OVERVIEW

Features and Applications

Standard

- Implements proposed baseline JPEG standard for image compression

High Performance

- Real-time compression and decompression of NTSC, square PAL, and CCIR 601 video frames
- Up to 2 MBytes/second sustained compressed data rate (CL550-35)
- Highly pipelined DCT/IDCT processor running at up to 35 MHz (CL550-35)

Flexibility

- Support for 8-bit grayscale, RGB, CMYK or 4:4:4:4, and YUV color space input/output
- User-accessible Quantizer and Huffman tables
- Frame-by-frame adjustment of compression ratio

High Integration

- On-chip DCT/IDCT processor
- On-chip Quantizer and Huffman tables
- On-chip video interface
- On-chip 16-bit or 32-bit Host Bus Interface
- Standard 144-pin PGA package for C-Cube CL550-35 and CL550-30
- Standard 144-pin MQUAD package for C-Cube CL550-30 and CL550-10
- CMOS: dissipates 3.4 watts maximum at 35 MHz

Applications

- Multimedia
- Video editing
- Color publishing and graphic arts
- Image processing, storage, and retrieval
- Color printers and scanners
- High-speed image transmission for LAN, modem, and color facsimile
- Digital cameras

Description

The C-Cube CL550® is a single-chip image compression/decompression processor. It implements the proposed baseline CCITT/ISO Joint Photographic Experts Group (JPEG) International Standard algorithm for image compression. The C-Cube CL550 processor is designed for applications that manipulate high-quality digital pictures and motion sequences.

The C-Cube CL550 processor can encode and decode grayscale and color images at video rates. The image compression ratio is controlled by the on-chip quantization tables. Images can be compressed from 8:1 to 100:1, depending on the quality, storage, and bandwidth requirements of each application.

The C-Cube CL550 has an on-chip video and host bus interface. The video interface supports 8-bit grayscale, RGB, CMYK or 4:4:4:4, and YUV (4:2:2 and 4:4:4) input and output. The Host Bus Interface provides a direct interface to the system bus for ease of system integration.

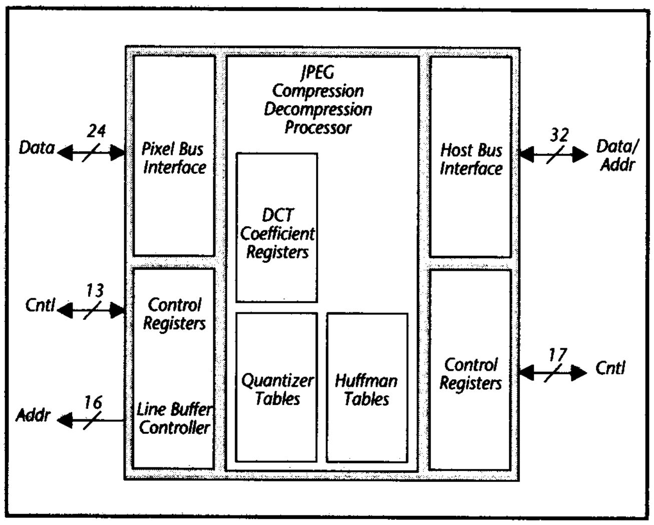


Figure 1-1. C-Cube CL550 Functional Block Diagram

This Page Intentionally Left Blank

1.0 INTRODUCTION

The C-Cube CL550® is a high-performance image compression/decompression processor. It implements the proposed baseline CCITT/ISO JPEG International Standard.

Section 2 of this document describes the baseline JPEG International Standard. The JPEG committee published a draft proposal in March of 1990. The draft is expected to be signed into an International Standard by the end of 1991.

Section 3 provides a functional overview of the C-Cube CL550. The C-Cube CL550 can be run in compression or decompression mode, as shown in Figure 1-2. In compression mode, pixel data is taken from the video bus, compressed, and made available on the host bus. In decompression mode, compressed data is acquired from the Host Bus Interface and decoded. Pixel data is made available on the pixel bus.

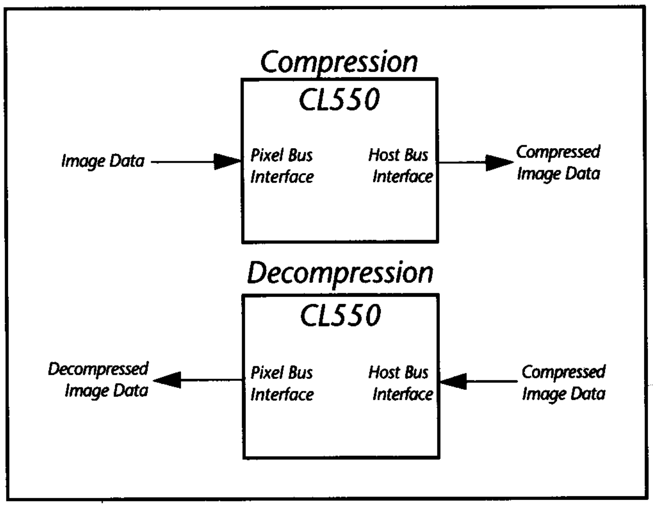


Figure 1-2. C-Cube CL550 Processor Data Flow

Section 4, "Hardware Designer's Guide," provides C-Cube CL550 hardware specifications as well as guidelines on how to integrate the compression processor into a system. The C-Cube CL550 processor integrates most of the interface functions, making it a cost-effective, easy-to-implement image compression solution.

Section 5, "Programmer's Guide," reviews the C-Cube CL550 programming model, the JPEG data structure and signaling parameters.

This Page Intentionally Left Blank

2.0 JPEG INTERNATIONAL STANDARD

2.1 Background of the JPEG Algorithm

The obvious advantages of digital image compression led to the formation of an international standards group: the Joint Photographic Experts Group (JPEG). JPEG is a joint ISO/CCITT technical committee (ISO/IECJTC1/SC2/WG10, Photographic Image Coding) whose goal has been to develop a general-purpose international standard for the compression of continuous-tone (grayscale or true color) digital images. The overall standard sets requirements and implementation guidelines for the image coding and decoding processes and for the coded representation of the compressed image data.

The standard defined by JPEG has usefulness in a broad range of applications. Because each application has different compression requirements, several processes for compression and decompression are specified within the JPEG standard. The processes fall into three general categories: the *Baseline Sequential Process*, the *Extended DCT-Based Processes*, and the *Lossless Process*. All JPEG coders and decoders must support the Baseline Sequential Process. All other processes are optional extensions that can be useful in specific applications. For detailed information on each of the processes, refer to the ISO Committee Draft document, ISO/IEC CD 10918-1.

The Baseline Sequential Process is based on the discrete cosine transform (DCT) followed by variable-word-length coding (Huffman coding). This process provides substantial compression (up to 100:1) while maintaining a high degree of visual fidelity in the reconstructed image. DCT-based processes, however, are *lossy* processes. The reconstructed images are not byte-for-byte equivalent to the source images. Further, the level of loss in the image varies with the compression ratio. Typically, the Baseline Sequential Process can compress image data to about 1 bit/pixel or less with very good visual quality in the reconstructed image. For example, a 24-bit RGB color image can be compressed to 1 bit/pixel (less than 5% of the original size), and the reconstructed image will be nearly indistinguishable from the original. The C-Cube CL550 is a VLSI implementation of the Baseline Sequential Process.

2.2 Operation of the JPEG Algorithm

The operation of the Baseline JPEG algorithm can be divided into three basic stages:

1. The removal of the data redundancy by means of the discrete cosine transform (DCT).
2. The quantization of the DCT coefficients using weighting functions optimized for the human visual system.
3. The encoding of the data to minimize the entropy of the quantized DCT coefficients. The entropy encoding is done with a Huffman variable-word-length encoder, as shown in Figure 2-1.

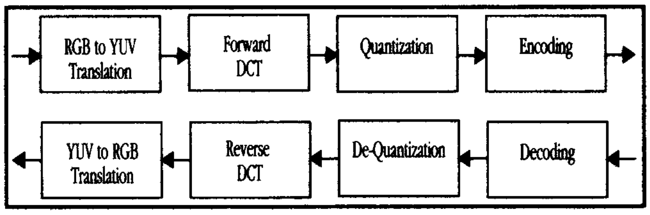


Figure 2-1. Basic Image Compression Scheme for Coder and Decoder

Although color conversion is a part of the redundancy removal process, it is not part of the JPEG algorithm. It is the goal of JPEG to be independent of the color space. JPEG handles colors as separate components. Therefore, it can be used to compress data from different color spaces, such as RGB, YUV, and CMYK.

However, the best compression results are achieved if the color components are independent (noncorrelated), such as in YUV, where most of the information is concentrated in the luminance and less in the chrominance. RGB color components can be converted via a linear transformation into YUV components, as shown in Table 2-1.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.3316 & 0.500 \\ 0.500 & -0.4186 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Table 2-1. Converting RGB Components to YUV Components

Another advantage of using the YUV color space comes from reducing the spatial resolution of the U and V chrominance components. Because chrominance does not need to be specified as frequently as luminance, every other U element and every other V element can be discarded. As a consequence, a data reduction of 3 to 2 is obtained by transforming RGB into YUV 4:2:2. The conversion in color space is a first step toward compressing the image.

2.2.1 Discrete Cosine Transform

For each separate color component, the image is broken into 8x8 blocks that cover the entire image. These blocks form the input to the DCT.

In the 8x8 blocks, typically the pixel values vary slowly. Therefore, the energy is of low-spatial frequency. A transform that can be used to concentrate the energy into a few coefficients is the two-dimensional 8x8 DCT. This transform, studied extensively for image compression, is extremely efficient for highly correlated data.

Conceptually, a one-dimensional DCT can be thought of as taking the Fourier Transform and retaining only the real (the cosine) part. The two-dimensional DCT can be obtained by performing a one-dimensional DCT on the columns and then a one-dimensional DCT on the rows. The transformed output from the two-dimensional DCT is ordered such that the mean value, the DC coefficient, is in the upper left corner of the 8x8 coefficient block and the higher frequency coefficients progress by distance from the DC coefficient. Higher vertical frequencies are represented by higher row numbers, and higher horizontal frequencies are represented by higher column numbers.

2.2.2 Quantization

The next step is the quantization of the frequency coefficients. The coefficients are quantized to reduce their magnitude and increase the number of zero-value coefficients. A uniform quantizer was selected for the JPEG baseline method. The step size is varied according to the coefficient location and tuned for each color component. This is shown in Figures 2-2 and 2-3. Figure 2-3 illustrates two functional matrices that have been optimized for CCIR 601 imagery.

The coding model rearranges the quantized frequency coefficients into a zigzag pattern, with the lowest frequencies first and the highest frequencies last. The zigzag pattern is used to increase the run-length of zero coefficients found in the block. The assumption is that the lower frequencies tend to have larger coefficients and the higher frequencies are, by the nature of most pictures, predominantly zero. As illustrated in Figure 2-4, the first coefficient (0,0) is called the DC coefficient and the remaining coefficients are AC coefficients. The AC coefficients are traversed by the zigzag pattern from the (0,1) location to the (7,7) location.

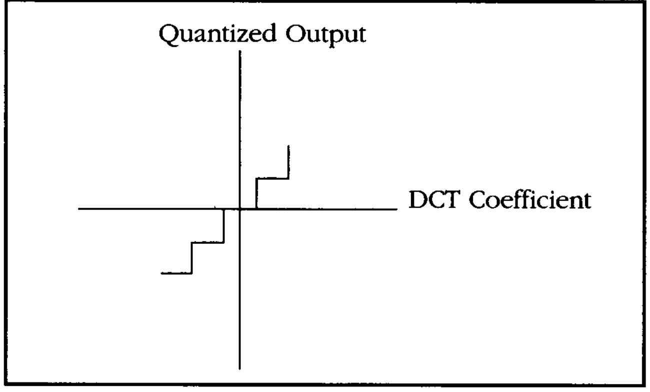
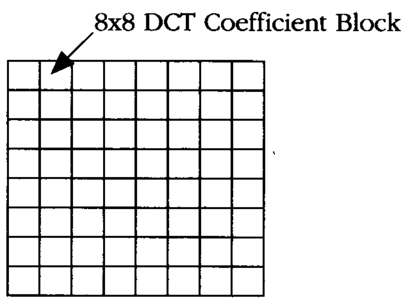


Figure 2-2. Quantizer Stepping (Uniform Quantization)



Y Component Matrix

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

Cb Cr Component Matrix

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

Figure 2-3. The Psychovisual Weighting Functions for the Luminance and Chrominance Components

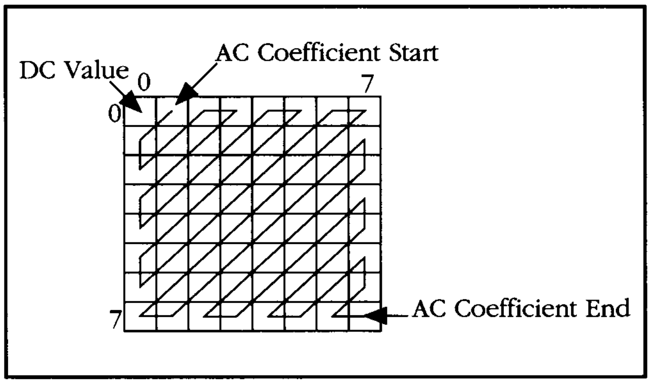


Figure 2-4. The Zigzag Pattern for Reordering the 8x8 DCT Coefficients

The DC coefficients of subsequent blocks often vary only slightly. Therefore, differences between successive DC coefficients are small. The coding of the DC coefficient exploits this property through Differential Pulse Code Modulation (DPCM). This technique codes the difference (Delta) between the quantized DC coefficient of the current block and the DC coefficient of the previous block. The formula for the encoding of the DC code is:

$$\text{Delta}_k = DC(0,0)_k - DC(0,0)_{k-1}$$

The inverse calculation takes place at the decoder.

2.2.3 Zero Run-Length Coding

The quantized AC coefficients usually contain runs of consecutive zeros. Therefore, a coding advantage can be obtained by using a run-length technique, where the upper four bits of the code symbol indicate the number of consecutive zeros before the next coefficient and the lower four bits indicate the number of significant bits in the next coefficient.

Following the code symbol are the significant bits of the coefficient, the length of which can be determined by the lower four bits of the code. The inverse run-length coder translates the input coded stream into an output array of AC coefficients. It takes the current code and appends to the output array the number of zeros corresponding to the four bits used for the run-length code. The coefficient placed in the output array has the number of bits determined by the lower four bits of the run-length code and a value determined by the number of trailing bits.

2.2.4 Entropy Encoding

The block codes from the DPCM and run-length models can be further compressed using entropy encoding. For the baseline JPEG method, the Huffman coder is used to reduce entropy. One reason for using the Huffman coder is that it is easy to implement by means of a look-up table in hardware. To compress data symbols, the Huffman coder creates shorter codes for frequently occurring symbols and longer codes for occasionally occurring symbols. Many applications may use predefined Huffman tables. Therefore, the baseline encoder can operate as a one-pass or two-pass system. In the one-pass system, predetermined Huffman tables are used, whereas in the two-pass system, Huffman tables are created that are specific to the image to be encoded.

The first step in creating the Huffman codes is to create a table assigning a frequency count to each symbol. Symbols with a higher probability are assigned shorter codes than the less frequently occurring symbols.

2.2.5 Summary of Baseline JPEG

The baseline system provides efficient lossy image compression. It supports four color components simultaneously, with a maximum number of eight input bits for each color pixel component.

The basic data entity is a block of 8x8 pixels. However, this block can represent a large sub-sampled image area (for example, sub-sampled by decimated chrominance signals). The blocks of the different color components are sent interleaved, thereby allowing the decoder to create the decompressed image and translate back to the original color space on the fly.

This Page Intentionally Left Blank

3.0 FUNCTIONAL OVERVIEW

3.1 Overview

This section describes the functional characteristics of each unit within the C-Cube CL550® processor. Figure 3-1 shows the processor's major functional blocks. The CL550 is a highly pipelined machine: there are over 320 processing stages in the data path. Each stage in the JPEG Baseline Sequential Process is implemented within this pipeline.

In compression operations, pixel data is written to the Pixel Bus Interface (PBI). The PBI handles raster-to-block conversion, pixel formatting, optional YUV-to-RGB color space conversion, and window sizing. Pixel data blocks are stored in the Block Storage Unit where they are sequenced onto the JPEG compression pipeline one component block at a time. Each component block is then processed by the DCT unit. The resulting DCT coefficients are quantized by the quantizer unit according to user-programmable quantization matrices. Up to four 64-word quantization matrices can be stored on-chip. Programmable sequence registers are used to select the appropriate matrix for each component block. The Quantized terms in each block are serialized by the Zig-Zag Scan Unit and the AC terms run-length coded by the Zero Packer/Unpacker Unit before being loaded into the FIFO. The FIFO serves as an intermediate buffer between the Zero Packer/Unpacker Unit and the Huffman Coder/Decoder (CODEC) Unit. The Huffman CODEC draws the packed symbols from the FIFO, performs DPCM calculations on the DC terms, and performs Huffman coding of both DC and AC terms. The Huffman codes are finally sent to the Host Bus Interface (HBI) as JPEG-compressed data. The CL550 allows the user to specify a particular Huffman table by providing on-chip table RAM that is loaded at initialization time. The host processor then reads out compressed data from the HBI in either 16-bit or 32-bit format.

Decompression operations follow the opposite procedure. Compressed data is written to the Huffman CODEC. Data is then Huffman decoded. The packed symbols are put back into the FIFO. The Zero Packer/Unpacker Unit accesses the FIFO symbols, generates the AC values, and passes them to the Zig-Zag Scan Unit for reordering into 8x8 block format. The DC terms are treated separately. Dequantization and Inverse DCT (IDCT) are then performed on the re-assembled blocks before they are sent back to the Block Storage Unit. The PBI takes the reassembled pixel blocks and outputs them to the pixel data bus.

With this architecture, it is possible to construct very-high-performance compression systems for both video and still-image applications. The CL550 can be reinitialized on a frame-by-frame basis, allowing the programmer to change compression ratios at the end of each frame or interleave between compress frames and decompress frames for half-duplex image communication applications. The following subsections describe some typical system configurations followed by a more detailed description of each functional block.

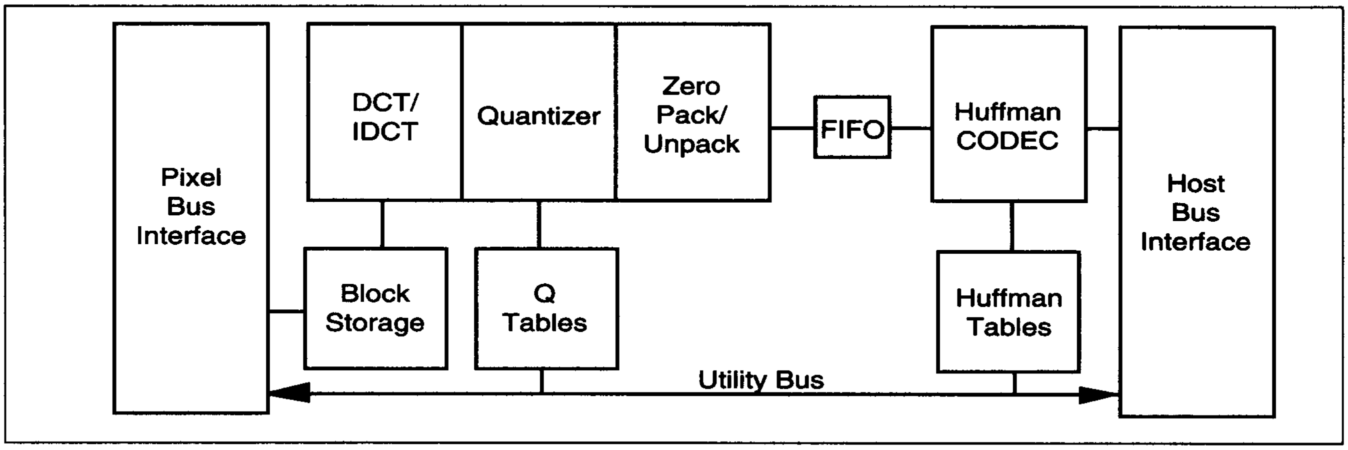


Figure 3-1. C-Cube CL550 Block Diagram

3.2 Application Examples

There are two basic system configurations for the CL550 processor. In high-speed systems, used to compress video or still frames, the pixel bus is connected directly to the video buffer, as shown in Figure 3-2. Digitized pixel data is stored in an intermediate buffer. This buffer can be either a VRAM frame buffer or a bi-directional FIFO buffer. The CL550 draws data asynchronously from this buffer at rates of up to 17.5 million pixels/second (CL550-35). The rate at which the CL550 inputs pixels is not constant. For complex image blocks, the CL550 will stop activity on the pixel bus in order to process the data. This mechanism for stopping the pixel bus is implemented using external circuitry (see Section 4.4.2.1). Once drawn from the pixel buffer, the data must be converted from raster line order into 8x8 block order for input to the CL550. The CL550 provides a very simple mechanism for this conversion using an external 8-line SRAM. The CL550 provides all of the address and control signals for the SRAM and handles all transfers to and from the SRAM. During compression, data from the pixel buffer is written to the SRAM strip buffer under the control of the CL550. During decompression the CL550 reads the data in block order back into the SRAM. The strip buffer addressing is such that only 8-lines of SRAM are required. (See Section 4.4.3.2 for more details on strip buffer addressing.) Compressed data is delivered to the Host Bus Interface. The HBI can operate in either a dedicated I/O fashion or under the control of a DMA controller. The maximum sustainable compressed data rate is approximately 2.0 MBytes/second at full clock rate (35 MHz).

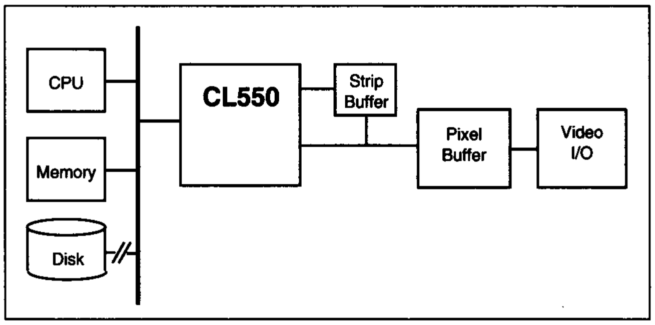


Figure 3-2. Typical Video Application Example

For more information concerning JPEG video system design techniques, refer to the technical note entitled "Designing JPEG Video Systems Using the C-Cube CL550."

A low-cost compression coprocessor system is shown in Figure 3-3. This system is useful for still-image compression applications, and it offers high compression/decompression performance with a minimum of hardware. In this configuration, a CL550 processor is used with both its pixel data bus and host data bus connected to a single system bus. The strip buffer is optional, as the host software can handle the pixel reordering. In a typical compression operation, the host writes pixel data to a pixel data latch, where it can be read by the CL550. The compressed data is read out of the HBI and taken by the system. The CL550 is put into a stalled state between pixel accesses and during host bus accesses.

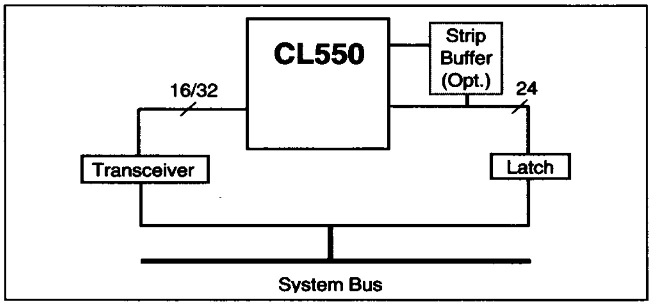


Figure 3-3. Typical Still-Frame Application Example

3.3 Pixel Bus Interface (PBI)

3.3.1 Signal Groups

Figure 3-4 shows a block diagram of the CL550 Pixel Bus Interface (PBI). Interface signals fall into the following groups:

• Data	PXDAT[23:0]
• Block address	PXADR[15:0]
• Control	PXRE#, PXWE#, PXIN#, PXOUT#, STALL#
• Status	VSYNC#, HSYNC#, BLANK#
• Clocks	PXCLK, PXPHSE, CLK3

The PXDAT[24:0] bus reads and writes pixels in the 8x8 block format required by the Baseline JPEG Sequential Process. A variety of pixel data types are supported, ranging from 8 bits/pixel to 32 bits/pixel. Transfers on this bus always take place in groups of 16 or 24 bits.

The PXADR[15:0], PXRE#, PXWE#, PXIN#, and PXOUT# signals are used to support the CL550's unique raster-to-block conversion mechanism. Typical display systems transfer pixel data one line at a time, whereas JPEG

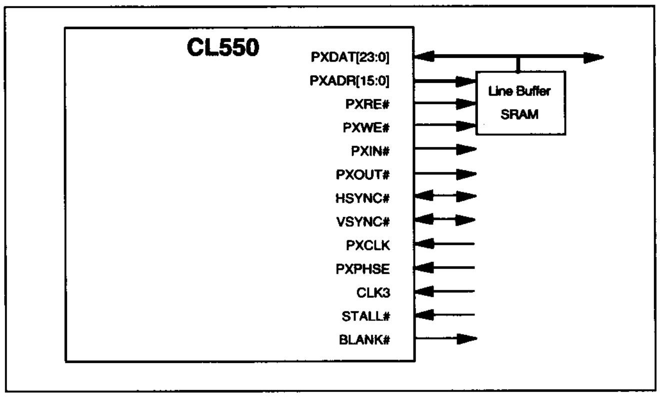


Figure 3-4. Pixel Bus Interface

requires input pixel data to be in 8x8 block order. The PBI performs this conversion automatically, greatly reducing the amount of system logic required to interface the CL550 to raster-oriented systems.

3.3.2 Window Management and Control

Several status signals, available on the PBI, are useful in window management and control. These signals are VSYNC#, HSYNC#, and BLANK#.

The VSYNC# and HSYNC# signals are bidirectional status signals that are used to indicate the beginning of a frame or field and the beginning of a new line, respectively. Note that the VSYNC# and HSYNC# lines are merely asynchronous reference signals, and they are not intended to be synchronized directly to a raster display system.

Register	Content/Function
HPeriod	Number of pixels in a line
VPeriod	Number of lines in an image
HDelay	Horizontal delay (in pixels) to the first active pixel
VDelay	Vertical delay (in lines) to the first active line
HActive	Active window width (in blocks)
VActive	Active window height (in blocks)
HSync	Horizontal sync-pulse width (in pixels)
VSync	Vertical sync-pulse width (in lines)
Vertical Line Count	Active window vertical line count

Table 3-1. Video Field Control Registers

The BLANK# signal is asserted to indicate that no pixels are being transferred to the external interface.

Window and frame parameters are programmed via a set of control registers. These registers are listed in Table 3-1. VPeriod and HPeriod are used to specify an image frame. HDelay, VDelay, HActive, and VActive are used to specify size and position of the active image area within the frame. Figure 3-5 illustrates the function of the video field registers.

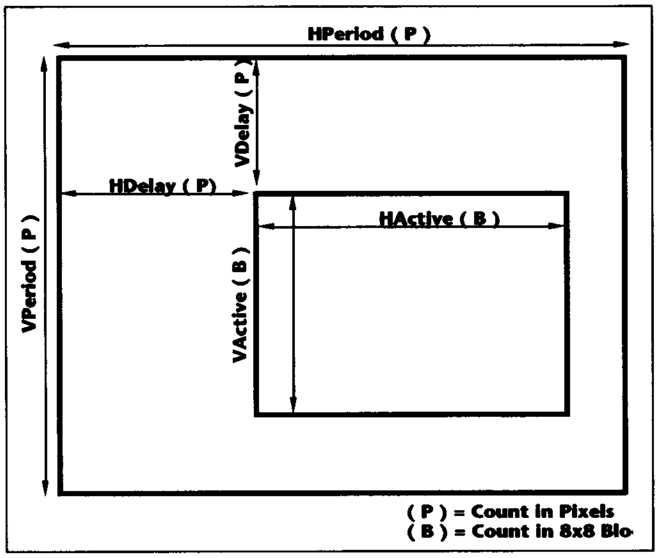


Figure 3-5. Video Field Descriptions

3.3.3 Color Space Conversion and Chrominance Sub-Sampling Mechanisms

The CL550 provides an internal RGB-to-YUV color space conversion and sub-sampling mechanism. Although not a part of the JPEG algorithm (JPEG is independent of color space), this mechanism is particularly useful in computer video and multimedia applications. For example, digitized data from a frame grabber or video digitizer is often presented in the 16-bit YUV4:2:2 format. This is the fomat required by NTSC and PAL monitors. However, typical computer graphics monitors require data to be presented in 24-bit RGB format. The CL550 has the ability to translate between these two color spaces by means of a matrix multiplier and chrominance sub-sampler, thus minimizing the need for external color space conversion logic and reducing overall system complexity and cost.

Conversion between the RGB color space and the YUV (also referred to as the Y, Cb, Cr) color space is accomplished using a matrix multiply operation. Nine registers are provided in the CL550 processor to allow programmability of the transform matrix (See Section 5).

In the RGB-to-YUV 4:2:2 pixel compression mode, for example, the 24-bit RGB pixels are read into the PXDAT bus. Once inside the CL550, they are transformed into 24-bit YUV format (YUV 4:4:4) using the on-chip matrix multiplier. Following this operation, the U and V components are sub-sampled to obtain a 4:2:2 ratio between luminance (Y) and chrominance (U and V). This is illustrated in Figure 3-6.

RGB 4:4:4		YUV 4:4:4		YUV 4:2:2
R1 R2 R3 R4	>>	Y1 Y2 Y3 Y4	>>	Y1 Y2 Y3 Y4
G1 G2 G3 G4	>>	U1 U2 U3 U4	>>	U1 V1 U3 V3
B1 B2 B3 B4	>>	V1 V2 V3 V4		

Figure 3-6. RGB-to-YUV 4:2:2 Conversion Operation

3.4 Block Storage Unit

The Block Storage Unit (BSU) provides on-chip storage for up to four 8x8 component blocks. Because the JPEG engine must operate on one 8x8 component block at a time, the BSU serves as a buffer for separating the multi-component input pixels into their individual component blocks. They can then be processed in sequential fashion. For example, in a CMYK compression operation, the PBI is inputting 16-bit values in the following format:

```
PXDAT[7:0]      C Y C Y C Y C Y C Y C Y
PXDAT[15:8]     M K M K M K M K M K M K
```

Before the CL550 can begin to process this data, it must assemble one 64-byte block each of C, M, Y, and K values. When this is done, the DCT/IDCT unit operates on each block in the following order.

C0...C63, M0...M63, Y0...Y63, K0...K63, C0...C63, M0...M63, ...

This feature of the CL550 is fundamental in allowing it to process multiple component pixels in a continuous fashion. In decompression, the process is reversed.

3.5 DCT/IDCT Unit

On compression, the DCT unit operates on data stored in the BSU. Because the DCT is a two-dimensional process, it is necessary to process the data in two passes: horizontal and vertical. The second-pass DCT data are sent out to the Quantizer Unit. In decompression, the process is reversed. The CL550 contains eight 16-bit registers for the loading of the DCT coefficients used in either DCT or IDCT operations. All multiply operations within this unit are 16-bit signed integer multiplies with results rounded to 16 bits.

3.6 Quantizer Unit and Q Table RAM

The Quantizer Unit, during compression operations, accepts data from the DCT unit and multiplies it by 16-bit quantization coefficients stored in one of four 64-word on-chip quantization tables. The quantization table provides an individual quantizer value for each of the 64 DCT terms. The quantization tables are loaded by the host processor at the start of the compression or decompression operation. Four tables are provided to support each of the four possible component blocks that are being processed. All multiply operations in the quantizer are 16 x 16 bits. In the compression direction, the quantized outputs are rounded and limited to 11 significant bits.

3.7 Zig-Zag Scan Unit

The next stage of the compression process requires the quantized DCT data blocks to be serialized for processing by the Zero Packer/Unpacker Unit. The Zig-Zag Scan Unit performs the scanning function descibed in Figure 2-4. For example, DCT data blocks come into the Zig-Zag Scan Unit in left-to-right, top-to-bottom linear order: (0,0),(0,1),(0,2),(0,3), ... (7,7). The Zig-Zag Scan Unit buffers and reorders the data in the following order: (0,0),(0,1),(1,0),(2,0),(1,1),(0,2),(0,3), ... ,(6,7),(7,6),(7,7). This ordering of data provides the greatest probability of long runs of zero terms, which makes possible effective run-lengthcoding of the AC terms by the Zero Packer/Unpacker Unit. For decompression, the Zig-Zag Scan Unit reverses this process.

3.8 Zero Packer/Unpacker Unit

The Zero Packer/Unpacker Unit generates run-length codes from the AC terms in the zig-zag scan data. Output data is sent to the FIFO RAM on-chip to await processing by the Huffman Coder/Decoder Unit. Output data from this unit is 13 bits wide, with the two most significant bits indicating the type of data as listed in Figure 3-7.

Bit 12	Bit 11	Data Type
0	0	Quantized DC value
0	1	Significant AC value
1	0	Run-length value (1 to 62)
1	1	End of block value

Figure 3-7. FIFO Data Types

As each block is processed by the Zero Packer/Unpacker Unit, the DC term is sent directly to the FIFO memory with '00' appended to bits 11 and 12. The Zero Packer/Unpacker will count the number of insignificant terms (zeros) that precede each significant AC value and send that count value to the FIFO. The significant AC value then follows. At the end of the block, an EOB term is sent to the FIFO. The data field of the EOB term (bits 10-0) is always zero. If there are no significant AC values (all 63 AC terms are zero) in the block, only the EOB is written.

3.9 FIFO RAM

The FIFO RAM provides a 128 x 13 intermediate storage buffer for data passing between the Zero Packer/Unpacker Unit and the Huffman Coder/Decoder Unit. Status level indicators are generated at empty, 1/4 full, 1/2 full, 3/4 full, and full. The FIFO level indicators can be read by software via the Flags register. Also, two output pins, NMRQ# and DRQ#, can be programmed to respond to various flag states. Another output pin, HALF_FULL, is hardwired directly to the 1/2 full level indicator.

During normal compression/decompression operations, it is possible for the Zero Packer/Unpacker Unit to generate data at a rate faster than the Huffman Coder/Decoder Unit can handle. When this occurs, it is necessary to halt the operation of the Zero Packer/Unpacker Unit to prevent the FIFO from overflowing (compression) or underflowing (decompression). This is accomplished by asserting the STALL# input of the CL550. The STALL# function halts all elements of the processing pipeline, from the Zero Packer/Unpacker Unit through the Pixel Bus Interface, in their current state. When STALL# is asserted, however, it is still possible to service the FIFO on the host end, allowing the host to maintain adequate FIFO levels.

The FIFO can overflow or underflow very rapidly. For instance, in a compression case in which all AC terms are significant, there will be one entry loaded into the FIFO for each PXCLK cycle, meaning that the FIFO can overflow in as little as 128 PXCLKs. Because it is difficult to control STALL# by software (because of the system latency), in real-time systems it is necessary to use a hardware signal such as NMRQ# to generate the STALL# input to the CL550. More detailed descriptions of this hardware implementation are given in Section 4.

3.10 Huffman Coder/Decoder Unit

The last stage of the compression process is the generation of the variable-length Huffman codes that create the JPEG data stream from data stored in the FIFO RAM. The Huffman Coder/Decoder (CODEC) module handles the following tasks:

- DPCM coding of the DC values in the FIFO
- Huffman coding of DPCM data and AC values
- Generation of restart (RST) marker codes at user-specified intervals
- Byte stuffing (padding) where needed

The CODEC has three main functional blocks: a Huffman coder unit, a Huffman decoder unit, and a look-up table RAM for the storage of data tables used in the coding and decoding processes. Compressed data for both compression and decompression directions is buffered in a 32-bit CODEC register that is accessible from the Host Bus Interface in either 16-bit or 32-bit transfers.

3.10.1 Huffman Coder Unit

In compression operations, the Huffman coder unit activates when the FIFO level reaches 1/4 full. As the coder reads an entry from the FIFO, it checks bits 11 and 12 to see what type of entry it is. For DC entries, difference values are first calculated by subtracting the DC value of the previous block of the same component type, stored in one of four DPCM registers (for the first block such value is zero). Next, a 10-bit table index into one of the two on-chip Huffman table RAMs is generated and the corresponding table entry is appended to the outgoing JPEG data in the CODEC register . (The process for generating the Huffman codes is described in detail in the JPEG specification document.)

Run-size values from the FIFO are handled by first converting the run-size entries into proper JPEG run-length codes. The maximum run-length allowed by JPEG is 16. However, the run-size values in the FIFO can be in the range of 1 through 62. In cases where the run-sizes are greater than 16, the coder breaks them into smaller values before coding them.

AC values are coded directly as specified in the JPEG standard specification. EOB values are treated as a special case AC term, with a table index of zero.

In addition to the coding of the FIFO data, the JPEG specification allows for the insertion of restart (RST) marker codes at user-specified intervals. The CL550 Huffman coder unit automatically inserts these marker codes into the compressed data stream.

3.10.2 Huffman Decoder Unit

For decompression operations, the decoder unit operates on the 32-bit data value that is loaded in the CODEC register. A marker code detector will automatically recognize and strip off RST markers from the code stream. If marker codes other than RST are encountered in the compressed data stream, the decoder will halt, the 'mark' bit in the Flags register will be set, and the marker code value will be placed in the Decoder Marker Code register. The host at that point must intervene in the decompression process and restart the Huffman decoder by writing to the Decoder Resume register (see Section 5 for more details). Note that within the JPEG scan field, only RST markers are allowed. Other markers ahead of the scan field are used to indicate decompression parameters and data tables. It is the responsibility of the host software to interpret and strip off this information and program the CL550 accordingly. Only data in the actual scan field should be sent to the CL550.

The decoder operates by taking two bits of JPEG data at a time, each time generating an address into the Huffman decode tables. The table values indexed are nine bits wide and contain either the actual data (which requires no further decoding) or a branch address in the Huffman tables, which requires access to another location. Some codes require several levels of indirection. The decoding process takes place in compliance with the JPEG specifications. For a detailed description of this process, refer to the JPEG standard document.

3.10.3 Huffman Table RAM

The Huffman table RAM is organized as a 736 x 18 SRAM array. The memory is organized into four sections. The Huffman Coder Unit accesses this memory in the address ranges shown in Figure 3-8. The Huffman Decoder Unit accesses this memory in the address ranges shown in Figure 3-9.

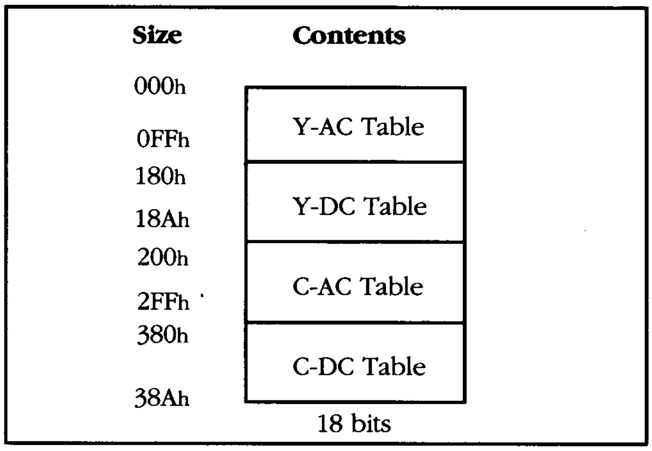


Figure 3-8. Huffman Table Sizes (Compression)

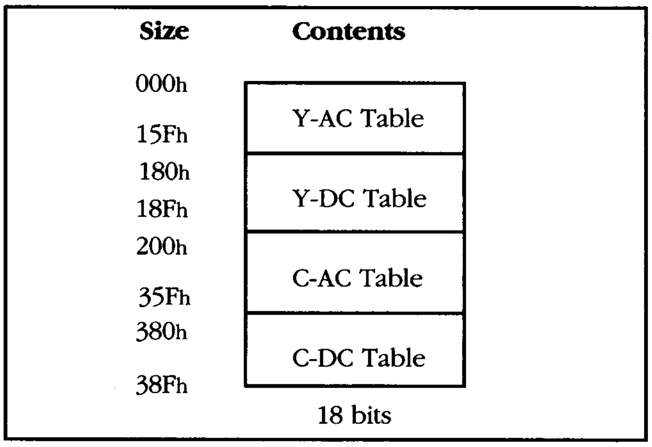


Figure 3-9. Huffman Table sizes (Decompression)

JPEG allows up to two Huffman code/decode tables for any particular image, each containing one DC and one AC table. The CL550 table RAM can contain two sets of these tables. For example, when compressing a YUV image, separate tables can be designated for either luminance (Y) or chrominance (C) blocks. During compression or decompression operations, the CL550 uses a programmable sequence register (Huffman Table Sequence Register) to select the appropriate set of tables as each component block is processed.

Each table entry is loaded from the Host Bus Interface in 9-bit halves from a 9-bit read/write port. Each 9-bit location is aligned to a 32-bit address boundary. When accessing the Huffman tables from the host interface, the lower half of the 18-bit table location appears at the lower address. Addressing for this port is given in Table 5-8.

The Huffman table RAM will support any JPEG-compliant Huffman table.

3.11 Host Bus Interface (HBI)

The Host Bus Interface (HBI) provides a 32-bit address/ data path for transferring compressed data or reading and writing the CL550's internal control/status registers and data tables. Figure 3-10 shows the interface signals that make up the HBI. Detailed signal descriptions, functional descriptions, timings, and operational considerations are given in Section 4.

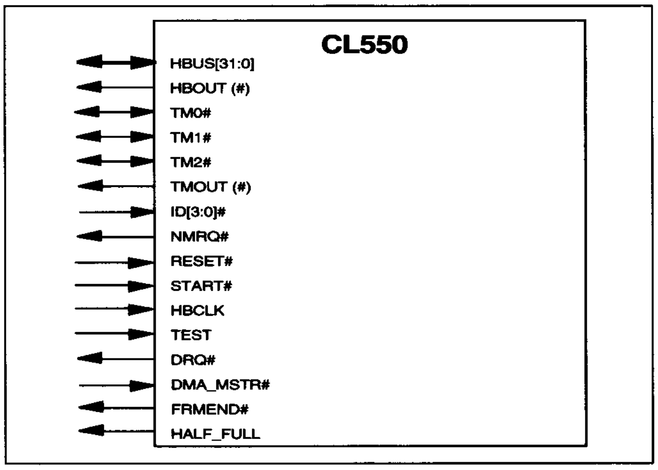


Figure 3-10. Host Bus Interface

The HBI can be sized to either 16 or 32 bits for both address and data. ID[3:0]# lines are used to specify address format, and the HBUS_32# line specifies data width. Transfers are accomplished using a Start/Acknowledge protocol. Data on this bus is transferred in "little endian" byte order, with the least significant byte on the low-order byte of the HBUS. The HBI architecture is such that it can be easily interfaced to most conventional 16-bit and 32-bit microprocessor busses.

This Page Intentionally Left Blank

4.0 HARDWARE DESIGNER'S GUIDE

This section provides the mechanical and electrical information necessary to design the C-Cube CL550 processor into a system.

4.1 Physical Dimensions

Figure 4-1 shows the CL550-35 and CL550-30 PGA physical dimensions. The CL550-35 and CL550-30 PGA come in a 144-pin, Pin Grid Array (PGA), cavity down package.

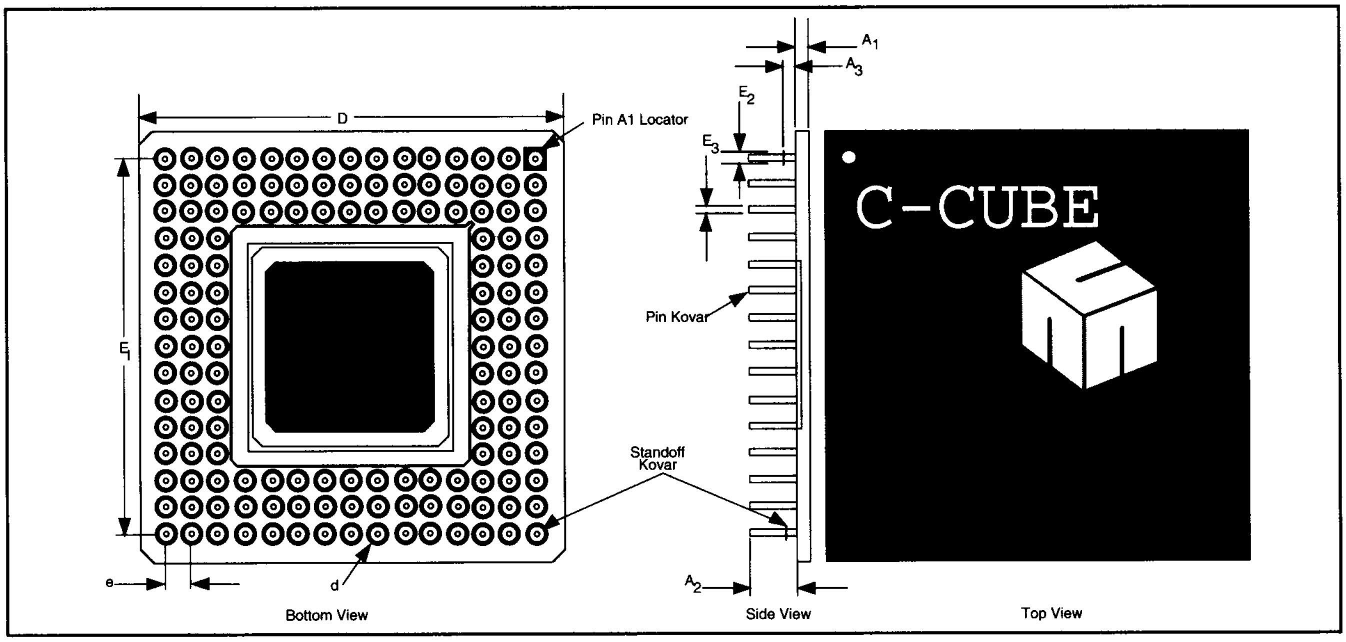


Figure 4-1. CL550-35 and CL550-30 Physical Dimensions

SYMBOL	DIMENSIONS	
	INCHES	MM
A1	0.100 ± 0.10	2.54 ± 0.25
A2	0.180 typ. ± 0.005	4.57 typ. ± 0.13
A3	0.050 typ. ± 0.005	1.27 typ. ± 0.13
D	1.560 sq. ± 0.016	39.6 ± 0.41
E1	1.400 typ. ± 0.014	35.56 ± 0.36
E2	0.050 dia. typ.	1.27 dia. typ.
E3	0.018 ± 0.002	0.46 ± 0.05
d	0.065 dia. typ.	1.65 dia. typ.
e	0.100 typ.	2.54 typ.

Key to Figure 4-1

Figure 4-2 shows the CL550-30 and CL550-10 MQUAD physical dimensions. The CL550-30 and CL550-10 MQUAD come in a 144-pin, Metal Quad Flat Pack (MQUAD), cavity down package.

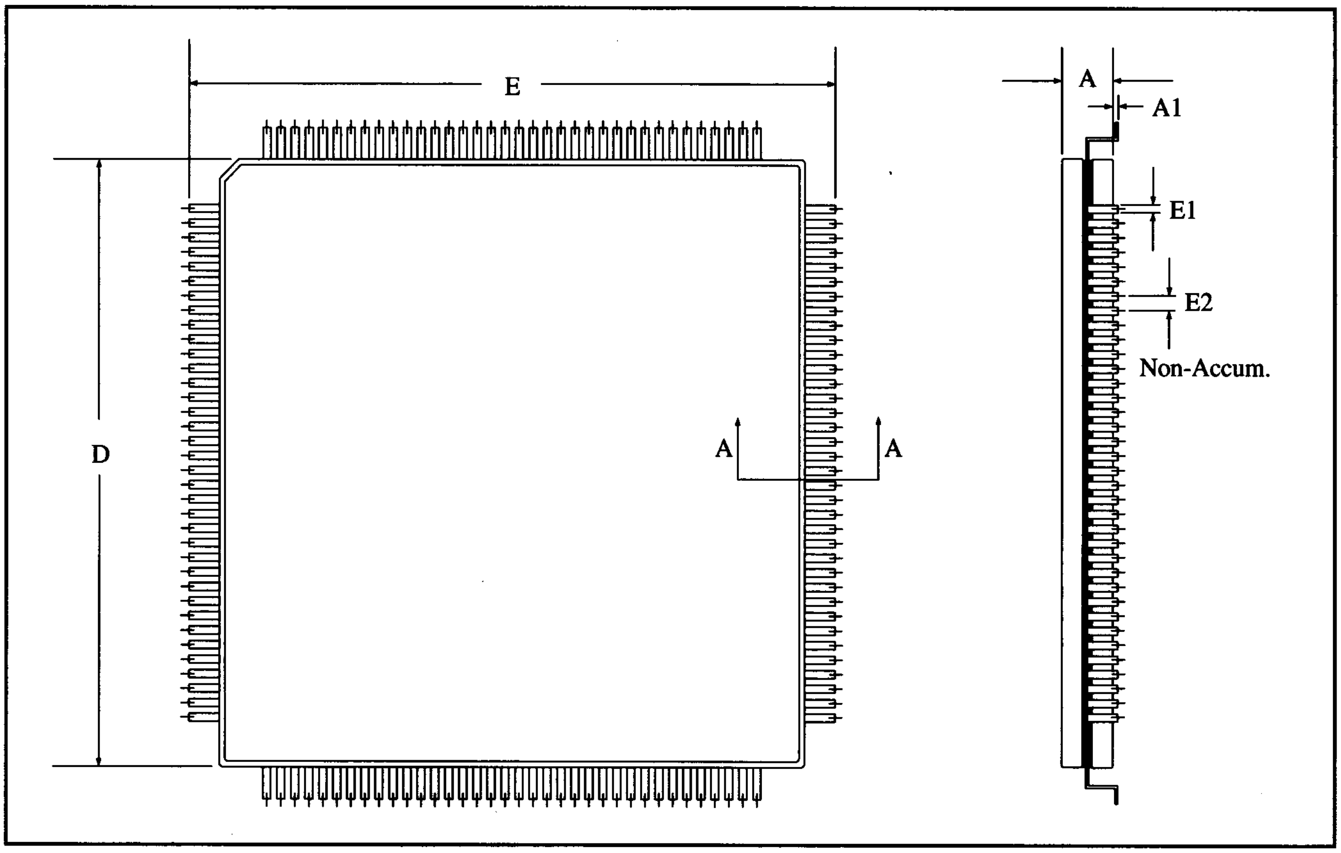


Figure 4-2. CL550-30 and CL550-10 MQUAD Physical Dimensions

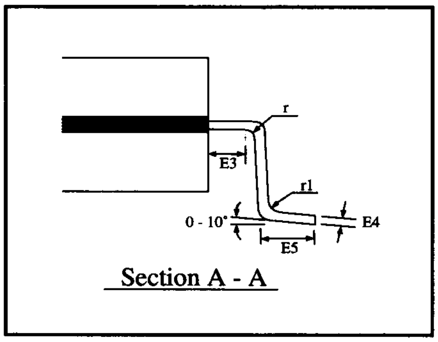


Figure 4-2A.

<table>
<tr><th rowspan="2">SYMBOL</th><th colspan="2">DIMENSIONS</th></tr>
<tr><th>INCHES</th><th>MM</th></tr>
<tr><td>A</td><td>0.130± 0.012</td><td>3.30 ± 0.30</td></tr>
<tr><td>A1</td><td>0.015 ± 0.008</td><td>0.38 ± 0.20</td></tr>
<tr><td>D</td><td>1.088 ± 0.016</td><td>27.60 ± 0.40</td></tr>
<tr><td>E</td><td>1.256 ± 0.016</td><td>31.90 ± 0.40</td></tr>
<tr><td>E1</td><td>0.012 typ.</td><td>0.30 typ.</td></tr>
<tr><td>E2</td><td>0.0256 typ.</td><td>0.65 typ.</td></tr>
<tr><td>E3</td><td>0.020 min.</td><td>0.50 min.</td></tr>
<tr><td>E4</td><td>0.006 typ.</td><td>0.15 typ.</td></tr>
<tr><td>E5</td><td>0.031 ± 0.008</td><td>0.80 ± 0.2</td></tr>
<tr><td>r</td><td>0.010 typ R</td><td>0.25 typ R</td></tr>
<tr><td>r1</td><td>0.010 typ R</td><td>0.25 typ R</td></tr>
</table>

Key to Figures 4-2 and 4-2A

4.2 C-Cube CL550 Pinout

Figure 4-3 is a pin-layout diagram of the CL550-35 and CL550-30 PGA package. Figure 4-4 shows the pinout labels.

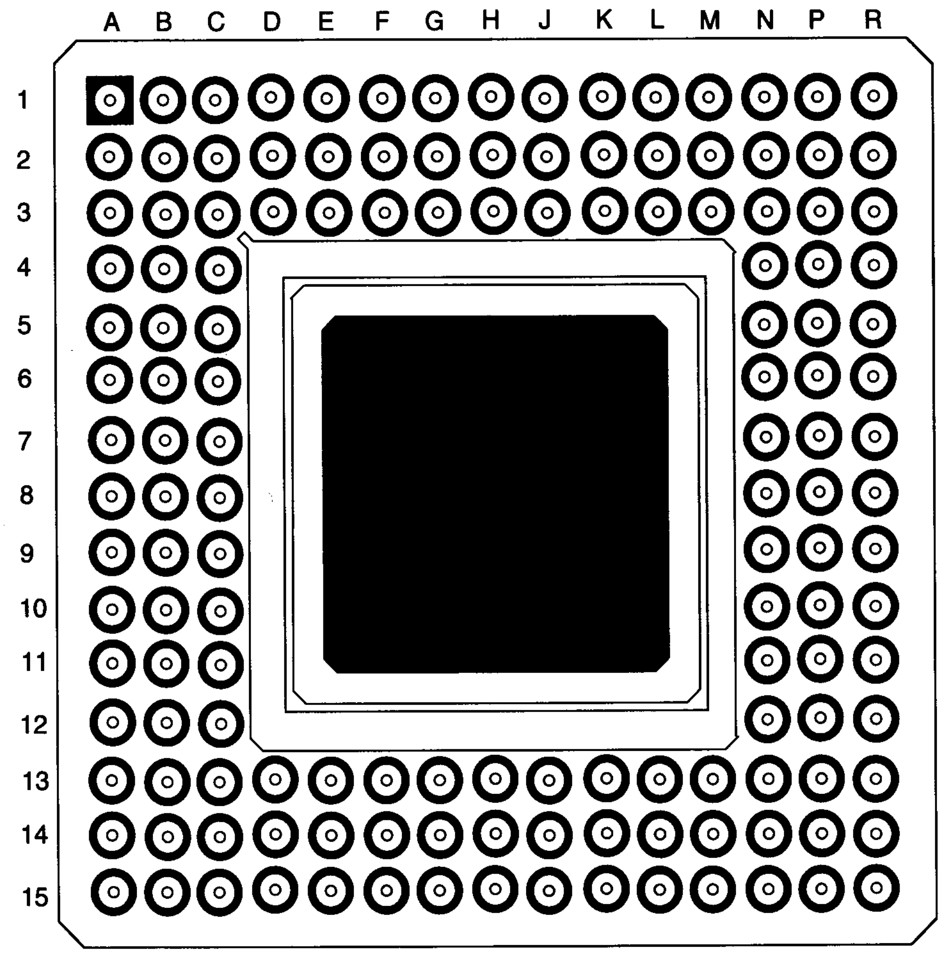


Figure 4-3. CL550-35 and CL550-30 (PGA Package) Pin Layout (Bottom View)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	VSS	PXADR[5]	PXADR[2]	PXDAT[23]	PXDAT[21]	PXDAT[19]	PXDAT[16]	VDD	PXDAT[14]	PXDAT[11]	PXDAT[9]	PXDAT[7]	PXDAT[4]	PXDAT[1]	TEST
B	PXADR[7]	PXADR[6]	PXADR[3]	PXADR[0]	PXDAT[22]	PXDAT[20]	PXDAT[17]	PXDAT[15]	PXDAT[13]	PXDAT[10]	PXDAT[8]	PXDAT[5]	PXDAT[2]	PXDAT[0]	$\overline{\text{TMOUT}}$
C	PXADR[9]	PXADR[8]	VSS	PXADR[4]	PXADR[1]	VSS	PXDAT[18]	VDD	PXDAT[12]	VSS	PXDAT[6]	PXDAT[3]	VSS	$\overline{\text{RESET}}$	$\overline{\text{TM1}}$
D	PXADR[12]	PXADR[11]	PXADR[10]										$\overline{\text{NMRQ}}$	$\overline{\text{TM2}}$	$\overline{\text{START}}$
E	PXCLK	$\overline{\text{PXOUT}}$	$\overline{\text{PXIN}}$										$\overline{\text{TM0}}$	HBCLK	$\overline{\text{HBOUT}}$
F	$\overline{\text{PXRE}}$	$\overline{\text{PXWE}}$	VDD										VDD	HBUS[0]	HBUS[1]
G	$\overline{\text{VSYNC}}$	$\overline{\text{HSYNC}}$	PXPHSE										HBUS[2]	HBUS[3]	HBUS[4]
H	VDD	PXADR[13]	VSS										VSS	HBUS[5]	HBUS[6]
J	PXADR[14]	PXADR[15]	CLK3										HBUS[9]	HBUS[8]	HBUS[7]
K	$\overline{\text{STALL}}$	$\overline{\text{BLANK}}$	VDD										VDD	HBUS[11]	HBUS[10]
L	$\overline{\text{FRMEND}}$	HALF-FULL	NC										HBUS[14]	HBUS[13]	HBUS[12]
M	NC	NC	NC										HBUS[17]	HBUS[16]	HBUS[15]
N	NC	NC	VSS	NC	NC	VSS	NC	VDD	ID[1]	VSS	HBUS[27]	HBUS[23]	VSS	HBUS[19]	HBUS[18]
P	NC	NC	NC	NC	NC	NC	$\overline{\text{HBUS\_32}}$	$\overline{\text{DRQ}}$	ID[2]	HBUS[31]	HBUS[29]	HBUS[26]	HBUS[24]	HBUS[21]	HBUS[20]
R	VSS	NC	NC	NC	NC	NC	$\overline{\text{DMA\_MSTR}}$	VDD	ID[3]	ID[0]	HBUS[30]	HBUS[28]	HBUS[25]	HBUS[22]	VSS

Figure 4-4. CL550-35 and CL550-30 Pinout Labels (Top View Through Chip)

Figure 4-5 is a pin-layout diagram of the CL550-30 and CL550-10 MQUAD.

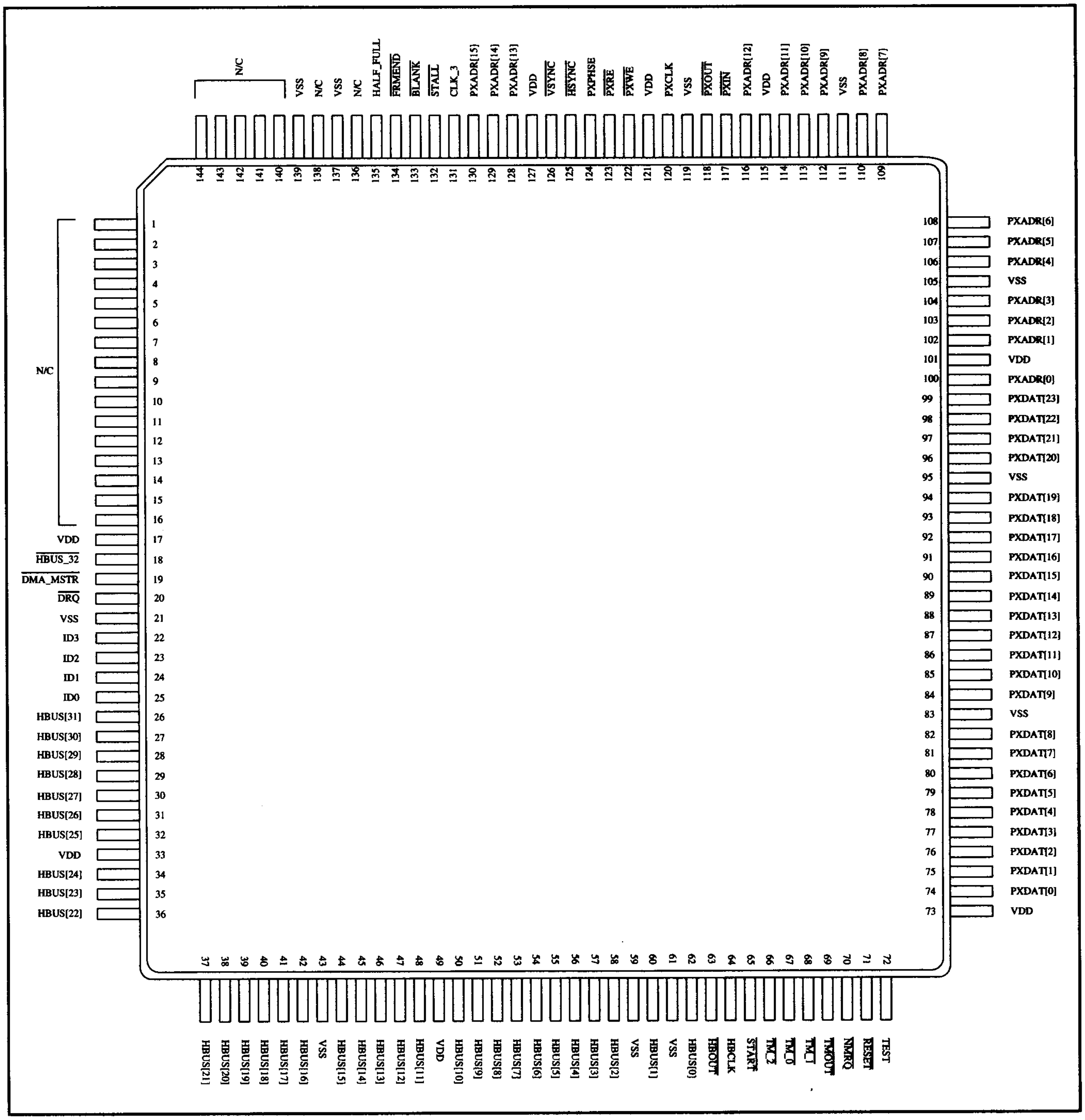


Figure 4-5. CL550-30 and CL550-10 MQUAD Pin Layout (Top View)

4.3 Specifications

ABSOLUTE MAXIMUM RATINGS

Supply voltage	-0.5 to 7.0 V
Input voltage	-1.5 to VDD
Output voltage	-0.5 to VDD
Storage temperature range	-65° to 150°C
Operating temperature range (case)	0° to 90°C

OPERATING CONDITIONS

Parameter		Test Conditions	Commercial		Unit
			Min	Max	
V_{DD}	Supply voltage		4.75	5.25	V
T_{case}	Operating temperature		0	85	°C

Figure 4-6. Operating Conditions

DC ELECTRICAL CHARACTERISTICS

Parameter		Test Conditions	Commercial		Unit
			Min	Max	
V_{IH}	High-level input voltage	V_{DD}= MAX (Note 1)	2.4		V
V_{IL}	Low-level input voltage	V_{DD}= MIN (Note 1)		0.8	V
V_{OH}	High-level output voltage	V_{DD}= MIN, I_{OH} = -8.0 mA	2.4		V
V_{OL}	Low-level output voltage	V_{DD}= MIN, I_{OL} = 12.0 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_{OZH}	High-level ouptut leakage current	Ouptut Disabled, V_{out} = V_{DD}		10	µA
I_{OZL}	Low-level output leakage current	Output Disabled, V_{out} = 0 V	-10		µA
I_{OPL}	HBUS output pull-up current	Output Disabled, V_{out} = 0 V	-50		µA
I_{DD}	Supply Current	V_{DD}= MAX, f = 0 MHz		260	mA
		V_{DD}= MAX, f = 10 MHz		375	mA
		V_{DD}= MAX, f = 30 MHz		590	mA
		V_{DD}= MAX, f = 35 MHz		670	mA
		V_{in} = 0 or V_{DD}, V_{OUT} = Open			
Capacitance		C_{in} = 10 typ., C_{out} = 12 typ.			pf
WARNING ! Remove power before insertion or removal.					

Figure 4-7. DC Electrical Characteristics Over Recommended Temperature Range

Note 1: Guaranteed but not 100% tested logic level in noise-free environment.

4.4 Pixel Bus Interface

The following subsection describes the Pixel Bus Interface (PBI). It includes detailed signal descriptions, a simple design example, AC timing specifications, and several timing diagrams that illustrate PBI operation.

The Pixel Bus Interface performs four major functions:

- Raster to/from 8x8 block pixel order conversion
- Blanking and active region control
- RGB-YUV conversion
- Interleaved pixel format conversion

4.4.1 Signal Descriptions

PXDAT[23:0]

PXDAT is a bidirectional 24-bit bus that handles the uncompressed pixel data. The format of the pixel data bus depends on the CL550 operating mode. This mode is determined by a value programmed into the Configuration register (see Section 5). In some modes, fewer than 24 bits of the PXDAT bus are used. The unused pins should be tied to ground through 10K-ohm resistors. The data formats supported by the CL550 are described in Table 4-1. The labels C0, C2, ..., CN are PXCLK cycle indices, and xx designates unused pins.The 0,1 ,2, ..., N indices on the color components indicate the temporal ordering of the information in the raster.

The PXDAT bus is multiplexed between RAM read and write cycles. During compression, the CL550 reads strip buffer data into the PBI and alternately writes external pixel data into the line buffer.

During decompression, the line buffer data is read and output to the external interface, and on the next cycle, the PBI data is written to the line buffer.

In both cases, the read-then-write sequence uses the same RAM address for read and write operations.

PXADR[15:0]

The PXADR bus is an output address bus for the line buffer RAM. The 16 bits of address support a line buffer of up to 64K entries. The addressing is a complex modulo counting scheme that converts raster-scan-ordered pixels into 8x8 blocks during compression and vice versa during decompression. Figure 4-8 illustrates the data format of one 8x8 block as it is ordered, zero being first or earliest.

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

Figure 4-8. 8x8 Pixels Block Ordering

PXWE#

The PXWE# output signal is designed to directly control the write enable input to the line buffer RAMs. During compression, PXWE# is active only during PXIN# cycles when pixel data is being written from the active portion of the video field into the line buffer RAM. During decompression, PXWE# is active only when active pixels are being written from the CL550 into the line buffer RAM.

PXRE#

The PXRE# output signal is designed to directly control the output enable input of the line buffer RAMs. During compression, PXRE# is active only when the CL550 is reading pixel data from the line buffer RAM. During decompression, PXRE# is active only when pixels are being read from the line buffer RAM out to the pixel destination.

PXIN#

PXIN# is an output signal designed to control the output enable signal on an external tristate buffer. This signal is active only when a pixel from the active region of the field is to be written into the line buffer RAM from the external pixel source.

PXOUT#

The PXOUT# output signal can be used to load the active pixel into a register as it is read out of the line buffer RAM. It is active only when pixels from the active region of the field are being read from the line buffer RAM.

TWO PIXELS PER TWO PXCLKs				
MODE	C0	C2	C4	C6
Single component (grayscale)				
PXDAT[23:16]	xx	xx	xx	xx
PXDAT[15:8]	X1[7:0]	X3[7:0]	X5[7:0]	X7[7:0]
PXDAT[7:0]	X0[7:0]	X2[7:0]	X4[7:0]	X6[7:0]
ONE PIXEL PER TWO PXCLKs				
MODE	C0	C2	C4	C6
YUV 4:2:2				
PXDAT[23:16]	xx	xx	xx	xx
PXDAT[15:8]	U0[7:0]	V0[7:0]	U1[7:0]	V1[7:0]
PXDAT[7:0]	Y0[7:0]	Y1[7:0]	Y2[7:0]	Y3[7:0]
YUV 4:4:4 to YUV 4:2:2				
PXDAT[23:16]	V0[7:0]	V1[7:0]	V2[7:0]	V3[7:0]
PXDAT[15:8]	U0[7:0]	U1[7:0]	U2[7:0]	U3[7:0]
PXDAT[7:0]	Y0[7:0]	Y1[7:0]	Y2[7:0]	Y3[7:0]
RGB 4:4:4 to YUV 4:2:2				
PXDAT[23:16]	B0[7:0]	B1[7:0]	B2[7:0]	B3[7:0]
PXDAT[15:8]	G0[7:0]	G1[7:0]	G2[7:0]	G3[7:0]
PXDAT[7:0]	R0[7:0]	R1[7:0]	R2[7:0]	R3[7:0]
ONE PIXEL EVERY FOURTH PXCLK				
MODE	C0	C4	C8	C12
4:4:4 (YUV pixel example)				
PXDAT[23:16]	V0[7:0]	V1[7:0]	V2[7:0]	V3[7:0]
PXDAT[15:8]	U0[7:0]	U1[7:0]	U2[7:0]	U3[7:0]
PXDAT[7:0]	Y0[7:0]	Y1[7:0]	Y2[7:0]	Y3[7:0]
4:4:4 (RGB pixel example)				
PXDAT[23:16]	B0[7:0]	B1[7:0]	B2[7:0]	B3[7:0]
PXDAT[15:8]	G0[7:0]	G1[7:0]	G2[7:0]	G3[7:0]
PXDAT[7:0]	R0[7:0]	R1[7:0]	R2[7:0]	R3[7:0]
ONE PIXEL PER FOUR PXCLKs				
MODE	C0	C2	C4	C6
4:4:4:4 (CMYK pixel example)				
PXDAT[23:16]	xx	xx	xx	xx
PXDAT[15:8]	M0[7:0]	K0[7:0]	M1[7:0]	K1[7:0]
PXDAT[7:0]	C0[7:0]	Y0[7:0]	C1[7:0]	Y1[7:0]

Table 4-1. CL550 Color Modes and Pixel Data Configurations

PXCLK

The PXCLK input signal is the main clock for the compression processor. All circuits except those directly related to the Host Bus Interface are driven by this clock. For single-component mode, this clock is equal to the pixel rate. For 4:2:2 modes, this clock is twice the actual pixel rate. For 4:4:4 and 4:4:4:4 modes, this clock is four times the actual pixel rate.

When the CL550 is used in single-component (grayscale) mode or 4:4:4:4 mode, there is a restriction on the skew between PXCLK and HBCLK when setting the configuration register (See Figure 4-9). If this restriction is not satisfied PXADR[15:0] output may not operate correctly until a hardware or software reset is issued to the CL550. In this invalid state, the CL550 is unable to correctly convert raster-formatted pixels to/from block-formatted pixels. (The designer must guarantee that the skew between the falling edge of HBCLK and the rising edge of PXCLK never falls in the failure window.)

Suggested Design Techniques:

For still-frame systems where HBCLK is equal to PXCLK the designer must adjust the clock skew to avoid this condition. Video system designers using YUV or RGB color spaces are not affected.

Note: For correct operation HBCLK must be the same rate or slower than PXCLK (see page 46).

PXPHSE

The value of the PXPHSE input signal indicates whether a line buffer read or write operation is occurring on the pixel bus. This signal should be one-half the frequency of PXCLK. If PXPHSE is high during the rising edge of PXCLK, the ensuing cycle will be a line buffer read operation. If PXPHSE is low during the rising edge of PXCLK, the cycle will be a line buffer write operation.

CLK3

The CLK3 clock input is one-half the frequency of PXPHSE. This signal is used only in 4:4:4 and 4:4:4:4 modes. In all other cases it can be tied to ground. In 4:4:4:4 video modes, CLK3 indicates which pair of components from a 4:4:4:4 pixel mode is on the pixel data bus. If CLK3 is high during the rising edge of PXCLK, it indicates that the first pair of components will be on the pixel bus. If CLK3 is low during the rising edge of PXCLK, it indicates that the second pair of components will be on the pixel bus. In 4:4:4 video modes, a high on CLK3 during the rising edge of PXCLK indicates that the cycle will be a line buffer RAM read cycle. A low on CLK3 indicates that the cycle will be a line buffer RAM write cycle.

HSYNC#

This bidirectional signal is used to indicate the start of a line. When the CL550 is programmed for master mode operation (Configuration register bit 3 = 1), the HSYNC# signal functions as an output and will be asserted when the

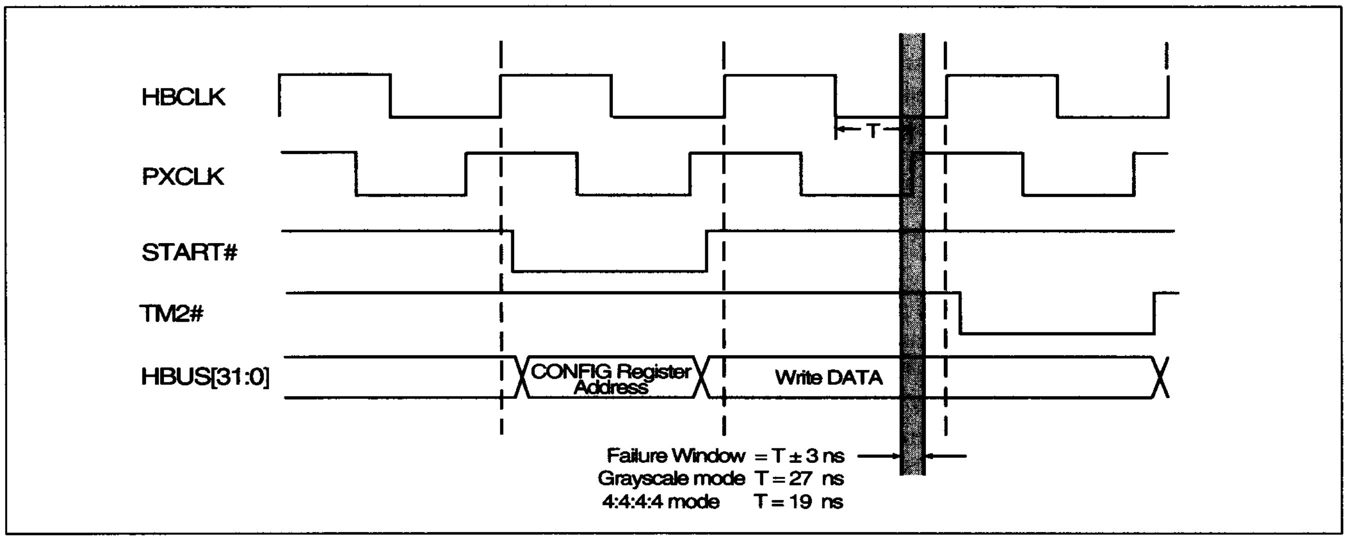


Figure 4-9. Host Bus Write Timing (Writing Configuration register)

CL550 is about to begin a new line. The duration of the pulse is programmed using the HSYNC# register.

When in slave mode (Configuration register bit 3 = 0), the HSYNC# line functions as an input, and the external pixel interface must assert this signal to begin the next line. The HSYNC# input is negative-edge triggered.

Note: When STALL# is asserted, the counters that control HSYNC# (master mode) are also halted. Thus, the HSYNC# transitions are not periodic and should not be used to synchronize directly to synchronous pixel interfaces. When in slave mode, the external interface must delay HSYNC# by the number of clock cycles that STALL# is asserted in the previous line.

VSYNC#

This bidirectional signal is used to indicate the start of a frame. When the CL550 is programmed for master mode operation (Configuration register bit 3 = 1), the VSYNC# signal functions as an output and will be asserted when the CL550 is about to begin a compression or decompression operation. The duration of the pulse is programmed using the VSync register.

When in slave mode (Configuration register bit 3 = 0), the VSYNC# line functions as an input and the external pixel interface must assert this signal after writing to the HVEnable and Start registers to begin a compression or decompression operation. The VSYNC# input is negative-edge triggered.

Note: When STALL# is asserted, the counters that control VSYNC# (master mode) are also halted. Thus, the VSYNC# transitions are not periodic and should not be used to synchronize directly to synchronous pixel interfaces. When in slave mode, the external interface must delay VSYNC# by the number of clock cycles that STALL# is asserted in the previous frame.

STALL#

This input signal, when asserted, will stop all activity on the Pixel Bus Interface in its current state. Signals affected by STALL# include PXADR[16:0], PXDAT[23:0], PXRE#, PXWE#, PXIN#, PXOUT#, BLANK#, VSYNC#, and HSYNC#. In addition, all internal logic modules in the JPEG processing pipeline between the FIFO and the PBI are stopped in their current state, so that no data transfers can take place between the FIFO and the JPEG pipeline. The Huffman CODEC is not affected by the assertion of STALL#, so that the host can access the CODEC register.

When STALL# is negated, processing will resume when the PXCLK, PXPHSE, and CLK3 signals have the same phase relationship as when STALL# was asserted. STALL# is sensed on the rising edge of PXCLK.

The STALL# signal must be used in the following cases:

1. On compression, STALL# is asserted to prevent the CL550's FIFO from overflowing. One of the CL550's FIFO status indicator pins (NMRQ#, DRQ#, or HALF_FULL) is used to generate the STALL# when the data in the FIFO reaches a certain threshold level, typically 1/2 or 3/4 full. The pixel pipeline then will halt, allowing the host to drain the FIFO below the threshold.

2. On decompression, STALL# is asserted to prevent the CL550's FIFO from underflowing. One of the CL550's FIFO status indicator pins (NMRQ#, DRQ#, or HALF_FULL) is used to generate STALL# when the data in the FIFO falls below a certain threshold level, typically 1/2 or 1/4 full. The pixel pipeline will then halt, allowing the host to fill the FIFO above the threshold.

3. During any compression or decompression operation, if the external pixel interface is not ready to deliver a pixel to or receive a pixel from the CL550, the STALL# signal should be asserted to hold off the CL550 processor.

BLANK#

BLANK# changes state at the same time as the PXADR bus, that is, at the beginning of the line buffer read cycle (PXPHSE high).

In compression, BLANK# negates one PXCLK before the first pixel in a line is written to the line buffer; it asserts after the last pixel is written.

In decompression, BLANK# negates simultaneously with the first active pixel read; it asserts one PXCLK after the last pixel read.

4.4.2 Pixel Bus Interface Example

4.4.2.1 CL550 Video System Design

The following example illustrates the use of the CL550 in a video environment. The CL550-30 and CL550-35 can handle either CCIR601, NTSC, or square PAL formats. Because of large variances in the rate of the JPEG data stream inside a given field, the CL550 Huffman CODEC may not keep up with the peaks in the compressed data. A peak in the data rate may cause the CL550's internal FIFO to see an error condition, that is, full or empty depending on the operating mode (compression or decompression). To avoid these occurrences, pixel buffering must be added to the pixel side of the CL550. Then, in case of a threatening FIFO error condition, the pixel pipeline can be stalled and the CODEC given time to recover. The buffering allows a continuous video stream to be maintained at all times. Time that the CL550 pixel pipeline loses while stalled may be recovered during blanking regions. A simplified block diagram is shown in Figure 4-10.

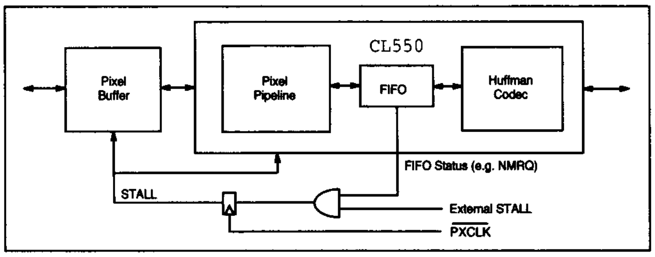


Figure 4-10. CL550 Video System Block Diagram

4.4.2.2 CL550 Still System Design

This subsection describes an example of an interface between the CL550 pixel bus and a host processor bus system. The CL550 is used as a coprocessor resource to the main CPU for still image compression/decompression. A block diagram of the system is shown in Figure 4-11.

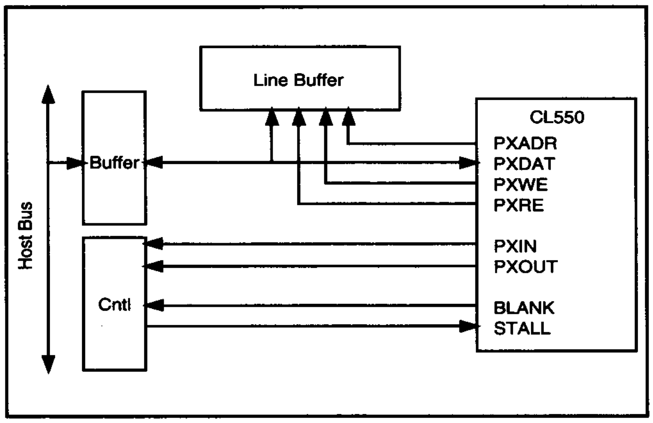


Figure 4-11. Pixel Bus Interface Example

During compression operations, the host writes data through the buffer to the CL550. Master-mode operation is assumed so that the chip manages all of the video timing. BLANK# inactive along with PXIN# active can be used to indicate when the CL550 is ready to accept the next pixel of data. If the host is not prepared to deliver the next pixel, the STALL# signal is asserted. After the host delivers the next pixel of data, STALL# is deasserted long enough to advance the CL550 to the next active pixel. The same control strategy is used for decompression, with the difference that PXOUT# now indicates that a pixel is available for reading. Until the host reads the pixel, STALL# is asserted.

4.4.3 Operational Considerations

4.4.3.1 Operation Without a Line Buffer

In some cases it may be desirable to use the CL550 without using the external line buffer RAM. For systems with a line buffer RAM, the PXIN# and PXOUT# signals provide the status required by the external pixel source or destination to control pixel flow. For systems without a line buffer RAM, during compression the interface circuitry must provide data to the CL550 when it is expecting to read data out of the nonexistent line buffer RAM. During decompression, the interface circuitry must accept pixels from the CL550 when it is expecting to write data into the line buffer RAM.

The interface circuitry is similar to that required when using the line buffer RAM, except that PXRE# is used instead of PXIN# for output enable control for the source data and PXWE# is used instead of PXOUT# to strobe pixel data into a register.

The final issue that must be resolved is the format of the pixel data. Since the line buffer RAM is no longer reordering raster data for the CL550, the pixel data must be handled in the 8x8 block format. For an image compression/decompression coprocessor where the image is stored in memory, this format is provided externally by the CPU or DMA controller. Pseudo-code to perform memory-to-coprocessor compression is shown in Figure 4-12.

```
;
;IMAGE - pointer to image stored in memory
;PXBUS - address of PXDAT bus transfers
;LINE_OFFSET = (line_length) - 8
;BLOCK_OFFSET = 8 - (8 * (line_length))
;EOL_LINE_OFFSET = (7 * line_length)

LOOP:
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
        MOVE.L      (IMAGE)+,       PXBUS
;
;After 8 pixels in line, point to first pixel in next
line
        ADD         LINE_OFFSET,IMAGE
        REPEAT      8_TIMES
;
;After 8 lines in a block, point to first pixel in first
line of next block (note that this is a negative
offset)
        ADD         BLOCK_OFFSET,IMAGE
        REPEAT      TILL_END_OF_LINE
;
;After the end of a line of 8x8 blocks, point to the
beginning of the next line of 8x8 blocks.
        ADD         EOL_LINE_OFFSET, IMAGE
        REPEAT      TILL_END_OF_IMAGE
;
;Note: with additional code, the redundant
additions can be avoided
;
```

Figure 4-12. Pseudo-Code for Data Addressing in Compression System Without a Line Buffer

4.4.3.2 Operation With a Line Buffer

The line buffer must be sized to accept an entire strip of pixels.

Line buffer depth = 8 * (# pixels per line) Bytes
Line buffer width = (# components/pixels) Bytes

For video systems, a typical line buffer uses 8K x 8 SRAMs, accommodating line sizes of up to 1K pixels. The CL550 will use the lowest order address space first. Thus if the required line buffer depth is 4K, then 4K x 8 SRAMs can be used.

The line buffer addressing algorithm is a complex modulo counting scheme that uses a read followed by a write to the same address.

The address remains constant for each pixel read/write pair, so that when a pixel is read into the CL550 from the strip buffer, the next incoming pixel is written into the vacated SRAM location. In this manner, the SRAM never needs to be greater than 8 lines deep. The first 8 lines are preloaded into the SRAM in linear memory locations as follows (at this point, nothing is being read into the CL550):

```
x(x):   x = Line number
       (x) = Pixel number

0(0)............ 0(7)  0(8)............ 0(15)  . . .
1(0)............ 1(7)  1(8)............ 1(15)  . . .
2(0)............ 2(7)  2(8)............ 2(15)  . . .
3(0)............ 3(7)  3(8)............ 3(15)  . . .
4(0)............ 4(7)  4(8)............ 4(15)  . . .
5(0)............ 5(7)  5(8)............ 5(15)  . . .
6(0)............ 6(7)  6(8)............ 6(15)  . . .
7(0)............ 7(7)  7(8)............ 7(15)  . . .
```

Once the 8 lines are loaded into the SRAM, the CL550 begins to read the first block of 8x8 pixel from the SRAM, replacing each pixel read with the next incoming pixel. At the end of the first block, the data in the buffer will look like:

```
x(x):   x = Line number
       (x) = Pixel number

8(0)............... 8(7)   0(8)............ 0(15)  . . .
8(8)............... 8(15)  1(8)............ 1(15)  . . .
8(16)............. 8(23)  2(8)............ 2(15)  . . .
8(24)............. 8(31)  3(8)............ 3(15)  . . .
8(32)............. 8(39)  4(8)............ 4(15)  . . .
8(40)............. 8(47)  5(8)............ 5(15)  . . .
8(48)............. 8(55)  6(8)............ 6(15)  . . .
8(56)............. 8(63)  7(8)............ 7(15)  . . .
```

From this point, after each 8 line strip is read into the buffer, the address patterns alternate from linear addresses to block-wise addresses. This addressing scheme is very handy for re-ordering raster data using a minimum of buffer space.

4.4.4 PBI AC Timing Characteristics

			PGA				MQUAD					
			CL550-35		CL550-30		CL550-30		CL550-10			
Time	**Fig.**	**Description**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Units**	**Note**
T50	4-13	PXCLK Pulse Width HIGH	13		15		15		30		ns	1
T51	4-13	PXCLK Pulse Width LOW	13		15		15		30(23)		ns	1,6
T52	4-13	PXCLK Clock Period	29		34		34		100		ns	1
T53	4-13	PXPHSE Setup Time	16		18		18		20		ns	1
T54	4-13	PXPHSE Hold Time	5		5		5		7		ns	1
T55	4-13	CLK3 Setup Time	16		18		18		20		ns	1
T56	4-13	CLK3 Hold Time	5		5		5		7		ns	1
T57	4-14	PXADR [15:0] Hold Delay	4		4		4		3		ns	1,5
T58	4-14	PXADR [15:0] Delay Time		14		15		15		22	ns	1
T59	4-14	PXDAT [23:0] Setup Time	3		4		4		6		ns	1,2,3
T60	4-14	PXDAT [23:0] Hold Time	4		5		5		7		ns	1
T61	4-14	PXRE Delay to LOW		12		13		13		18	ns	1
T62	4-14	PXRE Delay to HIGH		12		13		13		18	ns	1
T63	4-14	PXWE, PXRE Overlap	-3		-3		-3		-4		ns	1,5
T64	4-14	PXWE Delay to HIGH		12		13		13		18	ns	1
T65	4-14	PXWE Delay to LOW		12		13		13		18	ns	1
T66	4-14	PXIN Delay to HIGH		12		13		13		18	ns	1
T67	4-14	PXIN Delay to LOW		12		13		13		18	ns	1
T68	4-14	PXWE HIGH to Overlap	-2	2	-2	2	-2	2	-4	4	ns	1,5
T69	4-14	STALL Setup Time	14		15		15		20		ns	1
T70	4-14	STALL Hold Time	7		8		8		12		ns	1
T71	4-14	BLANK Delay Hold Time	4		4		4		3		ns	1,5
T72	4-14	BLANK Delay Time		13		15		15		18	ns	1
T73	4-14	HSYNC,VSYNC Delay Hold Time	4		4		4		3		ns	1,5
T74	4-14	HSYNC, VSYNC Delay Time		24		26		26		30	ns	1
T75	4-14	PXIN, PXRE Overlap	-2	2	-2	2	-2	2	-4	4	ns	1,5
T76	4-14	PXRE, PXIN Overlap	-2	2	-2	2	-2	2	-4	4	ns	1,5
T77	4-14	PXWE HIGH, PXADR Overlap	0		0		0		0		ns	1,5
T78	4-14	HSYNC, VSYNC Setup	7		8		8		12		ns	1
T79	4-14	HSYNC, VSYNC Hold	7		8		8		12		ns	1
T80	4-15	PXDAT Delay to Hi-Z		23		24		24		28	ns	1,4,5
T81	4-15	PXDAT [23:0] Delay Time		23		24		24		28	ns	1
T82	4-15	PXDAT, PXRE Overlap	-3		-3		-3		-4		ns	1,5
T83	4-15	PXRE, PXDAT Overlap	3	12	3	12	3	12	2	13	ns	1,5
T84	4-15	PXOUT to LOW Delay		12		13		13		18	ns	1
T85	4-15	PXOUT to HIGH Delay	4	12	4	13	4	13	3	18	ns	1
T86	4-15	PXOUT, PXRE Overlap	-2		-2		-2		-4		ns	1,5
T87	4-15	PXOUT, PXDAT Overlap	3	12	3	12	3	12	2	13	ns	1,5

Table 4-2. Pixel Bus Timing Table

Pixel Bus Timing Notes

1. Inputs switch between 0 and 3.5 V at 1 V/ns. Measurements were made at 1.5 V. CL = 50 pf.
2. PXDAT [23:0] is input-only during compression.
3. SRAM access time <= T_{52}-T_{58}-T_{59}.
4. Decompression parameter.
5. Not 100% tested Guaranteed by design characteristics.
6. Characteristics in parentheses apply to part number CL550-10N, required for NuBus applications.

4.4.5 Timing Diagrams

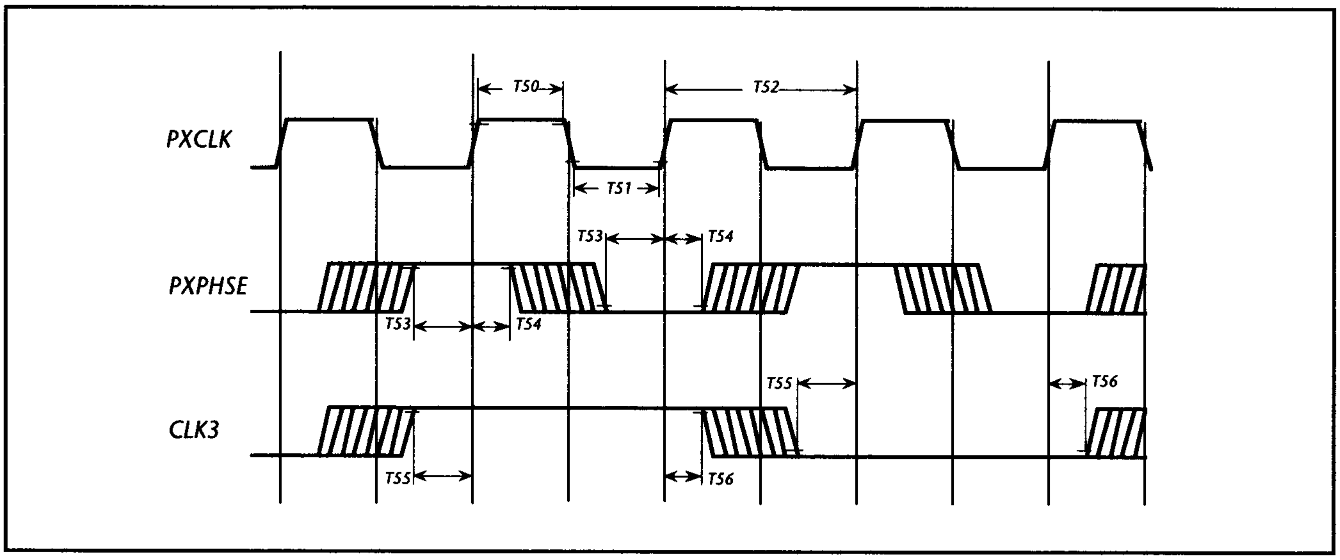


Figure 4-13. Pixel Bus Clock Timing

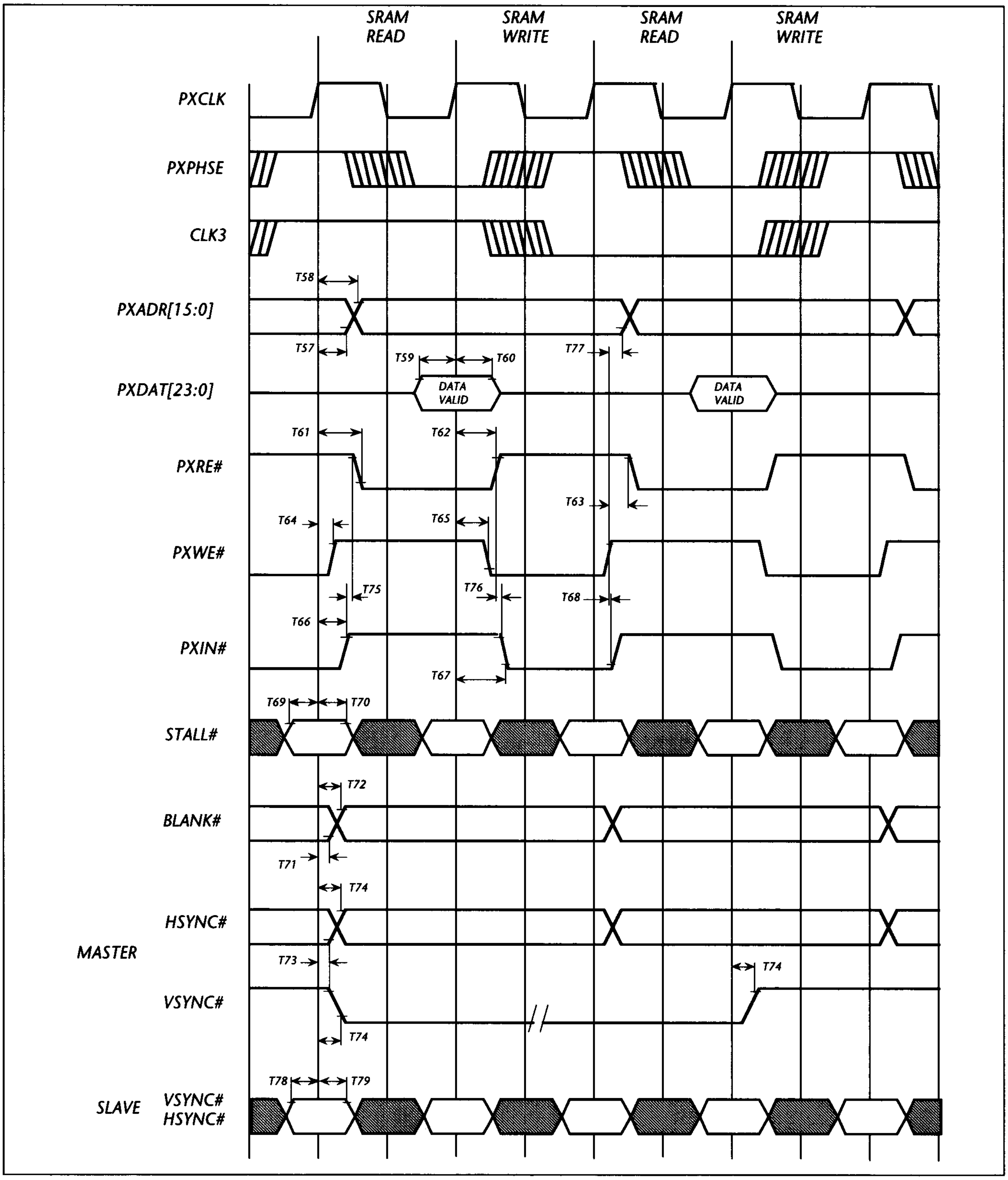


Figure 4-14. Pixel Bus Timing: Compression, Full-Rate Mode.

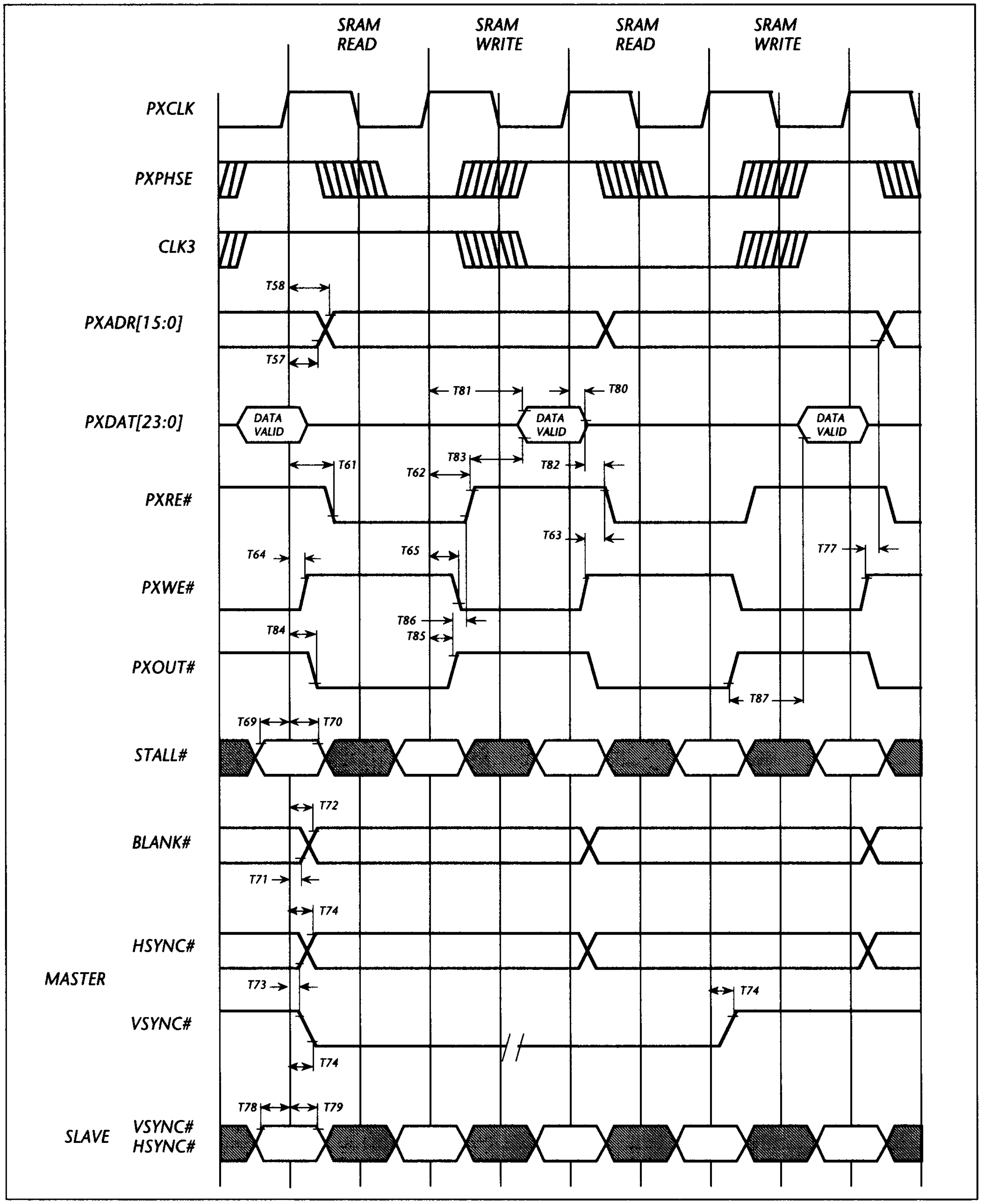


Figure 4-15. Pixel Bus Timing: Decompression, Full-Rate Mode.

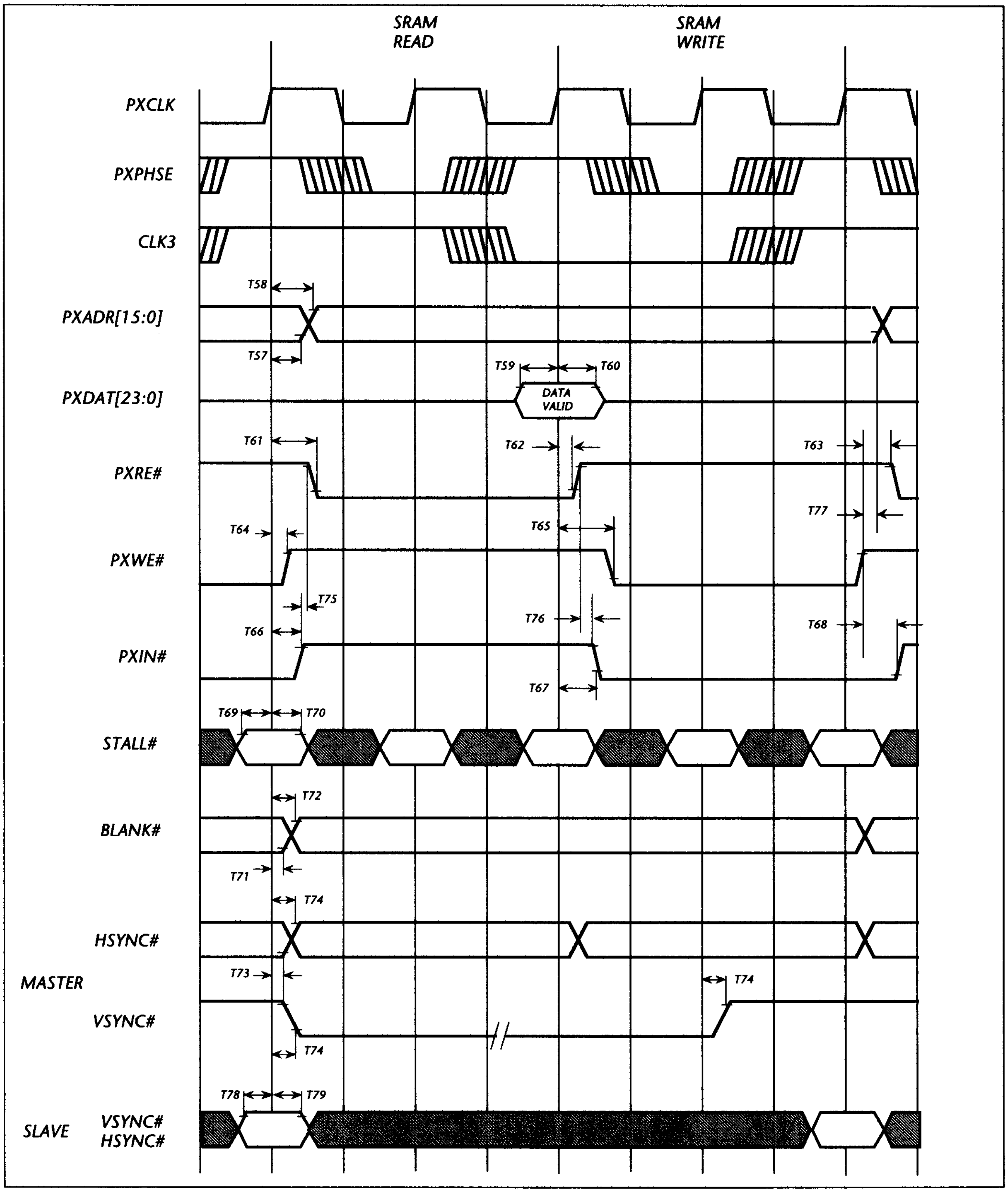


Figure 4-16. Pixel Bus Timing: Compression, Half-Rate Mode

Note: Half-rate modes apply to 4:4:4 modes only.

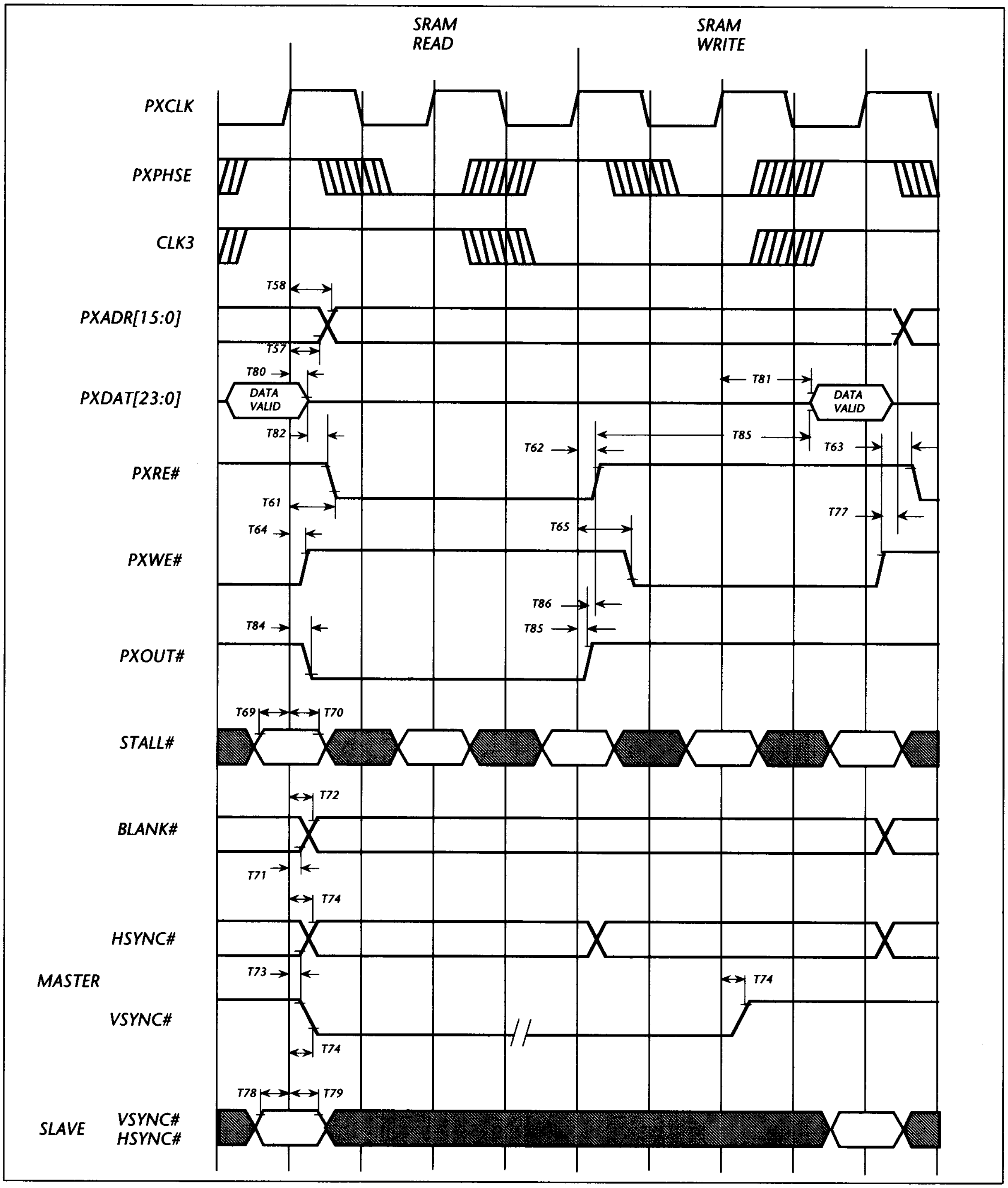


Figure 4-17. Pixel Bus Timing: Decompression, Half-Rate Mode

Note: Half-rate modes apply to 4:4:4 modes only.

4.4.5.1 Master-Mode Compression: 4:4:4 to 4:2:2 Format

The following set of diagrams (Figures 4-18 through 4-22) describe master-mode compression in the 4:4:4 to 4:2:2 mode with the following video parameters:

Register	Value	Comments
HPeriod	56	57 pixels per line
HSync	9	HSync pulse is 10 pixels wide
HDelay	6	6-pixel delay from falling edge of HSync to first active pixel
HActive	11	48 pixels per active line
VPeriod	53	53 lines
VSync	3	VSync pulse is 3 lines wide
VDelay	10	10-line delay from the falling edge of VSync to the first active line
VActive	4	32 active lines

Table 4-3. Operating Conditions

The behavior of the Pixel Bus Interface (PBI) signals is illustrated in different cases: beginning of frame, beginning of active lines, middle of frame, end of active lines, and end of frame.

4.4.5.1.1 Compression Overview

Figure 4-18 illustrates the phasing of the PXIN#, BLANK#, PXRE#, and PXWE# signals with respect to the beginning, middle, and end of the active window. At the beginning of the frame, the first 8 lines of data are written into the line buffer while no data is read out by the CL550. In the middle of the frame, data is both written into the line buffer by the external video source and read out of it by the CL550. At the end of the frame, the last 8 active lines are read out of the line buffer by the CL550 but no new data is written into it.

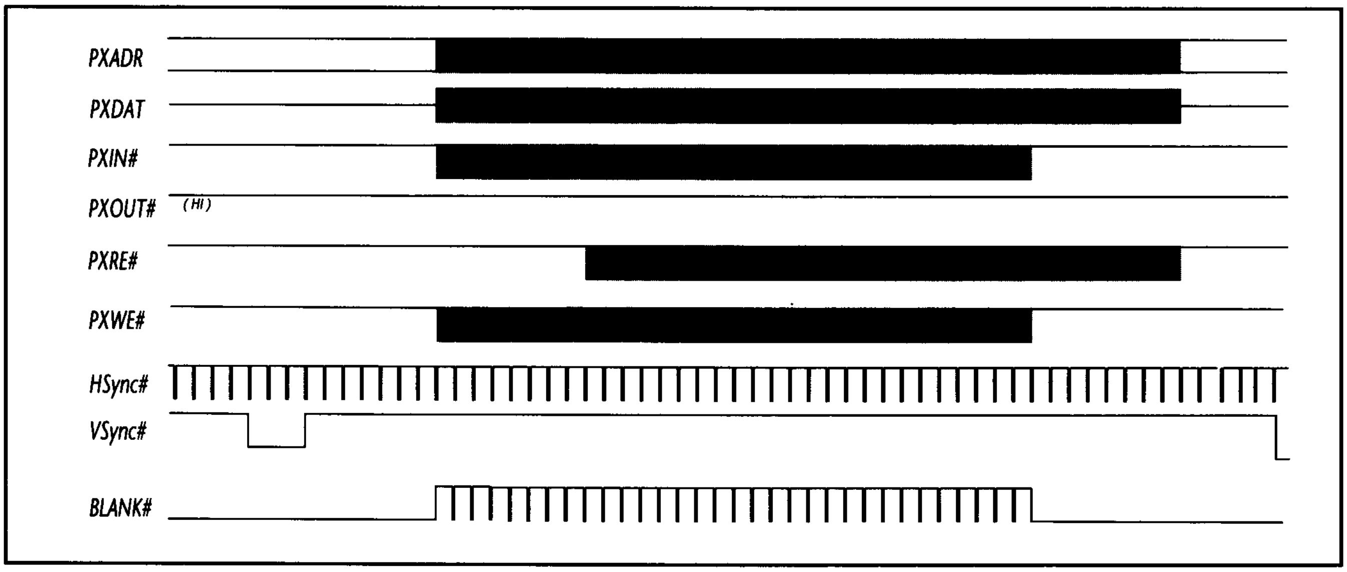


Figure 4-18. Compression Overview

4.4.5.1.2 First Line With Active PXIN#

Figure 4-19 illustrates the signal activity around the beginning of the first active line. The data driven onto the line buffer is from an external buffer whose OE# is controlled by the CL550 PXIN# output. Data is not read out of the line buffer until the end of the eighth active line. Therefore the PXRE# output is inactive.

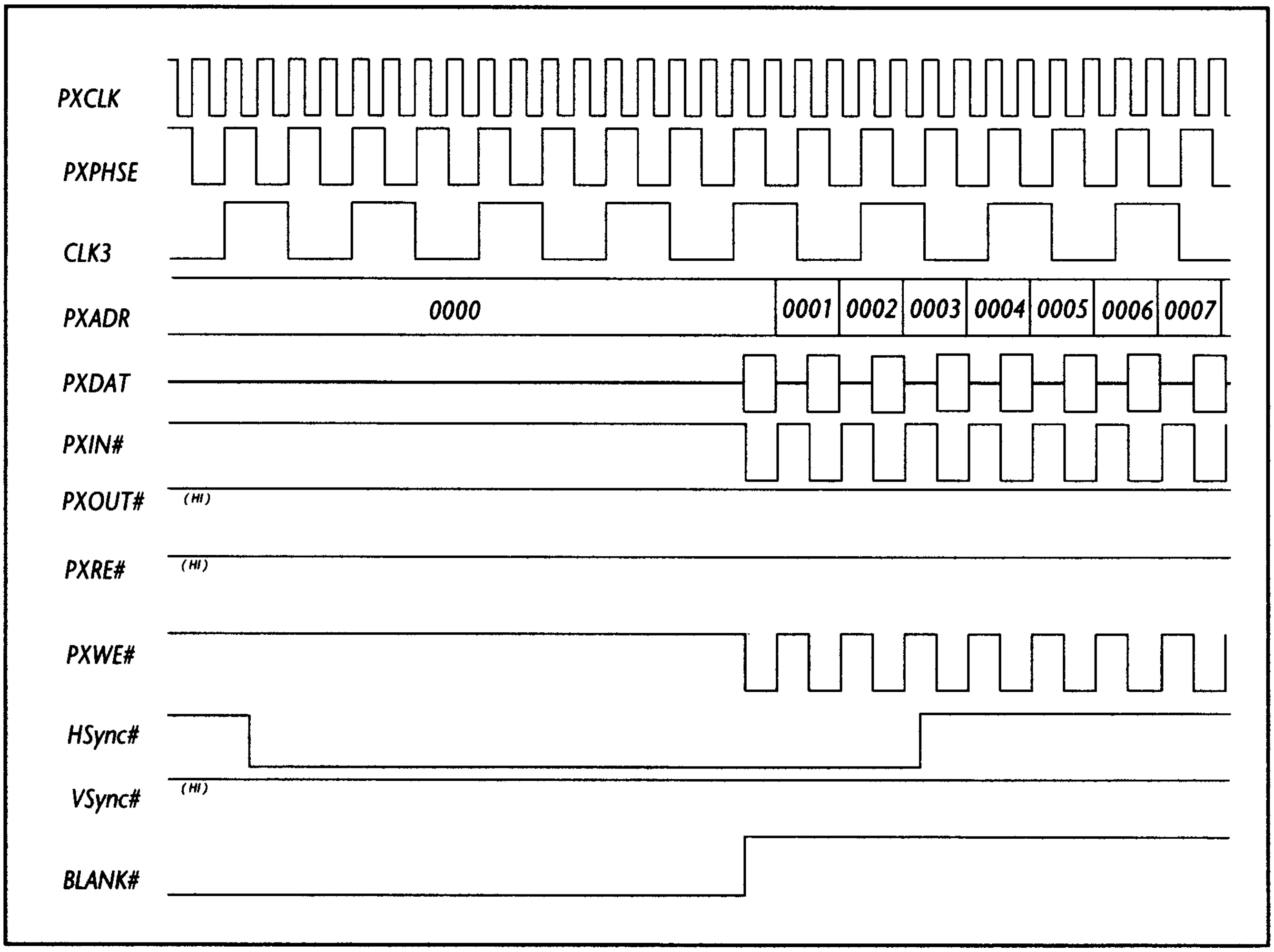


Figure 4-19. First Active Line (Compression)

4.4.5.1.3 First Line With Active PXRE#

Figure 4-20 illustrates the signal activity around the beginning of the ninth active line, when the CL550 starts reading data out of the line buffer and PXRE# becomes active. On every other cycle, data is written into the static RAM by the video source and read out of it by the CL550. The black regions on PXDAT that occur while PXRE# is active indicate periods when unknown data is driven onto the bus by the line buffer RAM.

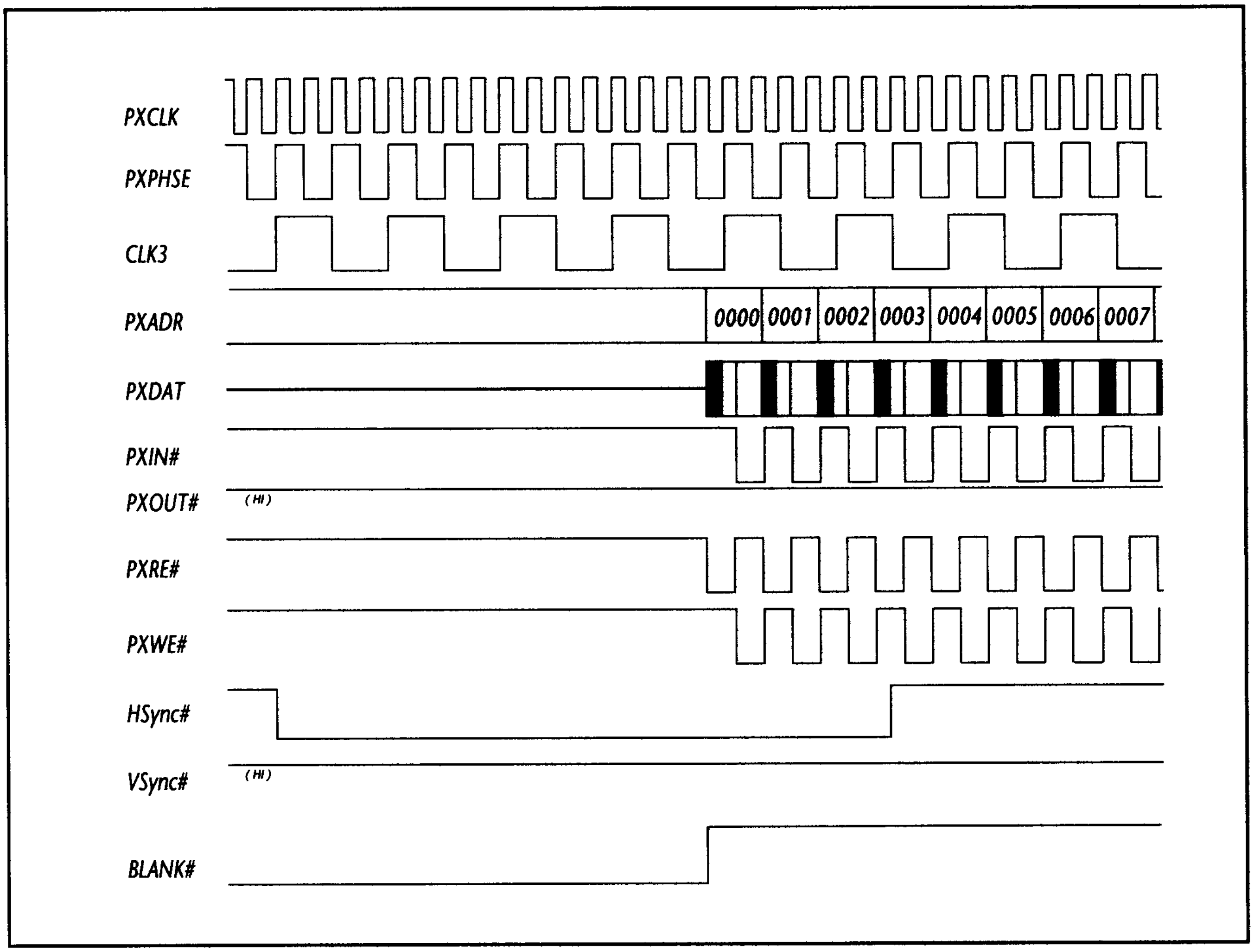


Figure 4-20. First Active Line With Active PXRE# (Compression)

4.4.5.1.4 Last Line With Active PXIN#

Figure 4-21 illustrates the signal activity around the last line with PXIN#, PXRE#, and PXWE# active. As the CL550 starts processing the last eight active lines, no new data is written into the SRAM line buffer. PXWE# and PXIN# become inactive, while PXRE# is still active as data is read out of the line buffer into the CL550.

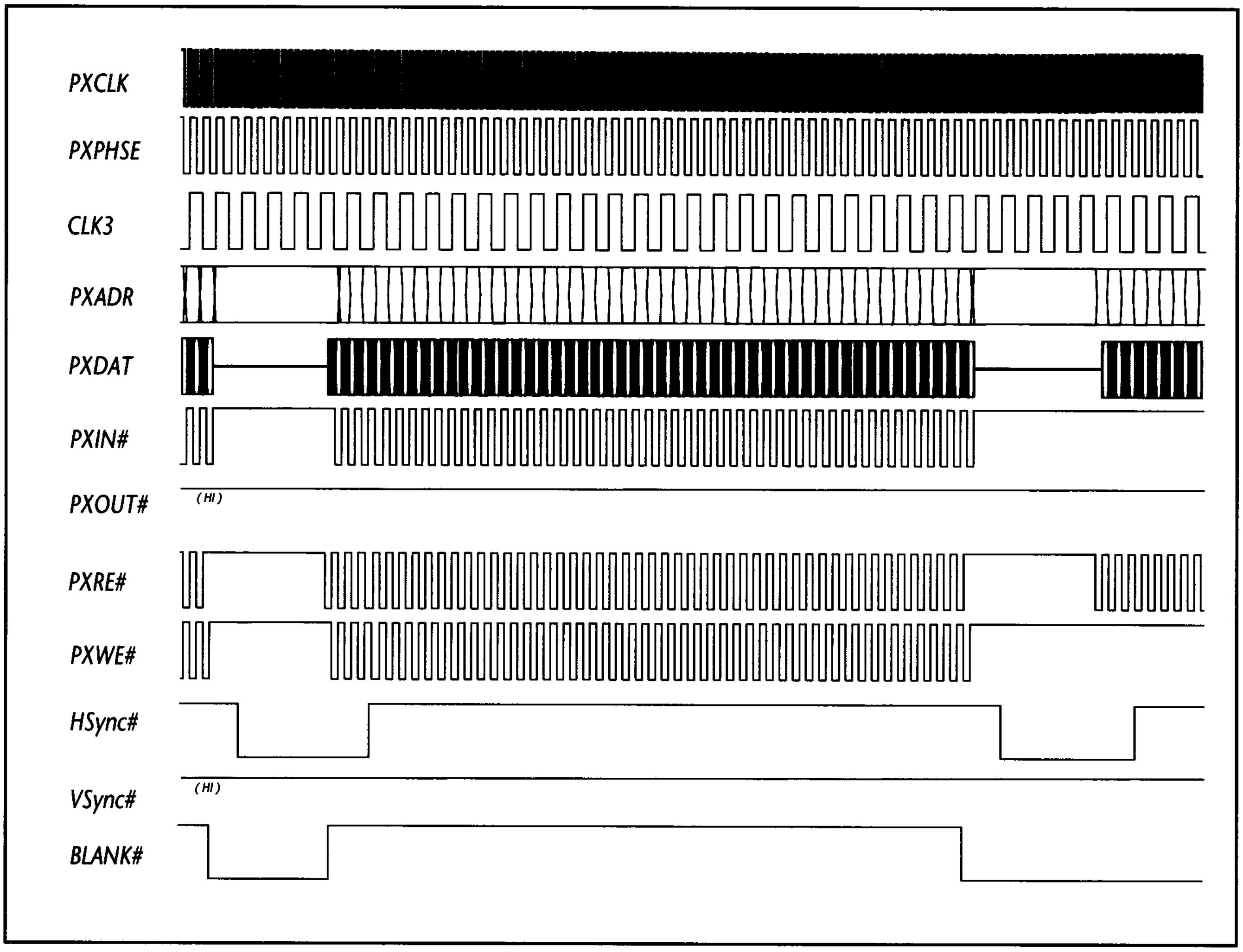


Figure 4-21. Last Line With Active PXIN#

4.4.5.1.5 Last Line With Active PXRE#

Figure 4-22 illustrates the signal activity around the last active line. When the last line is read out of the line buffer, the CL550 PXRE# signal becomes inactive.

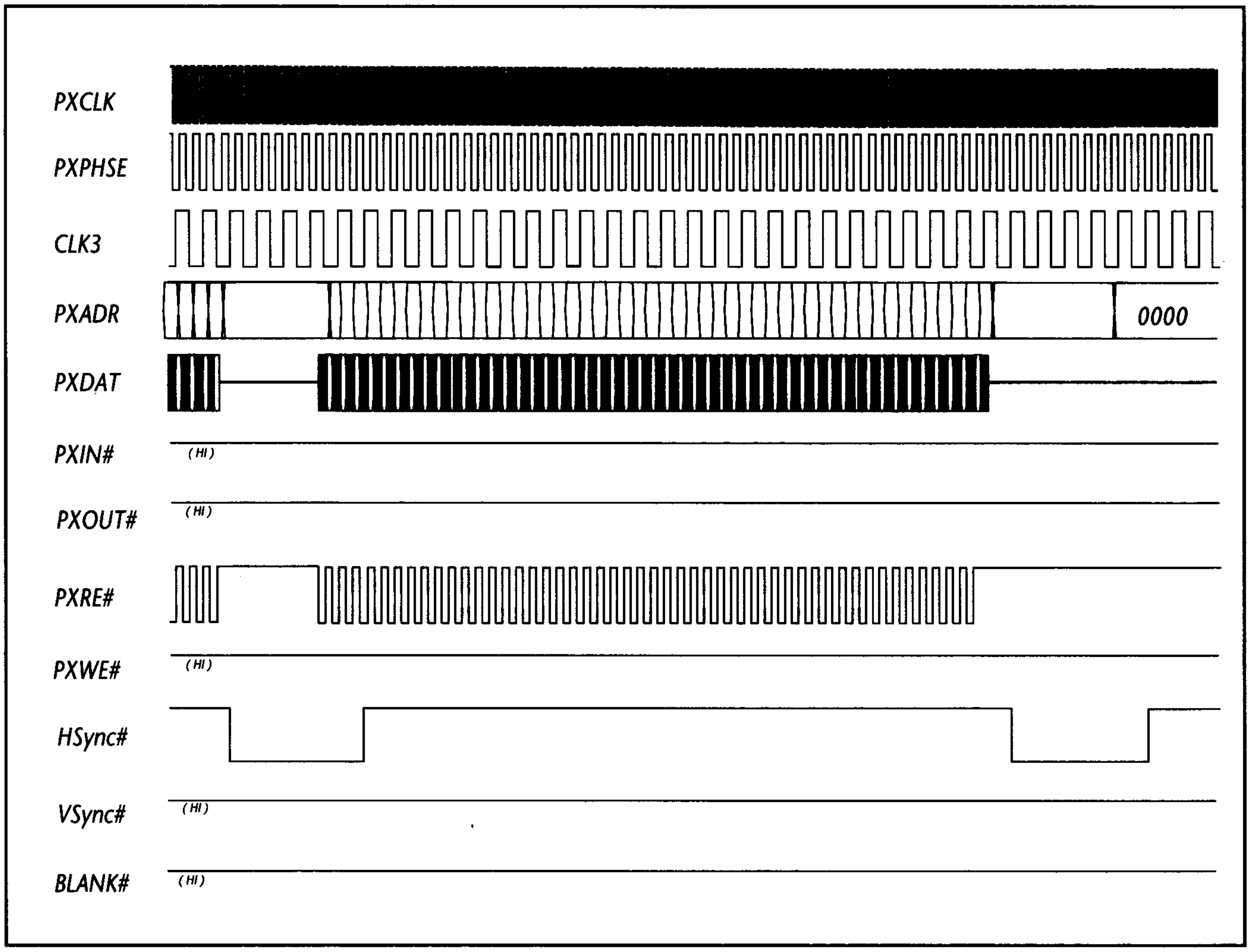


Figure 4-22. Last Line With Active PXRE#

4.4.5.2 Master-Mode Decompression 4:4:4 to 4:2:2 Format

This set of Figures (figures 4-23 through 4-27) describe master mode decompression in the 4:4:4 to 4:2:2 mode with the following video parameters:

Register	Value	Comments
HPeriod	56	57 pixels per line
HSync	9	HSync pulse is 10 pixels wide
HDelay	6	6-pixels delay from falling edge of HSync to first active pixel
HActive	11	48 pixels per active line
VPeriod	53	53 lines
VSync	3	VSync pulse is 3 lines wide
VDelay	10	10-line delay from the falling edge of VSync to the first line with active PXWE
VActive	4	32 active lines

Table 4-4. Operating Conditions

4.4.5.2.1 Decompression Overview

Figure 4-23 illustrates the phasing of the PXOUT#, BLANK#, PXRE#, and PXWE# signals with respect to the beginning, middle, and end of the frame. The initial eight active lines are written by the CL550 into the line buffer. The PXWE# output is active, while PXRE# and PXOUT# are inactive. On the ninth active line, PXOUT# and PXRE# become active, allowing an external source to read data out of the SRAM. After the last eight active lines of the frame are written by the CL550 into the line buffer, PXWE# becomes inactive while PXRE# and PXOUT# remain active, allowing the line buffer to be drained.

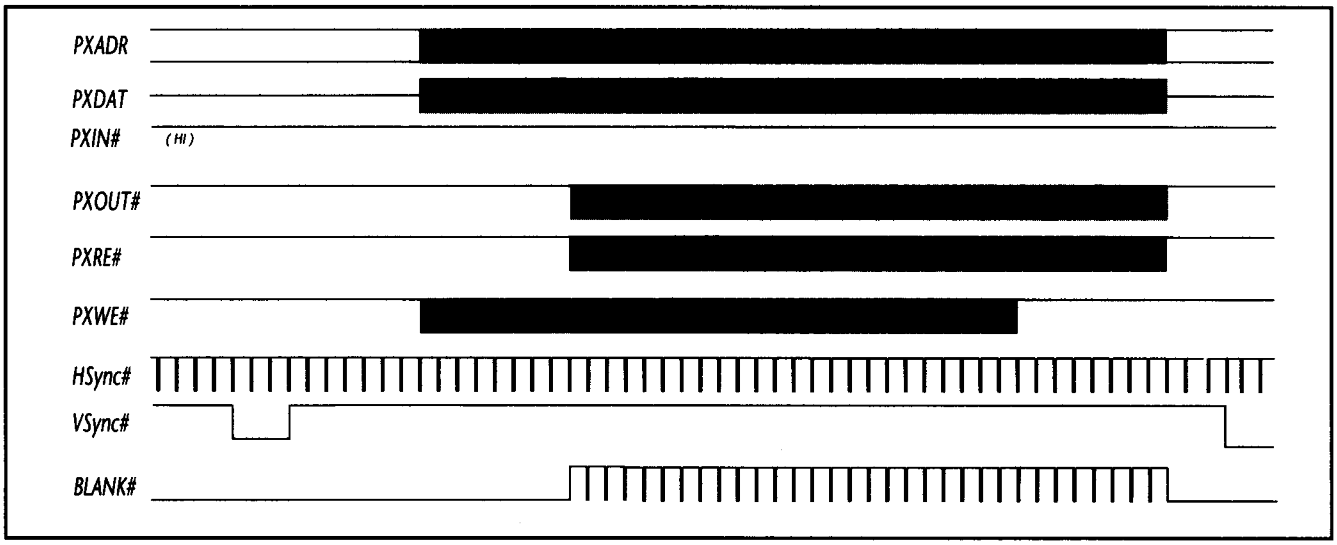


Figure 4-23. Decompression Overview

4.4.5.2.2 First Line With Active PXWE#

Figure 4-24 illustrates the signal activity around the first data written to the line buffer RAM. The data driven onto PXDAT is from the CL550. Note that the first group of pixels written to the line buffer RAM is not bounded by an HSYNC# interval. The decompression from the FIFO starts on one HSYNC and the data is written to the line buffer after a delay of HDelay plus the CL550 video pipeline latency. When the first line is completed, the CL550 waits until the start of the next line of data to start processing lines in sync with HSYNC#.

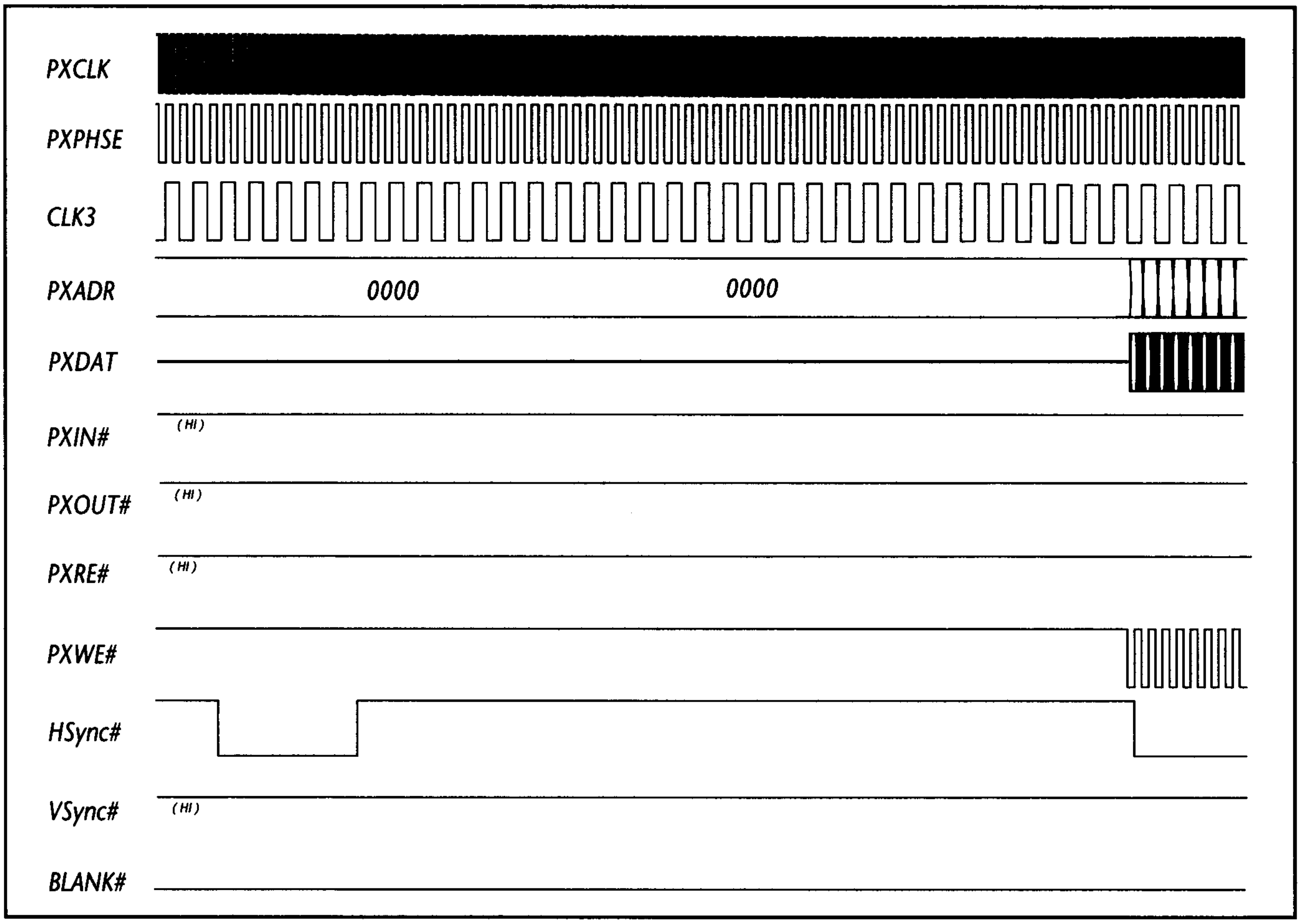


Figure 4-24. First Active Line With Active PXWE#

4.4.5.2.3 First Line With Active PXOUT#

Figure 4-25 illustrates the signal activity around the first line with PXOUT#, PXRE#, and PXWE# active (ninth active line). The black regions on PXDAT that occur while PXRE# is active indicate periods when unknown data is driven onto the bus by the line buffer RAM. The region following a black area indicates a period when the stable data is driven by the line buffer RAM onto PXDAT (and out to an external register). The data driven onto PXDAT while PXWE# is active is from the CL550.

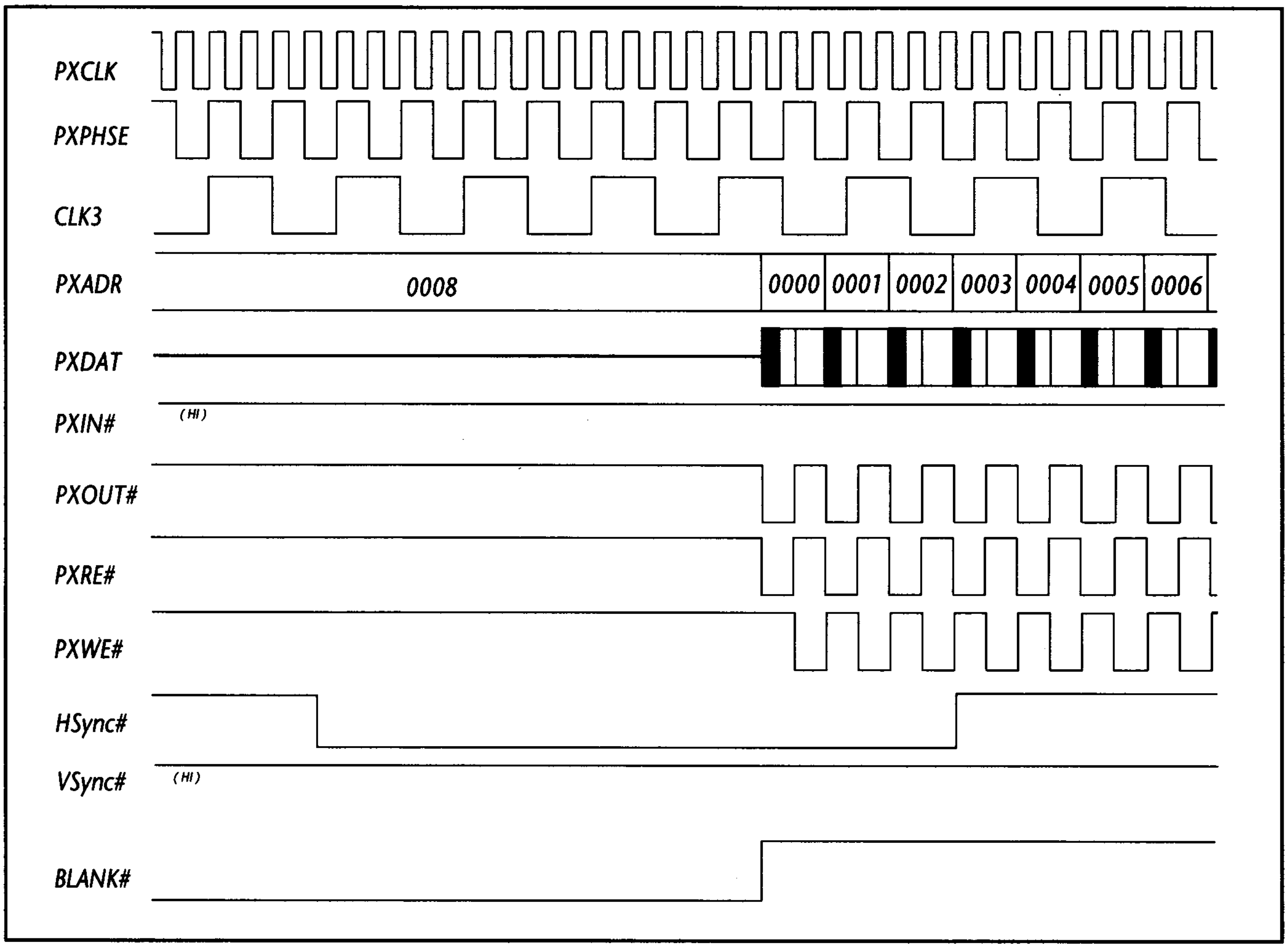


Figure 4-25. First Active Line With Active PXOUT#

4.4.5.2.4 Last Line With Active PXWE#

Figure 4-26 illustrates the signal activity around the last line with PXOUT#, PXRE#, and PXWE# active.

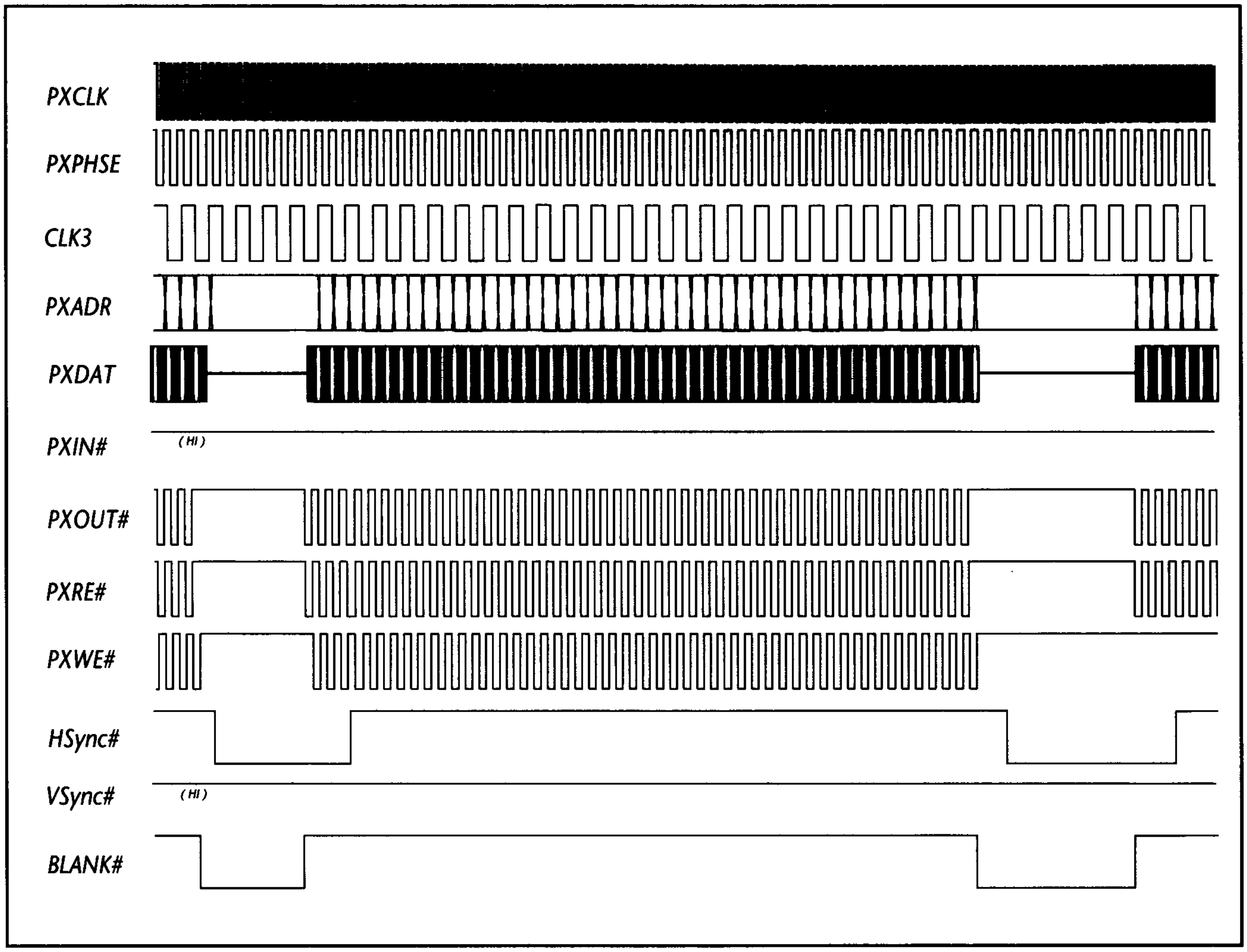


Figure 4-26. Last Line With Active PXWE#

4.4.5.2.5 Last Line With Active PXOUT#

Figure 4-27 illustrates the signal activity around the last line with PXIN#, PXRE#, and PXWE# active After the last active line is read out of the strip buffer, both PXRE# and PXOUT# become inactive.

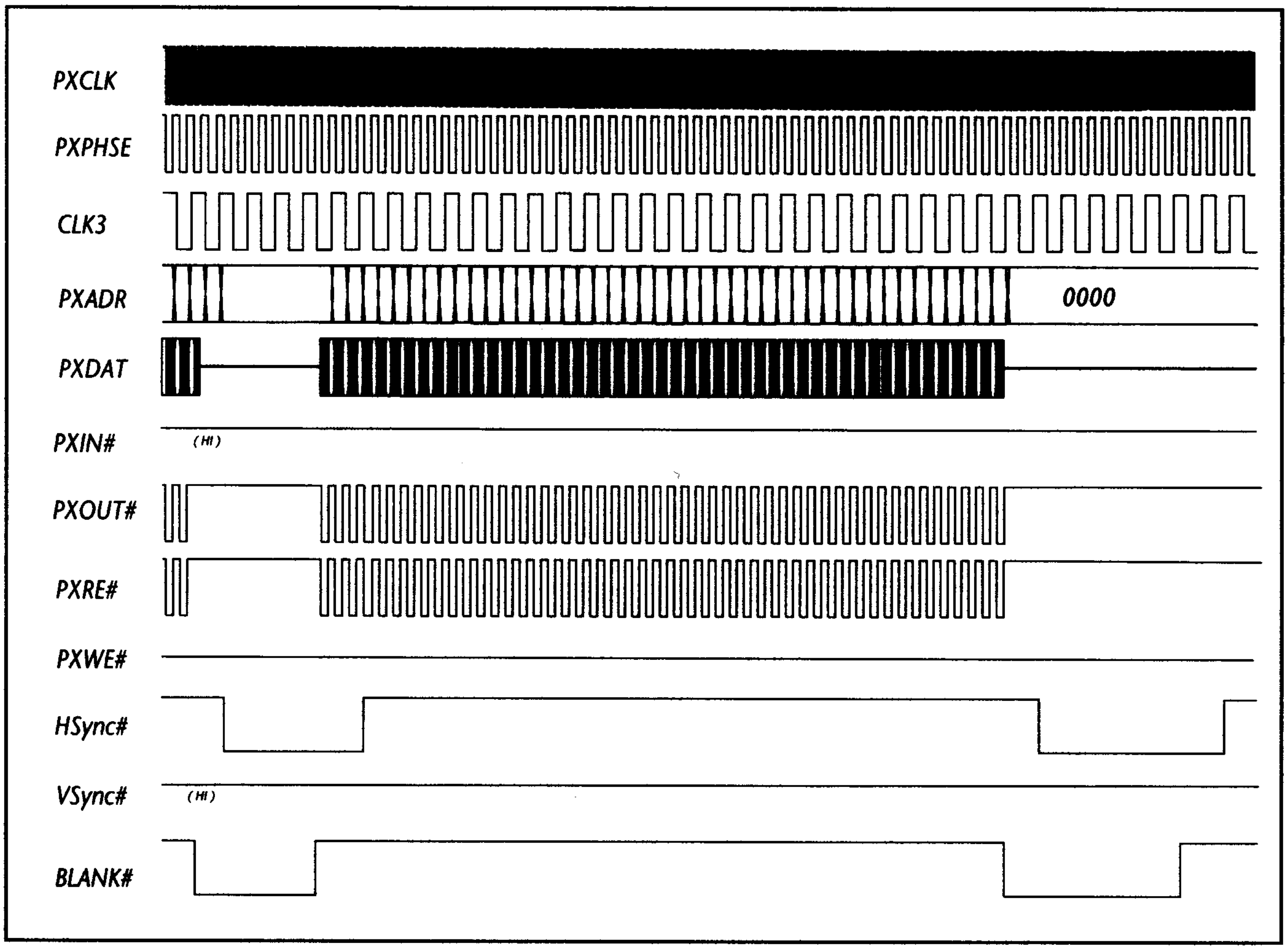


Figure 4-27. Last Line With Active PXOUT#

4.4.5.5 STALL# Timing

Figure 4-28 illustrates the effect that STALL# has on the pixel bus control signals when the CL550 is in RGB-to-YUV 4:2:2 master-mode compression.

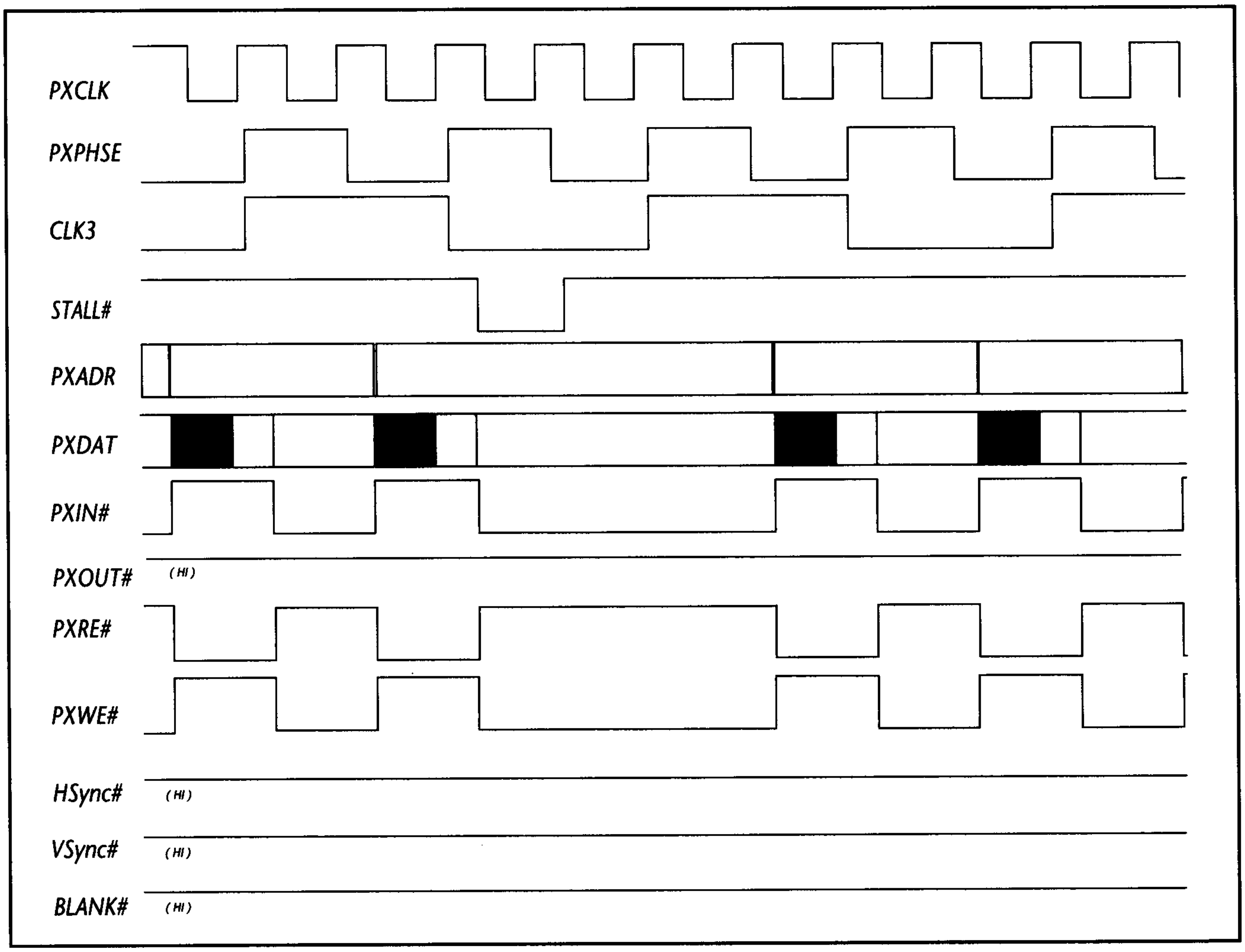


Figure 4-28. STALL# Timing

4.5 Host Bus Interface

This subsection describes the CL550 Host Bus Interface signals and data transfer procedures. It also includes AC specifications and detailed timing diagrams.

4.5.1 Signal Descriptions

4.5.1.1 Utility Signals

The utility signals on the CL550 include HBCLK, RESET#, ID[3:0]#, HBUS_32#, NMRQ#, and TEST.

HBCLK

The falling edge of HBCLK is used to sample the host bus data and control signals, while the rising edge of HBCLK is used to drive the output signals. Any duty cycle clock can be used as long as the specifications in Table 4-8 are met. For correct operation of the clock synchronization, HBCLK must be the same rate or slower than PXCLK.

Note: Refer to PXCLK skew description (see page 25) for HBCLK skew.

RESET#

The RESET# signal is an input that forces a hardware reset of the CL550. When the signal is asserted, most of the internal registers are forced to a known state. However, the values in the Huffman tables, DCT table, and Quantizer tables are unaffected. HBCLK must be running during RESET. The CL550 will not acknowledge any access until the end of the third cycle after RESET# is deasserted.

ID[3:0]#

The address space identification signals, ID[3:0], are inputs that select the address range of the chip. Setting ID to 1h through Eh selects an address region for the CL550. The ID signals enable placement of the CL550 in one of fourteen locations in the upper 1/16th of memory. In this range, the 32-bit address takes the form:

FSh (10xx)0 AA AA

where S is the value on the ID lines in the range 1h through Eh and AA AA is the address of one of the on-chip registers. Setting ID to 0h overrides the decoding of the upper 16 bits of the address, effectively putting the CL550 into a 16-bit address mode. Setting ID to Fh disables the Host Bus Interface.

HBUS_32#

HBUS_32# is a static signal used to configure the host bus data path width during CODEC accesses. When HBUS_32# is low, reads and writes to the CODEC register are 32 bits wide. When HBUS_32# is high, reads and writes are 16 bits wide. The utility bus, that is, the data path to all on-chip registers (except for the CODEC register) and tables, is always 16 bits wide.

The ID[3:0]# and HBUS_32# signals control the address/data mode as shown in Table 4-5.

ID[3:0]#	HBUS_32#	Description
0000	0	16-bit address, 32-bit data
0000	1	16-bit address, 16-bit data
1h-Eh	0	32-bit address, 32-bit data
1h-Eh	1	32-bit address, 16-bit data
1111	0/1	Chip disabled

Table 4-5. Address and Data Bus Configuration

NMRQ#

Interrupt request (NMRQ#) is an unlatched output signal, synchronous to HBCLK, that provides an indicator for both FIFO and video field status. This signal is open collector and should be tied to VCC through a resistor of at least 625 ohms. It can be programmed to selectively indicate active status flags as specified in the Interrupt Mask register. On power up, the CL550 should be hardware reset to prevent the generation of spurious interrupts.

TEST

The TEST signal is an input that forces all CL550 outputs to tristate. This feature is provided to simplify board-level diagnostics. TEST should be tied low for normal operation.

4.5.1.2 Data Transfer Signals

The CL550 host bus is transaction oriented, each transaction consisting of a start cycle and an acknowledge cycle. Several signals are used to implement the bus handshake.

START#

The START# input signal begins a data transfer. When asserted, it indicates that there is a valid address on the host bus (HBUS [31:0]). START# is sampled on the falling edge of HBCLK. START# should not be asserted for more than one HBCLK period.

TM[0]#

The value of the TM[0]# signal is ignored during a start cycle. In normal mode (bus slave mode) operation, the TM[0]# line returns to 0 during the acknowledge cycle. In bus master mode (entered by assertion of the DMA_MSTR# signal during a start cycle), this signal (along with TM[1]#) is sampled during the assertion of TM[2]# (acknowledge cycle) to determine whether a bus error has occurred.

TM[1]#

The TM[1]# signal is an input during a start cycle and indicates whether the transaction is a read or a write. A low (0) value indicates a write cycle and a high (1) value indicates a read cycle. In normal operation (bus slave mode), the TM[1]# signal returns to zero during the acknowledge cycle. In bus master mode (entered by assertion of the DMA_MSTR# signal during a start cycle), the sense of this signal is inverted so that a low value indicates a write cycle from CL550 to memory and a high value indicates a read cycle from memory to CL550. In bus master mode, this signal is sampled during the assertion of TM[2]# (acknowledge cycle) to determine whether a bus error has occurred (refer to Table 4-7 for details). Table 4-6 below illustrates the coding of the TM[1]# signal during a start cycle.

TM[2]#

TM[2]# is the acknowledge signal that is driven active (0) by the CL550 during a bus slave-mode ackowledge cycle. In bus master mode, this signal is driven by an external source to indicate that the transfer is complete.

HBUS[31:0]

HBUS is the multiplexed address and data port to the host. The HBUS signal has several different modes of operation, which are determined by the state of HBUS_32# and ID[3:0] as illustrated in Table 4-5.

HBOUT#

The HBOUT# output signal is for bus transceiver direction control. If this signal is low (0), the transceiver direction is from CL550 out to the host bus. If the signal is high (1), the transceiver direction is from the host bus into the CL550.

TMOUT#

The TMOUT# output signal provides transceiver direction control for the transfer mode control lines (TM[2:0]). If this signal is low (0), the transceiver direction is from CL550 out to the host bus. If the signal is high (1), the transceiver direction is from the host bus into the CL550.

4.5.1.3 DMA Signals

DRQ#

The DRQ# signal is an output, synchronous to HBCLK, that provides chip status for DMA interface control. The DRQ# output is also controlled by the Flag register bits, including the CODEC busy (CODECB) and Bus Error (Buser) flag bits. This signal is open collector and should be tied to VCC through a resistor of at least 625 ohms.

DMA_MSTR#

The DMA_MSTR# signal is an input that allows the CL550 to work with a DMA controller functioning as a bus master for CODEC register transfers. It is sampled on the falling edge of HBCLK when the START# signal is active. Refer to the Master Mode Transactions paragraph on page 49 and to the Operational Considerations on page 50 for more details.

TM[1]#	DMA_MSTR#	Buffer Direction	Operation
L	H	Host to CL550	Normal write cycle
H	H	CL550 to host	Normal read cycle
L	L	CL550 to memory	DMA write cycle
H	L	memory to CL550	DMA read cycle

Table 4-6. External Buffers Direction Control

4.5.1.4 System Status Signals

HALF_FULL

The HALF_FULL signal is an output that indicates the status of the internal FIFO. A value of 1 indicates that the FIFO contains at least 64 entries out of 128. Transitions of HALF_FULL are synchronous to PXCLK.

FRMEND#

The FRMEND# signal is an output that indicates that the end of an image has been reached. For compression, FRMEND# goes active when the Huffman coder has removed the last word from the FIFO. Note that there may be a delay before the last word can actually be removed from the CL550. For decompression, FRMEND# indicates when the last word is removed from the line buffer RAM. This signal can be disabled by setting bit 1 of the Configuration register to 0 (see Section 5). This signal is open collector and should be tied to VCC through a resistor of at least 625 ohms. Transitions of this signal are synchronous to HBCLK.

4.5.2 Data Transfer Examples

Transfer Modes

The host bus supports both 16-bit and 32-bit data transfers. Addresses can also be either 16 or 32 bits.

Data Transactions

The CL550 is capable of being a bus slave or cooperating with an external DMA controller to be a bus master. The host bus is transaction oriented, each transaction consisting of a start cycle and an acknowledge cycle. When the CL550 operates in bus slave mode, the bus master begins a transaction by asserting START# while driving TM[1] to indicate read/write and driving the address onto HBUS. Transactions are complete when the CL550 drives the TM signals to indicate status. For write transactions, the master is responsible for presenting the data on HBUS immediately following the start cycle until the acknowledge cycle. For read transactions, the CL550 drives the HBUS with the data during the Acknowledge cycle.

In bus master mode, the CL550 relies on an external DMA controller to supply an address to the slave.

Read Transactions (Bus Slave Mode)

The following steps are involved in a read transaction (see Figure 4-29):

1. The host places the address on HBUS and drives TM[1]# high and START# low. TM[2]# must also be at the high level at this time. In most applications, this can be accomplished by using a pull-up resistor.

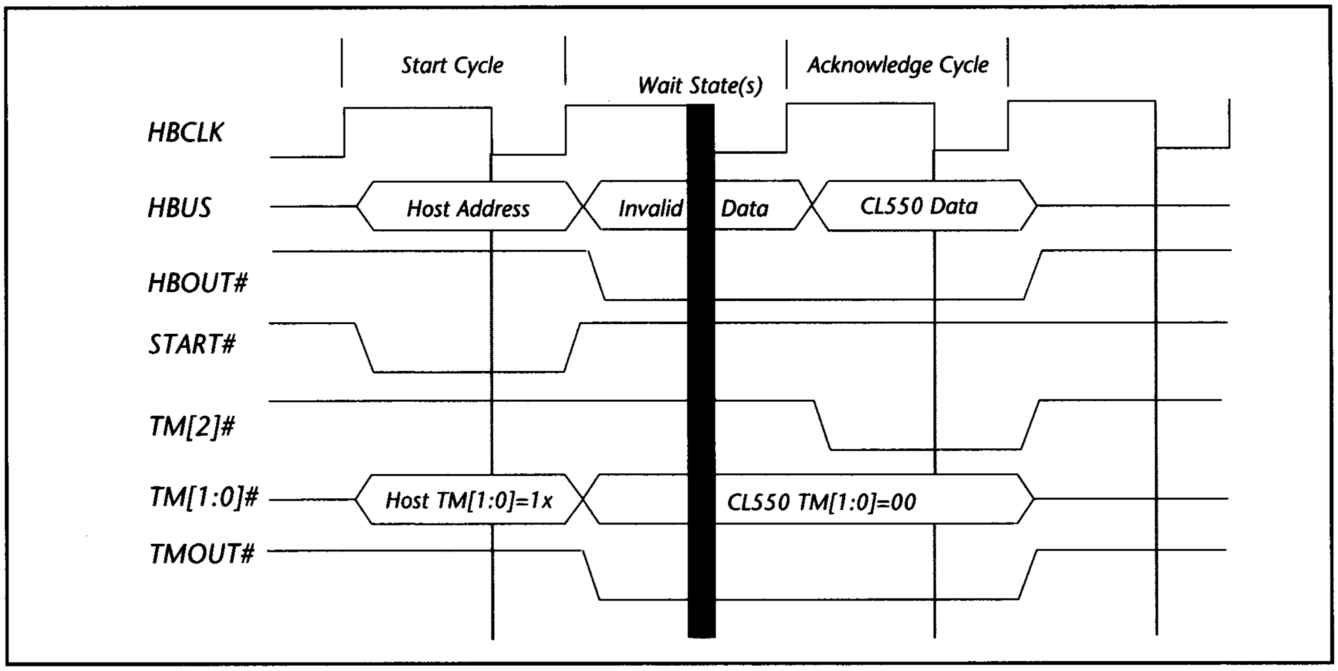


Figure 4-29. Read Transaction (Bus Slave Mode)

2. On the falling edge of HBCLK, the CL550 samples START#, TM[1]#, and TM[2]#. START# must not be held low for longer than one HBCLK cycle. If TM[2]# is not sampled high at this time, the CL550 will ignore START# and not return an acknowledge.

3. When the CL550 has data available, it places the data on the HBUS, the acknowledge on TM[2]#, and a result code on TM[1:0]#.

4. The host samples the data on the falling edge of HBCLK.

Wait states are implied between the start cycle and the acknowledge cycle. No bus control signals are driven during the wait states.

Write Transactions (Bus Slave Mode)

The following steps are involved in a write transaction (see Figure 4-30):

1. The host places the address on HBUS and drives TM[1]# low and START# low. TM[2]# must also be at the high level at this time. In most applications, this can be accomplished by using a pull-up resistor.

2. On the falling edge of HBCLK, the CL550 samples START#, TM[1]#, and TM[2]#. START# must not be held low for longer than one HBCLK cycle. If TM[2]# is not sampled high at this time, the CL550 will ignore START# and not return an acknowledge.

3. The host immediately places data on HBUS and waits for an acknowledge from the CL550.

4. When the CL550 is ready for data, it puts the acknowledge on TM[2]# and a result code on TM[1:0]#, and it samples the data on HBUS on the falling edge of HBCLK.

5. The host drives HBUS until the end of the acknowledge cycle.

Wait states are implied between the start cycle and the acknowledge cycle. Only HBUS is driven during the wait states.

Master Mode Transactions

The CL550 is capable of acting as a bus master with an external DMA controller (see Figure 4-31). The CL550 relies on the DMA controller to generate the start cycle, including providing the address and R/W signals to a bus slave. The CL550 then takes over as bus master, waiting for an acknowledge from the slave. To select master mode, the signal DMA_MSTR# must be asserted. In master mode, the CL550 does not assert any signals during the start cycle. The CL550 samples the R/W signal connected to TM[1]# to determine whether the cycle is a read cycle or a write cycle. Note that R/W is interpreted as the opposite of its normal meaning, that is, a read on TM[1]# indicates a read from the slave (which is a write to the CL550) while a write on TM[1]# indicates a write to the slave (which is a read from the CL550). See Table 4-6.

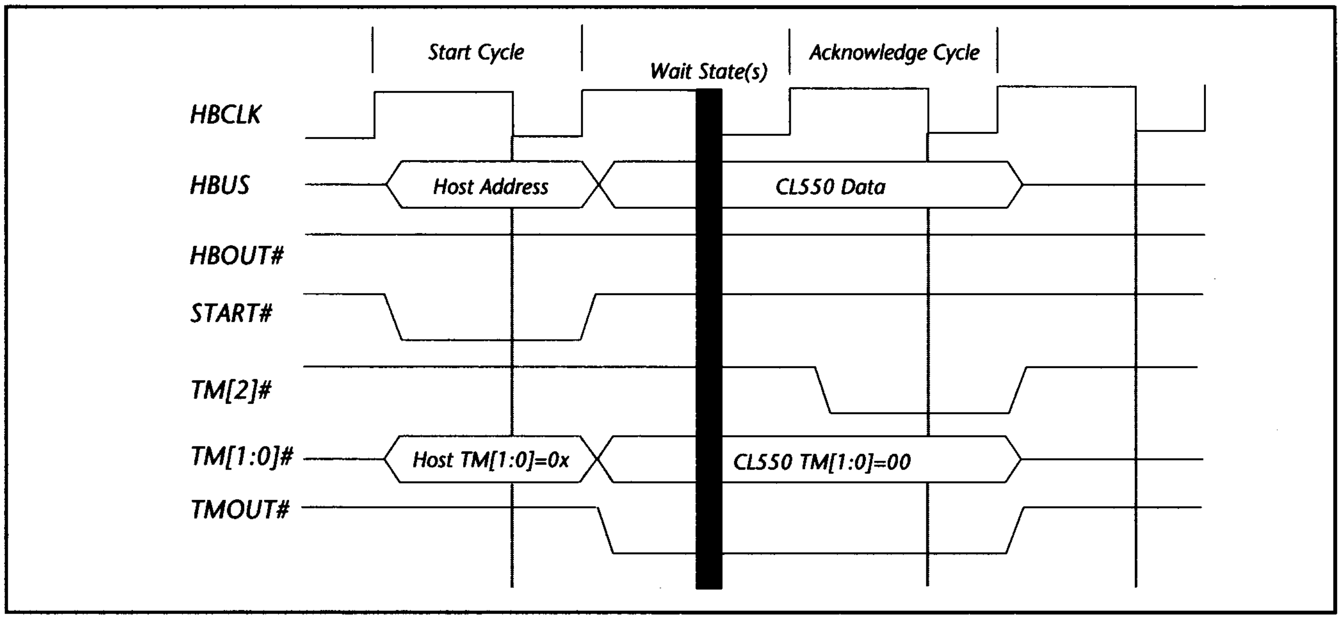


Figure 4-30. Write Transaction (Bus Slave Mode)

If the CL550 senses a read from the bus slave, it waits for an acknowledge from the slave, at which time it latches the data provided by the slave. If the CL550 senses a write to the bus slave, it provides the data on the cycle immediately following the start cycle and holds that data until it receives an acknowledge (TM[2]# asserted) from the slave. If more than one acknowledge occurs, a CL550 error condition is triggered, leading to incorrect coding/decoding. The system designer must make sure that, after a DMA_MSTR transfer is complete, the TM[2]# signal is suppressed until either:

1. The next DMA Master transfer occurs (START# and DMA_MSTR#) or
2. A CL550 access occurs (START# and valid CL550 address)

The CL550 also samples status bits during the acknowledge cycle to determine whether the Bus Error status flag (Buser) should be set.

All bus master transfers are to and from the CODEC register. The DMA_MSTR# signal must be asserted only when the CODEC register is prepared to accept or source data. This can be accomplished by asserting DMA_MSTR# in response to an appropriately programmed DMA_REQ# signal.

Bus Errors

The TM lines are used during the acknowledge cycle (TM[2]# asserted) to transmit transaction status information. In slave mode, the CL550 returns TM[1:0]# =00, that is, transaction complete. In master mode, the CL550 samples the TM[1:0]# lines and it sets a Bus Error flag if a bus error is indicated. The Bus Error flag status is determined as follows:

TM[1:0]#	TM[2]#	Operation
00	0	Transaction Complete
01	0	Bus Error
10	0	Bus Error
11	0	Bus Error

Table 4-7. Bus Error Conditions

Operational Considerations

Address Space When the START# signal is active, the address on HBUS is latched into the CL550 on the falling edge of HBCLK. The address is decoded to determine whether the cycle is intended for the CL550. When in 32-bit address mode, the chip responds to addresses of the form FS (10xx)0 AA AAh, where S is the value on the ID lines in the range from 1h through Eh and AA AA is the address of one of the chip registers. In 16-bit address mode (ID[3:0]# = 0), the chip decodes only AA AA to determine which register is being addressed. In 16-bit data mode, HBUS[31:16] remains disabled (tristated) at all times and all transfers take place over HBUS[15:0].

Wait States An access to the CODEC register inside the CL550 has a minimum of two HBCLK cycles: one for the address and one for data (zero wait state). All other registers within the CL550 are accessed in three cycles (one wait state). In worst-case conditions during an access to the CODEC register, the CL550 can insert wait states up to 70 pixel clocks in duration.To prevent this condition one can poll the Flags register or use the DRQ# signal to determine the CODEC state.

Drive Capability The HBUS signals do not have enough drive to meet the specifications of most system buses. Therefore, an external transceiver must be used to buffer HBUS from the system bus. HBOUT# is provided to facilitate this interface.

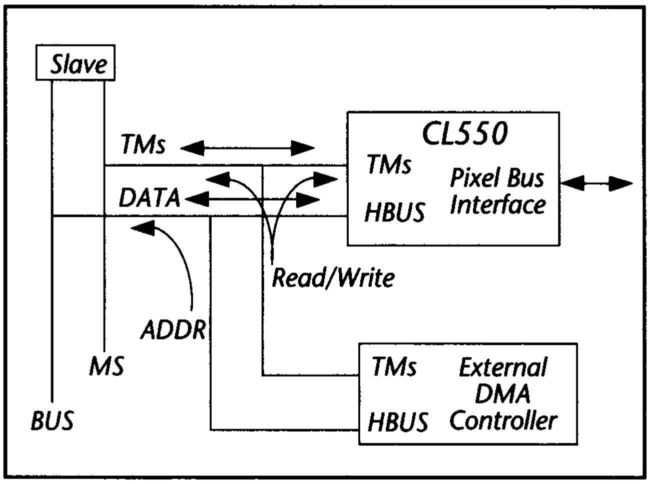


Figure 4-31. Master Mode Operation

Compression Mode When in compression mode, the CODEC register is read-only. If the host attempts to write to this register, the CL550 will not return an acknowledge on TM2 and the HBUS will remain in a locked state until a hard reset is issued. When in compression mode, the Huffman coder will not begin to operate until the FIFO reaches the 1/4 full mark. If the host attempts to read the CODEC register before the FIFO level reaches 1/4 full, the data stream will become corrupt.

Decompression Mode When in decompression mode, the CODEC register is write-only. If the host attempts to read from this register, the CL550 will not return an acknowledge on TM2 and the HBUS will remain in a locked state until a hard reset is issued. When in decompression mode, if the FIFO is full and the host attempts to write data to the CODEC register, the acknowledge on TM2 will be delayed until FIFO is not full (several hundred HBCLK cycles). Therefore, the host must never fill the FIFO past 3/4 full. When in decompression mode, the CL550 detects a marker code (a hex value FFxx) in the compressed data, the decoder will stop processing and the "mark" bit in the Fs register will be set. If the host attempts to write to the CODEC register before writing either a 0 or 1 to the Decoder Resume register, the CL550 will not return an acknowledge on TM2 and the HBUS will remain in a locked state until a hard reset is issued. Normally, the only marker codes that are allowed within the JPEG data scan field are the RST markers (FFD0 through FFD7). These markers are automatically detected and stripped off by the CL550 with no external intervention required.

Handshake When performing any access to the CL550 host bus, the TM2 line must be at the logic one level during the assertion of START#. If it is not, the CL550 will not recognize the access and no acknowledge will be given on TM2. A pull-up resistor on TM2 could be used for this purpose.

DMA_MSTR Mode The DMA_MSTR input on the CL550 allows the CL550 to behave as a bus master for CODEC transfers. When a DMA_MSTR transfer occurs, ACK from the addressed slave terminates the transfer. However, a subsequent ACK from a non-CL550 access (a "foreign ACK") will cause the CODEC to malfunction. The workaround is to prevent foreign ACKs from reaching the CL550 after the DMA_MSTR transfer completes. Once the DMA_MSTR transfer ACK occurs, the ACK signal to the CL550 must be suppressed until one of the following:

1. the next DMA_MSTR transfer (START and DMA_MSTR);

2. a valid CL550 access (START and valid CL550 Address)

This workaround applies only if other devices besides the CL550 and CL550 DMA slaves can drive ACK.

4.5.3 Interface Examples

Figure 4-32 shows an interface example between the CL550 host bus and the NuBus.

The CL550 can be interfaced to multiplexed or nonmultiplexed buses. Figure 4-33 shows the interface to a nonmultiplexed PC-type bus. The timing diagram in Figure 4-34 illustrates the bus transactions.

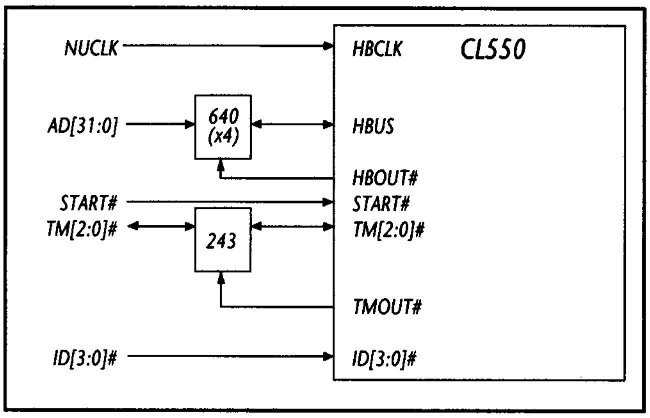


Figure 4-32. CL550 to NuBus Interface Example

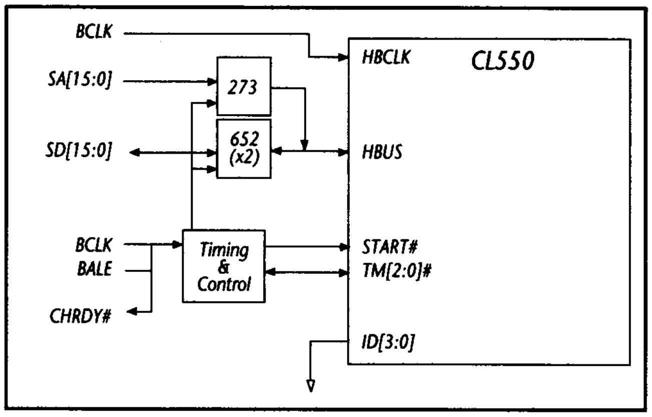


Figure 4-33. CL550 to PC-AT Bus Interface

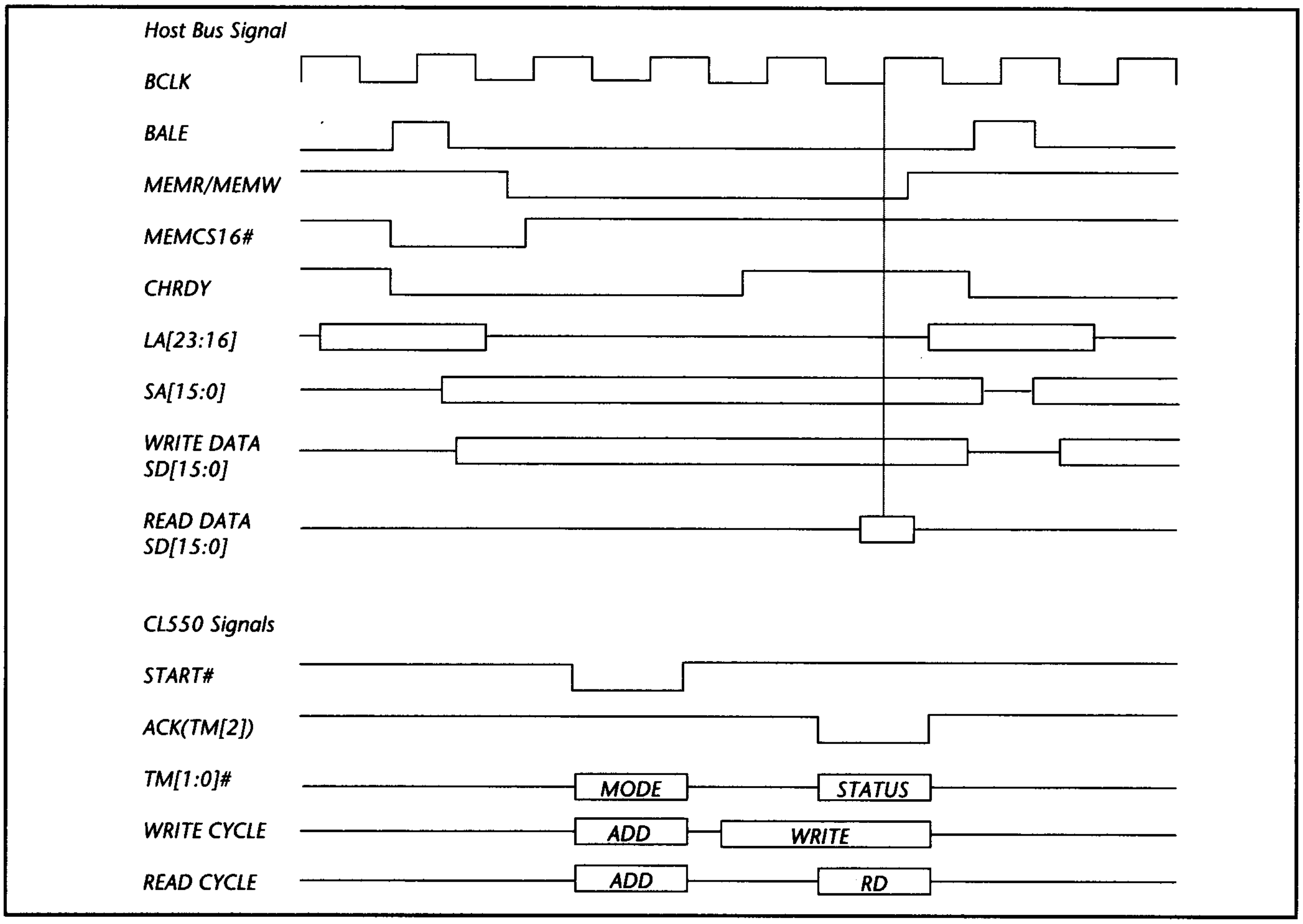


Figure 4-34. PC-AT Interface Timing Diagram

4.5.4 HBI AC Timing Characteristics

			PGA				MQUAD					
			CL550-35		CL550-30		CL550-30		CL550-10			
Time	Fig.	Description	Min	Max	Min	Max	Min	Max	Min	Max	Units	Note
T1	4-35	HBCLK Clock Period	84		100		100		100		ns	1
T2	4-35	HBCLK Pulse Width HIGH	50		50		50		50		ns	1
T3	4-35	HBCLK Pulse Width LOW	23		23		23		30(23)		ns	1,13
T4	4-35	RESET Setup Time	10		10		10		15		ns	1,2,3
T5	4-35	RESET Pulse Width LOW	170		200		200		200		ns	1,2
T6	4-36	START Setup Time	8		10		10		10		ns	1,4,5
T7	4-36	START Hold Time	12		15		15		15		ns	1
T8	4-36	TM2 Start Cycle Setup Time	8		10		10		10		ns	1,6
T9	4-36	TM2 Start Cycle Hold Time	12		15		15		15		ns	1
T10	4-36	TM2 Delay Hi-Z to HIGH		18		18		18		20	ns	1
T11	4-36	TM2 Delay to LOW		18		18		18		20	ns	1
T12	4-36	TM2 Delay Hold Time	5		5		5		4		ns	1,12
T13	4-36	TM2 Delay to Hi-Z		18		18		18		20	ns	1,12
T14	4-36	DMA_MSTR Setup Time	8		10		10		10		ns	1
T15	4-36	DMA_MSTR Hold Time	12		15		15		15		ns	1
T16	4-36	HBUS [31:0] Address Setup Time	8		10		10		10		ns	1
T17	4-36	HBUS [31:0] Address Hold Time	12		15		15		15		ns	1
T18	4-36	HBUS [31:0] Data Setup Time	8		10		10		10		ns	1
T19	4-36	HBUS [31:0] Data Hold Time	12		15		15		15		ns	1
T20	4-36	ID Setup Time	8		10		10		10		ns	1
T21	4-36	ID Hold Time	12		15		15		15		ns	1
T22	4-36	TM0, TM1 Start Cycle Setup Time	8		10		10		10		ns	1
T23	4-36	TM0, TM1 Start Cycle Hold Time	12		15		15		15		ns	1
T24	4-36	TM0, TM1 Hi-Z Hold Time	5		5		5		5		ns	1,12
T25	4-36	TM0, TM1 Hi-Z to LOW Delay		18		18		18		20	ns	1
T26	4-36	TM0, TM1 Delay Hold Time	5		5		5		5		ns	1,12
T27	4-36	TM0, TM1 Delay to Hi-Z		18		18		18		20	ns	1,12
T28	4-36	TMOUT Delay to LOW		22		22		22		26	ns	1,7
T29	4-36	TMOUT Delay to HIGH		22		22		22		26	ns	1
T30	4-41	NMRQ Hi-Z to LOW Delay		18		18		20		25	ns	1,8,9,11
T31	4-41	NMRQ LOW to Hi-Z Delay		18		18		20		25	ns	1,12
T32	4-41	NMRQ Delay Hold Time	3		3		3		3		ns	1,12
T33	4-40	DRQ Hi-Z to LOW Delay		18		18		20		25	ns	1
T34	4-38	DRQ LOW to Hi-Z Delay		18		18		20		25	ns	1,12
T35	4-38	DRQ Delay Hold Time	3		3		3		3		ns	1,12
T36	4-42	FRMEND Hi-Z to LOW Delay		23		25		25		28	ns	1
T37	4-42	FRMEND LOW to Hi-Z Delay		23		25		25		28	ns	1,12
T38	4-42	FRMEND Delay Hold Time	5		5		5		3		ns	1,12
T39	4-37	HBUS [31:0] Hi-Z Hold Time	2		2		2		2		ns	1,12
T40	4-37	HBUS [31:0] Delay Time		22		23		23		26	ns	1
T41	4-37	HBUS [31:0] Delay to Hi-Z		22		23		23		26	ns	1,12
T42	4-37	HBUS [31:0] Delay Hold Time	5		5		5		3		ns	1,12
T43	4-38	HBOUT Delay to LOW		22		22		22		26	ns	1
T44	4-38	HBOUT Delay to HIGH		22		22		22		26	ns	1,10
T45	4-38	TM2 Acknowledge Cycle Setup	8		10		10		10		ns	1
T46	4-38	TM2 Acknowledge Cycle Hold	5		5		5		5		ns	1,5
T47	4-38	TM0, TM1 Status Cycle Setup	8		10		10		10		ns	1
T48	4-38	TM0, TM1 Status Cycle Hold	12		15		15		15		ns	1
T88	4-42	Half-Full Delay		20		22		22		25	ns	1

Table 4-8. Host Bus Timing Table

Host Bus Timing Notes

1. Inputs switch between 0 and 3.5 V at 1 V/ns, and measurements are made at 1.5 V. CL = 50 pf.
2. RESET# is immediately seen when going low but is removed on positive edge of HBCLK.
3. Two HBCLK cycles are required for internal reset release.
4. START# should remain HIGH until transaction is complete.
5. A valid start cycle implies START# low, TM2# high, and the CS fields of HBUS [31:0] match the ID inputs.
6. TM2 is sometimes called ACK_ in design documentation.Reference part number CL550-10x for NuBus specifications
7. TMOUT# is direction control for I/O bus; when 1, TM2,1,0 are expected to be inputs.
8. NMRQ#, DRQ#, and FRMEND# are all open-collector signals.
9. NMRQ#, DRQ#, and FRMEND# change on positive edge of HBCLK and are not related to any specific transaction phase.
10. HOSTOUT is bidirectional control for the I/O HOSTBUS; when 1, HOSTBUS is expected to be inputs.
11. If DRQ# is LOW during a start cycle, it will go to Hi-Z on next positive edge of HBCLK, if the access is to the CODEC register.
12. Not 100% tested. Guaranteed by design characterization.
13. Characteristics in parentheses apply to part number CL550-10N, required for NuBus applications.

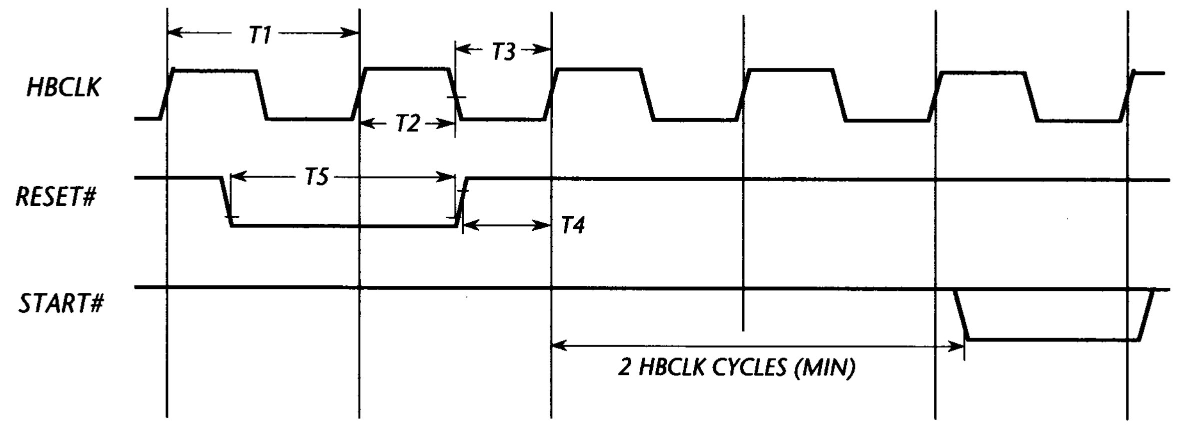


Figure 4-35. Host Bus Reset Timing Diagram

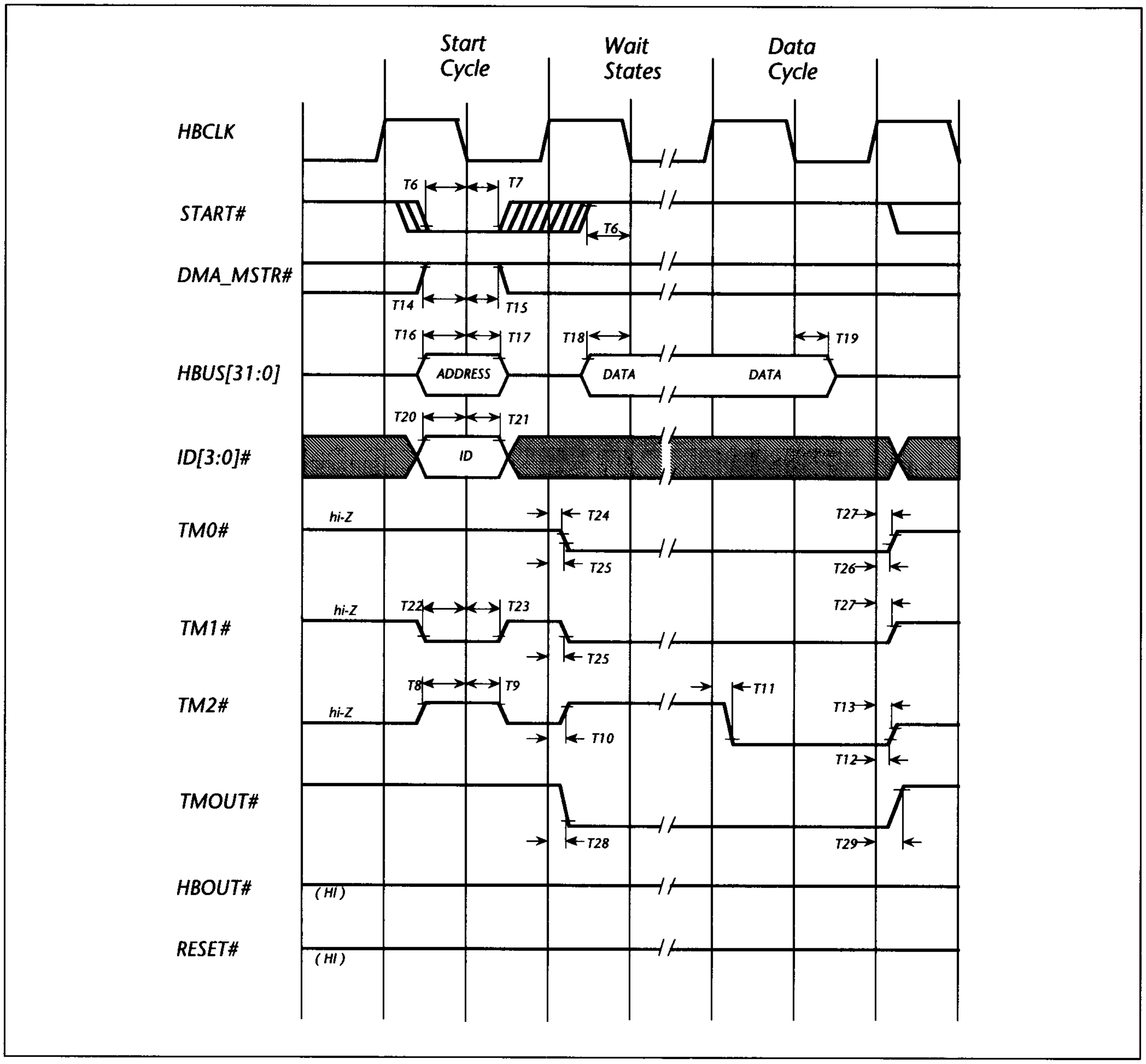


Figure 4-36. Host Bus Write Timing, Slave Mode

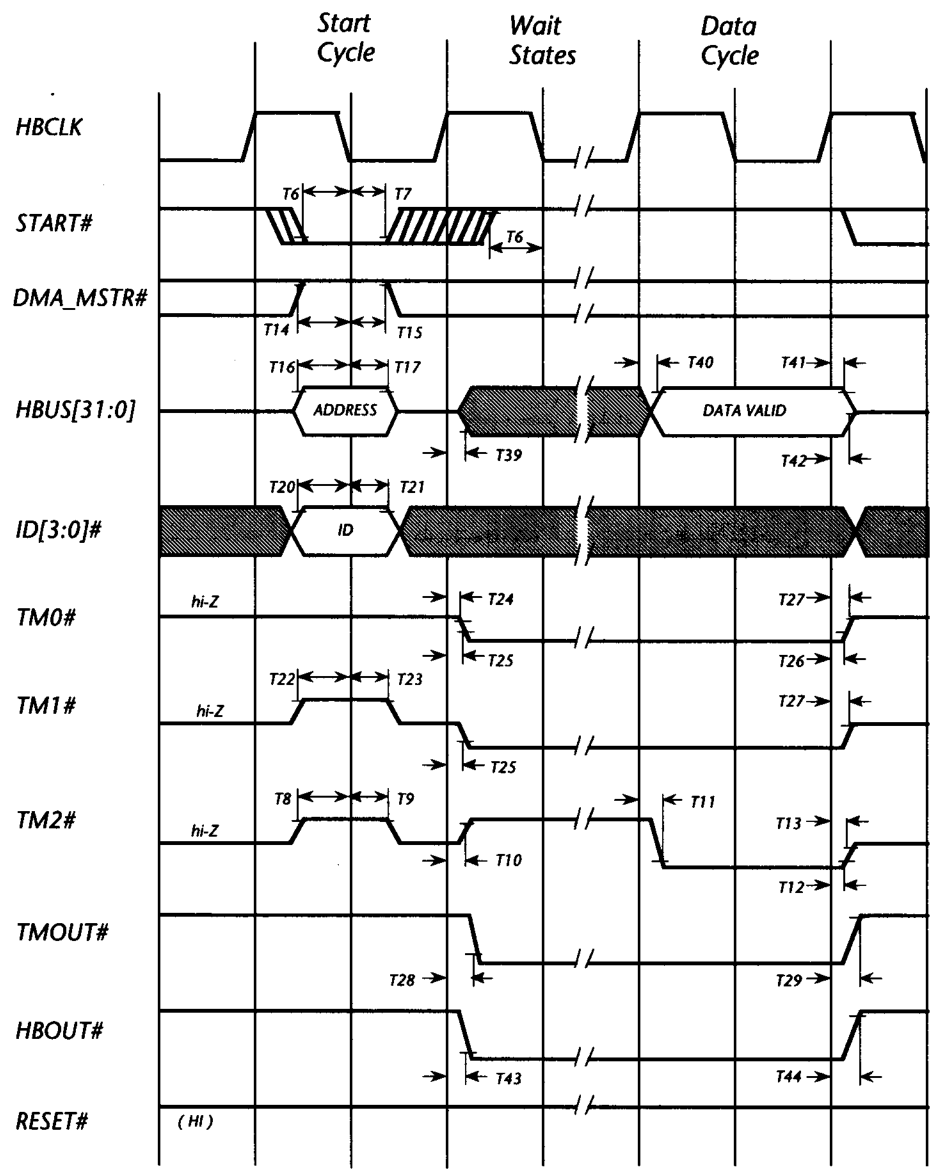


Figure 4-37. Host Bus Read Timing, Slave Mode

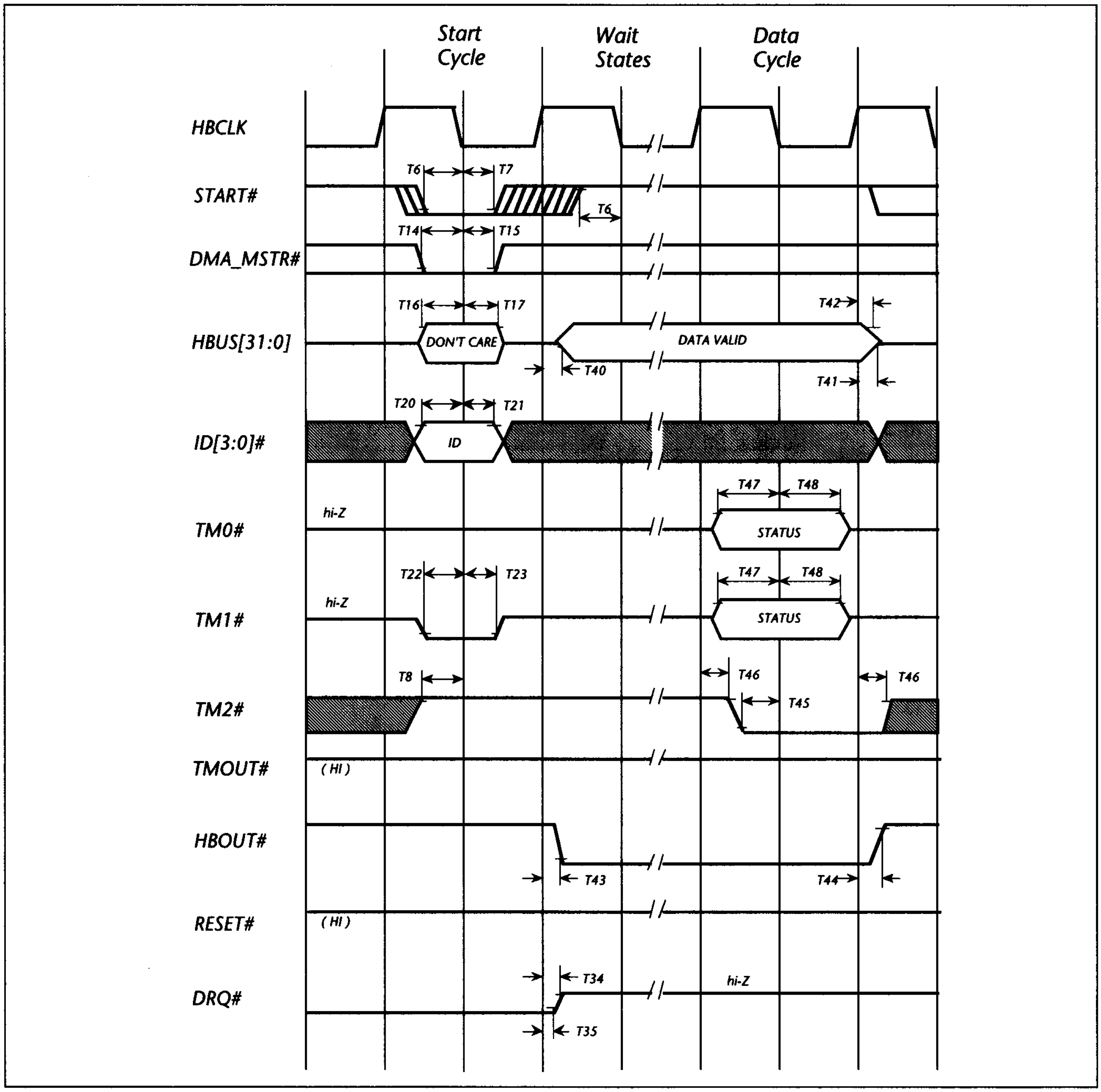


Figure 4-38. Host Bus Write Timing, DMA Master Mode (CL550-to-Memory DMA Transfer)

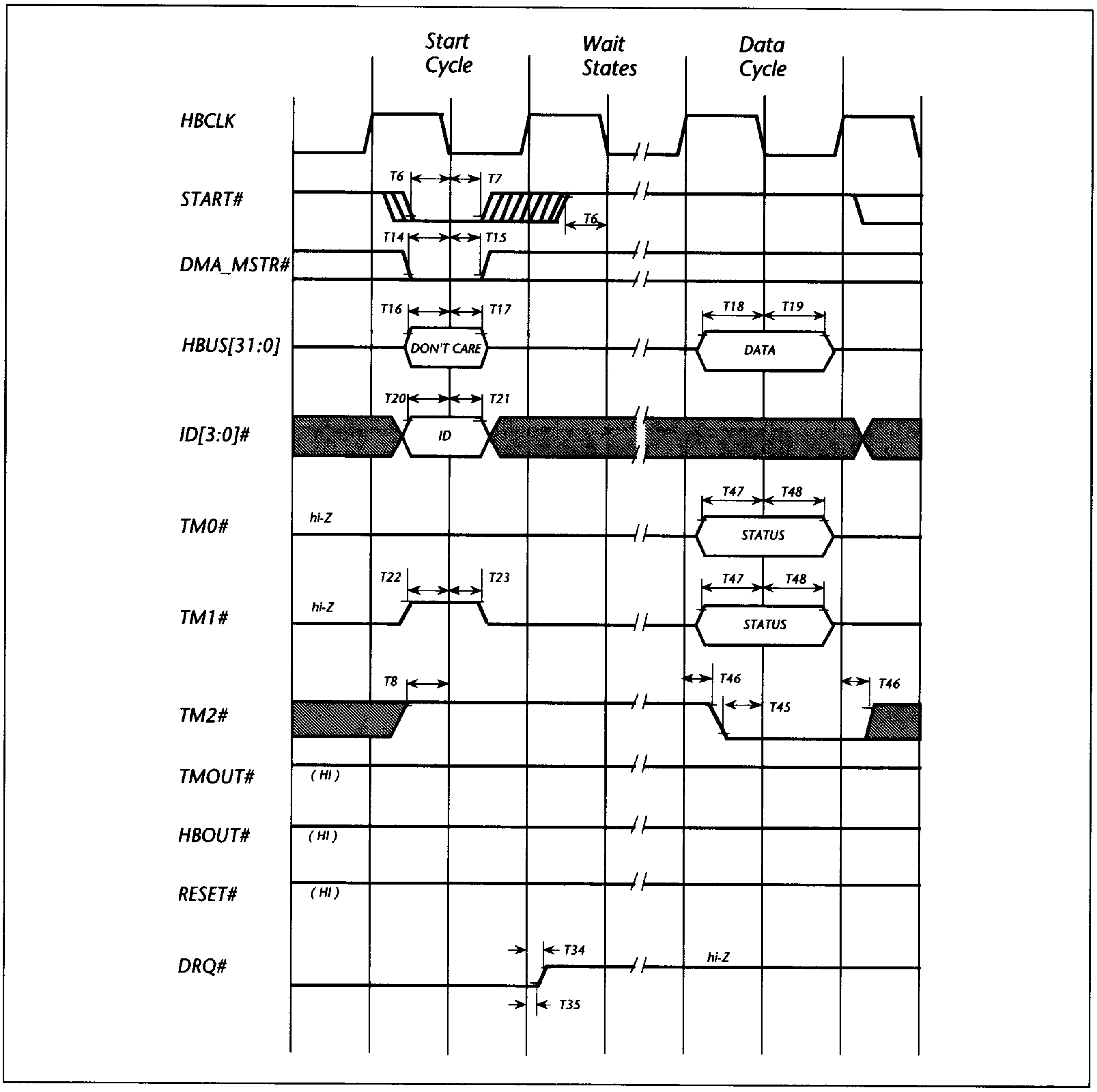


Figure 4-39. Host Bus Read Timing, DMA Master Mode (Memory-to-CL550 DMA transfer)

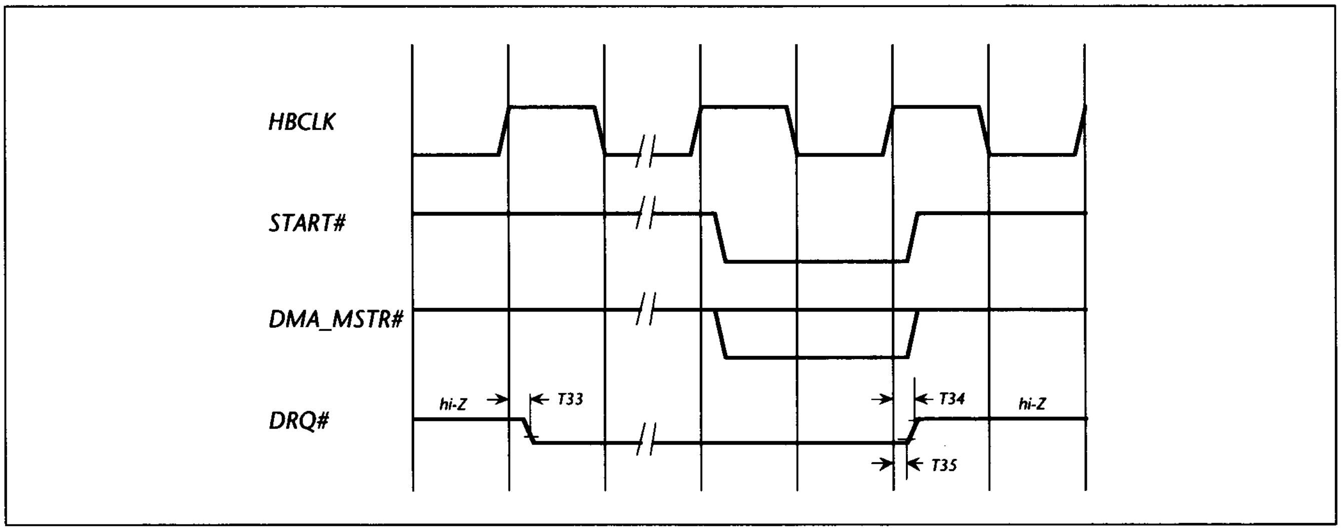


Figure 4-40. DRQ# Timing

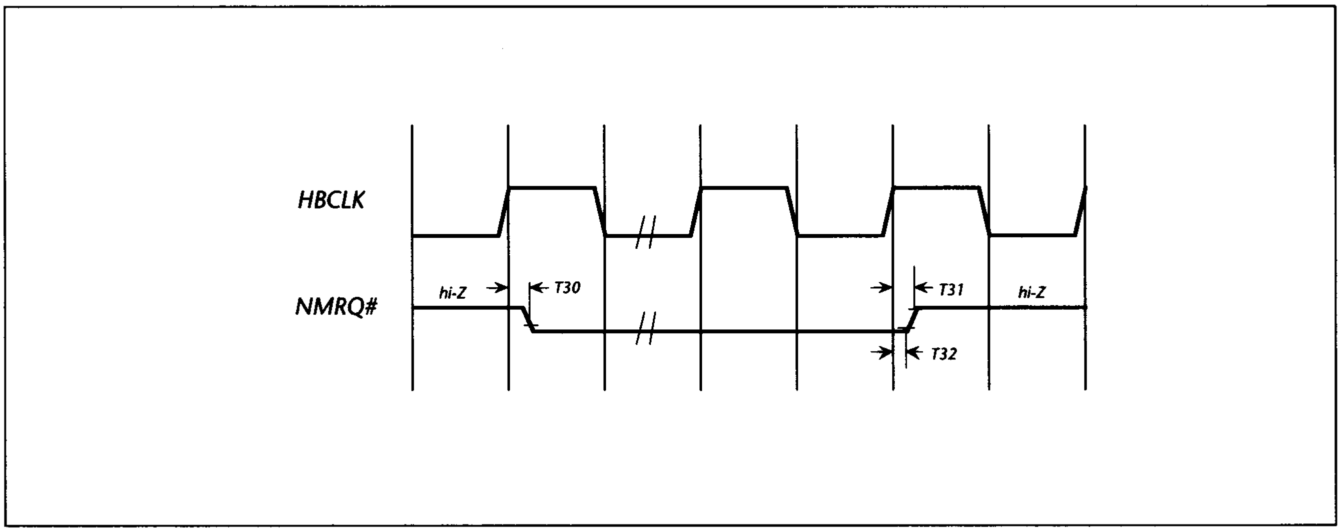


Figure 4-41. NMRQ# Timing

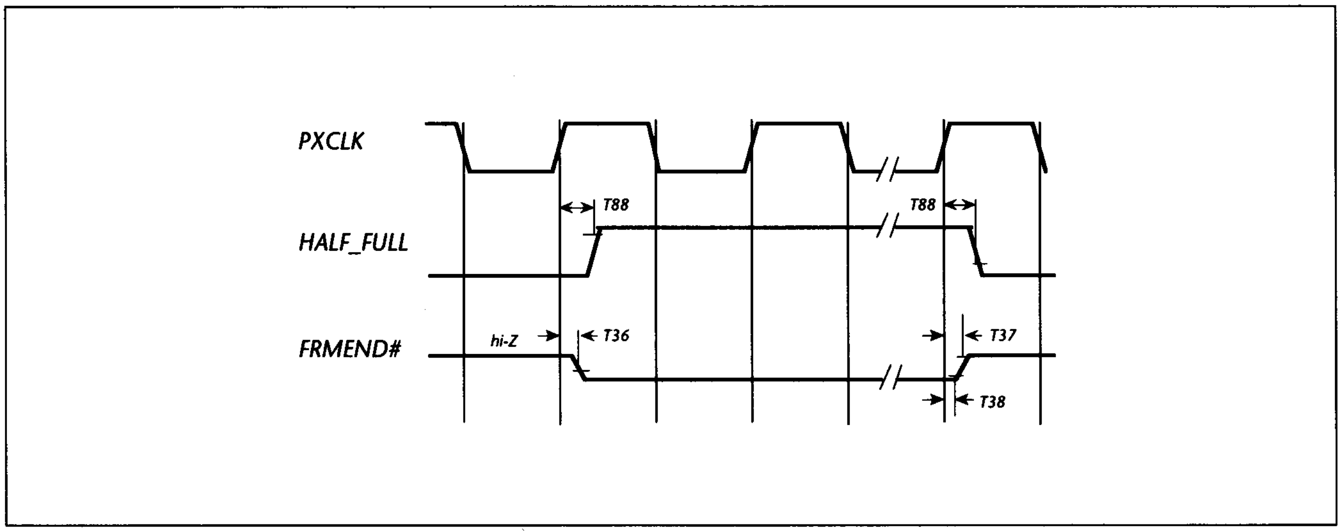


Figure 4-42. HALF_FULL, FRMEND# Timing

This Page Intentionally Left Blank

5.0 PROGRAMMER'S GUIDE

This section provides the information necessary to program the C-Cube CL550® processor. It includes:

- Description of the CL550 internal registers and tables
- Description of CL550 initialization routines (offered free of charge to CL550 customers)
- Initialization values examples
- CL550 compression/decompression control inner loop flow chart examples
- JPEG data structure and signaling parameters overview

5.1 C-Cube CL550 Registers

The CL550 data path is configured via a set of control registers. There are groups of registers and tables associated with each stage of the JPEG pipeline that control various aspects of the JPEG process.

The first group of registers is associated with the Pixel Bus Interface (PBI). The PBI registers are used to describe the horizontal and vertical image frame and window parameters. Tables used in the color transformation process are also included in this block.

The second group of registers and tables controls the operation of the JPEG Compression/Decompression Unit (CDU). Tables in this block include DCT transform coefficients, up to four quantizer tables, and Huffman code tables. Sequence registers are used to select these tables based on the input data sequence. In compression mode, for example, each incoming 8x8 pixel block is made up of 8x8 component subblocks (which can be either R, G, and B or Y, U, and V). The processing order for these blocks depends on the mode of operation. For 4:2:2 mode compression, these subblocks are ordered Y-Y-U-V-Y-Y-U-V. The sequence registers determine which of the tables are in use on a block-by-block basis.

Other registers in this block are used to specify the CL550's general operating parameters and to reset and start/stop the device. The Compression Monitor software, provided with the CL550, will load each of the registers and tables for the various modes of operation.

5.1.1 Pixel Bus Interface Unit Registers

HPeriod Register

Register Type: R/W
Address: 8000_h
Size: 14 bits

The HPeriod register serves two different functions depending on the mode (master or slave) the CL550 is operating in. In master mode it controls the number of pixels between consecutive HSYNC#s. HPeriod should be set to the number of pixels, minus 1, contained in one horizontal screen line.

In slave mode, no HSYNC# inputs will be acknowledged during the time interval starting at the falling edge of HSYNC# and lasting until a number of pixels, equal to the value stored in the HPeriod register, have been received by the processor. A typical value stored in the HPeriod register is the larger of the HDelay value and 90% of the number of pixels contained in one horizontal line.

In master mode, the following formulas will be used.

NumPixel = Number of pixels between HSYNC# assertions

Mode	HPeriod Value
Single Component	NumPixel / 2 - 1
RGB to YUV 4:2:2 YUV 4:2:2 4:4:4 to 4:2:2	NumPixel - 1
4:4:4 4:4:4:4 CMYK	2 * NumPixel - 1

HSync Register

Register Type: R/W
Address: 8004_h
Size: 14 bits

The HSync register performs two different functions depending on the mode (master or slave) the CL550 is operating in. In master mode, it is used to control the width of the HSYNC# pulse. HSync should be set to the number of pixels contained in one horizontal sync pulse width, minus 1.

In slave mode, HSync is not used and its value is insignificant.

HDelay Register

Register Type: R/W
Address: 8008_h
Size: 14 bits

The HDelay register controls the delay from the falling edge of HSYNC# to the first active pixel. The value of HDelay depends on the video mode.

Note: If HDelay = 0, the first HSync will not occur until one complete line after the first VSync.

Pixel Delay = number of pixels from falling edge of HSYNC# to the first active video pixel

Mode	HDelay Value
Single Component	1/2 * Pixel Delay
RGB to YUV 4:2:2 YUV 4:2:2 4:4:4 to 4:2:2	Pixel Delay
4:4:4 4:4:4:4 CMYK	2 * Pixel Delay

HActive Register

Register Type: R/W
Address: $800C_h$
Size: 12 bits

The HActive register controls the size of the active horizontal line. The number of pixels on a horizontal line must be a multiple of the minimum block size. In 4:4:4 modes, the minimum block width is 8 pixels, while in 4:2:2 modes it is 16 pixels (two 8x8 blocks).

NumActiveBlocks = (Number of active pixels in a horizontal line)/8

Mode	HActive Value
Single Component	NumActiveBlocks - 1
RGB to YUV 4:2:2 YUV 4:2:2 4:4:4 to 4:2:2	2 * NumActiveBlocks - 1
4:4:4 4:4:4:4 CMYK	4 * NumActiveBlocks - 1

Note: NumActiveBlocks must be a multiple of 2 for the 4:2:2 modes.

VPeriod Register

Register Type: R/W
Address: 8010_h
Size: 14 bits

In master mode, the VPeriod register is used to control the number of lines in a frame. It should be set to the number of lines between two consecutive falling edges of VSYNC#. In slave mode, VPeriod is ignored.

VSync Register

Register Type: R/W
Address: 8014_h
Size: 11 bits

In master mode the VSync register controls the width of the vertical sync pulse. It should be set to the vertical sync pulse width in lines.

In slave mode, VSync is not used and its value is insignificant.

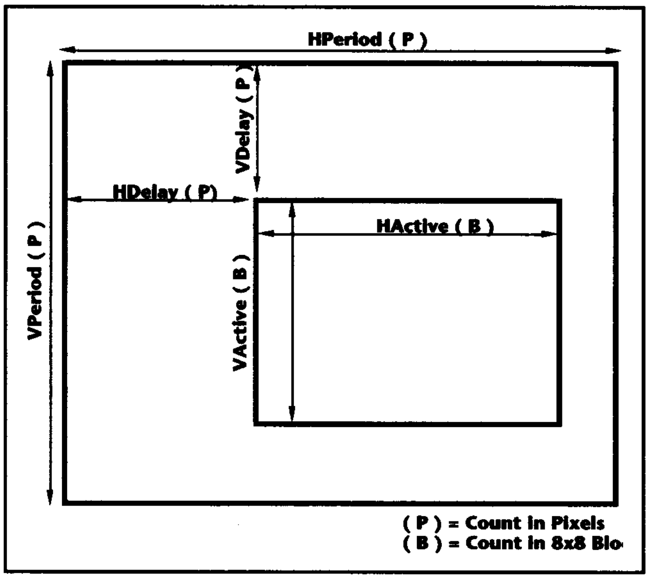


Figure 5-1. Video Field Registers

VDelay Register

Register Type: R/W
Address: 8018_h
Size: 14 bits

The VDelay register controls the vertical delay for the video compression/decompression process. For compression operations, the value should be set to:

VDelay Register Value = Vertical Delay = number of lines (HSYNC# pulses) from the falling edge of VSync to the first active video line.

For decompression operations, the value depends on the video mode and the horizontal video period. The minimum value for VDelay is:

Minimum VDelay = 9 + trunc[(Video Latency + HActive Clocks) / HPeriod Clocks]

Where: HActive Clocks = 2 * HActive * 8
HPeriod Clocks = 2 * HPeriod

The value that is loaded into the VDelay register is:

VDelay Register Value = Actual VDelay-Minimum VDelay

Note: The minimum Vertical Delay is 9. This minimum is due to the fact that the strip buffer is filled 8 lines before the start of the active region and there is at least one line of latency internal to the CL550. For horizontal periods less than 200 pixels, internal pipeline latencies may be greater than one line and the minimum Vertical Delay is adjusted upwards.

VActive Register

Register Type: R/W
Address: $801C_h$
Size: 11 bits

The VActive register controls the number of active lines. It should be set to the vertical block count of the active target window, that is, one eighth the number of active lines.

Video Latency Register

Register Type: R/W
Address: $C030_h$
Size: 14 bits

The Video Latency register controls internal pipelining timings and depends on the video mode. The values to be loaded into the Video Latency register are provided in the initialization table in Appendix A.

Vertical Line Count Register

Register Type: R
Address: $C03C_h$
Size: 14 bits

The Vertical Line Count register contains the vertical line count of the active window. This register can be read from the host bus by external devices that need to know the vertical line count.

HControl Register

Register Type: R/W
Address: $C034_h$
Size: 14 bits

HControl is a decompression parameter that determines the horizontal position at which the CL550 should stop removing data for the current frame from the internal FIFO. If the CL550 is being reset between frames, as in most still and some video systems, the HControl value should be set to 0x3FFFh. The equations for computing HControl for other systems are provided in the Compression Monitor. The HControl register is ignored in compression mode.

VControl Register

Register Type: R/W
Address: $C038_h$
Size: 14 bits

VControl is a decompression parameter that determines the vertical position at which the CL550 should stop removing data for the current frame from the internal FIFO. If the CL550 is being reset between frames as in most still and some video systems, the VControl value should be set to 0x3FFFh. The equations for computing VControl for other systems are provided in the Compression Monitor. The VControl register is ignored in compression mode.

Note: The functions of the video field registers, described above, are illustrated in Figure 5-1.

HV Enable Register

Register Type: R/W
Address: 9010_h
Size: 1 bit

The HV Enable register inhibits updates to the eight video parameter registers and the Start register during target window resizing. This function prevents partial updates of the video parameter registers if a VSYNC# occurs during the update. There are two sets of video parameter registers: a shadow register set that is written from the host bus, and an active register set that is used for the current frame. The shadow registers can always be updated by the host bus; however, the active register set can be updated only on the falling edge of VSYNC#.

When set to 0, HV Enable prevents updates of the active registers. When set to 1, the registers are updated on the falling edge of VSYNC# (but only if the Start register is set to 1 as well).

Important note: HV Enable must be 1 during the falling edge of VSYNC# in order for the Start register to function, both for starting and stopping the compression/decompression process.

Note: The horizontal and vertical video registers can be read back only after they have been loaded into the active registers.

Color Transformation Matrix

Register Type: W
Address: $C000\text{-}C020_h$
Size: 12 bits

There are nine 12-bit registers for loading the coefficients of the RGB-to-YUV transformation matrix. The nine register addresses are provided in Table 5-1.

Mij and M'ij (with i indicating the row number and j the column number) are the coefficients of the M and M' matrices for the RGB-to-YUV color space transformations:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = M \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = M' \times \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

Where M and M' are:

Address	Compression Mode	Decompression Mode
C000h	M_{11} (0133_h)	M_{11}' (0400_h)
C004h	M_{12} (0259_h)	M_{12}' (0400_h)
C008h	M_{13} (0074_h)	M_{13}' (0400_h)
C00Ch	M_{21} ($0F54_h$)	M_{21}' ($0FFE_h$)
C010h	M_{22} ($0EAD_h$)	M_{22}' ($0EA2_h$)
C014h	M_{23} ($01FF_h$)	M_{23}' ($071B_h$)
C018h	M_{31} ($01FF_h$)	M_{31}' ($059C_h$)
C01Ch	M_{32} ($0E53_h$)	M_{32}' ($0D23_h$)
C020h	M_{33} ($0FAE_h$)	M_{33}' ($0FFD_h$)

Table 5-1. Color Transformation Coefficients Addresses

These matrices can be used to implement a user defined color space conversion with the following two constraints:

1. The matrices should be approximate inverses of one another in order to minimize round-trip errors.

2. The forward matrix should produce pixel values in the range of [0,255.37] for unsigned values or [-128,127.37] for signed values.

Default values for the M and M' coefficients are provided in parentheses in Table 5-1.

5.1.2 Compression/Decompression Unit Tables and Registers

DCT Tables

Register Type: W
Address: 8800-881C$_h$
Size: 16 bits

The DCT table consists of eight 16-bit entries. The HBI can write directly to each entry. DCT values are used as multiplication coefficients for the DCT/IDCT operation during the video compression/decompression process. The values loaded into the DC tables depend on whether the CL550 is in compression or decompression mode and are provided in the initialization tables in Appendix A.

Quantizer Tables

Register Type: R/W
Address: B800-BBFC$_h$
Size: 16 bits

The Quantizer tables consist of two sets of 128 16-bit entries that the HBI can read or write directly. These values are used during the video compression/decompresion process. Table entries can be used in double buffer mode or four-table mode. When the CL550 operates in double-buffer mode, one set of tables can be loaded while the other set is active.

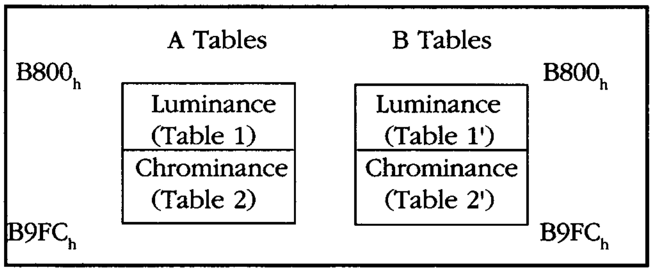


Figure 5-2. Quantizer Tables Configuration (Double-Buffer Mode)

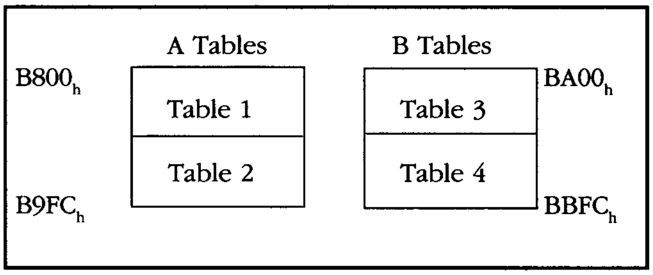


Figure 5-3. Quantizer Tables Configuration (Four-Table Mode)

The four-table mode is used only in 4:4:4 and 4:4:4:4 operating mode. Default Quantizer table values are provided in the Compression Monitor.

Note that there is a pipeline register between the Quantizer tables and the host bus. When reading data from the Quantizer tables, two consecutive reads must be performed for the first valid data to be presented to the host bus.

Quantizer A/B Table Select Register

Register Type: W
Address: BC00$_h$
Size: 1 bit

The value in the Quantizer A/B Table Select register selects the active Quantizer tables. When it is set to 1, the A tables are selected and used by the Quantizer. Selected tables cannot be loaded. When the CL550 is reset, the Quantizer A/B Table Select register points to the B tables, allowing the A tables to be loaded. When the CL550 operates in four-table mode, this register is not used.

Once the A tables are loaded, the Quantization A/B Table Select register must be set to 1 to point to the A tables for compression. However, the internal switch which controls the selection of the A or B tables will not toggle until the occurrence of VSYNC. Therefore, the B tables cannot be loaded until after Start = 1 and VSYNC falling edge.

There is another way, however, to load the B tables while the CL550 is not in active operation. Quantizer Sync register (address BE00) bits 8 and 9 control an internal test sync signal which, when set to 1 then 0, will toggle the table select switch in the same fashion as VSYNC.

Quantizer Y/C Table Sequence Register

Register Type: W
Address: BE08$_h$
Size: 14 bits

Both the Quantizer Y/C Table Sequence register and the Quantizer A/B Table Sequence register allow the Quantizer to select the proper quantization table in the compression and decompression process. The value to be loaded into this register depends on the operating mode (compression/decompression) and data formats being processed. Values for different operating modes and data formats are provided in the initialization tables in Appendix A.

Quantizer A/B Table Sequence Register

Register Type: W
Address: BE0C$_h$
Size: 10 bits

The value in the Quantizer A/B Table Sequence register determines which tables (A or B) are active. This register is used only when the CL550 operates in 4:4:4:4 and 4:4:4 mode. Register values for different operating modes and data formats are provided in the initialization tables in Appendix A.

Quantizer Sync Register

Register Type: W
Address: BE00$_h$
Size: 14 bits

The Quantizer Sync register supports 4:4:4:4 mode (four-table mode) and data synchronization.

Bits 13-11: Must be set to zero in normal operation.

Bit 10: This bit selects double-buffer versus four-table mode. When set to 1, the CL550 operates in four-table mode.

Bits 9-7: Must be set to zero in normal operation.

Bits 5-0: Data sync field. The proper value for this field is provided in the initialization tables in Appendix A.

Coder/Decoder DPCM Registers Sequence Registers

Register Type: W
Address: A004$_h$ (RH) and A008$_h$ (RL)
Size: 10 bits each

The CL550 has two registers (RH and RL) to select the active DPCM registers.

There are four DPCM registers specified by a two-bit address. Register values for different operating formats are provided in the Compression Monitor.

Decoder DPCM Reset Register

Register Type:W
Address: A818$_h$
Size: 1 bit

By writing to the Decoder DPCM Reset register, the host can reset the four DPCM registers. In decompression mode, after the decoder has flagged the EOI (End of Image) marker code, the host should reset the DPCM registers before restarting the decoder.

Huffman Code Tables

Register Type: R/W
Address: E000-FC7C$_h$
Size: 9 bits

There are two Huffman table layouts: one for coding and another for decoding. The default Huffman code tables for the baseline JPEG algorithm are defined in the JPEG Draft Proposal. The tables loaded into the CL550 are generated by processing the JPEG code table with the CL550 initialization software. Default Huffman table values are provided in the Compression Monitor.

Note: Unused Huffman table entries must be initialized to zero.

Huffman Table Load Enable Register

Register Type: W
Address: 9004$_h$
Size: 1 bit

The one-bit Huffman Table Load Enable register must be asserted before the Huffman code look-up table can be loaded. In normal compression and decompression operations, this bit must be deasserted.

Huffman Table Sequence Register

Register Type: W
Address: A000$_h$
Size: 10 bits

The Huffman Table Sequence register specifies the sequence in which the Huffman luminance (Y) and chrominance (C) tables are used by the coder/decoder. Register values for different operating modes and formats are provided in the initialization tables in Appendix A.

Coder Attributes Register

Register Type: W
Address: A00C$_h$
Size: 7 bits

The contents of the Coder Attributes register affect the coder function as follows:

Bit 6, EOB: When EOB is set to 1, an EOB code is inserted at the end of every block. When EOB is set to 0, an EOB is inserted only when the last coefficient in a block is equal to zero.

Bit 5, LSB: When LSB is set to 1, the coder outputs data with the least significant bit first.

Bit 4, RST-EN: When RST-EN is set to 1, the coder will set to 1 restart marker codes in the compressed data stream at the end of each new coding interval.

Bits 3-0: These bits specify the number of blocks per Mimimum Coded Unit (MCU).

Coder Coding Interval Registers

Register Type: W
Address: $A010_h$ (RH) and $A014_h$ (RL)
Size: 8 bits each

These two registers (RH and RL) define the number of MCUs per coding interval. The largest number of MCUs supported by the CL550 is 2^{16}-1, in accordance with the proposed baseline JPEG standard specifications. Assuming a coding interval of 15 MCUs, then the value of the RH and RL must be set to:

RH	0	0	0	0	0	0	0	0
	7							0
RL	0	0	0	0	1	1	1	1
	7							0

The contents of these registers are used by the coder to determine where to insert restart (RST) marker codes.

Start of Frame Register

Register Type: R/W
Address: 9020_h
Size: 1 bit

By writing either 0 or 1 to the Start of Frame register , the coder is reset. The host can use this register to reset the coder when the end of frame output (FRMEND#) pin is asserted and the host starts reading "FFFF..." data out of the coder.

Decoder Table Sequence Length Register

Register Type: R/W
Address: $A80C_h$
Size: 4 bits

The contents of the Decoder Table Sequence Length register specify the number of significant bits in the Huffman Table Sequence register when the CL550 is in decompression mode.

Decoder Marker Register

Register Type: R
Address: $A810_h$
Size: 8 bits

This register contains the actual Marker code generated by the Decoder.

Note: No resync codes or fill bits will be stored in this register.

Decoder Resume Flag

Register Type: W
Address: $A814_h$
Size: 1 bit

When the decoder detects a marker code (except the RST marker code), it writes it into the Decoder Marker register. The Mark bit in the Flags register is set and the decoder is stopped. By writing either 0 or 1 to the Decoder Resume flag, the host can restart the decoder.

Decoder Code Order Register

Register Type: R/W
Address: $A81C_h$
Size: 1 bit

This register must be set to 1.

Configuration Register

Register Type: R/W
Address: 9000_h
Size: 9 bits

The values loaded into the Configuration register determine the CL550 operating mode and environment, as shown in Table 5-2.

Bit Number					
8	**7 6 5 4**	**3**	**2**	**1**	**0**
dir	video mode select	mstr	Interlace Enable	End of Frame Enable	Undef.

Table 5-2. Configuration Register

dir (8): This field defines the direction of the processing path. When set to 0, the CL550 is in compression mode. When set to 1, the CL550 is in decompression mode.

video mode select (7-4): These four bits are used to select the video mode format:

0000:	invalid
0001:	YUV 4:2:2 mode
0010:	4:4:4 to 4:2:2 mode
0011:	RGB to YUV 4:2:2 mode
0100:	4:4:4 mode
0101:	invalid
0110:	4:4:4:4 mode
0111:	invalid
1000:	Single-component (grayscale) mode

mstr (3): This field determines whether the CL550 is in slave mode or master mode. When set to 0, it indicates that the CL550 is in slave mode. When set to 1, it indicates that the CL550 is in master mode, driving HSync and VSync.

Interlace Enable (2): In Master mode this bit should be set to zero. In Slave mode this bit, when asserted, enables the video interlace mode.

End of Frame Enable (1): When this bit is set to zero the FRMEND# output pin is never asserted. When set to 1, the FRMEND# output is equal to the logical "nand" of the Video Inactive (Vnac) and FIFO-empty (fi0e) bits in the Flags register.

(0): Bit zero is reserved and should be set to 0.

S-Reset Register

Register Type: W
Address: 9008_h
Size: 1 bit

The S-Reset register is used to initiate a soft reset of the CL550. When set to 1, the CL550 begins a soft reset sequence, which automatically resets the S-Reset register to zero. The soft reset sequence produces the same results as the hardware reset. A hardware reset resets this register to 0.

Start Register

Register Type: R/W
Address: $900C_h$
Size: 1 bit

The Start register is used to initiate the video compression/decompression process. In slave mode, when set to 1, video compression or decompression begins on the next negative edge of VSync. In master mode decompression, setting this register to 1 causes the start of a frame immediately. When returned to 0, the CL550 stops the compression or decompression process after completing the frame in progress.

Note: HV Enable register must also be set to 1 before the CL550 can be started or stopped.

Version Register

Register Type: R
Address: 9024_h
Size: 3 bits

The Version register contains the CL550 version number. The current version number is 011 (3).

Init Registers

The Init registers configure the compression/decompression unit pipeline for each video mode. The values to be loaded are provided in the initialization tables in Appendix A.

Register	Type	Address	Size
Init Register 1	W	9800_h	7
Init Register 2	W	9804_h	7
Init Register 3	W	$B600_h$	11
Init Register 4	W	$CF00_h$	11
Init Register 5	W	8820_h	16
Init Register 6	W	8824_h	16
Init Register 7	W	$D400_h$	16

Table 5-3. Init Registers

Flags Register

Register Type: R/W
Address: 9014_h
Size: 16 bits

The Flags register provides CL550 status information. Except for the Late bit, which latches a late condition, each flag indicates the current status of the CL550. Table 5-4 shows the layout of the Flags register.

fi0nf (15): When asserted, this field indicates that the FIFO is not full.

fi0ne (14): When asserted, this field indicates that the FIFO is not empty.

CodecNB (13): When equal to 1, this field indicates that the CODEC register is not busy.

Buser (12): When equal to 1, this field indicates that a bus error has occurred during a CODEC DMA operation.

Mark (11): When equal to 1, this bit indicates that a marker code has been detected in the data stream being decompressed.

Vsyn (10): This field reflects the state of the VSync pin. It is a 1 if the VSync pin is a 0.

Vnac (9): This field defines the state of the CL550 processor between the internal fifo and the pixel bus. Similar to vertical blank, it is low when the pixel bus is in the vertical active region. It is also low when any processing activity is taking place in the video pipeline (between the PBI and the FIFO) of the CL550. In compression, it will go low at the first active line of the pixel bus and return high after the last active line, plus the eight line strip buffer latency, plus the CL550's internal pipeline latency. In decompression, it will go low eight lines, (plus the internal pipeline latency), before the first active line of the pixel bus. It will return high at the end of the first active line of the pixel bus.

fld0 (8): In Slave mode, this field indicates whether the active pixels are in field 0 of an interlaced system. When equal to 1, it indicates that active pixels are in field 0. When equal to 0, it indicates that active pixels are in field 1. For non-interlaced systems, this field is not defined.

fi0e (7): When equal to 1, this field indicates that the FIFO is empty.

fi1q (6): When equal to 1, this field indicates that the FIFO is one-quarter full.

fi1h (5): When equal to 1, this field indicates that the FIFO is one-half full.

fi3q (4): When equal to 1, this field indicates that the FIFO is three-quarters full.

fin1q (3): When equal to 1, this field indicates that the FIFO is not one-quarter full.

fin1h (2): When equal to 1, this field indicates that the FIFO is not one-half full.

fin3q (1): When equal to 1, this field indicates that the FIFO is not three-quarters full.

Late (0): This field indicates that the FIFO has overflowed in compression or underflowed in decompression, causing the compressed data stream to be corrupted. Once set, the late flag must be cleared by software, by writing a 1 followed by a 0 into the Late field.

Bit Number															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
fi0nf	fi0ne	CodecNB	Buser	Mark	Vsyn	Vnac	fld0	fi0e	fi1q	fi1h	fi3q	fin1q	fin1h	fin3q	Late

Table 5-4. Flags Register

Interrupt Mask Register

Register Type: R/W
Address: 9018_h
Size: 16 bits

The Interrupt Mask register defines the interrupt mask for the CL550 NMRQ# output pin. The condition of the NMRQ# output pin is determined by the following equation:

NMRQ# = /(/flag [13] • mask [13]) • /(flag [12] • mask [12])•

((flag [0] • mask[0])
+ (flag [1] • mask[1])
+ (flag [2] • mask[2])
+ (flag [3] • mask[3])
+ (flag [4] • mask[4])
+ (flag [5] • mask[5])
+ (flag [6] • mask[6])
+ (flag [7] • mask[7])
+ (flag [8] • mask[8])
+ (flag [9] • mask[9])
+ (flag [10] • mask[10])
+ (flag [11] • mask[11])
+ (flag [14] • mask[14])
+ (flag [15] • mask[15]))

To mask (disable) an interrupt, that interrupt's mask bit must be set to 0. If an interrupt occurs and its corresponding interrupt mask bit is set to 1, the CL550 asserts the NMRQ# output. Asserting the RESET# input or the S-Reset Flag register sets all fields in the Interrupt Mask register to 0. Table 5-5 shows the fields of the Interrupt Mask register.

fi0nf (15): This field enables the FIFO Not Full interrupt.

fi0ne (14): This field enables the FIFO Not Empty interrupt.

CodecNB (13): This field enables the CODEC Register Not Busy interrupt.

Buser (12): This field enables the Bus Error interrupt.

Mark (11): This field enables the Marker Code interrupt.

Vsyn (10): This field enables the Vertical Sync interrupt.

Vnac (9): This field enables the Vertical Inactive interrupt.

fend (8): In Slave mode, this field enables the FRMEND interrupt.

fi0e (7): This field enables the FIFO Empty interrupt.

fi1q (6): This field enables the FIFO One-Quarter Full interrupt.

fi1h (5): This field enables the FIFO One-Half Full interrupt.

fi3q (4): This field enables the FIFO Three-Quarters Full interrupt.

fin1q (3): This field enables the FIFO Not One-Quarter Full interrupt.

fin1h (2): This field enables the FIFO Not One-Half Full interrupt.

fin3q (1): This field enables the FIFO Not Three-Quarters Full interrupt.

Late (0): This field enables the Data Late interrupt.

Bit Number															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
fi0nf	fi0ne	CodecNB	Buser	Mark	Vsyn	Vnac	fend	fi0e	fi1q	fi1h	fi3q	fin1q	fin1h	fin3q	Late

Table 5-5. Interrupt Mask Register

DMA Request Interrupt Mask Register

Register Type: R/W
Address: 901C$_h$
Size: 16 bits

The DMA Request Interrupt Mask register defines the operation of the CL550 DRQ# pin. The DRQ# pin is used to implement DMA transfers. It can also be used as a generic interrupt signal. The DMA Request Interrupt Mask register is identical to the Interrupt Mask register. The condition of the DRQ# output pin is determined by the following equation:

DRQ# = /(/flag [13] • mask [13]) • /(flag [12] • mask [12])•

((flag [0] • mask[0])
+ (flag [1] • mask[1])
+ (flag [2] • mask[2])
+ (flag [3] • mask[3])
+ (flag [4] • mask[4])
+ (flag [5] • mask[5])
+ (flag [6] • mask[6])
+ (flag [7] • mask[7])
+ (flag [8] • mask[8])
+ (flag [9] • mask[9])
+ (flag [10] • mask[10])
+ (flag [11] • mask[11])
+ (flag [14] • mask[14])
+ (flag [15] • mask[15]))

The DRQ# output pin is always deasserted in the cycle following the assertion of the START# signal.

Note: During a CODEC access, the DRQ# signal is suppressed.

fi0nf (15): This field enables the FIFO Not Full interrupt.

fi0ne (14): This field enables the FIFO Not Empty interrupt.

CodecNB (13): This field enables the CODEC Register Not Busy interrupt. If DMA transfer is used, this field should be set to 1.

Buser (12): This field enables the Bus Error interrupt.

Mark (11): This field enables the Marker Code interrupt.

Vsyn (10): This field enables the Vertical Sync interrupt.

Vnac (9): This field enables the Vertical Inactive interrupt.

fend (8): In Slave mode, this field enables the FRMEND interrupt.

fi0e (7): This field enables the FIFO Empty interrupt.

fi1q (6): This field enables the FIFO One-Quarter Full interrupt.

fi1h (5): This field enables the FIFO One-Half Full interrupt.

fi3q (4): This field enables the FIFO Three-Quarters Full interrupt.

fin1q (3): This field enables the FIFO Not One-Quarter Full interrupt.

fin1h (2): This field enables the FIFO Not One-Half Full interrupt.

fin3q (1): This field enables the FIFO Not Three-Quarters Full interrupt.

Late (0): This field enables the Data Late interrupt.

Bit Number															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
fi0nf	fi0ne	CodecNB	Buser	Mark	Vsyn	Vnac	fend	fi0e	fi1q	fi1h	fi3q	fin1q	fin1h	fin3q	Late

Table 5-6. DMA Request Interrupt Mask Register

CODEC Register

Register Type: R/W
Address: 0000_h - $7FFC_h$
Size: 32 or 16 bits

The CODEC register is the buffer between the Host Bus Interface and the Huffman coder/decoder. During decompression operations, the host writes a sequence of words into the CODEC register and the Huffman coder processes each word. Until a word is processed, a new word of data cannot be entered into the CODEC register. If the host attempts to write a new word into this register, the transfer acknowledge is withheld until the previous word is processed and the transfer can finish. During compression, the Huffman coder places compressed data words into the CODEC register. If the host attempts to read a new word from this register before the Huffman coder completes a new word, the transfer acknowledge is withheld until the transfer can finish. The size of the CODEC register depends on the state of the HBUS_32# pin: when HBUS_32# is tied low, the CODEC register is 32 bits, and when HBUS_32# is tied high, the CODEC register is 16 bits.

Important: During decompression, there are two restrictions placed upon writes to the CL550. Writes to the CL550 registers (other than the CODEC register) are prohibited while the CodecNB flag is not active (equal to 0). Writes to the CL550 registers (other than the CODEC register) while CodecNB is not active will corrupt the previously written CODEC register data. Furthermore, in 16-bit data mode, writes to the CODEC register must always consist of two consecutive 16-bit transfers. Other registers may not be written between the two transfers. Note that the last word written may need to be padded with 1's to fullfill this requirement.

Register and Code Table Summary

Table 5-8 gives summary information about the CL550 registers and tables. All addresses are 32 bit boundaries.

5.2 Initialization Procedure

The initialization of the CL550 requires several steps. C-Cube provides a set of C routines and object-code modules that can be modified for use within the host process or driver software. This section summarizes the operations performed by the initialization software. Basic initialization steps are as follows:

1. Reset the CL550 either by setting and resetting the S-Reset register or by using the hardware reset pin. This halts the chip and puts it in a known state, as shown in Table 5-7.
2. Set up the Configuration register as appropriate for compression/decompression, pixel input format, and master/slave mode.
3. Initialize the constant and diagnostics registers. Several of these values are based on system configuration and operating mode. The algorithms to compute these register values and constants are provided with the CL550 initialization software.
4. Set the Huffman Table Load Enable register to 1 to allow loading of the Huffman code look-up tables.
5. Write the Huffman code Y-AC, C-AC, Y-DC, and C-DC data into the Huffman RAM. The values loaded are specific to compression or decompression and are loaded into a different range of addresses for each operating mode.
6. Reset the Huffman Table Load Enable register to 0.
7. Write Color Transformation Matrix and DCT table constants. These constants are specific to compression or decompression and are provided with the CL550 initialization software.
8. Calculate Quantizer table values based on the desired compression ratio. Write Quantizer table values.
9. Write video field definition registers (HPeriod, HSync, HDelay, HActive, VPeriod, VSync, VDelay, and VActive).
10. Initialize Interrupt Mask registers. If the host is to use interrupt-driven service routines for reading/writing the compressed data, these registers should be set to generate interrupts when the FIFO fills/empties to one-quarter, one-half, or three-quarters full/empty. The level used is determined by the interrupt response latency of the host, the compression ratio of the data, and the pixel data rate.

After this procedure is completed, the CL550 can be started by setting the HV Enable and Start registers to 1. The HV Enable register should be reset to 0 after the falling edge of Vsync.

The following registers are reset to 1s.

Register	Reset Values
HPeriod	All 1s
HSync	All 1s
HDelay	All 1s
HActive	All 1s
VPeriod	All 1s
VSync	All 1s
VDelay	All 1s
VActive	All 1s
Video Latency	All 1s
Horizontal Control	All 1s
Vertical Control	All 1s

The following registers are reset to 0s.

Register	Reset Values
Configuration	0
Huffman Table Load Enable	0
S-Reset	0
Start	0
HV Enable	0
Interrupt Mask	0
DMA Request Interrupt Mask	0
Start of Frame	0
Decoder Code Order	0
Vertical Line Count	0
Init Register 4	0

The following registers are set to:

Register	Reset Values
Flags	1000 0x10 1000 1110
Init Register 3	001 0000 0000

The remaining CL550 registers are unaffected by reset.

Table 5-7. CL550 Registers Reset Value

Note: In the Flags register, bit 10 (x) is set to the inverse of the VSYNC pin.

Register/Table Name	R/W	Starting Address	Data Size
CODEC register	R/W	0000-7FFC	32 or 16
HPeriod register	R/W	8000	14
HSync register	R/W	8004	14
HDelay register	R/W	8008	14
HActive register	R/W	800c	12
VPeriod register	R/W	8010	14
VSync register	R/W	8014	11
VDelay register	R/W	8018	14
VActive register	R/W	801c	11
DCT;coef0 of multi1	W	8800	16
DCT;coef1 of multi1	W	8804	16
DCT;coef2 of multi1	W	8808	16
DCT;coef3 of multi1	W	880c	16
DCT;coef0 of multi2	W	8810	16
DCT;coef1 of multi2	W	8814	16
DCT;coef2 of multi2	W	8818	16
DCT;coef3 of multi2	W	881c	16
Init Register 5	W	8820	16
Init Register 6	W	8824	16
Configuration register	R/W	9000	9
Huffman Table Load Enable register	W	9004	1
S-Reset register	W	9008	1
Start register	R/W	900c	1
HV Enable register	R/W	9010	1
Flags register	R/W	9014	16
Interrupt Mask register	R/W	9018	16
DMA Request Interrupt Mask register	R/W	901c	16
Start of Frame register	R/W	9020	1
Version register	R	9024	3
Init Register 1	W	9800	7
Init Register 2	W	9804	7
Huffman Table Sequence register	W	a000	10
DPCM register Sequence High	W	a004	10
DPCM register Sequence Low	W	a008	10
Coder Attributes register	W	a00c	7
Coding Interval register H	W	a010	8
Coding Interval register L	W	a014	8
Decoder Table Sequence Length register	R/W	a80c	4
Decoder Marker register	R	a810	8
Decoder Resume Flag register	W	a814	1
Decoder DPCM Reset register	W	a818	1
Decoder Code Order register	R/W	a81c	1
Init Register 3	W	b600	11
Quantizer Table (double-buffer mode)	R/W	b800-b9fc	16
Quantizer Table (four table mode)	R/W	b800-bbfc	16
Quantizer A/B Table Select reg.	W	bc00	1
Quantizer Sync register	W	be00	14
Quantizer Y/C Table Sequence register	W	be08	14
Quantizer A/B Table Sequence Select register	W	be0c	10
Color Transformation Matrix	W	c000-c020	12
Video Latency register	R/W	c030	14
HControl register	R/W	c034	14
VControl register	R/W	c038	14
Vertical Line Count register	R	c03c	14
Init Register 4	W	cf00	11
Init Register 7	W	d400	16
FIFO memory	R/W	d800-d9fc	13
Huffman Y-AC	R/W	e000-eafc	9
Huffman Y-DC	R/W	ec00-ec7c	9
Huffman C-AC	R/W	f000-fafc	9
Huffman C-DC	R/W	fc00-fc7c	9

Table 5-8. CL550 Registers and Tables Summary

5.3 Data Flow Control

Once the CL550 registers and tables have been initialized, the actual compression and decompression of data can be started. The following paragraphs provide an overview of the flow of the control program managing the CL550 in compression and decompression mode, respectively.

5.3.1 Compression Mode

In compression mode, the CL550 takes pixel data from the pixel bus and produces a JPEG compressed data stream for output to the host bus. A FIFO is used to isolate the synchronous pixel bus side of the CL550 from the asynchronous host bus side. (See Figure 5-4.)

The host reads compressed data out of the chip. By reading from the host interface, the host causes the FIFO to be read by the Huffman coder. The host is responsible for preventing FIFO overflow. This condition can arise if the host read transfer rate is less than the compression rate for a sustained period. In order to prevent FIFO overflow conditions, the video pipeline can be stopped by asserting the STALL# signal. Data is processed by the Huffman coder while STALL# is asserted. In real-time operation, it is not possible to stop the video data input. While the CL550 pixel pipeline is stalled, incoming pixels must be stored in an external pixel buffer. Time that the CL550 pixel pipeline loses while stalled may be recovered during the blanking regions.

Sample Compression Program

A flow chart of a program using the CL550 Flags register to manage the flow of compressed data out of the CL550 is shown in Figure 5-5. The host starts the compression process: it reads data out at a rate that keeps the FIFO from underflowing or overflowing; when it recognizes the end of image, it drains the remaining data from the FIFO. Finally, it generates any header information required before processing the next image.

The process is started by setting the Start register high and waiting for the FIFO to start to fill. If the FIFO is more than one-quarter full, the host processor reads data out. The Vnac flag can be used by the host processor to indicate that no transfers remain and thus the end of the image has been reached. At this point, the program reads the remaining data from the FIFO. Finally, the host can compile any file information it may need, such as compressed file size, and then wait for the next frame.

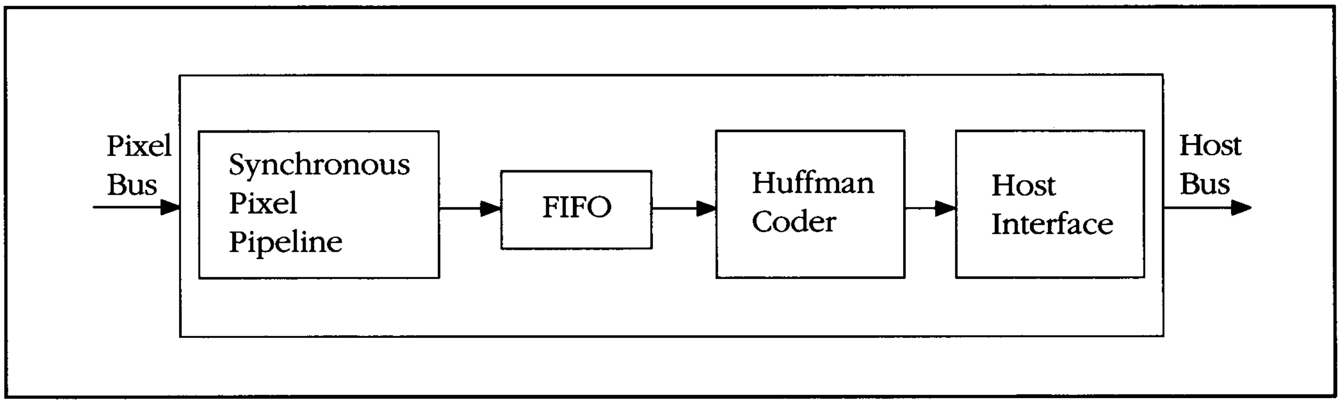


Figure 5-4. Compression Data Flow

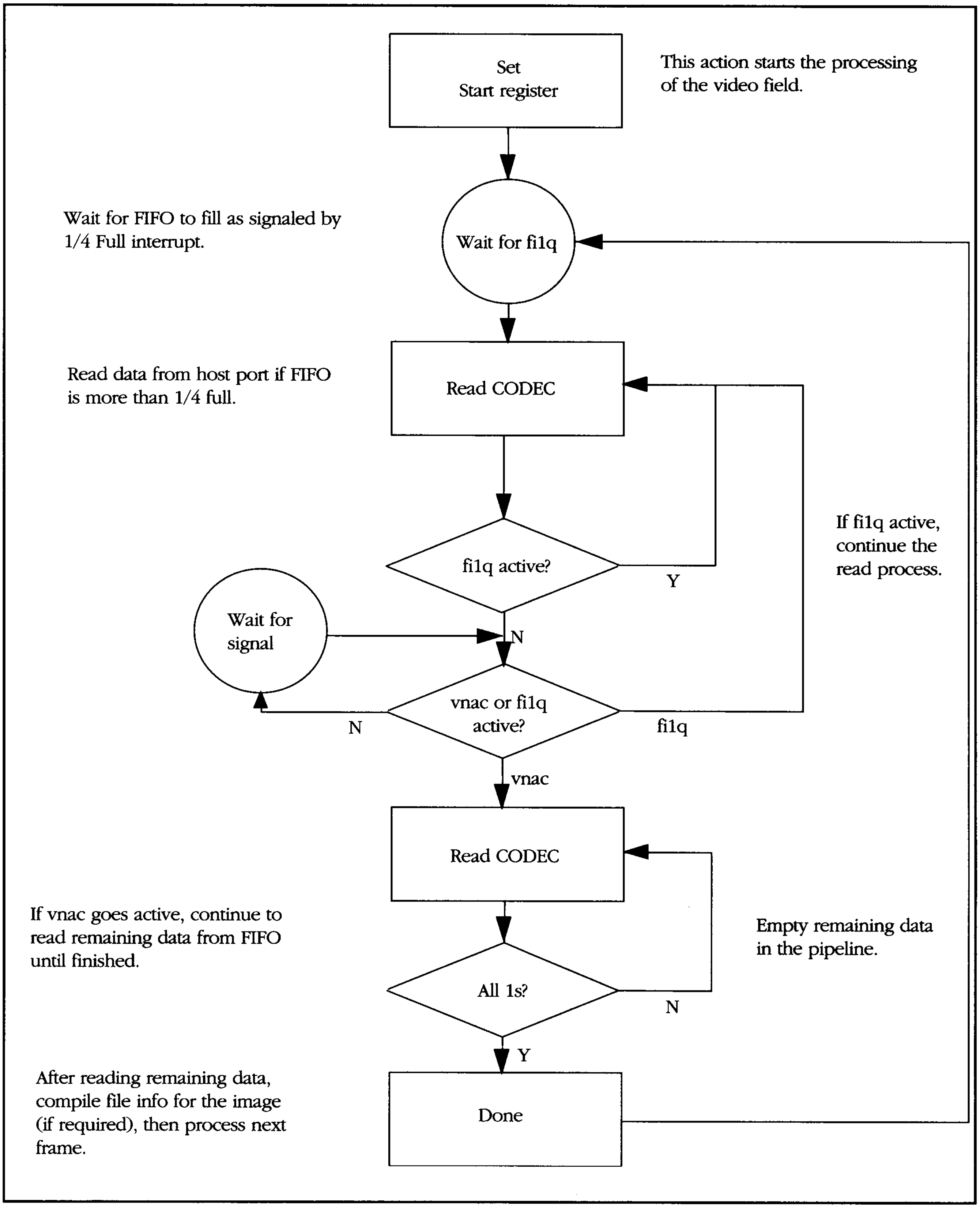

Set Start register
This action starts the processing of the video field.
Wait for fi1q
Wait for FIFO to fill as signaled by 1/4 Full interrupt.
Read CODEC
Read data from host port if FIFO is more than 1/4 full.
fi1q active?
Y
If fi1q active, continue the read process.
Wait for signal
N
vnac or fi1q active?
N
fi1q
vnac
Read CODEC
If vnac goes active, continue to read remaining data from FIFO until finished.
Empty remaining data in the pipeline.
All 1s?
N
Y
Done
After reading remaining data, compile file info for the image (if required), then process next frame.

Figure 5-5. Compression Program Flow Example

5.3.2 Decompression Mode

In decompression mode, the CL550 takes JPEG compressed data from the host bus and produces a pixel data stream for output to the pixel bus. (See Figure 5-6.)

The host writes compressed data into the chip. The host is responsible for preventing FIFO overflow or underflow. This condition can arise if the host write transfer rate is different from the decompression rate for a sustained period. The host must have enough bandwidth to keep up with the basic decompression rate. In order to prevent FIFO underflow conditions, the video pipeline can be stopped by asserting the STALL# signal. Data is processed by the Huffman coder while STALL# is asserted.

Sample Decompression Program

A flow chart of a program that uses the CL550 Flags register to manage the flow of compressed data is shown in Figure 5-7. The host starts the decompression process: it writes data into the CL550 at a rate that keeps the FIFO from underflowing or overflowing until it recognizes the end of image.

The process is started by loading data into the processor until the FIFO is three-quarters full. The Start register is then set high. When the FIFO is more than three-quarters full, the host processor stops writing data into the CL550. It restarts writing data when the FIFO is less than three-quarters full.

In general, the host can obtain status information from interrupts or by polling. Some interrupts, such as the Late Flag interrupt, should always be enabled and used to detect error conditions, such as FIFO underflows. In either case, only a single read is required to obtain status for all of the flags.

Important: During decompression, there are two restrictions placed upon writes to the CL550. Writes to the CL550 registers (other than the CODEC register) are prohibited while the CodecNB flag is not active (equal to 0). Writes to the CL550 registers (other than the CODEC register) while CodecNB is not active will corrupt the previously written CODEC register data. Furthermore, in 16-bit data mode, writes to the CODEC register must always consist of two consecutive 16-bit transfers. Other registers may not be written between the two transfers. Note that the last word written may need to be padded with 1's to fullfill this requirement.

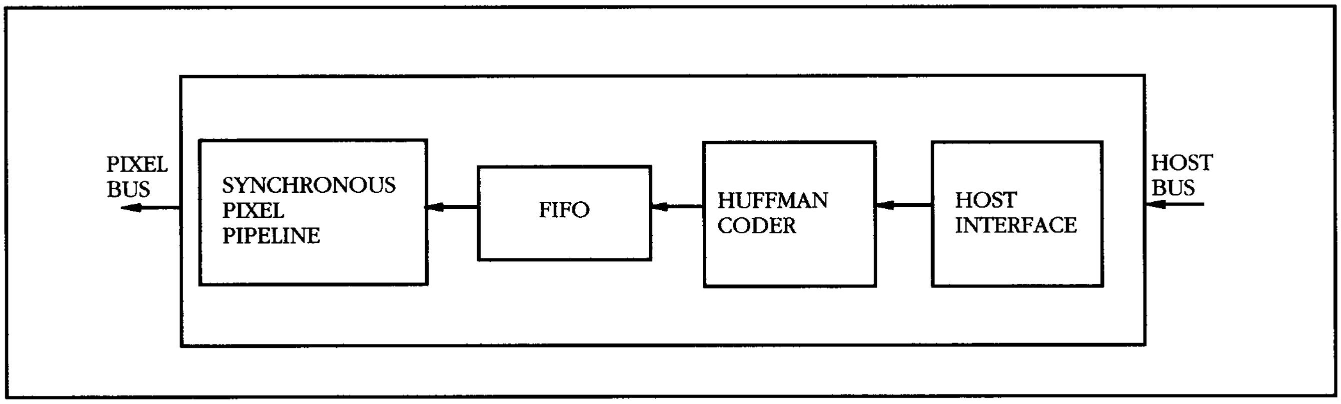


Figure 5-6. Decompression Data Flow

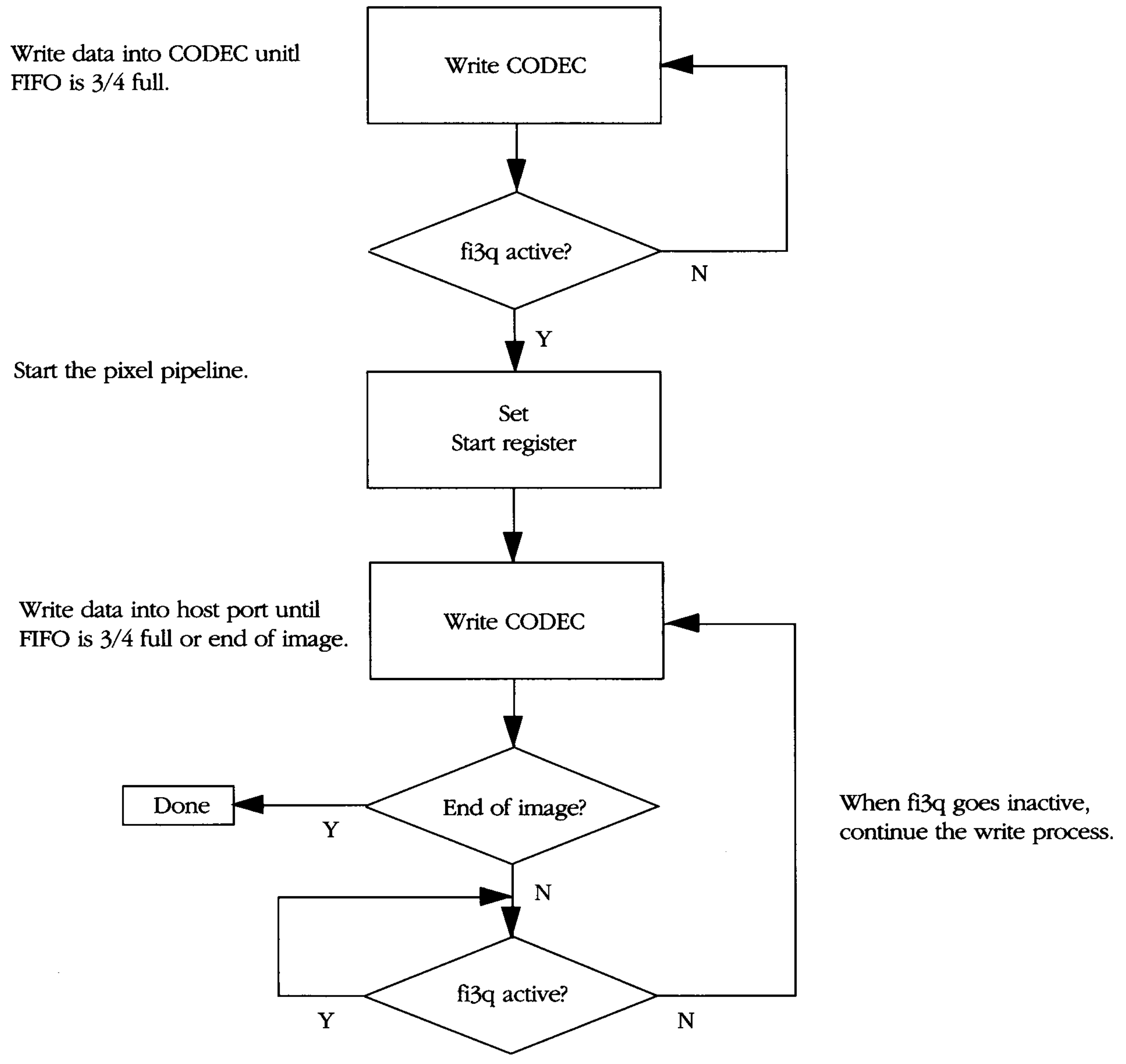

Write data into CODEC unitl FIFO is 3/4 full.
Write CODEC
fi3q active?
N
Y
Start the pixel pipeline.
Set Start register
Write data into host port until FIFO is 3/4 full or end of image.
Write CODEC
Done
End of image?
Y
N
When fi3q goes inactive, continue the write process.
fi3q active?
Y
N

Figure 5-7. Decompression Program Flow Example

5.4 Data Structure and Signaling Parameters

This subsection describes the data structures for the original and compressed data. It also defines the signaling parameters contained in the compressed data streams. For more detail, the reader should refer to the official 15-March-1991, WG10 "JPEG" Committee Draft document CD 10918-1.

Each component of the image (for example Y, U, or V) is represented internally in the compression/decompression system as a rectangular array of samples. The relationship between this internal representation and the placement of pixels in the physical image is defined by the application.

5.4.1 Frames and Scans

A nonhierarchical JPEG image consists of one frame. A frame consists of one or more scans through the image data for each component.

Within each scan, two basic types of data ordering are defined, interleaved and noninterleaved. With noninterleaved data, each scan contains only one component; with interleaved data, each scan contains data from all of the components in the frame.

Minimum Coded Unit (MCU)

The minimum coded unit (MCU) is the smallest unit of data that is allowed by the baseline JPEG compression algorithm.

For noninterleaved data, the MCU is an 8x8 block of samples. The 8x8 block units result from the division of each component into contiguous 8x8 sample blocks for purposes of computing the DCT. The upper left 8x8 block is aligned with the upper left 8x8 group of pixels in the array. The blocks in the component are processed from left to right along block rows, and from top block row to bottom block row of each component.

For interleaved data, the MCU is a block interleave. A block interleave consists of a sequence of one or more 8x8 blocks of samples from each component in the frame. The order of blocks and the number of blocks in a block interleave are determined from the sampling ratio signaling parameter. The sampling ratio defines the relative frequency both horizontally and vertically for the sampling of the individual components in the frame.

5.4.2 Compressed Data Organization and Conventions

The compressed data stream contains delineated segments for each scan in the image. Each scan contains a segment with signaling parameters and a compressed data segment. The compressed data segments contain data generated using the Huffman coding technique. Special resynchronization codes can also be embedded within the coded data segment.

The bit ordering and byte ordering conventions in the compressed data are as follows:

1. The coded data is byte aligned.
2. Integers are sent most significant byte first.
3. Huffman code roots are placed toward the most significant bit of the byte.
4. Integers associated with Huffman codes are appended with the most significant bit adjacent to leaf of the Huffman code.
5. If an integer is split between two or more bytes, the byte containing the most significant bit of the intger is sent first.

The frames and the segments within a frame are delineated by unique byte aligned "marker" codes. The marker codes and associated length fields allow the various segments in the compressed data to be located without decompression.

Marker Code Definitions

The marker codes consist of two byte-aligned 8-bit integers. The first 8-bit integer is always X'FF'. In the compressed data stream, any nonzero value following a X'FF' byte is defined to be a marker code.

The Huffman codes are constrained such that a marker code cannot be created by any valid sequence of normal coding operations. For Huffman coding, the constraint is introduced by inserting ("stuffing") a zero byte following any X'FF' byte that is created by any combination of Huffman codes or appended 1 bit. If these 1 bits happen to create an X'FF' byte, a zero byte is stuffed before appending the X'FF' prefix to the marker code. The marker codes fall into two classes: codes without fields and codes followed by a field whose length is specified by an unsigned integer in the first two bytes of the field. For those marker codes followed by a variable-length field (V), the first two bytes of the variable-length field contain an unsigned 16-bit integer giving the length of the field in bytes. The marker codes are summarized in the following table.

Marker Code	Field Length	Category
Frame signaling for non-hierarchical Huffman coding:		
X'CO'	V	S0F0 - Baseline DCT frame
X'C1'	V	S0S1 - Extended sequential DCT frame
X'C2'	V	S0F2 - Progressive DCT frame
X'C3'	V	S0F3 - Spatial DPCM frame
Frame signaling for hierarchical Huffman coding:		
X'C5'	V	S0F5 - Differential sequential DCT
X'C6'	V	S0F6 - Differential progressive DCT
X'C7'	V	S0F7 - Differential spatial
Frame signaling for nonhierarchical arithmetic coding:		
X'C8'	-	Reserved for JPEG extensions
X'C9'	V	S0F9 - Extended sequential DCT frame
X'CA'	V	S0F10-Progressive DCT frame
X'CB'	V	S0F11-Spatial DPCM frame
Frame signaling for hierarchical arithmetic coding:		
X'CD'	V	S0F13-Differential sequential DCT
X'CE'	V	S0F14-Differential progressive DCT
X'CF'	V	S0F15-Differential spatial
Huffman table load:		
X'C4'	V	DHT - Define Huffman table(s)
Arithmetic coding conditioning load:		
X'CC'	V	DAC - Define arithmetic coding conditioning(s)
Restart interval termination:		
X'D0'-X'D7'	0	RSTm - Restart with modulo 8 count "m".
Other marker codes:		
X'D8'	0	SOI - Start of image
X'D9'	0	EOI - End of image
X'DA'	V	SOS - Start of scan
X'DB'	V	DQT - Define quantization table(s)
X'DC'	V	DNL - Define number of lines
X'DD'	V	DRI - Define restart interval
X'DE'	V	DHP - Define hierarchical progression
X'DF'	V	EXP - Expand reference image(s)
X'EO'-X'EF'	V	APPn - Reserved for application signaling
X'F0'-X'FD'	-	JPGn - Reserved for JPEG extensions
X'FE'	V	COM - Comment
X'FF'	0	FIL - Fill byte

Table 5-9. Marker Codes

The SOI (start of image) marker code always starts the compressed data stream. Note that the SOI marker code can be used to detect problems with either bit order or bit sense of the data.

The DQT (define quantization table) marker code is used to define quantization tables. It is followed by a two-byte length and a field that specifies one or more quantization tables.

The DRI (define restart interval) marker code provides a mechanism for setting the restart interval. It is followed by a two-byte length, which is always equal to 4, and a two-byte value giving the number of MCUs in the restart interval.

The SOFn (start of frame) marker codes start the signaling parameter sequence for all scans within a given image frame. The length field gives the length in bytes for this portion of the signaling parameters. The SOF marker code contains a four-bit field (n) in which it identifies the class of compression algorithm used within a frame.

The DHT (define Huffman table) marker code is used to define Huffman tables. It is followed by a two-byte length and a field that specifies one or more Huffman tables.

The SOS (start of scan) marker code starts the signaling parameter sequence for a scan within the image frame. The length field gives the length in bytes for this portion of the signaling field. Each scan in the frame must start with this marker code.

The RST_m (restart) marker code can be added to the compressed data at the start of each coding interval. If used, it provides a unique byte-aligned code that can be located by scanning the compressed data. A three-bit field in this marker code provides a modulo 8 resynchronization interval count.

The DNL (define number of lines) marker code provides a mechanism for transmitting the line count at the end of the scan. If the DNL marker code is used, it is added to the compressed data at the end of the first scan of the frame.

The EOI (end of image) marker code terminates the compressed data stream.

The FIL (fill) marker code provides a mechanism for extending the 1-byte sequence in the marker code prefix (X'FF'). The FIL marker code must always be followed by another marker code.

The APPn marker codes are marker code categories reserved for application-specific use. Information contained in these fields should not affect the decoding of the compressed data. These codes and associated fields can be inserted into the compressed data before the SOF and SOS marker codes (which then occur immediately after the inserted field). They can also be inserted immediately after the scan signaling field. APPn marker codes and associated fields can be followed by other APPn marker codes and by any other marker codes that are allowed at that position in the compressed data.

JPGn marker codes are reserved for future JPEG extensions.

CL550 Handling of Marker Codes

Compression Mode

The only marker code the CL550 can generate is the restart (RST) code.

If restart marker code generation is enabled (that is, bit 4 of the Coder Attributes register is asserted), the CL550 will insert a restart code "FFDX" (where X is modulo 8 count) after the coder encodes the last pixel data of the first MCU of every coding interval.

If necessary, before an RST code is generated, the current byte will be "1" filled to the byte boundary.

All of the other JPEG marker codes are inserted by software.

Decompression Mode

The CL550 decoder can detect marker codes. If a restart marker code is detected, the CL550 resets the DPCM registers and then proceeds normally. If a different marker code is detected, the CL550 coder completes the proceeding decoding operation, asserts the Marker code flag bit in the Flags register, writes the marker code byte to the marker code register, and stops. If the marker code interrupt is enabled, the host will be interrupted and software will react accordingly. The decoder operation is resumed by software by writing to the Decoder Resume Flag. The decoder will reset the Marker Code Flag bit after the decoder operation is resumed. If end of image (EOI) marker code is encountered by software in the interrupt routine, the DPCM registers must be reset before the Decoder Resume Flag is set.

5.4.3 Structure of the Compressed Data Stream

The structure of a typical compressed data set is presented in Figure 5-8.

```
SOI
        DQT,  length, quantization table definition
                DRI,   length, restart interval
                        SOFn, length, frame parameters
                                DHT,   length, Huffman table definition
                                        SOS, length, scan parameters
                                                compressed data for restart interval, RST0
                                                ......                               , RST1
                                                ........ etc. .....
                                                compressed data for final restart interval, RSTm
                                        SOS, length, scan parameters
                                                ......, RST0
                                                ......, RST1
                                                ....... etc. .....
                                                compressed data for final resart interval, RSTm
EOI
```

Figure 5-8. Compressed Data Structure

The coded data is the portion of the data stream created by Huffman coding. Coded data segments are always terminated by a marker code.

Sequential Control Structure for Encoding an Image

The next paragraphs provide a detailed explanation of the control structure necessary to create a compressed data stream. The control structure for compression of an image is shown in Figure 5-9.

Each frame in the image is coded independently. Only the quantization matrices and code tables may be retained from one frame to the next. Component identification should also be consistent from frame to frame.

Control Structure for a Frame

The control structure for compression of a frame is oriented around the scans in the frame. If noninterleaved data ordering is used, each component is sent in a separate scan. If interleaved data ordering is used, all of the components in the frame are sent in a single scan. Figure 5-10 provides a sketch of the frame control structure.

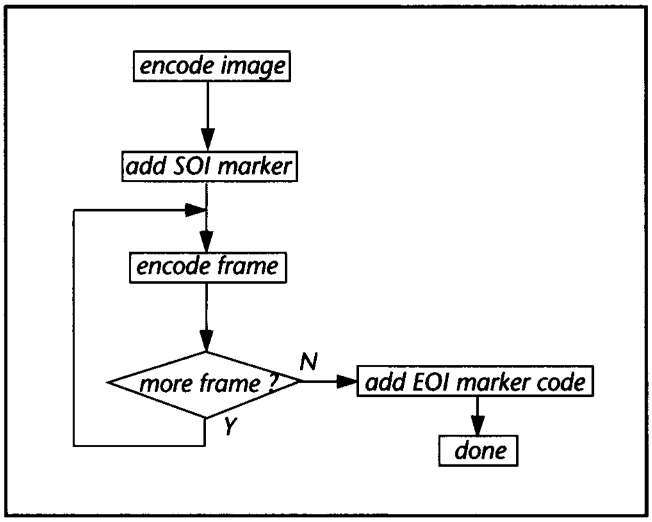


Figure 5-9. Encoder Control Structure for an Image

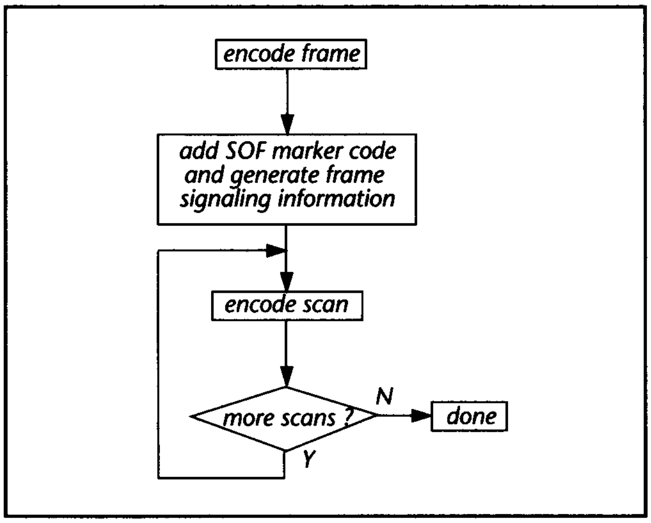

encode frame
add SOF marker code and generate frame signaling information
encode scan
more scans ?
N
Y
done

Figure 5-10. Encoder Control Structure for a Frame

Control Structure for a Scan

A scan is composed of a sequence of restart intervals.

An RST marker code can be placed in the coded data between restart intervals. A scan is always terminated by a SOS, SOF, or EOI marker code. The DNL marker code can precede one of these marker codes at the end of the first scan in the frame.

Figure 5-11 provides a simplified scan control structure. The loop is terminated when the encoder has coded the expected number of minimum coded units (MCUs).

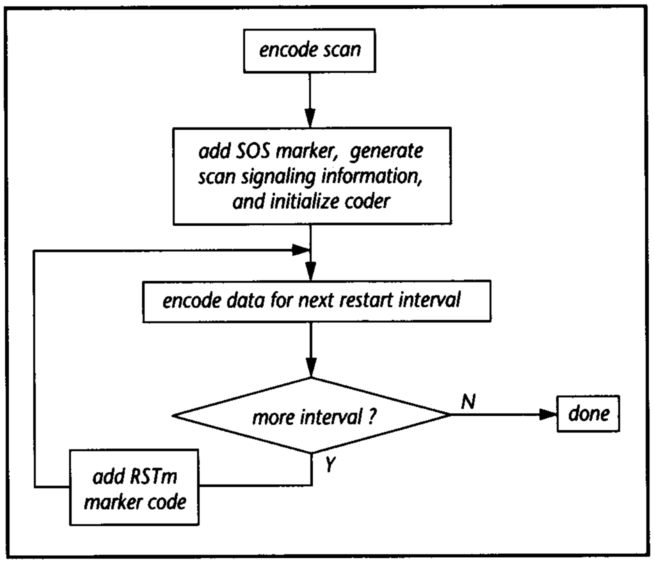

encode scan
add SOS marker, generate scan signaling information, and initialize coder
encode data for next restart interval
more interval ?
N
done
Y
add RSTm marker code

Figure 5-11. Encoder Control Structure for a Scan

Coding Interval Control Structure

Figure 5-12 provides a simplified version of the encoder control structure for a restart interval. The loop is terminated either when the encoder has coded the number of lines of minimum coded units (MCUs) in the restart interval or when it has completed the image frame.

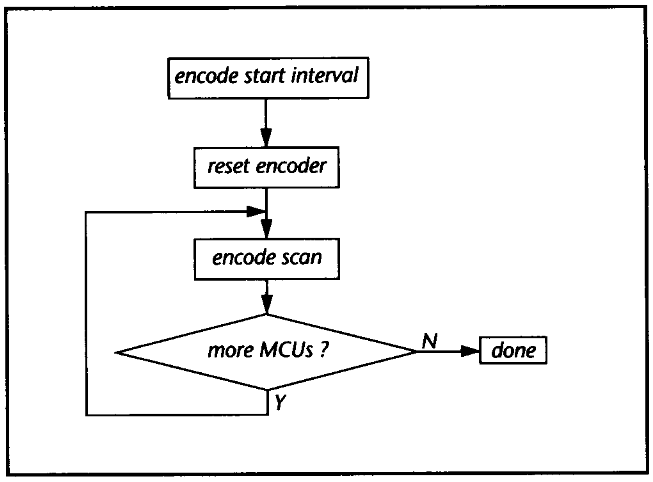


Figure 5-12. Encoding of a Restart Interval

This Page Intentionally Left Blank

6.0 Ordering Information

C-Cube CL550 Processor

Part Description	Temperature Range	Order Number
35 MHz CL550 Processor (CL550-35 PGA package)	Tcase = 0 to 85° C	C0003
30 MHz CL550 Processor (CL550-30 PGA package)	Tcase = 0 to 85° C	C0002
30 MHz CL550 Processor (CL550-30 MQUAD package)	Tcase = 0 to 85° C	C0004
10 MHz CL550 Processor (CL550-10 MQUAD package)	Tcase = 0 to 85° C	C0005
10 MHz CL550 NuBus Processor (CL550-10N MQUAD package)	Tcase = 0 to 85° C	C0006

This Page Intentionally Left Blank

Appendix A: Register Initialization Values

	BYPASS	422	1 Table 444	2 Table 444	3 Table 444	4444	RGB -> YUV_422	444 -> 422
Init_1	0x0002L	0x0042L	0x0040L	0x0040L	0x0040L	0x0042L	0x003CL	0x0042L
Init_2	0x0001L	0x0001L	0x0000L	0x0000L	0x0000L	0x0001L	0x003DL	0x0001L
Init_3	0x0141L	0x0081L	0x007FL	0x007FL	0x007FL	0x0081L	0x007BL	0x0081L
Init_4	0x00F7L	0x00F7L	0x00F5L	0x00F5L	0x00F5L	0x00F7L	0x00F1L	0x00F7L
Lnit_5	0x0000L	0x0000L	0x000EL	0x000EL	0x000EL	0x0000L	0x000AL	0x0000L
Init_6	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L
Init_7	0x005DL	0x001DL	0x001BL	0x001BL	0x001BL	0x001DL	0x0017L	0x001DL
QuantSync	0x0406L	0x0406L	0x0404L	0x0404L	0x0404L	0x0406L	0x0400L	0x0406L
QuantYC	0x2000L	0x2099L	0x1800L	0x20CCL	0x2044L	0x2055L	0x2099L	0x2099L
QuantAB	0x0000L	0x0000L	0x0000L	0x0000L	0x0088L	0x0099L	0x0000L	0x0000L
VideoLatency	0x00BFL	0x017FL	0x0181L	0x0181L	0x0181L	0x017FL	0x0185L	0x017FL

Table A-1. Initialization Register Values for Compression

Init_1	0x003EL	0x003EL	0x003EL	0x003EL	0x003EL	0x003EL	0x003EL	0x003EL
Init_2	0x0037L	0x0037L	0x0037L	0x0037L	0x0037L	0x0037L	0x0037L	0x0037L
Init_3	0x01FFL	0x01FFL	0x01FFL	0x01FFL	0x01FFL	0x01FFL	0x01FFL	0x01FFL
Init_4	0x0049L	0x0049L	0x0049L	0x0049L	0x0049L	0x0049L	0x0049L	0x0049L
Init_5	0x0005L	0x0005L	0x0005L	0x0005L	0x0005L	0x0005L	0x0005L	0x0005L
Init_6	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L
Init_7	0x0022L	0x0022L	0x0022L	0x0022L	0x0022L	0x0022L	0x0022L	0x0022L
QuantSync	0x043EL	0x043EL	0x043EL	0x043EL	0x043EL	0x043EL	0x043EL	0x043EL
QuantYC	0x2000L	0x2033L	0x1800L	0x2099L	0x2088L	0x20AAL	0x2033L	0x2033L
QuantAB	0x0000L	0x0000L	0x0000L	0x0000L	0x0011L	0x0033L	0x0000L	0x0000L
VideoLatency	0x00BFL	0x017FL	0x017FL	0x017FL	0x017FL	0x017FL	0x017FL	0x017FL

Table A-2. Initialization Register Values for Decompression

Config	0x0080L	0x0010L	0x0040L	0x0040L	0x0040L	0x0060L	0x0030L	0x0020L
CodecSeq	0x0000L	0x00CCL	0x0000L	0x0036L	0x0000L	0x0000L	0x00CCL	0x00CCL
CodecCompH	0x0000L	0x0088L	0x0024L	0x0024L	0x0024L	0x00CCL	0x0088L	0x0088L
CodecCompL	0x0000L	0x0044L	0x0012L	0x0012L	0x0012L	0x00AAL	0x0044L	0x0044L
CoderAttr	0x0001L	0x0004L	0x0003L	0x0003L	0x0003L	0x0004L	0x0004L	0x0004L
CodingIntH	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L	0x0000L
CodingIntL	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0008L	0x0001L
DecLength	0x0008L	0x0008L	0x0006L	0x0006L	0x0006L	0x0008L	0x0008L	0x0008L
CodeOrder	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L
QuantSelect	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L	0x0001L

Table A-3. Configuration Register Values