sequence.
- Any time that the decode parameters may have changed.

When a sequence_header_code is decoded, the microapplication writes the sequence parameters in DRAM with the values contained in the bitstream unless *SeqWP* is1. (See Section 14.1.1 on page 14-2, Sequence Variable Group.)

> *Note: The SEQ-V interrupt is also caused by detection of a* sequence_header_code, *but is not issued until the VSYNC prior to the display of the first picture (in display order) found in the bitstream after the* sequence_header_code. *(See Section 12.1.3, VSYNC Interrupts, and SEQ-V on page 12-25.)*

SEQ-V

Name: First I-picture display after `sequence_header_code`
Category: VSYNC
Mask Bit: 3

The SEQ-V interrupt (first I-picture display after `sequence_header_code`) event occurs at the rising edge of VSYNC prior to the first display field of the first I-picture decoded following the detection of a `sequence_header_code`. Each time a SEQ-V event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues an SEQ-V interrupt or places an SEQ-V interrupt in the pending interrupt queue.

The microapplication uses an internal flag to produce the SEQ-V interrupt. The flag is set each time a `sequence_header_code` is decoded. This flag is cleared when the microapplication reaches the IDLE state, the FlushBitstream() command is executed, a `sequence_end_code` is decoded, and each time an SEQ-V interrupt event occurs.

An SEQ-V interrupt event occurs if all of the following are true:

- The timing for VSYNC interrupts is satisfied (see Section 12.1.3, VSYNC Interrupts).
- The picture to be displayed in the following field time is an I-picture which has not been previously displayed.
- The SEQ-V interrupt flag is set.
- The SEQ-V interrupt is enabled (**mask**[3] of the last SetInterruptMask() macro command executed equals 1).

  Note: The SEQ-D interrupt is also caused by detection of a `sequence_header_code`*, but is issued as soon as the decode process detects it. (See Section 12.1.2, Decode-time Interrupts, and SEQ-D on page 12-25.)*

Name: Bitstream buffer underflow error
Category: Decode-time
Mask Bit: 8

UND

The UND interrupt (bitstream buffer underflow) event occurs when the number of valid bytes in the bitstream buffer in DRAM is 0. This interrupt can be used by the host to detect a bitstream buffer underflow error condition. Each time a UND event occurs, the microapplication performs the operations shown in Figure 12-4 and either issues a UND interrupt or places a UND interrupt in the pending interrupt queue.

A UND interrupt event occurs if all of the following are true:

- The number of valid words in the bitstream buffer is 0, and a new word is needed by the decoder process.
- The UND interrupt is enabled (**mask**[8] of the last SetInterruptMask() macro command executed equals 1).

  Note: The UND interrupt is sensitive to the buffer's emptiness being 0, not becoming 0. This means that, once a UND interrupt is produced, new UND interrupts will be produced continuously until the host takes steps to make the bitstream buffer non-empty.

Unlike the RDY interrupt, the UND interrupt does *not* depend on the NewPacket() command, and production of the UND interrupt cannot be suppressed by issuing NewPacket() commands. UND interrupt production can only be stopped by (1) transferring new data to the CL450 or (2) disabling the UND interrupt. And, as described in Section 15.2.3, During Bitstream Underflow, the microapplication is unlikely to execute a SetInterruptMask() macro command if the bitstream buffer is empty since low-priority macro commands may only be executed after an entire picture has completed decoding.

12.4 Interrupt Examples

This section contains two examples of how the CL450's interrupts behave in relation to one another and the host's actions. Each example

12.4 Interrupt Examples

This section contains two examples of how the CL450's interrupts behave in relation to one another and the host's actions. Each example consists of introductory text explaining the cases shown, a numbered list of events which occur in the example, and a corresponding figure illustrating how those events relate to each other in time sequence.

While reading the examples, remember that the situations and the example host actions shown have been chosen to best illustrate the behavior of the CL450. These situations are not necessarily typical, and the host's responses and actions may not be optimal in an actual system.

The examples show some, but not necessarily all, of the macro commands which the host issues during the period shown. In general, only those macro commands whose timing is important to the interrupts in the example are shown. For example, in a typical system the host would be issuing many more NewPacket() commands than are shown, and there are several other commands (SetBorder, SetWindow, InquireBufferFullness, etc.) which would not substantively affect the examples if the host issued them at times not explicitly shown. However, *all* macro commands which can affect the microapplication's command processing state or the interrupts which are enabled are explicitly shown.

The figures in each example are arranged similar to timing diagrams, showing the timing relationship between changes in the CL450's state and output. All of the figures show the same state elements, although not all elements affect the outcome of each example.

Figure 12-8 is a key to some of the items in the example figures. In particular, miniature versions of the display-field picture from Figure 12-2 are used to represent the CL450's video output. However, because the example figures are organized with time passing from left to right, the video display has been rotated so the top of the display field appears on the left and scan lines run bottom-to-top within the field.

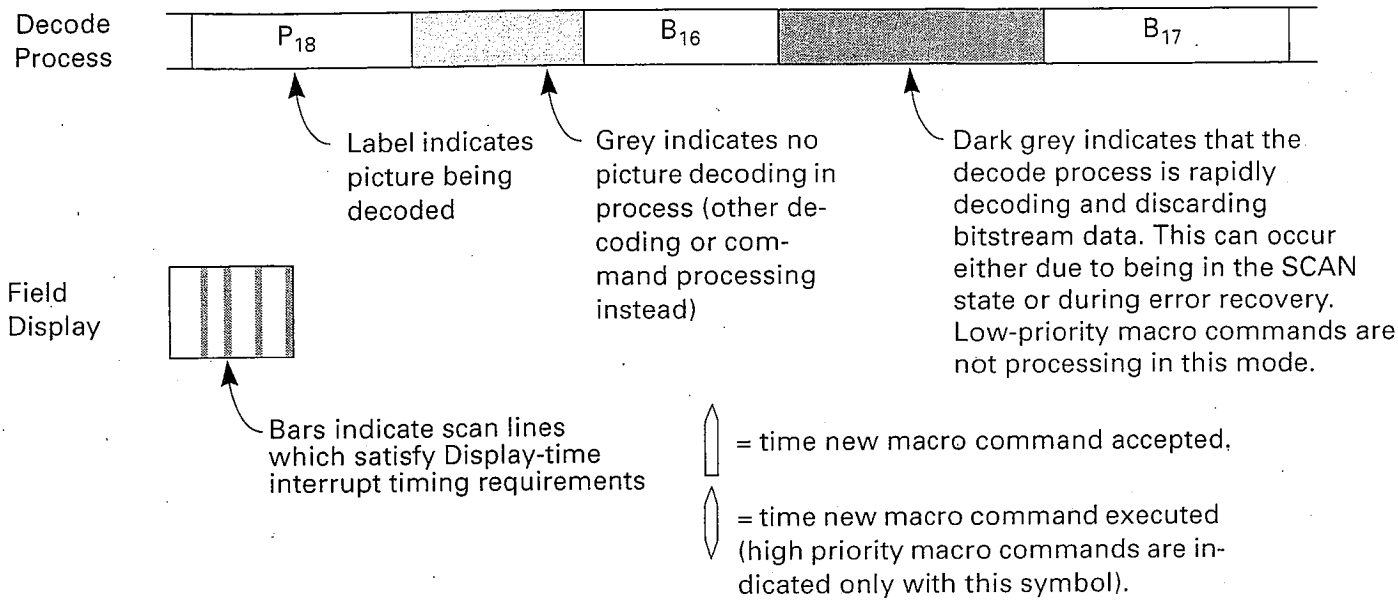


Figure 12-8 Key to Interrupt Examples

12.4.1 Interrupt Example 1: RDY, UND, ERR

This example shows how the RDY interrupt is produced, how a bitstream data error causes the ERR interrupt and triggers the microapplication's error concealment, and how this in turn can cause both the RDY and UND interrupts.

At the beginning of this example, the CL450 is in the middle of decoding and displaying a bitstream and only the RDY, UND, and ERR interrupts are enabled. (see Figure 12-9).

1. The bitstream buffer's fullness increases until the host pauses the incoming data supply, which happens after the host sends all the data for all the NewPacket() commands that have been issued. This pause satisfies *one* of the requirements for the creation of the RDY interrupt.
2. As the microapplication continues to decode, the number of empty bytes exceeds the **threshold** value. However, because the microapplication polls the fullness level according to the constraints for Display-time interrupts, the RDY interrupt is not yet produced.

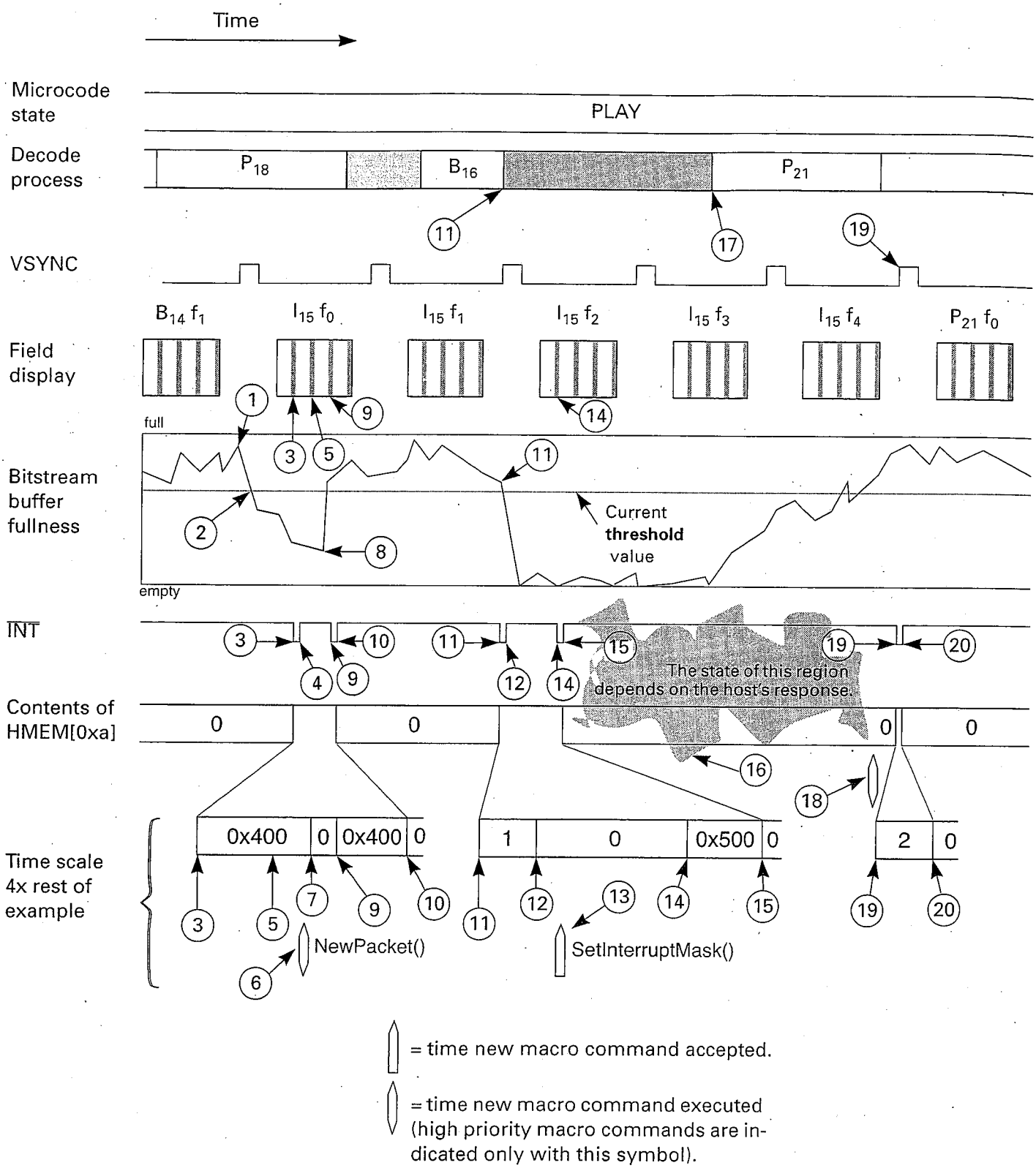


Figure 12-9 Interrupt Example 1: RDY, UND, ERR

3. After the first block-row from the picture I_{15} is displayed, the microapplication recomputes the buffer fullness, and discovers that there are more empty bytes than **threshold**. Because the other interrupt requirements have also been satisfied, the microapplication recognizes an RDY interrupt event. Because HMEM[0xa] (Interrupt Status) is 0, the RDY interrupt can be issued to the host. Therefore the microapplication makes the $\overline{\text{INT}}$ pin active and sets HMEM[0xa][10] equal to 1. No other bits are set because no other interrupts are pending.
4. The host receives the interrupt and begins its interrupt service process by deactivating the $\overline{\text{INT}}$ pin. While the host may now be able to receive more interrupts, the microapplication cannot issue another interrupt because Interrupt Status is still non-zero.
5. For whatever reason, the host does not issue a NewPacket() command to begin transfer of more data before the end of display of the second block row of I_{15} occurs. Therefore, the conditions necessary for the RDY interrupt all become true again, so a second RDY interrupt event occurs. Because the host has also not yet cleared Interrupt Status, the RDY interrupt cannot be issued (even though $\overline{\text{INT}}$ is inactive) and is queued internally instead.
6. The host then issues a NewPacket() command. While this does not directly add more data to the bitstream buffer, it does make one of the conditions required for the RDY interrupt false.
7. The host then clears Interrupt Status. The host must do this *after* issuing the NewPacket() command if a new RDY interrupt can be issued between these two actions.
8. Finally, coded data begins to be received by the CL450 again and the bitstream buffer fullness begins to rise.
9. When the end of display of the third block row of I_{15} occurs, even though the number of empty bytes in the bitstream buffer is less than **threshold**, the microapplication issues a new (second) RDY interrupt to the host. This interrupt is issued because the internal interrupt queue is examined each time a Display-time interrupt *could be* produced, and because there is a queued RDY interrupt from the end of the previous block row. This interrupt would not have been queued if the host had responded more quickly to the original RDY interrupt.

10. The host receives the interrupt and determines that it is a remnant from the previous RDY interrupt (possibly by examining Buffer Fullness Status). Therefore, it deactivates the $\overline{\text{INT}}$ pin and clears Interrupt Status. Note that these operations should be done in that order because the microapplication will not modify the state of the $\overline{\text{INT}}$ pin unless Interrupt Status is 0.
11. The decode process discovers an error in the bitstream while decoding picture B_{16}. This error causes the microapplication to perform two actions. First, the microapplication attempts error recovery (see Section 15.2, Error Recovery and Concealment). Second, because the ERR interrupt is enabled, a ERR interrupt event occurs; and because Interrupt Status is 0, the interrupt is issued directly to the host, activating the $\overline{\text{INT}}$ pin and writing Interrupt Status with 1.
12. The host receives the ERR interrupt. In response, the $\overline{\text{INT}}$ pin is made inactive and Interrupt Status is cleared. The host also sets an internal flag because it will have to interpret future RDY and UND interrupts differently than if the microapplication were not performing error recovery.
13. The host issues and the CL450 accepts a SetInterruptMask() macro command. This command is issued with a **mask** argument of 0x503, leaving RDY, UND and ERR enabled, and adding PIC-V. The host enables the PIC-V interrupt so that it will be informed when the first new picture is displayed, which roughly corresponds to the end of error recovery. Note that, because SetInterruptMask() is a low priority command, the microapplication does not execute it until item number 18, below.
14. The next time the timing requirements for Display-time interrupts are satisfied (in this example, the end of the first block row of the third field time in which picture I_{15} is displayed), the microapplication issues both a UND and an RDY interrupt (assuming that an adequate number of bitstream words has been transferred to satisfy the RDY condition).
15. The host receives the interrupts from the CL450. It ignores the UND interrupt, because it knows that the microapplication is in error recovery. It uses the RDY interrupt as it normally would. The $\overline{\text{INT}}$ pin is deactivated and Interrupt Status cleared.

16. In this region, the microapplication will issue UND and/or RDY interrupts for every block row displayed. The host should continue to ignore the UND interrupts, and supply data as quickly as possible. Some or all of the RDY interrupts may be suppressed by issuing NewPacket() commands one or more in advance of sending data to the CL450.
17. The decode process finds a `picture_start_code` for a picture that it has the information to decode. At this point, it resumes normal decoding operation.
18. Microcode executes the SetInterruptMask() command issued at item number 13, above. It is only at this point that the PIC-V interrupt is actually enabled.
19. The VSYNC prior to the first display of the newly-decoded picture P_{21} occurs. This creates a PIC-V interrupt event. The PIC-V interrupt is immediately issued to the host because (in this example) Interrupt Status is 0 when VSYNC occurs. The microapplication therefore activates $\overline{\text{INT}}$ and sets Interrupt Status[1].
20. The host receives the PIC-V interrupt. It clears Interrupt Status and deactivates the $\overline{\text{INT}}$ pin. In addition, the host clears its internal flag, indicating that the microapplication is performing error recovery. Subsequently, if the host receives a UND interrupt, it will be indicative of a system error condition. The host may also subsequently issue another SetInterruptMask() macro command to disable the PIC-V interrupt if the host has no other use for it.

12.4.2 Interrupt Example 2: PIC-V, SCN

This example shows how the PIC-V interrupt is produced, how the Scan() macro command might be used, and how the SCN interrupt is produced.

At the beginning of this example, the CL450 is in the middle of decoding and displaying a bitstream in the PLAY state, and only the PIC-V and SCN interrupts are enabled. (See Figure 12-10.)

1. The active edge of VSYNC preceding the first display of picture B_{15} occurs. This causes a PIC-V interrupt event and, because Interrupt Status was previously 0, the microapplication issues the interrupt immediately. Therefore, the $\overline{\text{INT}}$ pin is activated and Interrupt Status[1] is set.
2. The host issues and the CL450 accepts the Scan() command.
3. When the decoding for picture B_{15} completes, the microapplication processes accumulated low-priority macro commands. The previously-accepted Scan() command is executed, causing the microapplication to transition from the PLAY state to the SCAN state.
4. The host finally receives and processes the PIC-V interrupt issued in item 1. The host deactivates $\overline{\text{INT}}$ and clears Interrupt Status.
5. The microapplication finishes its internal SCAN initialization and begins to process the portion of the bitstream ahead of the I-picture at a very high rate. MPEG header information is still decoded and processed and only P- and B-pictures are discarded completely.
6. When the decoding process encounters a `picture_start_code` for an I-picture (in this example, I_{38}), it stops discarding bitstream data and begins normal picture decoding.
7. When the decoding of I_{38} is complete, the microapplication automatically transitions to the PAUSE state (and stops consuming bitstream data). In addition, the completion of decode causes an SCN interrupt event. Because Interrupt Status is 0, the interrupt is issued immediately and the $\overline{\text{INT}}$ pin is activated and Interrupt Status[11] set.

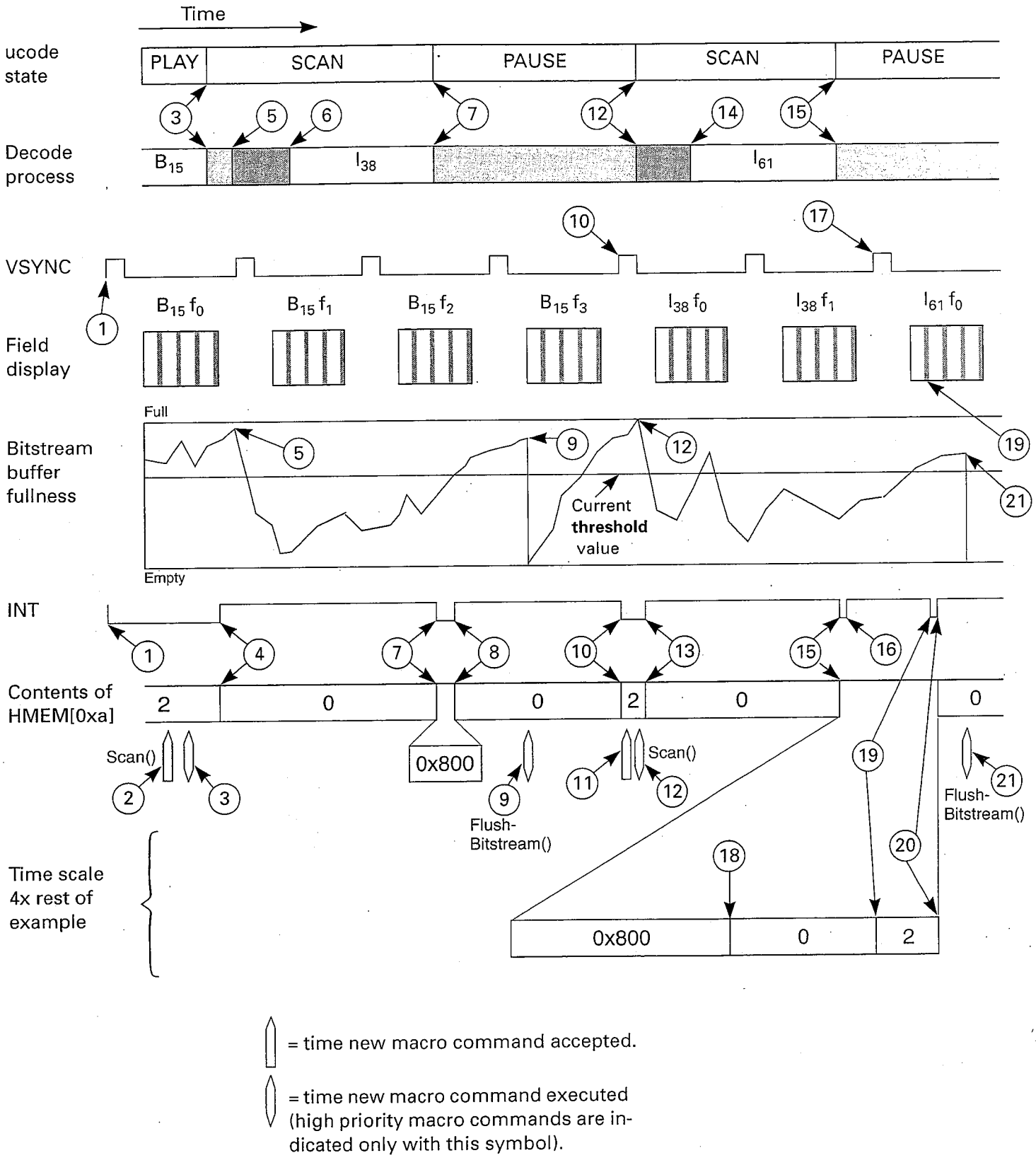


Figure 12-10 Interrupt Example 2: PIC-V, SCN

8. The host then receives the SCN interrupt. In response, the host's interrupt handler sends a message to a portion of non-interrupt software (which will respond later), deactivates the $\overline{\text{INT}}$ pin, and clears Interrupt Status.
9. After the interrupt handler exits, the host issues a FlushBitstream() command, which the microapplication immediately executes because it is high priority. This macro command discards all of the bitstream data (and associated NewPacket() macro commands) which followed the I-picture and have been transmitted to the CL450 since the decoding of I_{38} completed (while the microapplication was in the PAUSE state). After this command has executed, the host begins transmitting bitstream information and NewPacket() macro commands in preparation for the next Scan() command.
10. When the next VSYNC occurs, the microapplication begins the display of I_{38}. Because this is the first display field of a new picture, it also causes a PIC-V interrupt event. The PIC-V interrupt is issued immediately (because Interrupt Status is 0), so the microapplication activates $\overline{\text{INT}}$ and sets Interrupt Status[1].
11. The host (continuing the non-interrupt processing which began in item # 9) then issues a new Scan() command, which the microapplication accepts and places in the Command FIFO.
12. Because the microapplication is in the PAUSE state and no other low priority macro commands are pending, the Scan() command issued in item #11 is executed almost immediately. This command causes the microapplication to transition back to the SCAN state and begin the rapid consumption of bitstream data.
13. The host finally receives the PIC-V interrupt (issued in item #10) which indicates that display of I_{38} has begun. In response, the $\overline{\text{INT}}$ pin is deactivated and Interrupt Status cleared.
14. When the decoding process encounters a `picture_start_code` for an I-picture (in this case I_{61}), it stops discarding bitstream data and begins normal picture decoding.
15. When decoding of I_{61} is complete, the microapplication transitions back to the PAUSE state and an SCN interrupt event occurs. The interrupt is issued immediately; $\overline{\text{INT}}$ is activated and Interrupt Status[11] is set to 1.

16. The host receives the SCN interrupt and begins its interrupt processing by deactivating $\overline{INT}$ (rather than delaying processing as in item# 8).
17. While the microapplication still thinks the SCN interrupt is pending (Interrupt Status[11] = 1 even though the host has deactivated $\overline{INT}$), the active edge of VSYNC occurs which begins the first field display of I_{61}. This edge causes a PIC-V interrupt event. However, because Interrupt Status is non-zero, the interrupt is queued within the CL450 to be issued in the future.
18. The host finally finishes its interrupt processing and clears Interrupt Status.
19. At the end of display of the first block row from I_{61}, the microapplication checks for Display-time interrupts. Although RDY is not enabled, the bitstream buffer fullness is still recomputed (and written to Buffer Fullness Status, HMEM[0xb]), and the microapplication checks for queued interrupts. The PIC-V interrupt event which occurred in item #17 is in the queue, and since Interrupt Status is now 0, the PIC-V interrupt is issued: $\overline{INT}$ is activated and Interrupt Status[2] set.
20. The host receives the PIC-V interrupt, deactivates $\overline{INT}$, and clears Interrupt Status.
21. Finally, the host issues FlushBitstream() to clean out any data in the bitstream buffer which was not consumed while I_{61} was being decoded or which was received since the decoding of I_{61} completed.

12.5 Servicing Interrupts

The following are the sequences for servicing non-vectored and vectored interrupts. Note that these sequences assume that the host processes all CL450 logical interrupts simultaneously if they were received with the same physical interrupt. For other service mechanisms, see Section 12.2.2.

The sequence for servicing a non-vectored interrupt is:

1. The CL450 asserts external signal $\overline{INT}$ low by setting $\overline{Int}$ to 0 in the HOST_control register.
2. The host reads the interrupt type from the Interrupt Status register.

3. The host executes an interrupt service routine (ISR) located at a pre-determined host internal address.
4. The host sets the *Int* bit to 1 in the HOST_control register.
5. The host writes 0 to the Interrupt Status register.

The sequence for servicing a a vectored interrupt is:

1. The CL450 asserts external signal INT low by setting *Int* to 0 in the HOST_control register.
2. The host acknowledges the interrupt by asserting INTACK. If the *AIC* bit in HOST_control is 1, the CL450 sets *Int* to 1 in the HOST_control register.
3. The host broadcasts the interrupt priority ID (IPID) on A[3:1].
4. If the IPID on A[3:1] matches the value of *IPID* in the HOST_intvecr register, the CL450 responds by driving the interrupt vector *IVect* onto D[7:0].
5. The host executes the interrupt service routine (ISR) pointed to by the interrupt vector. As part of the ISR, the host reads the interrupt type from the Interrupt Status location in HMEM.
6. If the *Int* bit was not set in item #3 above, then the host sets *Int* to 1 in the HOST_control register.
7. The host writes 0 to the Interrupt Status register.

13
Audio/Video Synchronization

13.1 MPEG Conditions and Constraints

The MPEG standard includes information and restrictions intended to allow elementary audio and video decoders to be synchronized to:

- Each other (so that audio data is played when the appropriate pictures are on the video display and vice versa)
- The bitstream source (to prevent bitstream buffer underflows or overflows in both decoders)

MPEG synchronization is primarily based on the following two pieces of information which are present in the system level of an MPEG bitstream:

- *System clock references (SCRs)*, which serve as a time base for elementary decoders.
- *Presentation time stamps (PTSs)*, which indicate the time relative to the SCR at which a picture should be displayed on the screen.

13.2 CL450 Synchronization Mechanism

The CL450 provides support for each of these MPEG constructs, respectively, through use of its on-chip SCR (System Clock Reference) counter, and its NewPacket() and AccessSCR() macro commands.

13.2.1 CL450 SCR Counter

The CL450 implements an on-chip 33-bit counter which it uses as a local copy of the system clock (see Section 13.3). This counter must be initialized by the host, after which the CL450 will increment it at a nominal frequency of 90 KHz. The host may also periodically read or write this counter to keep it in step with the decoder global SCR, if any, using the AccessSCR() macro command.

It is recommended that the host update the CL450's SCR (if the CL450 is not the SCR master for the system) at least every 0.7 seconds, which is the minimum rate at which the MPEG standard requires that SCR update values be placed in a system bitstream.

13.2.2 CL450 Presentation Time Stamps

The CL450 receives PTS information from the host in the three **timeStamp** arguments of the NewPacket() macro command. Each time the host issues a NewPacket() command, it *may* place a PTS in the **timeStamp** arguments. Each PTS received by the CL450 is then associated with a `picture_start_code` in the corresponding packet according to the requirements given in the MPEG standard.

PTS values given to the CL450 may be created by the host or extracted from the packet headers in the system bitstream. All PTS values present in an MPEG system stream do not have to be passed to the CL450, but for smooth synchronization it is recommended that valid PTS values be given to the CL450 at least once every 0.7 seconds.

Figure 13-1 illustrates the association between the parameters of the NewPacket() command and `picture_start_codes` within the bitstream. Note that the words "no PTS" are used to indicate the **timeStamp** arguments to NewPacket() when the *Vld* bit (**timeStamp2**[15]) is 0, and "PTS" is used to represent the **timeStamp** arguments if *Vld* is 1.

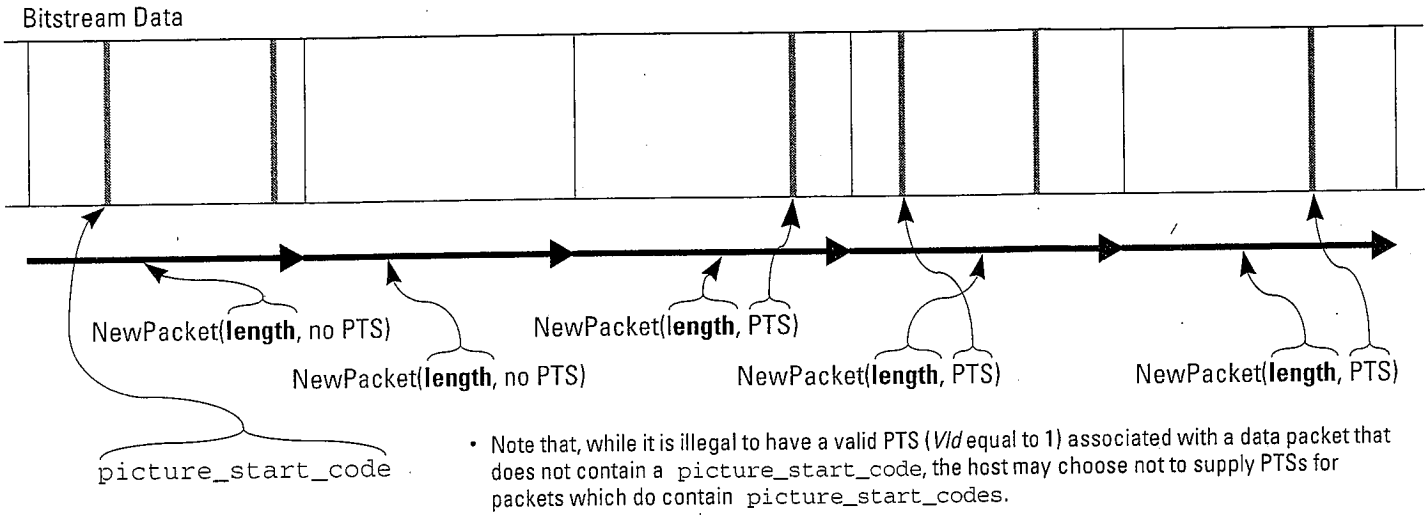


Figure 13-1 Association of NewPacket() Commands and Bitstream

13.2.3 CL450 Synchronization

The CL450 performs synchronization as follows: Each time a `picture_start_code` is detected in the bitstream, the microapplication determines if there is a valid PTS associated with that start code. There are three possible outcomes:

- No valid PTSs have been received from the host since the last time the microapplication was in the IDLE state, the FlushBitstream() command was executed, or a data error was detected. In this case, the microapplication takes no further action for synchronization and behaves as described in Section 13.5.3 on page 13-10 (Unpacketed/Packeted Data Transfer without PTSs).
- This picture does not have a valid PTS, either because none was supplied in the associated NewPacket() command or because this is not the first `picture_start_code` in this packet. In this case, the microapplication synthesizes a PTS for this picture based on the PTS for the previous picture and the current `picture_rate` parameter.
- This picture has a valid PTS which was supplied by the host via the NewPacket() command.

Once the microapplication has a PTS for the picture, it samples the value of the on-chip SCR counter. The value read is then biased (because this operation is being performed before the picture is decoded rather than at the instant of "presentation") and compared with the picture's PTS. If the PTS and the biased SCR *match* to within ±3000 SCR counts, then the picture is decoded and posted for display.

If the PTS is more than 3000 counts *greater* than the SCR, then the CL450 has been decoding too quickly or there has been a discontinuity in the PTS sequence. In either of these cases, the microapplication posts the last picture displayed for display again, waits a frame time, and then repeats the synchronization operation. In this manner, pictures are repeated as many times as necessary to allow the SCR to catch up to the decoding and display processes.

Alternately, if the PTS is more than 3000 counts *less* than the SCR, then the CL450 has been decoding too slowly or there has been a discontinuity in the bitstream. In either of these cases—if and only if the next picture to be decoded is a B-picture—the CL450 discards the picture about to be decoded and advances to the next portion of the bitstream.

Skipping the decode (and therefore the display) of a picture allows the CL450's decode process to catch up to the "actual time" as defined by the SCR counter. Only B-pictures are skipped to ensure that the CL450 always has the necessary reference frames decoded for normal decoding to continue following the skip. If the next picture to be decoded is an I- or P-picture, the PTS/SCR mismatch is ignored and will be resolved the next time a B-picture is encountered.

13.3 Updating the CL450 SCR Counter

The host may initialize or update the value in the CL450's SCR counter by issuing the AccessSCR() macro command (see page 11-13). (The AccessSCR() command may also be used to read the current state of the SCR counter.) The counter is updated in the interval between when the host sets HOST_newcmd[0] and when the microapplication clears HOST_newcmd[0].

13.3.1 SCR Source and Drift

The SCR counter is incremented based on the CL450's GCLK input, which is nominally 40 MHz. To generate the 90-KHz frequency at

which the SCR should be incremented, the CL450 divides GCLK by 444, which produces a frequency of 90,090 Hz. Because this frequency is not exactly 90 KHz, the CL450's SCR counter will run fast by roughly 0.1% relative to an ideal SCR. Not counting any inaccuracies in the GCLK frequency, this difference causes a drift of +63 SCR counts in 0.7 seconds. Since typical applications update the SCR counter at least every 0.7 seconds, and 63 is much less than the built-in jitter tolerance of 3000 SCR counts, this difference has no effect on the CL450's synchronization.

13.3.2 Automatic SCR Modifications

Immediately before the microapplication transitions from the STILL command state to the PAUSE state upon completing the decode of a still picture (see page 11-15), the microapplication clears the SCR counter to 0. The host should account for this and initialize the CL450 with the correct SCR before issuing a Play() command (which will resume decoding with synchronization).

13.4 Using SCR Masters for Synchronization

Systems that perform audio/video synchronization must have one master time-base within the decoder (the "real" SCR) to which local SCRs, such as the SCR counter within the CL450, are locked. The following sections describe several alternatives for the master SCR.

13.4.1 Synchronizing to the Bitstream

The SCR values which are embedded in the pack headers in an MPEG system bitstream are intended to indicate the time (relative to the decoder's System Clock Reference) at which the SCR fields are received by the system-level decoder. In systems where the bitstream is delivered to the decoder with the timing indicated by the SCR fields (either because the bitstream is coming from a fixed bit-rate source or because some other system element is explicitly timing the delivery of MPEG packs), the SCR values can be used to update all of the SCR counters in the decoder.

> *Note: The host, which serves as the system-level decoder and demultiplexer in systems containing the CL450, must minimize the timing skew introduced when the*

CL450's SCR and any other SCRs in the system are updated.

13.4.2 Synchronizing to the Audio Decoder

The CL450 (and the system-level decoder) can also be synchronized to the audio decoder. If the audio decoder contains its own SCR counter, then this counter can be initialized based on the first SCR contained in the bitstream. Subsequently, the audio decoder will synchronize its own operation to its SCR, and the host can update its SCR (if any) and the CL450's SCR with the contents of the audio decoder's SCR. Again, it is important that a minimal amount of SCR skew be introduced when updating the CL450.

Alternately, if the audio decoder does not perform synchronization and/or does not contain an SCR counter, SCR updates may be synthesized by the host based on the amount of audio data consumed by the audio decoder. Because MPEG-coded audio data is decoded at a fixed bit rate, the number of bits of coded data consumed by the audio decoder can be used to accurately calculate the amount of time elapsed since audio decoding began (relative to the decoder's clock source). This duration may be combined with an audio PTS to synthesize a value representing the System Clock Reference at the time the audio decoder's data consumption was sampled.

These methods effectively lock the entire decoder subsystem to the audio decoder's output clock so that any instability in that clock will be reflected in the audio and video outputs.

Note: Locking the decoder subsystem to the audio clock may cause bitstream underflow or overflow if drift occurs with bitstream delivery relative to the decoder.

13.4.3 Synchronizing from the CL450's SCR

The CL450 can also be used as the master SCR for the decoder. Assuming that the audio decoder performs synchronization and accepts SCR updates, the CL450's SCR can be initialized using data from the bitstream and then periodically sampled to provide updates for the host, if necessary, and the audio decoder.

Note: The CL450's SCR runs 0.1% fast (plus any instability in the GCLK source), so the host might wish to periodically adjust the CL450's SCR to correct this inaccuracy.

13.4.4 Synchronizing from the CL450 and VSYNC

The CL450 can be used as an SCR master without using the CL450's internal synchronization mechanism. In this case, the NewPacket() commands sent to the CL450 by the host, if any, are never given validated PTS values. Instead, the host uses the `time_code` and `temporal_reference` information written into the CL450's DRAM variable area when pictures are decoded to compute the amount of time which has elapsed since video decoding began. This timing information can then be combined with SCR or PTS information from the bitstream to create SCR updates for the host and/or audio decoder.

Two significant differences between this method and using the CL450's SCR as the SCR master are:

- This method locks the decoder subsystem to the CL450's VSYNC input via the CL450's transcoding algorithm, rather than GCLK.
- If a bitstream error occurs which causes a large portion of the bitstream to be decoded, both the CL450 and the audio decoder will immediately jump forward in time together. In the alternate case where the CL450 is synchronizing to its SCR, the next picture from the bitstream which can be decoded would be delayed until the correct display time, as specified by the PTS.

13.5 Transferring Coded Data to the CL450

Data may be transferred to the CL450 either with or without the host issuing the NewPacket() macro command. However, most systems use NewPacket() because of the microapplication features (especially audio/video synchronization) which are available only if the NewPacket() command is used.

The host must decide whether or not the NewPacket() command will be used *before* the first word of coded data is transmitted to the CL450. Typically, the engineer writing the software controlling the host will make this decision based on the type of application and bitstreams to be decoded.

However, there may be systems which will dynamically change between decoding bitstreams with and without using NewPacket(). For these systems, it is critical to remember that the microapplication is modal (when not in the IDLE state); it assumes that if it receives one NewPacket() command, NewPacket()'s are associated with all coded data.

13.5.1 Unpacketed Data Transfer

MPEG coded data can be transferred to the CL450, correctly decoded, and be displayed without using the NewPacket() macro command. However, as stated in the description of this command (page 11-21), some microapplication functions cannot be performed without the additional system-level information provided by the NewPacket() command.

13.5.2 Packeted Data Transfer

When NewPacket() is used, the microapplication associates each NewPacket() command with a section of the coded data stream. The command's **length** parameter specifies the amount of coded data.

To associate data and NewPacket() commands, the microapplication simply starts counting coded data bytes and adding **length** parameters starting with the first word of coded data and the first NewPacket() command received, respectively. The NewPacket() command associated with a section of the bitstream must be *accepted* by the microapplication before the first word of data from that section is transmitted to the CL450.

When using NewPacket(), bitstream data must be associated with a NewPacket() command. However, the host may issue any number of NewPacket() commands ahead of the arrival of bitstream data at the CL450, provided that the Command FIFO does not overflow (see Section 11.2.2, Command FIFO).

Figure 13-2 shows an example of how NewPacket() commands and bitstream data could be sent to the CL450. Note that the times marked as NewPacket() commands are the times when the CL450 *accepts* the macro command (microcode clears HOST_newcmd[0] to 0), not the earlier time when the host *issued* the macro command (host sets HOST_newcmd[0] to 1).

Items are represented in the figure as follows:

- The horizontal dimension is the time axis.
- The bitstream data is divided into numbered sections to show the portions which correspond to the similarly-numbered NewPacket() commands. Note that the horizontal size of a marked section of bitstream data has nothing to do with the size of that section in bytes but instead corresponds to the time within which it is transmitted to the CL450.
- Stars indicate the acceptance of NewPacket() commands by the microapplication.
- Arrows indicate the amount of time between the acceptance of a NewPacket() command and the reception of the associated data.

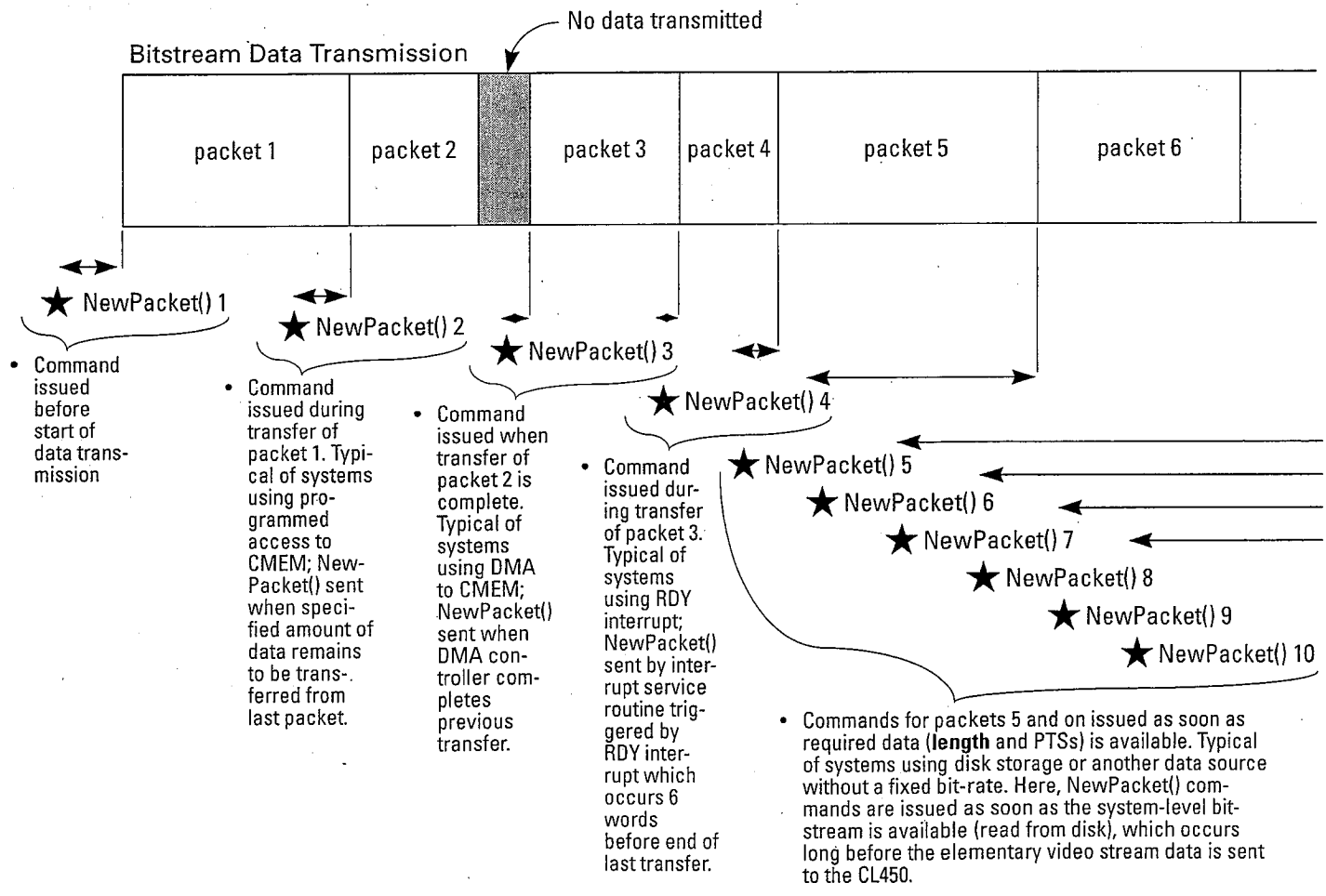


Figure 13-2 Bitstream/NewPacket() Association

13.5.3 Unpacketed/Packeted Data Transfer without PTSs

When unpacketed data is sent to the CL450 or when NewPacket() is used but the *Vld* bit in **timeStamp2** is never 1, the microapplication does not have information to perform audio/video synchronization. In this case, the rate at which pictures are displayed (and therefore bitstream data consumed) is controlled solely by the nominal VSYNC frequency specified by the host (see SetVideoFormat() on page 11-40) and by the current `picture_rate` parameter, as described in Section 15.1.

In general, if the VSYNC frequency is stable and the bitstream has been properly encoded and multiplexed *and* no bitstream errors are encountered, very little drift occurs between the CL450's video display and the output of an audio decoder operating from the same bitstream (depending on the transcoding case). However, there are several important reasons why it is recommended that audio/video synchronization be performed:

- If a bitstream error occurs, the CL450 will probably skip decoding one or more pictures. If synchronization is not being used, the decoding and display processes will jump ahead by an amount of time equal to the number of pictures skipped (potentially an entire GOP even for a single-bit error) times the nominal frame rate. This is sufficient to destroy "lip sync" with an associated audio decoder as well as potentially causing buffer fullness problems depending on how the system bitstream is demultiplexed and transferred to the CL450.
- The quality achieved by letting both the CL450 and the audio decoder run independently is dependent on the stability of audio decoder timing.
- Either the CL450 or the audio decoder could suffer from bitstream buffer underflow or overflow even if no errors occur.

13.5.4 Packeted Data Transfer with one PTS

The microapplication treats specially the first NewPacket() with a valid PTS (*Vld* bit equal to 1). Until the first picture with an associated valid PTS is decoded, the microapplication does not perform audio/video synchronization and behaves as described above. Once a valid PTS is received, the microapplication assumes that the SCR counter contains a valid value.

In addition, the microapplication will subsequently synthesize PTS values for pictures which do not have PTSs supplied by the host, based on the last host-supplied PTS value.

If the host supplies only a single valid PTS, then all pictures after the picture corresponding to the valid PTS will still be synchronized. Because the host does not provide any additional PTS values, the microapplication synthesizes PTSs for the rest of the pictures, causing them to be displayed with the exact timing specified by the `picture_rate` parameter.

This approach is recommended for systems in which elementary video streams are being played (so the host does not have a system stream to retrieve PTS values from) unless the host wants the improved error recovery behavior that synchronization can provide. Note that the "synchronized" display rate will be 0.1% fast unless the host periodically updates the SCR counter.

13.5.5 Packeted Data with Periodic PTSs

This is the typical use for audio/video synchronization, and it occurs in two basic steps:

1. PTS values are extracted from the system bitstream by the host as part of the system demultiplexing function and given to the microapplication as arguments to the NewPacket() command.
2. The bitstream is reassembled without adding or deleting any bytes, and the reassembled coded data is transferred to the CL450 as 16-bit words.

Generally, the SCR counter is periodically updated to prevent the CL450 from synchronizing to an SCR value which is divergent from the system SCR, and the host must still ensure that the SCR counter is initialized before synchronization begins.

It is also important when demultiplexing an MPEG system bitstream to create CL450 NewPacket() commands to provide for the fact that (1) MPEG packets may be an odd number of bytes, and (2) the **length** argument to the NewPacket() command must be an even number of bytes. To do this, system demultiplexers may need to adjust packet boundaries, logically moving one byte of data between two adjacent packets. This

adjustment affects the **length** parameters of the two successive NewPacket() commands but not the demultiplexed bitstream data.

> *Note: It is particularly important when reassigning a byte from one packet to the next or previous packet to ensure that, if that byte is the first byte of a* `picture_start_code` *with an associated PTS, the PTS is moved to the correct packet as well.*

How this problem is resolved depends on the particular application area. For some applications, system-level packets are guaranteed to be an even number of bytes, eliminating the problem altogether. In other applications, the frequency of video PTS values in the system bitstream is significantly higher than minimum, or slightly less quality is required in audio/video synchronization. In these cases, the PTS values associated with modified-length packets can simply be discarded and not sent to the CL450.

14
Host Access of DRAM Variables

14.1 Types of Variables

The host may access several variable locations in DRAM while the microapplication is executing. These areas include:

- *Scratch storage*: 16 words of DRAM are allocated exclusively for this purpose. The host may read and write these locations at any time, provided that the restrictions in this section are met.
- *Semaphores for host/microcode communication*
- *Bitstream parameters*

The precise organization of these locations is given in Appendix A, CL450 DRAM-variable Allocation.

The DRAM locations containing bitstream parameters are divided into the following two groups, each of which uses a semaphore to arbitrate between the CL450 and the host for access to the variables:

- *The Sequence Variable Group*: Contains information from the `sequence_header` of an MPEG bitstream and is accessed using the SEQ_SEM semaphore location.

- *The Picture Variable Group*: Contains information from the GOP and picture headers and is accessed using the PIC_SEM semaphore.

14.1.1 Sequence Variable Group

The sequence group contains the variables shown in Figure 14-1. Note that each entry occupies a separate word address, and that these addresses are specified in Appendix A.

All bits within the 16-bit DRAM words which do not correspond to a bit within the sequence header are 0 and must be written to 0 if these values are written by the host (see Section 14.2.2).

The microapplication can write to all the locations in the sequence variable group. However, writes to locations containing bitstream information (locations other than SEQ_SEM and SEQ_CONTROL) are controlled by the *SeqWP* and *SeqNoDef* bits of SEQ_CONTROL. The microapplication sets *NewSeq* (in SEQ_CONTROL) each time a `sequence_header_code` or `sequence_end_code` is decoded, and clears *NewSeq* each time new information is read and digested (triggered by decoding a `group_start_code`).

> *Note: When the microapplication writes to SEQ_CONTROL, the values of SeqWP and SeqNoDef will be retained. The host should never clear NewSeq.*

```
SEQ_SEM              /* semaphore controlling access to variable group */
SEQ_CONTROL          /* bit 0 NewSeq:   new sequence information – need
                      *                 update
                      * bit 1 SeqWP:    write-protect sequence information
                      *                 from bitstream
                      * bit 2 SeqNoDef: write-protect sequence information
                      *                 from default restoration
                      */
HORIZONTAL_SIZE
VERTICAL_SIZE
PICTURE_RATE
FLAGS                /* bit 0 == load_intra_quantizer_matrix
                      * bit 1 == load_non_intra_quantizer_matrix */
INTRA_Q[64]          /* 64-element intra_quantizer_matrix */
NON_INTRA_Q[64]      /* 64-element non_intra_quantizer_matrix */
```

Figure 14-1 DRAM-Resident Sequence Variable Group Variables

Also note that the the elements of the INTRA_Q and NON_INTRA_Q matrices are *not* stored in DRAM in the same order they are stored in the bitstream. In particular:

- Each matrix entry occupies a 16-bit word, not a byte, and the most-significant byte for all entries must be 0.
- Matrix entries are stored in row-major order, not in zig-zag order.

Figure 14-2 below shows the word-address offset from the beginning of each matrix for each matrix element.

+0x0	+0x1	+0x2	+0x3	+0x4	+0x5	+0x6	+0x7
+0x8	+0x9	+0xa	+0xb	+0xc	+0xd	+0xe	+0xf
+0x10	+0x11	+0x12	+0x13	+0x14	+0x15	+0x16	+0x17
+0x18	+0x19	+0x1a	+0x1b	+0x1c	+0x1d	+0x1e	+0x1f
+0x20	+0x21	+0x22	+0x23	+0x24	+0x25	+0x26	+0x27
+0x28	+0x29	+0x2a	+0x2b	+0x2c	+0x2d	+0x2e	+0x2f
+0x30	+0x31	+0x32	+0x33	+0x34	+0x35	+0x36	+0x37
+0x38	+0x39	+0x3a	+0x3b	+0x3c	+0x3d	+0x3e	+0x3f

Figure 14-2 INTRA_Q and NON_INTRA_Q Matrix Ordering

The contents of the sequence variable group are invalid from the time the microapplication is initialized or the Reset() command is executed until the microapplication leaves the PLAY-SETUP state.

When initialization or Reset() is complete, SEQ_SEM and SEQ_CONTROL are both 0. Once initialized, the host may allocate SEQ_SEM and write to the sequence variable group at any time.

While in the PLAY-SETUP state, the microapplication writes default values to the sequence variable group and sets the *NewSeq* bit, unless the *SeqNoDef* bit is 1.

The contents of INTRA_Q and NON_INTRA_Q are invalid any time the corresponding bits of FLAGS are 0.

14.1.2 Picture Variable Group

The picture group contains the variables listed in Figure 14-3, which includes information from both GOP and picture headers. Note that each entry occupies a separate word address and that each address is specified in Appendix A. All bits within the 16-bit DRAM words which do not correspond to a bit within the MPEG syntax are 0.

```
PIC_SEM              /* semaphore controlling access to variable group */
TIME_CODE_0          /* time_code:
                      *    [11]   drop_frame_flag
                      *    [10:6] time_code_hours
                      *    [5:0]  time_code_minutes
                      */
TIME_CODE_1          /* time_code:
                      *    [11:6] time_code_seconds
                      *    [5:0]  time_code_pictures
                      */
TEMPORAL_REFERENCE
```

Figure 14-3 DRAM-Resident Picture Variable Group Variables

Note: The microapplication writes values into these locations at the time the picture and GOP headers are decoded.

The contents of the picture variable group are invalid from the time the microapplication is initialized or the Reset() macro command is executed until the corresponding header information has been decoded.

14.2 Access Synchronization Using Semaphores

Because neither the CL450 nor the host has a "test-and-set" method for accessing DRAM locations, both the host and the microapplication must use a method in which a value is written to a semaphore and then the location is polled to determine if the value written is replaced.

The semaphores for both variable groups are independent, and the host should allocate only one at a time. Each semaphore may be in one of three states, encoded according to Table 14-1.

Table 14-1 PIC_SEM / SEQ_SEM DRAM Location Encoding

Semaphore Value	Meaning
0	Unallocated; neither host nor microapplication may access variables, but either may attempt to allocate the semaphore
0x4242	Allocated to host; microapplication may not access variables
other	Allocated to microapplication; host may not access variables

When attempting to allocate the semaphore, the host must perform operations equivalent to the pseudocode shown in Figure 14-4.

```
#define          HOST_CODE     0x4242
   /* "semAddr" (the argument to the function defined below) is one of
    * the following: */
#define          SEQ_SEM       <see Appendix A>
#define          PIC_SEM       <see Appendix A>

void AllocateSemaphore(unsigned long semAddr)
{

   do {

      while (CL450DRAM[semAddr] != 0)
         ;                                    /* make sure microcode doesn't
                                               * already have semaphore
                                               * allocated */

      CL450DRAM[semAddr]= HOST_CODE;           /* try to allocate semaphore for
                                               * host*/

      wait 0.5µs or more;
   } while (CL450DRAM[semAddr] != HOST_CODE);
                                               /* try again if we didn't win */
}
```

Figure 14-4 Allocate Semaphore Function Pseudocode

Once allocated, either semaphore may be released simply by writing a 0 to the correct DRAM location.

Note: The total time between allocating the semaphore and releasing the semaphore should not exceed 100µs; otherwise, too much CL450 bandwidth is wasted in the case where the CL450 has to wait for the host to release the semaphore before decoding can continue. Assuming the bitstream buffer does not underflow while the CL450 is performing decoding with a semaphore allocated, the CL450 will not hold a semaphore for longer than 5µs.

The sequence header information (accessed by SEQ_SEM) is accessed by the CL450 each time a `sequence_header_code`, `group_start_code`, or `sequence_end_code` is found in the bitstream. Similarly, the CL450 accesses the information controlled by PIC_SEM each time a `group_start_code` or `picture_start_code` is encountered in the bitstream.

If the host can guarantee that it will not access a group of variables at the same time the microapplication does, then the maximum time that the semaphore is allocated can be ignored. However, the host should *never* access the DRAM variables without allocating the corresponding semaphore.

14.2.1 Reading DRAM-Resident Variables

Both variable groups may be read by the host any time the appropriate semaphore has been allocated (provided that the timing restrictions in Section 14.1 are met), although there are times when the data in the variables will be stale or invalid. The procedure for reading from either group of variables is the same and is outlined in the following pseudocode:

```
#define          SEQ_SEM        <see Appendix A>
#define          PIC_SEM        <see Appendix A>

        /* "semaphore address" is either SEQ_SEM or PIC_SEM */
AllocateSemaphore(semaphore address);       /* function in Figure 14-4 */
read from DRAM address(es) containing desired variable(s);
CL450DRAM[semaphore address]= 0;            /* release semaphore to allow
                                             * CL450 to access variables */
```

Figure 14-5 DRAM Variable Read Pseudocode

14.2.2 Writing DRAM-Resident Variables

The host may write values to the sequence variable group in DRAM. This can be done to provide sequence-layer parameters to the picture decode process when random-accessing a bitstream in a location which does not contain a sequence header.

Each time the microapplication decodes a `group_start_code`, the value of SEQ_CONTROL is read. If the *NewSeq* bit is set, the microapplication reads in the values from the sequence variable group and uses them to update internal variables which control the decoding and display processes.

Note that the *NewSeq* bit is also used internally by the microapplication and is automatically set when new values are decoded from the bitstream (a `sequence_header_code` is decoded and *SeqWP* is 0) or when the default values have been restored (a `sequence_end_code` is decoded and *SeqNoDef* is 0).

The procedure for writing to the sequence variable group is shown in Figure 14-6.

```
#define          SEQ_SEM       <see Appendix A>

AllocateSemaphore(SEQ_SEM);                    /* function in Figure 14-4 */
write to DRAM address(es) containing desired variable(s);
set SEQ_CONTROL[NewSeq];
CL450DRAM[SEQ_SEM]= 0;                         /* release semaphore to allow
                                                * CL450 to access variables */
```

Figure 14-6 DRAM Variable Write Pseudocode

Note that the new values written do not take effect until the microapplication decodes the next GOP header in the bitstream, at which point it observes that *NewSeq* is 1 and reads in the new sequence information.

14.3 Timing Restrictions

The microapplication does not support host access to DRAM locations other than those listed in Appendix A while microapplication is executing; attempting to access other locations will cause indeterminate results.

For accessible locations, the following is true:

- The maximum cycle time for host access to DRAM varies according to the current state of microcode execution.
- Some worst-case cycle times are too long to permit access to the CL450's DRAM in some host environments.

There are three different maximum cycle times for host access to DRAM, depending on the operations the microcode is currently performing. Each of these maximum cycle times is determined by the maximum low time for the $\overline{\text{DTACK}}$ pin, which is used by the CL450 to extend access cycles.

The three maximum $\overline{\text{DTACK}}$-low times and their corresponding operations are given in Table 14-2.

Table 14-2 Maximum $\overline{\text{DTACK}}$ Pulse Width

Operating Mode	Maximum $\overline{\text{DTACK}}$ Pulse (ns)
No microapplication executing (CPU_control[0] == 0)	TBD
Microapplication executing: output window blanked or in vertical border time	TBD
Microapplication executing: in vertical active display time	TBD

15 CL450 Host-independent Operations

15.1 Frame Rate Conversion

The CL450 microapplication uses two parameters to determine how many times each coded picture is displayed:

- *The display's nominal field rate*: Supplied by the SetVideoFormat() macro command (see Chapter 11).
- *The coded picture rate (the sequence-layer parameter* `picture_rate`): Supplied by default from the bitstream sequence header (see Table 10-4), or from the host via the DRAM variable PICTURE_RATE (see Figure 14-1).

Note that the display field rate for NTSC is treated as exactly 30 Hz. Similarly, `picture_rates` of 29.97 and 23.976 are treated as 30 Hz, and 24 Hz, respectively.

The CL450 compares both field rate and `picture_rate` parameters to each other, with the following three possible outcomes:

- When both the nominal field rate and the `picture_rate` are equal (30 Hz or 25 Hz), the display microapplication displays each

decoded picture for two field times (VSYNC periods). Note that audio/video synchronization continues to operate, possibly causing some pictures to be displayed for multiple frame times but always an even number of field times.

- When the `picture_rate` is higher than the nominal field rate (for example, a bitstream coded at 30 Hz displayed in a 25 Hz system), the display microapplication causes some frames to be displayed for only one field time rather than two.
- When the `picture_rate` is slower than the nominal field rate (for example, a 25-Hz bitstream display with a 60-Hz VSYNC frequency), the display microapplication causes some frames to be displayed for an extra field time (three fields per frame rather than two).

Field repeating and skipping is done at the required frequency for the nominal `picture_rate` to be correct. Note that, unlike synchronization, these rate conversion operations take place at the *field* level rather than the frame level.

For purposes of rate conversion, the CL450 microapplication treats 25 Hz (either input `picture_rate` or display frame rate) as 24 Hz. Because of this, bitstreams coded at 25 Hz and displayed at 30 Hz, or bitstreams coded at 24 Hz and displayed at 25 Hz, will play at a slightly incorrect speed. Table 15-1 indicates the cases in which the actual play speed does not match the nominal speed, and the nominal number of synchronization events per second if A/V synchronization is being performed.

Table 15-1 Picture Display Rates: Play Speed vs. Nominal Speed

Format	`picture_rate`	Rate Error (%)	Average Synchronizations per Second
NTSC (30 Hz)	25 Hz	4.2	Slow 1.25
PAL (25 Hz)	24 Hz/23.976 Hz	4.0	Fast 1.00

Conceptually, the synchronization and rate conversion processes operate entirely independently. The synchronization process decides how many times a particular picture should be displayed, and the pull-down process decides for each picture *time* how many fields that picture

should be repeated. Extra fields can be added to the second frame display of a picture as easily as the first, and if a picture is repeatedly displayed for enough frame times, it will have extra fields added to its display repeatedly.

> *Note: When performing this kind of rate conversion, whether the first field displayed for a picture is odd or even depends solely on what has occurred before and is very difficult to predict.*

15.2 Error Recovery and Concealment

The CL450 handles signaled and unsignaled errors the same way. That is, when the CL450 detects an illegal (non-MPEG) construct (unsignaled error) or a `sequence_error_code` (signaled error) in the bitstream, it assumes that a transmission or storage error has occurred and that a portion of the bitstream is corrupt. (A signaled error is denoted by a `sequence_error_code` hex value of 0x000001B4.)

In general, the CL450's error-recovery strategy assumes that a relatively small section of the bitstream has been corrupted. For example, if an error is detected while decoding a B-picture, the CL450 assumes that it may decode the next B-picture correctly. If, instead, the error resulted in an entire reference frame being lost, then all pictures decoded until an I-picture and a reference picture have been found and decoded will be in error. Similarly, if an error causes a sequence header to be lost, then subsequent decode operations may be conducted with incorrect dequantization matrices, resulting in incorrect images.

In practice, this error-recovery strategy allows the CL450 to resume decode and display as soon as possible. A strict algorithm which avoided all possibility of displaying an erroneous picture would very often have to wait until a new sequence header was found before resuming decode, which could be an arbitrarily long period.

15.2.1 During Picture Decode

If the error occurs while a picture is being decoded, the microapplication abandons that picture and begins scanning through the bitstream to find the next picture for which it has enough information to decode. Because of the forward and backward prediction linkages between pictures, the kind of picture searched for depends on the picture which was being decoded when the error was detected. For example:

- If a B-picture was being decoded, then the microapplication resumes decoding and display with the next picture of any type found in the bitstream. Because B-pictures contain no information used for future decoding, decoding can resume with the next B-, P-, or I-picture found.
- If a P- or an I-picture was being decoded when the error was detected, decoding does not resume until the next I-picture is found. Intervening P- and B-pictures are discarded because both kinds could contain motion vectors referring to the incompletely decoded reference picture.

While the decode process is searching for the next picture to decode, sequence and GOP headers are decoded normally if encountered.

15.2.2 During Header Decode

If an error is encountered while decoding header information, the CL450 behaves as described above for errors detected in B-pictures. When an error occurs in a header, the corresponding variable area in DRAM may be corrupted. In particular, it is impossible to determine which values are from the new, partially-decoded header and which are old. It is also impossible to tell if the actual error occurred significantly before it was detected.

15.2.3 During Bitstream Underflow

Because bitstream buffer underflow ordinarily results only from catastrophic failure at the system level, the CL450 microapplication does not necessarily behave as otherwise indicated if bitstream buffer underflow occurs. In general, any time that the microapplication is in a command processing state in which bitstream decoding occurs and the bitstream buffer underflows, microapplication execution stalls in the middle of the decode operation. The most noticeable result is that low-priority macro commands no longer are executed because the criteria for execution (given in Section 11.4, Command Latency) are based on the completion of picture decoding operations.

Functions implemented by the CL450's internal interrupt handlers (including video display, Display-time interrupt production, macro command acceptance, and high-priority macro command execution) continue even if the bitstream buffer has underflowed.

Note: The host could create behavior similar to a bitstream buffer underflow by retaining control of the DRAM variables for longer than the maximum specified time (see Section 14.1, Types of Variables). This action blocks the decoding process and results in similar functional behavior.

15.3 Operating Restrictions

The CL450 is designed to work over a particular set of input data and parameters. The contents of the rest of this document is true only within this operating domain. The most significant restrictions on the operating domain are given in this section.

15.3.1 Bitstream

The CL450 is intended for use with "constrained parameters" video bitstreams only. In general, only those bitstreams which have (or could have) the MPEG `constrained_parameters_flag` set are consistently decoded correctly. The only exceptions to this are:

- Still pictures meeting the requirements of the MPEG standard for bitstreams which can be decoded by an STD and which contain individual field pictures which do not exceed the requirements for a constrained parameters bitstream
- Bitstreams with coded data rates up to 5.0 Mbits/second but meeting all other requirements for the `constrained_parameters_flag`.

The bitstream's `picture_rate` parameter must have a value between 0001_2 (23.976 Hz) and 0101_2 (30.0 Hz), inclusive.

The CL450 also cannot decode bitstreams consisting of D-pictures (DC intra-coded pictures, indicated by a `picture_coding_type` of 100_2).

15.3.2 Output Window and Timing

Because the CL450 microapplication begins the display of B-pictures before they have been completely decoded, there are some otherwise-legal combinations of coded picture size and SetBorder() and SetWindow() parameters which may produce incorrect visual results.

Typically, the microapplication will have decoded approximately 55% of the macroblocks in a B-picture before the first possible time a scan line from the picture can be displayed. This condition is shown in Figure 15-1, which assumes use of the tallest possible output window located at the top of the video display area (minimum **topBorder**). If the output window starts further down the display screen, a larger portion of the picture is decoded before display of the picture begins, assuming that display of the picture begins with the first scan line of the picture.

Any combination of SetWindow() and SetBorder() parameters which cause scan lines to be displayed at or below their position on the screen shown in Figure 15-1 will function correctly. However, combinations of windowing parameters which cause scan lines to be moved *up* relative to their default positions may cause visual glitches.

The host may compute the sections of the picture that will be completely decoded at a given time by assuming that:

- 45% of the macroblocks in the picture remain to be decoded at the end of the top border before a B-picture is displayed.
- Macroblocks are decoded at a constant rate through the remainder of the picture.
- All macroblocks are decoded 16 scan lines before the beginning of the bottom border.

For example, assume that an NTSC-resolution (15-macroblock high) picture is being decoded. In this case, 8.25 macroblock rows are completely decoded when the display of each B-picture begins (55% of 15). During picture display, macroblock rows are decoded at a rate of 0.03 macroblocks per scan line (15-8.25 / 240-16). Thus, when display scan line 150 occurs (counting from the top of the default output window), 12.75 macroblock rows will be decoded (150 x 0.03 + 8.25). Note that this number should always be truncated because only completely decoded macroblock rows can be displayed without glitches.

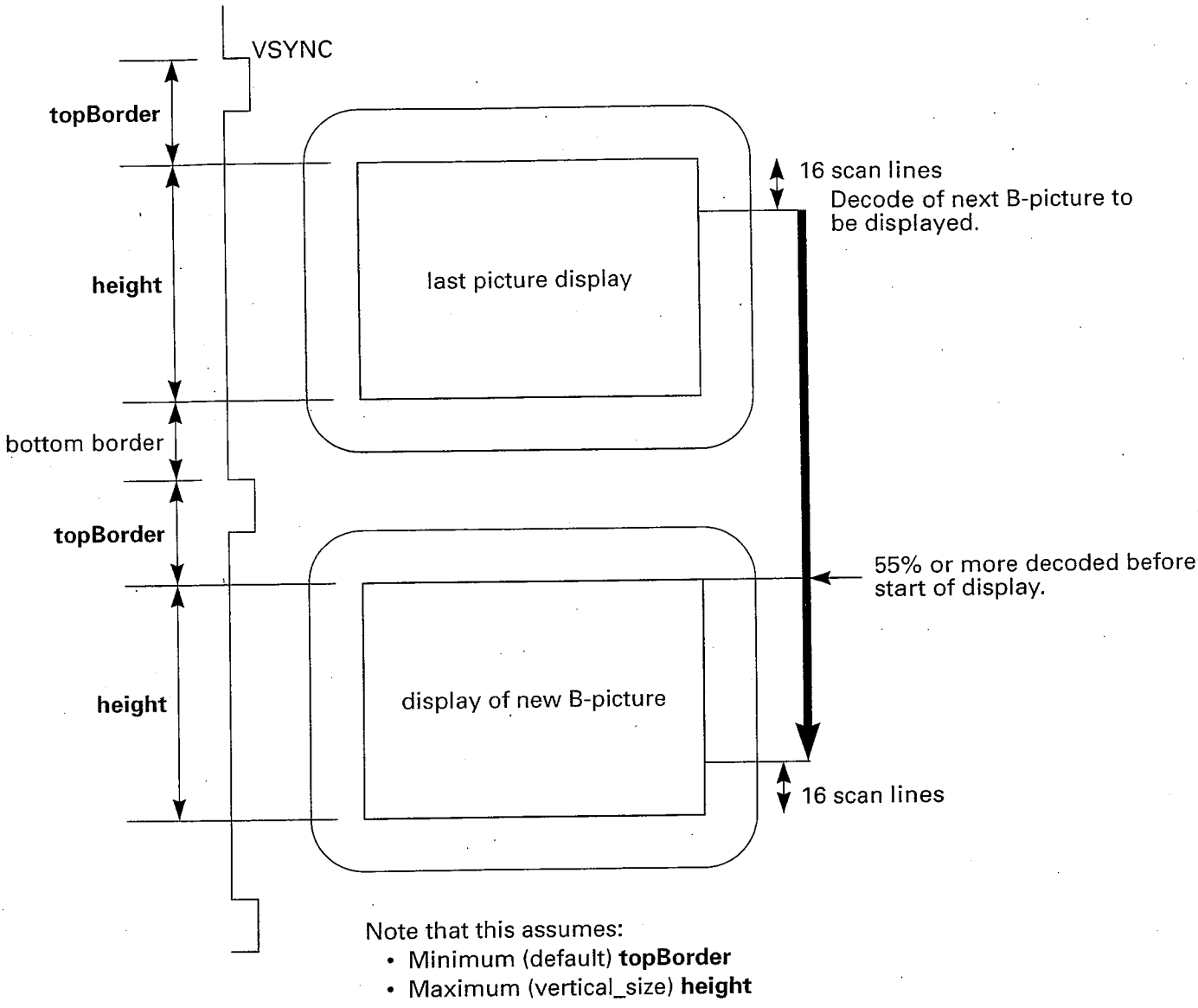


Figure 15-1 B-picture Decode and Display

The method just described is a conservative way of determining (1) which scan lines are available for display, and (2) which macroblock rows will always display correctly (based on the macroblock rows which this method indicates are completely decoded). However, this method will also show that the bottom macroblock row of a picture can never be displayed above its default location on the screen. This result is too conservative for many applications, so experimentation with the actual system and bitstream are recommended.

These restrictions are also present when decoding and displaying pictures with extreme aspect ratios (very tall and narrow, or very short and wide). In these cases, the default window configuration does not cause all of the picture to be displayed.

To calculate when portions of odd shaped pictures can be displayed, you can (1) assume that 55% of the picture is decoded before display begins, (2) compute the number of macroblocks per *second* (not scan lines) that are decoded in the NTSC case, and then (3) multiply this value by your $\overline{\text{HSYNC}}$ period to determine the number of macroblocks decoded per scan line. This should provide a conservative estimate of when macroblock rows from the picture may be displayed.

Appendix A
CL450 DRAM-Variable Allocation

This appendix contains the DRAM addresses for each of the host-accessible DRAM variables, starting with the host scratch storage listed in Table A-1.

Table A-1 DRAM Addresses for Host Scratch Storage

DRAM Address	Variable Name
0x0	HOST_WORD_0
0x1	HOST_WORD_1
0x2	HOST_WORD_2
0x3	HOST_WORD_3
0x4	HOST_WORD_4
0x5	HOST_WORD_5
0x6	HOST_WORD_6
0x7	HOST_WORD_7
0x8	HOST_WORD_8

DRAM Address	Variable Name
0x9	HOST_WORD_9
0xa	HOST_WORD_A
0xb	HOST_WORD_B
0xc	HOST_WORD_C
0xd	HOST_WORD_D
0xe	HOST_WORD_E
0xf	HOST_WORD_F

Table A-2 lists the locations of the scalar variables in the sequence group.

Table A-2 DRAM Addresses for Sequence Variable Group

DRAM Address	Variable Name
0x10	SEQ_SEM
0x11	SEQ_CONTROL
0x12	HORIZONTAL_SIZE
0x13	VERTICAL_SIZE
0x14	PICTURE_RATE
0x15	FLAGS

In Table A-3 and Table A-4, the INTRA_Q and NON_INTRA_Q matrix element addresses are shown arranged in the same physical order as the corresponding default matrices in the MPEG standard (given with the definitions of the `load_intra_quantizer_matrix` and `load_non_intra_quantizer_matrix` flags). Matrix element (0,0) is located in the upper left, and the matrices are stored in row-major format within DRAM.

Table A-3 DRAM Addresses for INTRA_Q Matrix

0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27
0x28	0x29	0x2a	0x2b	0x2c	0x2d	0x2e	0x2f
0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37
0x38	0x39	0x3a	0x3b	0x3c	0x3d	0x3e	0x3f
0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47
0x48	0x49	0x4a	0x4b	0x4c	0x4d	0x4e	0x4f
0x50	0x51	0x52	0x53	0x54	0x55	0x56	0x57
0x58	0x59	0x5a	0x5b	0x5c	0x5d	0x5e	0x5f

Table A-4 DRAM Addresses for NON_INTRA_Q Matrix

0x60	0x61	0x62	0x63	0x64	0x65	0x66	0x67
0x68	0x69	0x6a	0x6b	0x6c	0x6d	0x6e	0x6f
0x70	0x71	0x72	0x73	0x74	0x75	0x76	0x77
0x78	0x79	0x7a	0x7b	0x7c	0x7d	0x7e	0x7f
0x80	0x81	0x82	0x83	0x84	0x85	0x86	0x87
0x88	0x89	0x8a	0x8b	0x8c	0x8d	0x8e	0x8f
0x90	0x91	0x92	0x93	0x94	0x95	0x96	0x97
0x98	0x99	0x9a	0x9b	0x9c	0x9d	0x9e	0x9f

Table A-5 lists the DRAM word locations of variables in the picture group.

Table A-5 DRAM Addresses for Picture Variable Group

DRAM Address	Variable Name
0x16	PIC_SEM
0x17	TIME_CODE_0
0x18	TIME_CODE_1
0x19	TEMPORAL_REFERENCE

Table A-6 indicates the location in DRAM for the encoding of the microapplication-executable header information.

Table A-6 DRAM Addresses for Microapplication Version Information

DRAM Address	Contents
0xa0[15:8]	Microapplication version number, high byte (REV_HI). This contains the major version (should be 0x02 for the microapplication covered by this document).
0xa0[7:0]	Microapplication version number, low byte (REV_LO). This contains the minor version number.
0xa1[15]	This bit is used to denote experimental microapplication executables. If this bit is set, it is indicative of an intermediate development version or a version with internal debugging functionality.
0xa1[14:8]	Extensions ID. Should be 0x00. Other values indicate that the microapplication is an experimental (debugging) revision and/or contains functions not specified in this document.
0xa1[7:0]	Product ID. Should be 0x02.
0xa2	Initial value for CPU_pc register.
0xa3	RESERVED

Appendix B Microapplication Executable File Format

This appendix describes the format of the CL450 executable file and applies to microapplication versions 2.00 through 2.FF *only*.

The CL450 microapplication executable is distributed in a single file with a ".BIN" extension. Once a microapplication executable file has been copied from the distribution disk, its name can be changed.

B.1 Syntax Conventions

The syntax for the file is represented with the same conventions used in the compressed bitstream syntax of the MPEG standard. In addition to the data-type symbols used in the MPEG standard, the new symbols shown in Table B-1 will be used.

Table B-1 Executable File Syntax Symbols

Symbol	Definitions
ubyte	Unsigned integer, occupying a single byte
uilsbf	Unsigned integer, least-significant *byte* first
silsbf	Signed integer, least-significant *byte* first
strfbf	String, null-terminated, first (left-most) byte first

It is assumed that the file is read one *byte* at a time, and that bit ordering within the file bytes is preserved when the executable file is read.

B.2 File Structure

The microapplication executable consists of a single header, followed by one or more code segments. The top-level syntax of the file is as follows:

```
microcode_executable()
{
   file_header;
   for (c=0;c < num_segments;c++)
   code_segment();
}
```

The file_header includes the total number of code segments in the executable file (num_segments), information about the microapplication, and information needed to initialize the CL450 on-chip CPU's instruction memory (IMEM) and program counter.

The code_segments contain the CL450 executable code, and must be loaded into the CL450's local DRAM by the host. In addition, one or more of the code segments must *also* be loaded into IMEM to serve as a bootstrap for the CL450's CPU.

B.3 File Header Structure

The file header contains 10 entries as shown below. Each entry is described following the file_header syntax shown on the next page.

```
file_header()
{
   magic_number                          16 uilsbf
   rev_hi                                8 ubyte
   rev_lo                                8 ubyte
   productID                             16 uilsbf
   init_PC                               16 uilsbf
   imem_dram                             16 uilsbf
   exe_length                            32 ubyte
   num_segments                          16 uilsbf
   RESERVED                              32 uilsbf
   comment_string                        80.8 strfbf

}
```

magic_number — This entry in the file_header contains the value 0xC3C3 for all valid CL450 microapplication files. If a file does not begin with this value, then it is not a valid ".BIN" microapplication executable.

rev_hi — This 8-bit value gives the major version number of the microapplication executable contained in the file. For microapplication executables and executable files covered by this document, the value is 0x02.

rev_lo — This 8-bit value gives the minor version number of the microapplication executable. Typically, larger values indicate later microapplication releases.

productID — This value is used to encode the feature set of the microapplication executable. If productID[15] is 1, then the executable is an "experimental" microapplication release and should be regarded with caution. Experimental microapplication releases are typically made as part of the microapplication support and system debugging process and will not match the microapplication functional specification in one or more ways. Experimental microapplications should not be used except under the direction of C-Cube Microsystems technical support.

The least significant bits of productID are used to encode the feature set implemented by the microapplication. For a microapplication with a major version number (rev_hi) of 0x02, a productID with the lower bits equal to 2 indicates that the feature set described by this document is implemented. Previous microapplication releases with major version numbers of 0x01 used the values 0 and 1 to indicate the discontinued "CL450 Baseline" and "CL450i" feature sets.

Finally, the high-order bits of productID (bit 14 down to the bits described above) are reserved for the encoding of customer-specific microapplication modifications.

init_PC — This entry contains the value the host should write to the CL450's CPU_pc register during the microapplication loading process prior to enabling CPU execution.

imem_dram — This value is a *byte* address in the CL450's DRAM at which the DRAM and IMEM address spaces are conceptually overlapped. All data from the code_segments which is written to the DRAM range from byte address "imem_dram" through byte address "imem_dram+2044" inclusive must also be written by the host to the corresponding IMEM address. IMEM addresses can be computed from DRAM addresses within the "overlapped" range as follows:

IMEMaddress= (DRAMaddress - imem_dram) / 4;

Note that IMEM may only be written 32-bits at a time (requiring two sequential 16-bit write operations from the host).

exe_length — This entry gives the length in *bytes* of the complete executable file, including the header.

num_segments — This entry gives the number of code_segments contained in the file. If the end of the last code segment in the file and the actual end of the file do not coincide exactly, the executable file has been corrupted and should not be used.

RESERVED — This 32-bit quantity is reserved for future use and executable file readers should ignore its contents.

comment_string — Thisfile_header:entries:magic_number string contains ancillary information about the microapplication contained in the file. Typically this includes at least the microapplication copyright notice.

B.4 Code Segment Structure

Each code segment contains two entries and an array of values, as shown below. Each entry is described following the code_segment syntax:

```
code_segment()
{
   seg_length ;                              32 uilsbf
   seg_address;                              32 ubyte
   for (c=0;c < (seg_length/4);c++){

      instruction_word_1 ;                   16 uilsbf
      instruction_word_0 ;                   16 uilsbf
}
```

seg_length — This entry gives the length of the segment in *bytes*. Each CL450 microinstruction is 32-bits long, occupying four bytes.

seg_address — The entry gives the DRAM *byte* address at which this segment should begin loading. The loaded segment should occupy DRAM byte addresses "seg_address" through "seg_address + seg_-length - 1" inclusive. Note that if part or all of the DRAM area occupied by a segment coincides with the "DRAM/IMEM overlap" region (see imem_dram, above), then that part of the segment must be written to the appropriate portion of IMEM *as well as DRAM*.

instruction_word_1 and **instruction_word_0** — These values are the actual microapplication instructions. They should be loaded into the CL450's local DRAM (and possibly IMEM). The first value from the file (instruction_word_1) is the *most* significant half of each CL450 instruction, and should be written to the CL450 *first*. When writing code_-segment data to the CL450's DRAM, the first word of data from the

executable file (instruction_word_1) goes in the lower DRAM *word* address, and instruction_word_0 goes in the higher DRAM *word* address. When writing code_segment data to IMEM, instruction_word_1 is written to the CPU_imem register *first*, followed by instruction_word_0.

Appendix C
CL450 Microapplication Distribution Disk

This appendix describes the C-Cube Microsystems microapplication distribution disk, which contains the following files:

- *\DRAM450.BIN*: This file contains the CL450 microapplication executable in the format described in Appendix B.
- *\450DEMO.EXE*: This file is a demonstration program, which runs under Microsoft Windows™ and is compiled for execution on the CL450 development board. When executed, the microapplication executable located in the same directory in the file "DRAM- 450.BIN" is loaded, and the MPEG elementary video bitstream located in the same directory in the file "MPEG.NFL" is decoded and displayed.
- *\MPEG.NFL*: This file contains an MPEG elementary video bitstream.
- *\README.DOC*: This file contains a complete description of the contents of the subdirectories, as well as release notes and errata (if any) for the particular microapplication version contained on the disk.

- \KERMIT.INI: This application contains a configuration file read by 450DEMO.EXE (by the C3VIO library) at program start-up for information on the configuration of the CL450 development boards. The version of this file supplied with the microapplication corresponds with the factory settings of the development board.

The C-Cube Microsystems microapplication distribution disk also contains the following directories:

- *\450DEMO* : This directory contains the source, header, and make files used to compile 450DEMO.EXE.
- *\C3LIB* : This directory contains the source, header, and make files for an example library of CL450 I/O routines. This library may be built either for DOS or Windows applications, although a version complied for DOS is supplied and is required to build 450DEMO.EXE. By changing the routines in this library, it should be possible to port 450DEMO.EXE to any other CL450-based subsystem.

Appendix D
CL450 Troubleshooting

This appendix contains information which should be collected before reporting a suspected bug in the CL450 microapplication to C-Cube Microsystems customer support:

1. Determine the version number of your microapplication by examining the appropriate DRAM location (see Table A-6) while the microapplication is loaded.
2. Determine the value of CPU_control[0] after the failure is observed.
3. If CPU_control[0] is 0, then either the host or the microapplication halted the CL450's CPU. Capture the contents of CPU_pc. If CPU_control[0] is 1, then the CL450's CPU is still running. Read the contents of CPU_pc and attempt to determine if the microapplication is looping within one or more areas of IMEM. Report the IMEM addresses (CPU_pc values) for these areas.
4. Capture the contents of CPU_intenab and CPU_int.
5. Capture the contents of DRAM addresses 0x0 through 0x360.

6. Capture the contents of TMEM addresses 0x0 through 0x7f. Note that reading TMEM contents from the host only retrieves the least-significant 16 bits from each 24-bit TMEM word. This is sufficient for an initial bug report, but you may be asked to run a special microapplication which reads and dumps the full TMEM contents.
7. Capture the contents of HMEM addresses 0x0 through 0xf.

When reporting the content of a CL450 register, it is important to get all 16 bits read from the CL450, not just the bits which are defined.

Most of the items listed above may change while the microapplication is executing. Because of this, values should be captured twice. First, perform the operations which cause the suspected bug and capture the values while the microapplication is still executing. Second, reset and restart the system and execute until the bug occurs again. Then clear CPU_control[0] and capture all of the values. This will ensure that the microapplication is not changing the state while it is being captured.

The registers which must be used to read from TMEM (CPU_taddr and CPU_tmem) are used by the microapplication. Because of this, reading TMEM while the microapplication is executing will not always be successful. Sometimes the microapplication will interfere with the results of reading TMEM, and sometimes the act of reading TMEM will cause the microapplication to malfunction.

Finally, a description of the host operations and/or bitstream which cause the suspected bug to occur should be supplied. It is possible that C-Cube Microsystems will request a copy of the bitstream to duplicate the problem.

Index

C

D

E

F

G

H

I

N

O

P

R

S

T

U

V

W

X

Y

Z

Customer Feedback

C-Cube Microsystems is always working to improve the quality of our documentation. If you have comments or suggestions about this document, please send us a marked-up copy of the page or pages or send us an e-mail message. We will acknowledge all comments received. Our address is:

Technical Publications Department
C-Cube Microsystems
1778 McCarthy Boulevard
Milpitas, CA 95035

phone: (408) 944-6300
fax: (408) 944-6314

e-mail: techpubs@c-cube.com

North American Representatives

Alabama

M^2I
1910 Sparkman Avenue
Huntsville, AL 35816
phone: 205-830-0498
fax: 205-837-7049

Arkansas

TL Marketing
14850 Quorum Dr., Suite 100
Dallas, TX 75240
phone: 214-490-9300
fax: 214-960-6075

California

Bager Electronics
519 Encinitas Blvd.
Encinitas, CA 92024
phone: 619-632-8816
fax: 619-632-8810

Bager Electronics
17220 Newhope Street, Suite 209
Fountain Valley, CA 92708
phone: 714-957-3367
fax: 714-546-2654

Bager Electronics
6324 Variel, Suite 314
Woodland Hills, CA 91367
phone: 818-712-0011
fax: 818-712-0160

Norcomp
2140 Professional Drive, #200
Roseville, CA 95661
phone: 916-782-8070
fax: 916-782-8073

Norcomp
1267 Oakmead Parkway
Sunnyvale, CA 94086
phone: 408-733-7707
fax: 408-774-1947

Colorado

Promotech Sales, Inc.
2901 S. Colorado Blvd.
Denver, CO 80222
phone: 303-692-8484
fax: 303-692-8416

Florida

M^2I
402 S. North Lake Blvd
Suite 1016
Altamonte Springs, FL 32701
phone: 407-260-6422
fax: 407-260-6460

Georgia

M^2I
3000 Northwoods Parkway #110
Norcross, GA 30071
phone: 404-447-6124
fax: 404-447-0422

Hawaii

Bager Electronics
17220 Newhope Street, Suite 209
Fountain Valley, CA 92708
phone: 714-957-3367
fax: 714-546-2654

Illinois

Beta Technology Sales, Inc.
1009 Hawthorn Drive
Itasca, IL 60143
phone: 708-250-9586
fax: 708-250-9592

Iowa

Cahill, Schmitz & Howe
226 Sussex Drive. N.E.
Cedar Rapids, IA 52402
phone: 319-377-8219
fax 319-377-0958

Louisiana

TL Marketing
14850 Quorum Dr., Suite 100
Dallas, TX 75240
phone: 214-490-9300
fax: 214-960-6075

Massachusetts

Advanced Technical Services
348 Park Street, Suite 102
North Reading, MA 01864
phone: 508-664-0888
fax: 508-664-5503

Minnesota

Cahill, Schmitz & Cahill, Inc.
315 N. Pierce Street
St. Paul, MN 55104
phone: 612-646-7217
fax: 612-646-4484

Mississippi

M^2I
1910 Sparkman Avenue
Huntsville, AL 35816
phone: 205-830-0498
fax: 205-837-7049

Montana

Promotech Sales, Inc.
2901 S. Colorado Blvd.
Denver, CO 80222
phone: 303-692-8484
fax: 303-692-8416

Nevada

Clark County Only
Bager Electronics
6324 Variel, Suite 314
Woodland Hills, CA 91367
phone: 818-712-0011
fax: 818-712-0160

Other
Norcomp
2140 Professional Drive, #200
Roseville, CA 95661
phone: 916-782-8070
fax: 916-782-8073

New Jersey

Parallax
734 Walt Whitman Road
Melville, NY 11747
phone: 516-351-1000
fax: 516-351-1606

North American Representatives (cont.)

New York

Empire Technical Associates
29 Fennell Street, Suite A
Skaneateles, NY 13152
phone: 315-685-5703
fax: 315-685-5979

Empire Technical Associates
349 W. Commercial Street
Suite 2920
East Rochester, NY 14445
phone: 716-381-8500
fax: 716-381-0911

Parallax
734 Walt Whitman Road
Melville, NY 11747
phone: 516-351-1000
fax: 516-351-1606

North Carolina

M^2I
1200 Trinity Road
Raleigh, NC 27607
phone: 919-851-0010
fax: 919-851-6620

North Dakota

Cahill, Schmitz & Cahill, Inc.
315 N. Pierce Street
St. Paul, MN 55104
phone: 612-646-7217
fax: 612-646-4484

Oklahoma

TL Marketing
14850 Quorum Dr., Suite 100
Dallas, TX 75240
phone: 214-490-9300
fax: 214-960-6075

South Carolina

M^2I
1200 Trinity Road
Raleigh, NC 27607
phone: 919-851-0010
fax: 919-851-6620

South Dakota

Cahill, Schmitz & Cahill, Inc.
315 N. Pierce Street
St. Paul, MN 55104
phone: 612-646-7217
fax: 612-646-4484

Tennessee

M^2I
3000 Northwoods Parkway #110
Norcross, GA 30071
phone: 404-447-6124
fax: 404-447-0422

Texas

TL Marketing
14850 Quorum Dr., Suite 100
Dallas, TX 75240
phone: 214-490-9300
fax: 214-960-6075

TL Marketing
8100 Shoal Creek, Suite 250
Austin, TX 78758
phone: 512-371-7272
fax: 512-371-0727

TL Marketing
14343 Tory Chase Blvd. Suite I
Houston, TX 77014
phone: 713-587-8100
fax: 713-580-7517

Wisconsin

Western
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce Street
St. Paul, MN 55104
phone: 612-646-7217
fax: 612-646-4484

Eastern
Beta Technology Sales, Inc.
9401 W. Beloit Road, Suite 409
Milwaukee, WI 53227
phone: 414-543-6609
fax: 414-543-9288

Canada

Electrosource
6875 Royal Oak
Burnaby, BC
Canada V5J 4J3
phone: 604-435-2533
fax: 604-435-2538

Electrosource
230 Galaxy Blvd.
Rexdale, ONT
Canada M9W 5R8
phone: 416-675-4490
fax: 416-675-6871

Electrosource
340 March Road, Suite 503
Kanata, ONT
Canada
K2K 2E4
phone: 613-592-3214
fax: 613-592-4256

Electrosource
6600 Trans Canada Highway,
Suite 420
Pointe Claire, Quebec
H9R 4S2
phone: 514-630-7486
fax: 514-630-7421

Others (Not Listed)

Contact nearest office of
C-Cube Microsystems

International Representatives and Distributors

France

NEWTEK (Rep/Dist.)
8, rue de l'Esterel
SILIC 583
94663 Rungis Cedex
phone: (33) 1-46.87.22.00
fax: (33) 1-46.87.80.49

United Kingdom

Kudos Thame Ltd. (Rep/Dist.)
55 Suttons Park, London Rd.
Reading, BERKS RG6 1AZ
phone: (44) 734-351010
fax: (44) 734-351030

Germany

Metronik GmbH (Rep/Dist.)
Leonhardsweg 2
8025 Unterhaching
phone: (49) 89-61108-0
fax: (49) 89-6116858

Australia

ZATEK Components Pty Ltd. (Rep/Dist.)
Suite 8, 1059 Victoria Rd.
West Ryde 2114
Sydney
phone: (61) 2-874-0122
fax: (61) 2-874-6171

Hong Kong

MEMEC Asia Pacific (Rep/Dist.)
Unit No 2520-2525
Tower 1
Metroplaza
Hing Fong Road
Kwai Fong, N.T.,
phone: (852) 410-2780
fax: (852) 418-1600

Japan

Kubota C-Cube Inc.
Fuso Building, 7F, 2-12-8
Shin-Yokohama
Kohoku-Ku
Yokohama, Kanagawa 222
phone: (81) 45-474-7571
fax: (81) 45-474-7570

Korea

MEMEC Asia Pacific (Rep/Dist.)
4th Floor, Jae Woong Bldg
176-11 Nonhyun-Dong
Kangnam-ku
Seoul
phone: (82) 2-518-8181
fax: (82) 2-518-9419

Singapore

Serial System Marketing (Rep/Dist.)
11 Jalan Mesin
Standard Industrial Bldg, #06-00
Singapore 1336
phone: (65) 280-0200
fax: (65) 286-6723

Republic of China

MEMEC Asia Pacific(Rep)
14F-1, 171 Section 58
Min Sheng East Road
Hai Hwa Building
Taipei
Taiwan, R.O.C.
phone: (886) 2 760-2028
fax: (886) 2 765-1488

ALLY, Inc. (Dist.)
7F, 18, Alley 1 Lane 768, Sec. 4
Pa Teh Rd.,
Taipei
Taiwan, R.O.C.
phone: (886) 2 788-6270
fax: (886) 2 786-3550

C-Cube Microsystems Sales Offices

Home Office

C-Cube Microsystems
1778 McCarthy Boulevard
Milpitas, CA 95035
phone: 408-944-6300
fax: 408-944-6314

Eastern Area Office

C-Cube Microsystems
One Kendall Square, Suite 220
Cambridge, MA 02139
phone: 617-621-7180
fax: 617-621-7179

Southwestern Area Office

C-Cube Microsystems
453 Bristol Avenue
Cardiff, CA 92007
phone: 619-632-0864
fax: 619-632-0864

European Office

C-Cube Microsystems
44 Dartford Road
Sevenoaks, Kent
UK TN 133 TQ
phone: (44) 732 743 256
fax: (44) 732 450 151

Japan Office

Kubota C-Cube Inc
Fuso Building 7F
2-12-8 Shin-Yokohama
Kohoku-ku
Yokohama, Kanagawa 222
phone: 81-45-474-7571
fax: 81-45-474-7570