

**CL550/CL560**

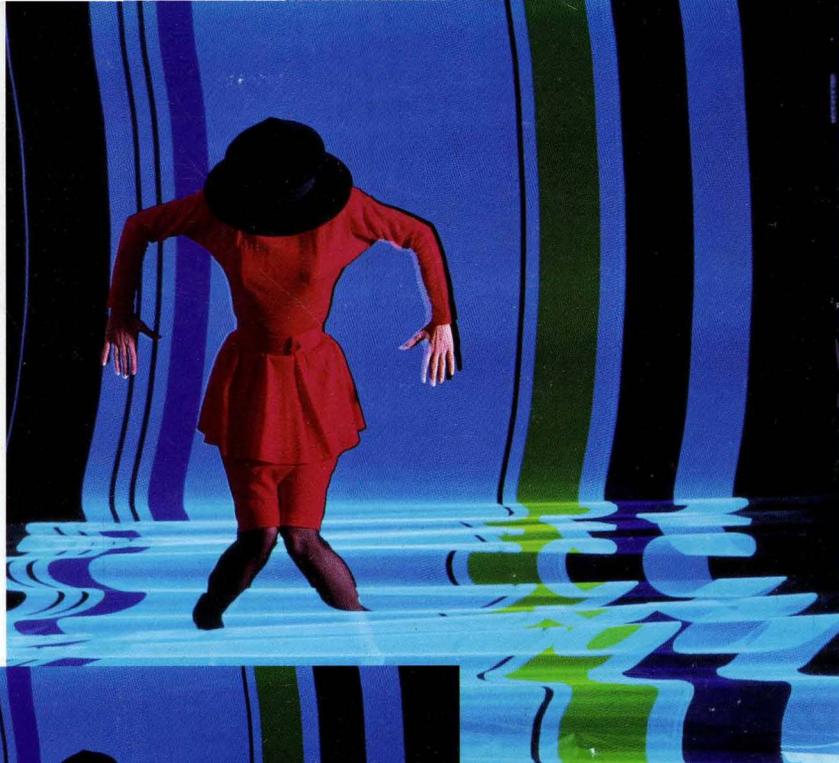
**JPEG**

**COMPRESSION**

**PROCESSOR**

**USER'S**

**MANUAL**



**C-Cube  
Microsystems**

# **CL550/CL560 JPEG Compression Processor User's Manual**



C-Cube  
Microsystems

This document is preliminary. The specifications contained herein are derived from functional specifications and performance estimates, and have not been verified against production parts.

C-Cube Microsystems reserves the right to change any products described herein at any time and without notice. C-Cube Microsystems assumes no responsibility or liability arising from the use of the products described herein, except as expressly agreed to in writing by C-Cube Microsystems. The use and purchase of this product does not convey a license under any patent rights, copyrights, trademark rights, or any other intellectual property rights of C-Cube Microsystems.

**Trademark Acknowledgment:**

C-Cube, CL550, and CL560 are trademarks of C-Cube Microsystems. The corporate logo is a registered trademark of C-Cube Microsystems.

Windows and Video-for-Windows are trademarks of Microsoft Corporation. Apple and Apple QuickTime are trademarks of Apple Computer, Inc.

© C-Cube Microsystems 1993  
All rights reserved

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

**Customer Comments and Feedback:**

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

techpubs @ c-cube.com

C-Cube Part # 90-1556-101, Revision A

# Preface

This manual is the primary users guide for the C-Cube CL550 and CL560 JPEG Compression Processors. It contains detailed information about the CL550 and CL560 hardware and also provides general information on how to program the parts.

This manual is intended for:

- System designers and managers who are evaluating the CL550 and CL560 for possible use in a system
- Designers and hardware engineers who are designing a system based on the CL550 and CL560
- Programmers and software engineers who are writing application programs that interact with the CL550 and CL560

This manual is divided into these chapters:

- Chapter 1, Introduction, presents an introduction to the architecture of the CL550 and CL560

---

**Audience**

---

---

**Organization**

---

- Chapter 2, JPEG Overview, provides a brief overview of the JPEG algorithm used by the compression processors.
- Chapter 3, Signal Descriptions, describes the function of each of the external signals on the CL550 and CL560.
- Chapters 4 and 5, Host Interface and Video Interface, present functional descriptions for the main interfaces of the CL550 and CL560.
- Chapter 6, Specifications, includes detailed electrical and mechanical specifications.
- Chapter 7, Registers, describes in detail each of the internal registers of the CL550 and CL560.
- Chapter 8, System Designer's Guide, provides a general overview on programming the CL550 and CL560 with initialization procedures and compression/decompression procedure flowcharts.

---

## **Conventions**

---

Please note the following conventions that are used in this manual:

- Hexadecimal numbers are indicated by the prefix 0x, for example, 0xFF. Binary numbers are indicated by a subscript, for example, 10<sub>2</sub>. Otherwise, all numbers used in this guide are decimal numbers unless otherwise noted.

---

## **Revision History**

---

This manual, part # 90-1556-101 Rev A., supersedes the previous revision by the same name. The major content changes include:

- Chapter 6, Specifications: AC timing parameters are listed for 7 products and speed grades:
  - CPGA package (3): CL550-35, CL550-30, CL560-30
  - MQUAD package (4): CL550-10, CL550-30, CL560-15, CL560-30
- Chapter 8, System Designer's Guide, is new.

# Contents

## **1 Introduction**

1.1	CL550 Features	1-2
1.2	CL560 Improvements	1-2
1.3	Applications	1-3
1.4	Product Family	1-3
1.5	CL550 Functional Description	1-4
1.6	CL560 Functional Description	1-6

## **2 JPEG Overview**

2.1	JPEG Background Information	2-2
2.2	Operation of the JPEG Algorithm	2-3
2.3	Discrete Cosine Transform	2-4
2.4	Quantization	2-5
2.5	Zero Run-length Coding	2-7
2.6	Entropy Encoding	2-8
2.7	Summary of JPEG Baseline	2-8

## **3 Signal Descriptions**

3.1	Host Interface	3-3
3.1.1	Data Transfer Signals	3-3
	HBUS[31:0]	3-3

	<u>HBOUT</u>	3-4
	<u>HBUS_32</u>	3-4
	ID[3:0]	3-4
3.1.2	DMA Signals	3-4
	<u>DRQ</u>	3-4
	<u>DMA_MSTR</u>	3-4
3.1.3	Interrupt Signals	3-5
	<u>NMRQ</u>	3-5
	HALF_FULL	3-5
	<u>IRQ1</u>	3-5
	IRQ2	3-5
3.1.4	Timing and Control Signals	3-6
	START	3-6
	TMO	3-6
	TM1	3-6
	TM2	3-7
	<u>TMOUT</u>	3-7
	<u>FRMEND</u>	3-7
	HBCLK	3-7
	RESET	3-7
	TEST	3-8
3.2	Video Interface	3-8
3.2.1	Pixel Data, Address and Handshake Signals	3-8
	PXDAT[23:0]	3-8
	PXADR[15:0]	3-8
	PXRE	3-8
	PXWE	3-9
	PXIN	3-9
	PXOUT	3-9
	STALL	3-10
3.2.2	Video Synchronization Signals	3-10
	<u>HSYNC</u>	3-10
	<u>VSYNC</u>	3-10
	<u>BLANK</u>	3-10
3.2.3	Video Clock Signals	3-11
	PXCLK	3-11
	PXPHASE	3-11
	CLK3	3-11
<b>4 Host Interface</b>		
4.1	Register Access Timing	4-3
4.1.1	Signal Descriptions	4-3

4.1.2	Register Access Timing	4-7
4.1.3	Host Bus Register Access	4-9
4.1.4	Host Bus Register Write	4-11
4.2	DMA Access Timing	4-12
4.2.1	CL560 DMA Transfers	4-14
4.2.2	CL560 DMA Write Transaction Timing	4-15
4.2.3	CL560 DMA Read Transaction Timing	4-17
4.2.4	Alternative Method of CL550/560 DMA Transfers	4-19
4.2.5	CL550 Burst-mode Write Transaction Timing	4-20
4.2.6	CL550 Burst-mode Read Transaction Timing	4-22
4.2.7	Operational Considerations	4-23
4.3	Control Signals	4-25
4.3.1	RESET	4-25
4.3.2	NMRQ	4-26
4.3.3	HALF_FULL	4-26
4.3.4	TRQ1, IRQ2	4-26
<b>5</b>	<b>Video Interface</b>	
5.1	Overview	5-2
5.1.1	Signal Descriptions	5-3
5.1.2	Video Interface Clocks	5-4
5.1.3	Master/Slave Mode Operation	5-7
5.1.4	STALL Operation	5-7
5.2	Video Interface Logic	5-10
5.2.1	Pixel Order Conversion	5-10
5.2.2	Window Management and Control	5-12
5.2.3	Color Conversion	5-12
5.3	Basic System Configurations	5-14
5.4	Timing Diagrams Compression Mode	5-17
5.5	Timing Diagrams Decompression Mode	5-29
<b>6</b>	<b>Specifications</b>	
6.1	Operating Conditions	6-2
6.2	AC Characteristics	6-3
6.2.1	Host Interface Control Signal Timing	6-4
6.2.2	Video Interface Signal Timing	6-16
6.3	Package Specifications	6-23
6.3.1	144-Pin Ceramic Pin-Grid Array	6-24
6.3.2	144-Pin MQUAD Package	6-29
<b>7</b>	<b>Registers</b>	
7.1	Video Interface Registers	7-6
	HPeriod Register	7-7

HSync Register	7-8
HDelay Register	7-8
HActive Register	7-9
VPeriod Register	7-9
VSync Register	7-10
VDelay Register	7-10
VActive Register	7-11
Video Latency Register	7-11
Vertical Line Count Register	7-12
HControl Register	7-12
VControl Register	7-12
HV Enable Register	7-13
Color Transformation Matrix	7-13
DCT Tables	7-15
7.2 Compression and Decompression Registers	7-15
Quantizer Tables	7-15
Quantizer A/B Table Select Register	7-16
Quantizer Y/C Table Sequence Register	7-17
Quantizer A/B Table Sequence Register	7-18
Quantizer Sync Register	7-18
Coder/Decoder DPCM Reg. Seq. Registers, RH	7-19
Coder/Decoder DPCM Reg. Seq. Registers, RL	7-19
Decoder DPCM Reset Register	7-20
Huffman Code Tables	7-20
Huffman Table Load Enable Register	7-20
Huffman Table Sequence Register	7-21
Coder Attributes Register	7-21
Coder Coding Interval Registers	7-22
Coder Sync Register(CL560)	7-22
Compressed Word Count Register, High (CL560)	7-23
Compressed Word Count Register, Low (CL560)	7-23
Coder Rate Control Active Register(CL560)	7-24
Coder Rate Control Enable Register (CL560)	7-24
Coder Robustness Active Register (CL560)	7-25
Coder RST Padding Control Register (CL560)	7-25
Start of Frame Register	7-26
Decoder Table Sequence Length Register	7-26
Decoder Marker Register	7-26
Decoder Resume Flag (CL550)	7-26
Decoder Code Order Register	7-27
Decoder Start Register (CL560)	7-27
Decoding Mismatch Register (CL560)	7-27

Decoding Mismatch Error Code Register (CL560)	7-28
Configuration Register	7-28
S-Reset Register	7-29
Start Register	7-29
Version Register	7-30
Init Registers	7-30
Flags Register	7-31
NMRQ Interrupt Mask Register (CL550)	7-33
TRQ1 Mask Register (CL560)	7-34
DMA Request Interrupt Mask Register	7-35
IRQ2 Interrupt Mask Register (CL560)	7-37
FRMEND Enable Register (CL560)	7-38
CODEC Register (CL550)	7-39
CODEC FIFO (CL560)	7-40
FIFO Level Register (CL560)	7-40

## 8 System Designer's Guide

8.1	Typical System Configurations	8-1
8.2	JPEG Video Concepts	8-4
8.3	CL550 Compression Operation	8-5
8.3.1	Overview	8-5
8.3.2	CL550 Operation as a Still-image Coprocessor	8-7
8.3.3	Operation as a Video Compression Processor	8-10
8.3.4	Use of DMA and Interrupts with the CL550	8-12
8.3.5	DMA System Programming Example	8-14
8.4	CL560 Compression Operation	8-16
8.4.1	CL560 Enhancements	8-16
8.4.2	CL560 Operational Differences from the CL550	8-18
8.4.3	Compressed Data Rate Control Mechanisms	8-19
8.4.4	Typical Compression System Using the CL560	8-21
8.4.5	Slave-mode Compression Operation	8-23
8.5	CL550 Decompression Operation	8-25
8.5.1	Overview	8-25
8.6	CL560 Decompression Operations	8-28
8.7	CL5xx Initialization Procedures	8-30
8.7.1	Overview	8-30
8.7.2	Programming the Video Interface Control Registers	8-32
8.7.3	HControl, VControl Registers Programming	8-34
8.7.4	DCT Lookup Table	8-35
8.7.5	Color Conversion Matrix	8-35
8.7.6	Sequence Control Registers	8-35
8.7.7	Selection of the RST Interval (Compression)	8-36

8.7.8	Pipeline Configuration Registers	8-36
8.7.9	Quantizer Tables Loading	8-37
8.7.10	Huffman Tables Loading	8-37
8.7.11	CL5xx Start-up Sequence	8-37
8.7.12	Device Reset Considerations	8-38
8.8	Programming the Huffman Tables	8-41
8.8.1	Extracting Huffman Tables from ISO JPEG Interchange Format	8-41
8.8.2	CL550 Huffman Table Formats	8-41
8.8.3	CL560 Huffman Table Format	8-47
8.9	CL5xx Quantizer Table Programming	8-50
8.9.1	Overview	8-50
8.9.2	Q Table Scaling Function	8-51
8.9.3	Generating Machine-Loadable Q Tables for CL5xx Devices	8-53
8.10	Custom Block Sequencing	8-56
8.10.1	Restrictions	8-56
8.10.2	CL5xx Internal Component Sequencing	8-56
8.10.3	CL550 Component Sequence Programming	8-57

# Figures

1-1	CL550 Block Diagram	1-4
1-2	CL560 Block Diagram	1-6
2-1	Basic Image Compression Scheme for Coder and Decoder	2-3
2-2	Quantizer Stepping (Uniform Quantization)	2-5
2-3	Psychovisual Weighting Functions for the Luminance and Chrominance Components	2-6
2-4	Zigzag Pattern for Reordering the 8 x 8 DCT Coefficients	2-7
3-1	CL550 and CL560 Logic Diagrams	3-2
3-2	Strip Buffer RAM Connections	3-9
4-1	CL550 and CL560 Host Interface Block Diagram	4-2
4-2	ID[3:0] Chip Select Address Format	4-4
4-3	Register Read Transaction	4-10
4-4	Register Write Transaction	4-11
4-5	DMA Mode Operation	4-13
4-6	CL560 DMA Write (Decompression - Host Write to CODEC)	4-15
4-7	CL560 DMA Read (Compression - Host Read from CODEC)	4-17
4-8	CL550 DMA Write (Decompression - Host Write to CODEC)	4-20
4-9	CL550 DMA Read (Compression - Host Read from CODEC)	4-22
5-1	Video Interface Block Diagram	5-2
5-2	CL550 Host Bus Write Timing for the Configuration Register	5-5

5-3	STALL Timing, YUV 4:2:2 Compression Example	5-8
5-4	Pixels in Raster Order	5-10
5-5	Same Pixels in 8 x 8 Block Order	5-10
5-6	Video Field Descriptions	5-13
5-8	Typical Video System Application	5-14
5-7	RGB to YUV Conversion Operation	5-14
5-9	Typical Still-frame Application	5-15
5-10	Typical Multimedia System Application	5-16
5-11	Typical CL560 Synchronous Interface	5-18
5-12	Compression Overview	5-19
5-13	First Active Line (Compression)	5-22
5-14	Beginning of First Active Line with Active PXRE (Compression)	5-24
5-15	Last Line with Active PXIN (Compression)	5-26
5-16	Last Line with Active PXRE (Compression)	5-28
5-17	Decompression Overview	5-30
5-18	First Line with Active PXWE (Decompression)	5-32
5-19	First Active Line with Active PXOUT	5-34
5-20	Last ActiveLine with PXWE	5-36
5-21	Last Line with Active PXOUT	5-38
6-1	HBCLK and RESET Timing	6-4
6-2	DRQ Timing	6-5
6-3	NMRQ, IRQ1 Timing	6-6
6-4	CL550 HALF_FULL, FRMEND Timing	6-7
6-5	CL560 IRQ2, FRMEND Timing	6-7
6-6	CL550 Host Interface Timing: Register and Memory Write	6-8
6-7	CL560 Host Interface Timing: Register and Memory Write	6-9
6-8	CL550 Host Interface Timing: Register and Memory Read	6-10
6-9	CL560 Host Interface Timing: Register and Memory Read	6-11
6-10	CL560 Host Interface Timing: Burst Mode Read	6-12
6-11	CL560 Host Interface Timing: Burst Mode Write	6-13
6-12	Video Interface Clock Timing	6-16
6-13	Video Interface Timing: Compression, Full Rate Mode	6-17
6-14	Video Interface Timing: Decompression, Full Rate Mode	6-18
6-15	Video Interface Timing: Compression, Half-Rate Mode	6-19
6-16	Video Interface Timing: Decompression, Half-Rate Mode	6-20
6-17	CL550 and CL560 CPGA Physical Dimensions	6-24
6-18	CL550 and CL560 CPGA Pin Layout (Bottom View)	6-25
6-19	CL550 and CL560 CPGA Pinout Diagram (Top View Through Chip)	6-26
6-20	MQUAD Physical Dimensions	6-29
6-21	CL550 MQUAD Pinout Diagram	6-30
7-1	Video Field Registers	7-6
7-2	Quantizer Tables Configuration (Double-buffer Mode)	7-15

7-3	Quantizer Tables Configuration (Four-table Mode)	7-16
8-1	Coprocessor Configuration	8-2
8-2	I/O Peripheral Configuration	8-3
8-3	Combined I/O Peripheral and Coprocessor Configuration	8-3
8-4	JPEG Video Data in Frame-by-Frame Organization	8-4
8-5	JPEG Video Data Organized as a Continuous-Stream	8-5
8-6	Flow Chart for the CL550 FIFO Drain Loop	8-8
8-7	Flow Chart for Flushing the CL550 FIFO	8-9
8-8	Typical CL550 Video Compression System	8-11
8-9	Programmed I/O System Architecture	8-12
8-10	Direct DMA Architecture	8-13
8-11	Interrupt Timing for 30 Frame/Second Compression	8-15
8-12	CL550 and CL5660 Architecture Differences	8-16
8-13	Flow Chart for the CL560 FIFO Drain Loop	8-20
8-14	Typical CL560-based Compression System	8-22
8-15	Typical CL560-based Slave-mode Compression System	8-24
8-16	Flow Chart for Filling the CL550 FIFO	8-27
8-17	Program Flow for Filling the CL560 FIFO	8-29
8-18	Programming Parameters for Video Window Example	8-33
8-19	CL550 Huffman Table Formats for Compression	8-43
8-20	CL550 Huffman Table Format for Decompression	8-45
8-21	Detecting a Bad Huffman Table	8-46
8-22	CL550 Huffman Table Layouts in Compression Mode	8-48
8-23	CL560 Machine-Specific Huffman Table Entry Format	8-49
8-24	Example for Accessing a Table Entry at Address 0xE000	8-49
8-25	Typical Flow for Generating a CL5xx Quantizer Table	8-51
8-26	Quantizer Table Scaling Function	8-52
8-27	Make CL5xx Quantizer Table Function Listing	8-54
8-27	Make CL5xx Quantizer Table Function Listing	8-55
8-28	CL5xx Block Storage Unit Component Sequencing	8-57
8-29	Quantizer Y/C Sequence Register	8-58
8-30	Quantizer Y/C Sequence Register Example	8-59
8-31	Quantizer A/B Sequence Register	8-59
8-32	DCPM Register Sequence	8-60



# Tables

2-1	Converting RGB Components to YCbCr Components	2-4
2-2	Zigzag Sequence of Quantized DCT Coefficients	2-7
3-1	CL550 and CL560 Redefined Signal Pins	3-3
3-2	Address and Data Bus Configuration	3-3
4-1	CL550 and CL560 Redefined Signal Pins	4-3
4-2	Address and Data Bus Configuration	4-5
4-3	TM Signals During a Host Bus Register Read (CL550 to Host)	4-6
4-4	TM Signals During a Host Bus Register Write (Host to CL550)	4-6
4-5	TM Signals During a Host Bus Register Read (CL560 to Host)	4-6
4-6	TM Signals During a Host Bus Register Write (Host to CL560)	4-7
4-7	TM Signals During a DMA Read (CL560 to Host)	4-7
4-8	TM Signals During a DMA Write (Host to CL560)	4-7
4-9	CL550 Bus Error Conditions	4-9
4-10	External Buffers Direction Control	4-14
5-1	CL550/560 Color Modes and Pixel Data Configurations	5-6
5-2	Video Field Control Registers	5-12
5-3	Compression Timing Example Register Values	5-17
5-4	Timing Example Video Parameters	5-29
6-1	Absolute Maximum Ratings	6-2
6-2	Operating Conditions	6-2

6-3	DC Characteristics	6-2
6-4	HBCLK and RESET Timing Parameters, CPGA Package	6-4
6-5	HBCLK and RESET Timing Parameters, MQUAD Package	6-4
6-6	<u>DRQ</u> Timing, CPGA Package	6-5
6-7	<u>DRQ</u> Timing, MQUAD Package	6-5
6-8	<u>NMRQ</u> , <u>IRQ1</u> Timing, CPGA Package	6-6
6-9	<u>NMRQ</u> , <u>IRQ1</u> Timing, MQUAD Package	6-6
6-10	HALF_FULL, <u>FRMEND</u> Timing, CPGA Package	6-7
6-11	HALF_FULL, <u>FRMEND</u> Timing, MQUAD Package	6-7
6-12	Host Interface Timing, CPGA Package	6-14
6-13	Host Interface Timing, MQUAD Package	6-15
6-14	Video Interface Clock Timing, CPGA Package	6-16
6-15	Video Interface Clock Timing, MQUAD Package	6-16
6-16	Video Interface Timing Table, CPGA Package	6-21
6-17	Video Bus Timing Table, MQUAD Package	6-22
6-18	CPGA Pin List Sorted by Pin Number	6-27
6-19	CPGA Pin List Sorted by Pin Name	6-28
6-20	CL550/CL560 MQUAD Pinout Differences	6-30
6-21	CL550 (CL560) MQUAD Pin List Sorted by Pin Number	6-31
6-22	CL550 (CL560) MQUAD Pin List Sorted by Pin Name	6-32
7-1	CL550 Register and Summary	7-2
7-2	CL560 Register and Summary	7-4
7-3	HPeriod Register Value Calculation	7-7
7-4	HDelay Register Value Calculation	7-8
7-5	HActive Register Value Calculation	7-9
7-6	Video Latency Register Values	7-11
7-7	RGD- YUV Color Transformation Coefficient Addresses	7-14
7-8	DCT Values	7-15
7-9	Quantizer Y/C Values	7-17
7-10	Quantizer A/B Values	7-18
7-11	Quantizer Sync Register Data Sync Field Values	7-19
7-12	DPCM Sequence Register Values	7-19
7-13	Huffman Sequence Register Values	7-21
7-14	MCU Block Number Values	7-22
7-15	Coder Sync Register Initialization Values	7-23
7-16	FIFO Threshold Levels	7-25
7-17	Video Mode Select Bits	7-28
7-18	Version Number Register Contents	7-30
7-19	Initialization Registers	7-30
8-1	CL5xx Register Reset Values	8-39
8-2	Registers Required for Minimized Restart Procedure	8-40
8-3	CL550 Huffman Layouts in Compression Mode	8-43

# 1 Introduction

The C-Cube CL550 and CL560 are high-performance single-chip compression/decompression processors that implement the baseline CCITT/ISO Joint Photographic Experts Group (JPEG) digital image compression algorithm. The CL550 and CL560 processors are designed for applications that require manipulation of high-quality digital pictures and motion sequences.

These parts can encode and decode grayscale and color images at video rates. The image compression ratio is controlled by the on-chip quantization tables. Compression ratios from 8:1 to 100:1 are possible depending on the quality, storage and bandwidth requirements of each application.

The CL550 and CL560 have on-chip video and host bus interfaces. 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.

---

**1.1**  
**CL550 Features**

---

The CL550 compression/decompression processor features the following:

- Compressed output conforms to the JPEG Baseline Process as defined by ISO IS 10918-1
- Real-time compression and decompression of CIF (320 x 240 x 30 fields per second) and 1/2 CCIR 601 video (640 x 240 x 25 or 30 fields per second).
- Up to 2 Mbytes/second sustained compressed data rate (CL550-35)
- Highly pipelined DCT/IDCT processor running at up to 35 Mhz (CL550-35)
- Support for 8-bit grayscale, RGB, CMYK or 4:4:4:4, and YUV color space input and output
- User-accessible quantizer and Huffman tables
- Frame-by-frame adjustment of compression ratios
- 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 MQUAD and ceramic PGA packages
- CMOS technology

---

**1.2**  
**CL560**  
**Improvements**

---

The CL560 Compression processor has all of the features of the CL550, with these improvements:

- Up to 60 Mbytes/second sustained compression rate
- Up to 15 million pixels/second processing rates
- Highly pipelined DCT/IDCT processor runs at up to 30 MHz
- Real-time compression of CCIR 601 video frames at broadcast-quality levels
- Improved Huffman table architecture allows the same table to be used for compression and decompression, allowing faster switch-

ing between modes

- Single cycle per 32-bit word Huffman CODEC
- Synchronous or asynchronous video interface operation
- On-chip 128 x 32 compressed data FIFO supports burst access
- Improved interrupt structure and DMA support
- Compression rates as high as 50:1 for real-time video applications
- Compression rates as low as 1:1 for high-quality printer, copier and professional video applications

The CL560 pinout is a superset of the CL550 pinout. Although the function of two pins has changed, most CL550 users can upgrade to the CL560 with only minor changes to printed circuit board layouts.

These JPEG processors can be used in any of the following applications:

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

The CL550 and the CL560 are the two members in the JPEG compression/decompression processor family. The CL560 is an enhanced version of the CL550.

The CL550 is the first product in the family. It is designed for use in PC multimedia and still-image based systems where cost is a factor.

The CL560 is a new-generation JPEG processor designed for high-end still image and real-time video compression and decompression. The CL560 can compress and decompress full CCIR 601-resolution video frames in real time, at compression ration as high as 50:1 or as low as

---

### **1.3 Applications**

---



---

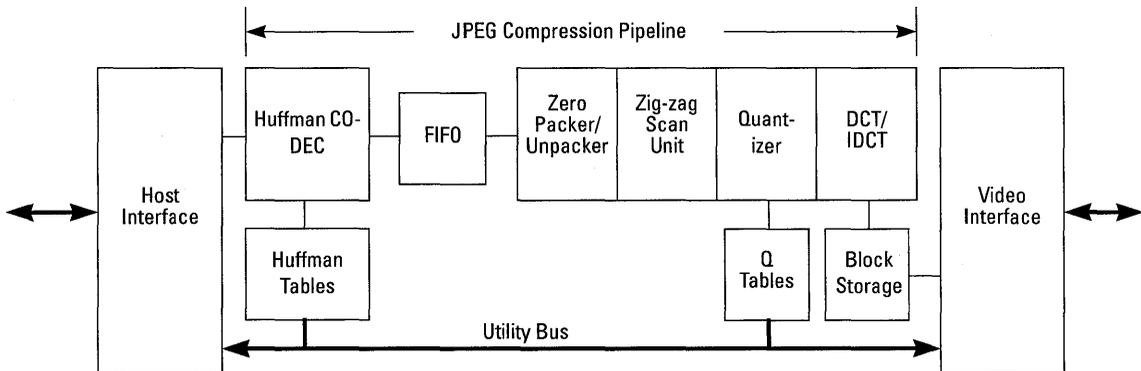
### **1.4 Product Family**

---

1:1. The CL560 is ideally suited for used in high-end printing and scanning systems, high-speed digital copiers and printers, and a wide range of broadcast-quality video editing applications.

## 1.5 CL550 Functional Description

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



**Figure 1-1 CL550 Block Diagram**

During compression operations, uncompressed pixel data is written into the Video interface. The first operation that the video interface performs is a raster-to-block conversion of the pixel data. This is necessary because video generation and display devices normally deal with pixel data as raster lines, while the JPEG compression algorithm requires that the pixel data be organized as 8 x 8 blocks. Logic in the CL550 device performs that conversion.

The next step is the optional RGB-to-YUV color space conversion. This is also done in the video interface. Video generation and display devices frequently present data to the CL550 as RGB pixels. The CL550 can also perform the color space conversion. Other functions done by the Video interface are pixel formatting and window sizing.

Once the Video interface has formatted the pixel data, it writes the data into the Block Storage unit. The Block Storage unit stores the 8 x 8 blocks until the JPEG compression pipeline is ready to process them. It then sequences them into the pipeline one block at a time.

Each component block is then processed by the Discrete Cosine Transform (DCT) unit. The resulting DCT coefficients are quantized by the quantizer according to user-programmable quantization matrices. The CL550 allows up to four 64-word quantization matrices to be stored on-chip, and provides programmable sequence registers to allow the user to select the appropriate matrix for each component block.

The quantized terms are then serialized by the Zig-zag scan unit and the AC terms are 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 Differential Pulse Code Modulation (DPCM) calculations on the DC terms, and performs Huffman coding of both the DC and the AC terms. Huffman codes are specified by the user, and stored in on-chip table RAM that is loaded at initialization.

The Huffman codes are finally sent to the Host interface as JPEG compressed data. The Host interface is designed to operate in either slave mode or master mode. In slave mode, the CL550 acts as a peripheral device to the host processor, using a data request/data available handshake to control the transfer of data. In master mode, the CL550 works in conjunction with an external DMA controller chip to allow high-speed DMA transfers of data. The Host interface is explained in detail in Chapter 4, Host Interface.

Compression operations follow the opposite procedure. JPEG compressed data is written to the Host interface. The Host interface then transfers the data to the Huffman CODEC, where it is 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 8 x 8 block format. The DC terms are treated separately. Dequantization and Inverse DCT (IDCT) are then performed on the reassembled blocks before they are

sent to the Block Storage unit. The Video interface optionally performs YUV-to-RGB color space conversion of the pixel data, realigns the 8 x 8 block data as raster lines, and outputs the lines to the external video display device.

With this architecture, it is possible to construct very high-performance compression systems for still-frame applications or motion video. The CL550 parts can be reinitialized on a frame-by-frame basis, allowing the programmer to change compression ratios at the end of each frame. It also allows systems to be designed where the CL550 switches back and forth between compressing and decompressing frames for half-duplex image communication.

## 1.6 CL560 Functional Description

This section describes the functional characteristics of each block within the C-Cube CL560 processor. Figure 1-2 shows the processor's major functional blocks. The CL560 is a highly pipelined machine with over 320 processing stages in the data path. Each stage in the JPEG Baseline Sequential Process is implemented within this pipeline. The major difference between the CL560 architecture and the CL550 architecture is in the Huffman CODEC. The synchronous CODEC in the CL560 allows data to be encoded or decoded in a single clock cycle, whereas the asynchronous CODEC in the CL550 takes several clock cycles, thus allowing higher throughput.

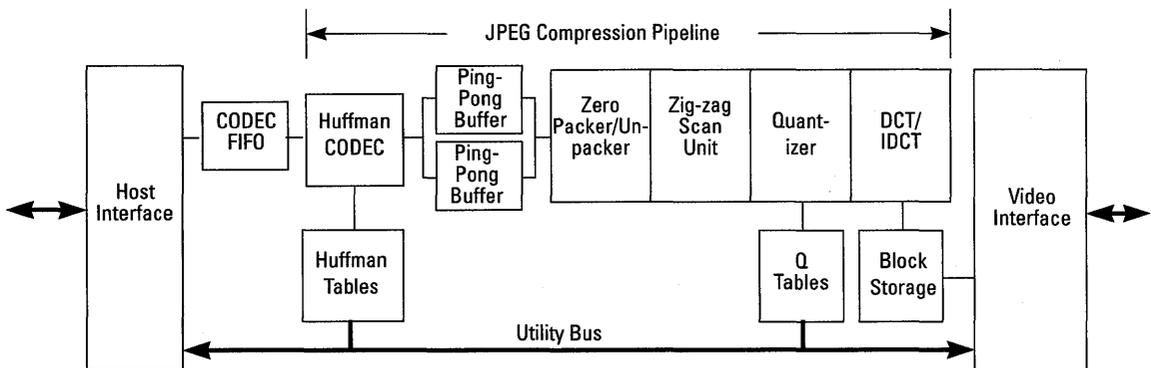


Figure 1-2 CL560 Block Diagram

During compression operations, uncompressed pixel data is written into the Video interface. The first operation that the Video interface performs is a raster-to-block conversion of the pixel data. This operation is necessary because video generation and display devices normally deal with pixel data as raster lines, while the JPEG compression algorithm requires that the pixel data be organized as 8 x 8 blocks. Logic in the CL560 device performs that conversion.

The next step is the optional RGB-to-YUV color space conversion, also performed by the Video interface. Video generation and display devices frequently present data to the CL560 as RGB pixels. The CL560 also performs optional color space conversion. Other functions done by the Video interface are pixel formatting and window sizing.

Once the video interface is through formatting the pixel data, it writes the data into the Block Storage unit. The Block Storage unit stores the 8 x 8 blocks until the JPEG compression pipeline is ready to process them. It then sequences them into the pipeline one block at a time.

Each component block is then processed by the Discrete Cosine Transform (DCT) unit. The resulting DCT coefficients are quantized by the quantizer according to user-programmable quantization matrices. The CL560 allows up to four 64-word quantization matrices to be stored on-chip, and provides programmable sequence registers to allow the user to select the appropriate matrix for each component block. Up until this point, the CL560 compression process has been identical to the CL550 compression process.

The quantized terms are then serialized by the Zig-zag scan unit and the AC terms are run-length coded by the Zero Packer/Unpacker unit before being loaded into the Ping-pong buffer. The Ping-pong buffer is a pair of synchronous 64-word registers used to smooth the flow of data to and from the Huffman CODEC.

The Huffman CODEC draws the packed symbols from the Ping-pong buffer, performs Differential Pulse Code Modulation (DPCM) calculations on the DC terms, and performs Huffman Coding of both the DC and the AC terms. Huffman codes are specified by the user, and stored in on-chip table RAM that is loaded at initialization.

The Huffman codes are then stored in a 128 x 32 CODEC FIFO. The FIFO acts as a rubber-band buffer between the synchronous JPEG com-

pression pipeline and the asynchronous Host Bus interface. The FIFO is used to filter out fluctuations in the data rate. It allows fast-burst access to the CL560 to minimize the time needed to transfer data.

The Host interface is designed to operate in either register access mode or DMA access mode. In register access mode, the CL560 acts as a peripheral device to the host processor, using a data request/data available handshake to control the transfer of data. In DMA access mode, the CL560 works in conjunction with an external DMA controller chip to allow high-speed DMA transfers of data. The Host interface is explained in detail in Chapter 4, Host Interface.

Compression operations follow the opposite procedure. JPEG compressed data is written to the Host interface. The Host interface then stores the compressed data in the CODEC FIFO until it can be transferred to the Huffman CODEC for decoding. After decoding, the packed symbols are stored in the Ping-pong buffer. The Zero Packer/Unpacker Unit reads the Ping-pong buffer to retrieve the packed symbols, generates the AC values, and passes them to the Zig-zag Scan unit for reordering into 8 x 8 block format. The DC terms are treated separately. Dequantization and Inverse DCT (IDCT) are then performed on the reassembled blocks before they are sent to the Block Storage unit. The Video interface optionally performs YUV -to-RGB color space conversion of the pixel data, realigns the 8 x 8 Block data as raster lines, and outputs the lines to the external video display device.

With this architecture, it is possible to construct very high-performance compression systems for both video and still-frame applications. The CL560 parts can be reinitialized on a frame-by-frame basis, allowing the programmer to change compression ratios at the end of each frame. It also allows systems to be designed where the CL560 switches back and forth between compressing and decompressing frames for half-duplex image communication.

# 2

## JPEG Overview

This chapter presents an overview of the JPEG video compression standard. The chapter is divided into these sections:

- 2.1, JPEG Background Information
- 2.2, Operation of the JPEG Algorithm
- 2.3, Discrete Cosine Transform
- 2.4, Quantization
- 2.5, Zero Run-Length Coding
- 2.6, Entropy Encoding
- 2.7, Summary of JPEG Baseline

---

**2.1**  
**JPEG Background**  
**Information**

---

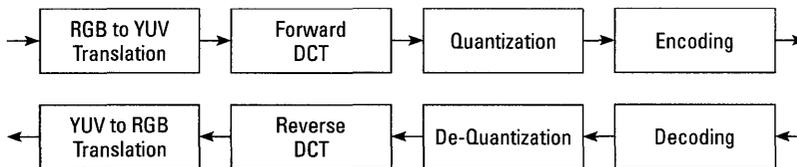
The obvious advantages of digital image compression led to the formation of an international standards group: the Joint Photographic Experts Group (JPEG). JPEG is a joint ISO/CCITT technical committee (ISO/IEC JTC1/SC2/WG10, Photographic Image Coding) whose goal has been to develop a general-purpose 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.

The operation of the Baseline JPEG algorithm can be divided into three basic stages, as shown in Figure 2-1:

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.



**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, YCbCr, and CMYK.

However, the best compression results are achieved if the color components are independent (noncorrelated), such as in YCbCr, 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 YCbCr components, as shown in Table 2-1.

---

## 2.2 Operation of the JPEG Algorithm

---

**Table 2-1 Converting RGB Components to YCbCr Components**

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.169 & -0.3316 & 0.0500 \\ 0.500 & -0.4186 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

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

---

**2.3**  
**Discrete Cosine Transform**

---

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

In the 8 x 8 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 8 x 8 DCT. This transform, studied extensively for image compression, is extremely efficient for highly correlated data.

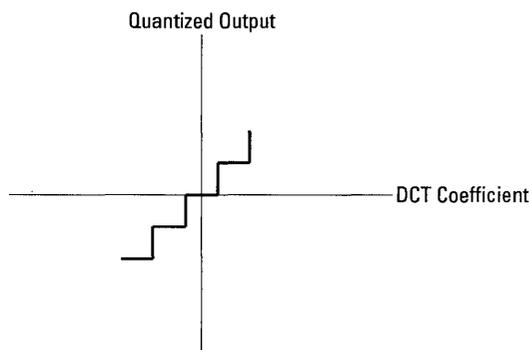
Conceptually, a one-dimensional DCT can be thought of as taking the Fourier Transform and retaining only the real (the cosine) part. The two-dimensional DCT can be obtained by performing a one-dimensional DCT on the columns and then a one-dimensional DCT on the rows. The transformed output from the two-dimensional DCT is ordered such that the mean value, the DC coefficient, is in the upper left corner of the 8 x 8 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.

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 Figure 2-2 and Figure 2-3. Figure 2-3 illustrates two functional matrices that have been optimized for CCIR 601 imagery.

---

## 2.4 Quantization

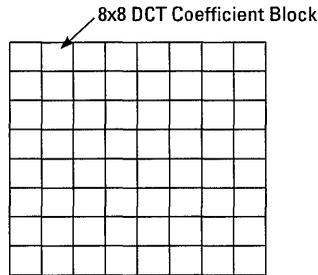
---



**Figure 2-2** Quantizer Stepping (Uniform Quantization)

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 (shown graphically in Figure 2-4 and numerically in Table 2-2) 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.

# Quantization

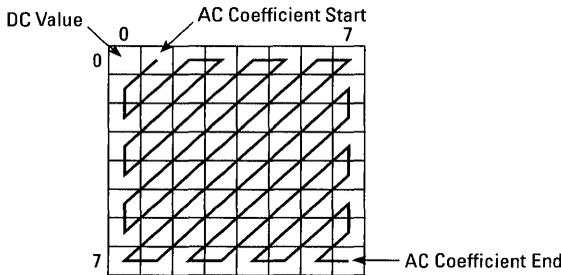


<i><b>Y Component Matrix</b></i>							
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	58	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

<i><b>Cb Cr Component Matrix</b></i>							
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** Psychovisual Weighting Functions for the Luminance and Chrominance Components



**Figure 2-4 Zigzag Pattern for Reordering the 8 x 8 DCT Coefficients**

**Table 2-2 Zigzag Sequence of Quantized DCT Coefficients**

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

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:

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

The inverse calculation takes place at the decoder.

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

---

## 2.5 Zero Run-length Coding

---

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.

---

### 2.6 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.

---

### 2.7 Summary of JPEG Baseline

---

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 8 x 8 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.

# 3

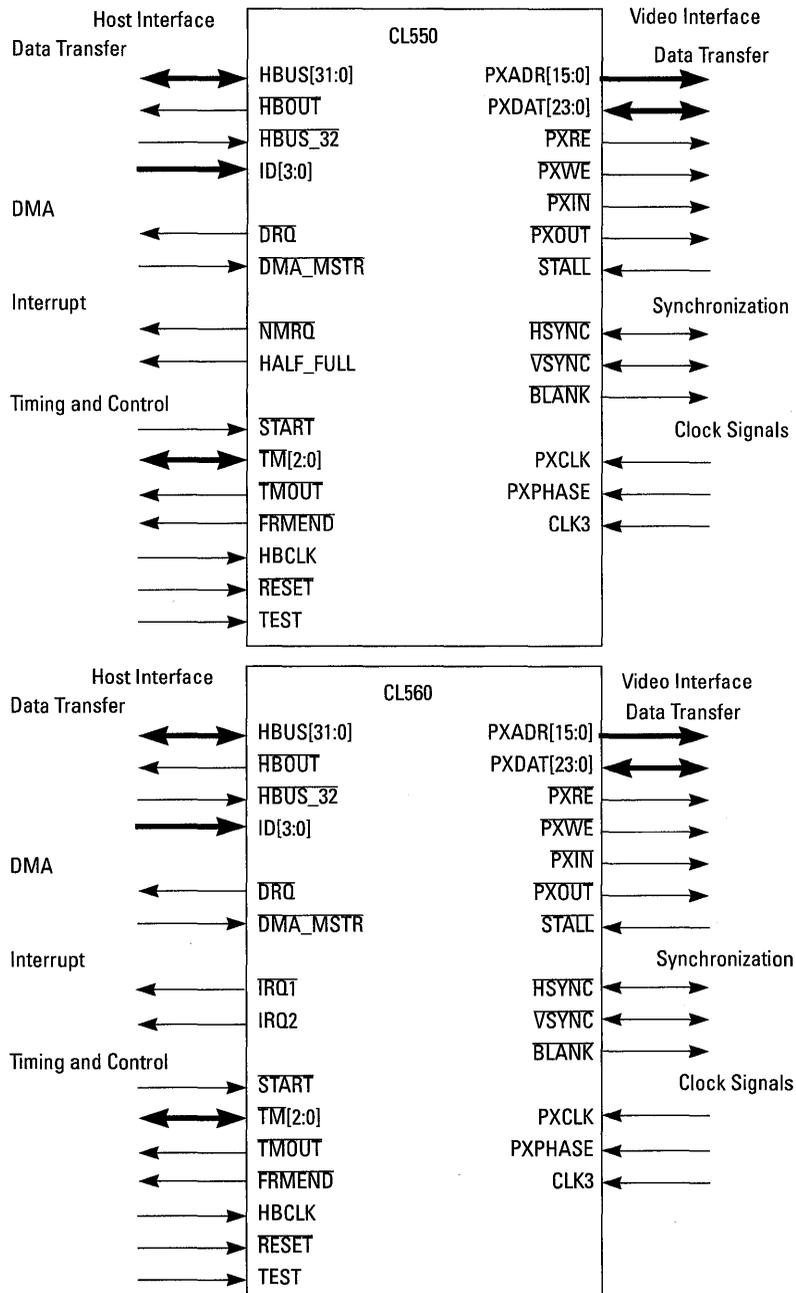
## Signal Descriptions

This chapter describes the signals that comprise the external physical interface to the CL550 and CL560. The information presented for each signal includes the signal name and mnemonic, type (input, output, or bidirectional), and description. For information about the functional operation of these parts, see Chapters 4 and 5. For timing information, see Chapter 6.

This chapter is divided into two sections that correspond to the components that interface to the CL550 and CL560:

- 3.1, Host Interface
- 3.2, Video Interface

Figure 3-1 shows a diagram of each of the parts with the various signals grouped together by function.



**Figure 3-1 CL550 and CL560 Logic Diagrams**

The CL550 external interface differs from the CL560 by only two pins:

**Table 3-1 CL550 and CL560 Redefined Signal Pins**

CL560 Signal	CL550 Signal	CPGA Pin #	MQUAD Pin #
IRQ1	NMRQ	70	D13
IRQ2	HALF_FULL	135	L2

The Host Interface signals divide logically into these functional groups:

- *Data transfer signals:* These signals comprise the address and data bus and various control signals used to complete the data transfer handshake.
- *DMA signals:* These signals are used to implement a handshake during a DMA transfer.
- *Interrupt signals:* These signals provide interrupt requests to the host processor.
- *Timing, control and status signals:* These signals include the clocks and reset signals.

## 3.1 Host Interface

### 3.1.1 Data Transfer Signals

#### HBUS[31:0] Host Bus

#### Bidirectionals

HBUS is the multiplexed host processor data and address bus. The width of both the address bus and the data bus can be programmed to be 16-bits or 32-bits wide. The signals **HBUS\_32** and **ID[3:0]** determine the widths as shown in Table 3-2. **HBUS\_32** and **ID[3:0]** are discussed later in this section.

**Table 3-2 Address and Data Bus Configuration**

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 (Fh)	0/1	Chip disabled

**HBOUT      Host Bus Output      Output**

The Host Bus Output signal controls the direction of the host bus transfer. The CL550 and CL560 do not have sufficient drive capability to be connected directly to most computer host buses. When external drivers are used, HBOUT controls the direction of these buffers. When this signal is low (0), the transceiver direction is from the CL550 or CL560 to the host bus (output). When this signal is high (1), the direction is from the host bus to the CL550 or CL560 (input).

**HBUS\_32      Host Bus Width = 32-bits      Input**

HBUS\_32 is a static signal used to configure the width of the host bus data path during CODEC accesses. When HBUS\_32 is low (0), reads and writes to the CODEC register are 32-bits wide. When HBUS\_32 is high (1), reads and writes to the CODEC register are 16-bits wide. The data path to all on-chip registers except the CODEC register is always 16-bits wide.

**ID[3:0]      Address Space ID Signals      Inputs**

The address space identification signals, ID[3:0], are inputs that select the address range of the chip. Setting ID[3:0] to 0x1 through 0xE selects an address region for the CL550 or CL560. The ID signals allow the CL550 or CL560 to be placed in one of fourteen locations in the upper 1/16 of the memory.

**3.1.2 DMA Signals**

The CL560 is capable of acting as either a bus slave for CODEC transfers, or a bus master when used with an external DMA controller. DMA transfers are fully discussed in Chapter 4, Host Interface.

**DRQ      DMA Request      Open-Drain Output**

The DRQ signal is an output that provides chip status for DMA interface control. The DRQ output is controlled by the Flag Register bits, and enabled using the DMA mask register described in Chapter 7, Registers. The DRQ output is an open-drain output and should be tied to VCC through a resistor of at least 625 ohms.

**DMA\_MSTR      DMA Master      Input**

DMA\_MSTR is an input that allows the CL560 to work with a DMA controller functioning as a bus master for CODEC trans-

fers. It is sampled on the falling edge of HBCLK when the **START** signal is active. DMA transfers are fully discussed in Chapter 4, Host Interface.

*Note: The CL550 does not work correctly in the DMA master mode, and in systems that use the CL550, **DMA\_MSTR** should always be held HIGH. Refer to Chapter 4, Host Interface for a solution to this problem.*

### 3.1.3 Interrupt Signals

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

*Note:  $\overline{\text{NMRQ}}$  is a CL550 signal only. The CL560 uses  $\overline{\text{IRQ1}}$  instead.*

Interrupt Request ( $\overline{\text{NMRQ}}$ ) is an unlatched output signal, synchronous to HBCLK, that provides an indicator of both FIFO and video field status. It can be programmed to selectively indicate active status flags as specified in the Interrupt Mask Register. This signal is an open-drain output and should be tied to VCC through a resistor of at least 625 Ohms (4.7K Ohms recommended). On power-up, the CL550 or CL560 should be hardware reset to prevent the generation of spurious interrupts.

**$\text{HALF\_FULL}$**       **FIFO HALF\\_FULL**      **Output**

*Note:  $\text{HALF\_FULL}$  is a CL550 signal only. The CL560 uses  $\text{IRQ2}$  instead.*

The  $\text{HALF\_FULL}$  signal is an output that indicates the status of the internal FIFO. A value of 1 (HIGH) indicates that the FIFO contains at least 64 entries out of 128. Transitions of  $\text{HALF\_FULL}$  are synchronous to PXCLK.

**$\overline{\text{IRQ1}}$**       **Interrupt Request**      **Open-Drain Output**  
 **$\text{IRQ2}$**       **Interrupt Request**      **Output**

*Note:  $\overline{\text{IRQ1}}$  and  $\text{IRQ2}$  are CL560 signals only. The CL550 uses  $\overline{\text{NMRQ}}$  and  $\text{HALF\_FULL}$  instead.*

$\overline{\text{IRQ1}}$  and  $\text{IRQ2}$  are general-purpose status outputs from the CL560. The assertion of these signals is programmable based on masks contained in the  $\overline{\text{IRQ1}}$  and  $\text{IRQ2}$  mask registers described in Chapter 7, Registers. Transitions of  $\text{IRQ2}$  are syn-

chronous to HBCLK.  $IRQ1$  is an open-drain output while  $IRQ2$  has a totem-pole output.

### 3.1.4 Timing and Control Signals

A host bus transaction consists of two (or more) bus clock cycles. During the first cycle, the Start cycle, the host processor must indicate to the CL550 or CL560 what kind of transaction will occur (bus slave mode read, bus master mode write, etc.) by placing specific values on  $TM[2:0]$  and  $DMA\_MSTR$ .  $TM[2:0]$  act as outputs during the last bus clock cycle of the transaction, called the Acknowledge cycle. The value output on  $TM[2:0]$  indicates either that the transfer completed successfully or that an error occurred. In between the start and the acknowledge cycle, an indeterminate number of wait cycles can occur. Host bus transactions are fully discussed in Chapter 4, Host Interface.

#### **START      Start a Transfer      Input**

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

#### **TM0      Transfer Mode 0      Bidirectional**

In bus slave mode operation, **TM0** is an output line that transitions to 0 (along with **TMT**) during the bus transaction acknowledge cycle to indicate that the transaction completed. In CL550 bus master mode only, **TM0** is an input, sampled during the assertion of **TM2** (along with **TMT**), to determine whether a bus error has occurred. The value of **TM0** is ignored during a Start cycle.

#### **TMT      Transfer Mode 1      Bidirectional**

The CL550 and CL560 sample the **TMT** input during the Start cycle to determine whether the transaction is a read or a write. A low value (0) indicates a write cycle, and a high value (1) indicates a read cycle. During a bus slave cycle, **TMT** returns to 0 during the acknowledge cycle. During a bus master mode Start cycle, the sense of this signal is inverted so that a low value indicates a write cycle and a high value indicates a read cycle.

**TM2                      Transfer Mode 2                      Bidirectional**

**TM2** is the acknowledge signal that is driven active LOW (0) by the CL550 or CL560 during a bus slave mode acknowledge cycle. In bus master mode, this signal is driven by an external source to indicate that the transfer is complete.

**TMOU                      Transfer Mode Output                      Output**

The **TMOU** signal provides transceiver directional control for the transfer mode control lines **TM**[2:0]. If this signal is low (0), the transceiver direction is from CL550 or CL560 out to the host bus. If the signal is high (1), the transceiver direction is from the host bus into the CL550 or CL560.

**FRMEND                      Frame End                      Open-Drain Output**

The **FRMEND** signal is an output that indicates that the end of an image has been reached. This signal is an open-drain output and should be tied to VCC through a resistor of at least 625 Ohms (recommended value = 4.7K Ohms). In the CL550, during compression, **FRMEND** goes active when the Huffman coder has removed the last word from the FIFO. During decompression, **FRMEND** indicates that the last word has been removed from the Strip buffer RAM. This signal can be disabled by setting bit 1 of the Configuration register to a zero (See Chapter 7, Registers). In the CL560 this signal is also controlled by the Frame End Enable register (See Chapter 7, Registers). Transitions of **FRMEND** are synchronous to **HBCLK**.

**HBCLK                      Host Bus Clock                      Input**

**HBCLK** is the clock signal used to synchronize host bus data transfers. 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. **HBCLK** must be the same rate or slower than **PXCLK**.

**RESET                      Reset                      Input**

The **RESET** signal is an input that forces a hardware reset of the CL550 or CL560. When the **RESET** signal is asserted LOW, most of the internal registers are forced to a known state. The values in the Huffman tables, DCT table, and Quantizer tables are unaffected. **HBCLK** and **PXCLK** must be running during **RESET**. The CL550 or CL560 will not acknowledge any access until the end of the third cycle after **RESET** has been deasserted.

**TEST                      Test    Input**

The TEST signal is an input the forces all CL550 or CL560 outputs to a high-impedance state. This feature is provided to simplify board-level diagnostics. TEST should be tied low for normal operation.

---

**3.2  
Video Interface**

---

The Video Interface signals divide logically into these functional groups:

- *Pixel Bus Data Transfer Signals:* These signals comprise the video data bus, the Strip Buffer address bus, and the handshake signals necessary to transfer data.
- *Video Synchronization Signals:* These are the signals used to control the horizontal and vertical placement of the video frame.
- *Video Clock Signals:* These are the timing signals necessary for the CL550 and CL560 to operate.

**3.2.1 Pixel Data, Address and Handshake Signals**

**PXDAT[23:0] Pixel Data Bus    Bidirectionals**

PXDAT is a bidirectional 24-bit bus that handles uncompressed or decompressed pixel data. In the compression mode, uncompressed video data is input on PXDAT[23:0] and compressed data is output on the host bus. In the decompression mode, compressed data is input on the host bus and decompressed video data is output on PXDAT[23:0]. PXDAT is also used to transfer data to and from the strip buffer RAM. In some modes (Gray-scale, YUV 4:2:2 and CMYK), only 16 of the 24 bits are used. The unused pins should be tied to ground through 10K-ohm resistors.

**PXADR[15:0] Pixel Address    Outputs**

PXADR is the address bus for the strip buffer RAM. The 16 bits of address support a strip buffer of up to 64K entries.

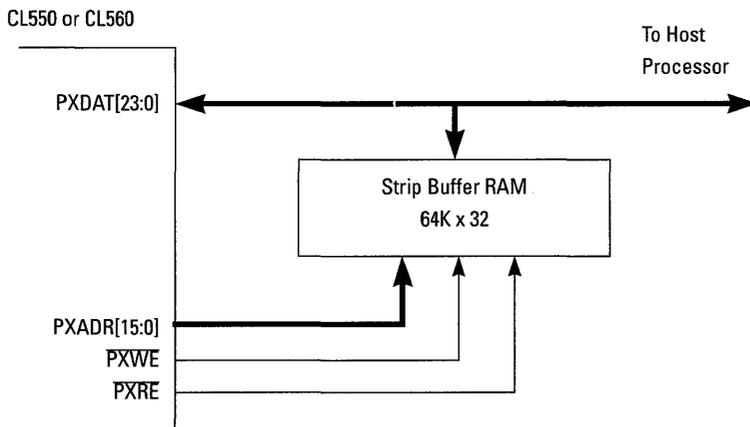
**PXRE                      Pixel Read    Output**

The PXRE output signal is designed to directly control the output enable pin of the strip buffer RAMs. During compression, PXRE is active only when the CL550 or CL560 is reading pixel data from the strip buffer RAM. During decompression, PXRE

is active only when pixels are being read from the strip buffer RAM out to the pixel destination.

### **PXWE** Pixel Write Output

The  $\overline{\text{PXWE}}$  output is designed to directly control the write enable input to the strip buffer RAMS. During compression,  $\overline{\text{PXWE}}$  is active only during PXIN cycles when pixel data is being written from the active portion of the video field into the strip buffer RAM. During decompression,  $\overline{\text{PXWE}}$  is active only when active pixels are being written from the CL550 or CL560 into the strip buffer RAM.



**Figure 3-2 Strip Buffer RAM Connections**

### **PXIN** Pixel Input Control Output

The  $\overline{\text{PXIN}}$  signal is used to activate an input buffer on the Pixel Data bus, PXDAT, during input cycles. It is asserted (LOW) only when pixel data is being input from the active portion of the video field into the Strip buffer RAM.

### **PXOUT** Pixel Output Control Output

The  $\overline{\text{PXOUT}}$  signal can be used to load the active pixel into a register as it is read out of the Strip buffer RAM. It is asserted (LOW) only when pixels from the active region of the field are being read from the Strip buffer RAM.

<b>STALL</b>	<b>Stall</b>	<b>Input</b>
--------------	--------------	--------------

This input signal, when asserted (LOW), will stop all activity on the Video Interface in its current state. Signals affected by **STALL** include **PXADR[16:0]**, **PXDAT[23:0]**, **PXRE**, **PXWE**, **PXIN**, **PXOUT**, **BLANK**, **VSYNC** and **HSYNC**. Use of the **STALL** signal is discussed completely in Chapter 5, Video Interface.

### 3.2.2 Video Synchronization Signals

<b>HSYNC</b>	<b>Horizontal Sync</b>	<b>Bidirectional</b>
--------------	------------------------	----------------------

**HSYNC** is a bidirectional signal used to indicate the start of a horizontal line. When the CL560 is programmed for master mode operation (Configuration register bit 3 = 1), the **HSYNC** signal functions as an output and is asserted (LOW) when the CL560 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.

<b>VSYNC</b>	<b>Vertical Sync</b>	<b>Bidirectional</b>
--------------	----------------------	----------------------

**VSYNC** is a bidirectional signal used to indicate the start of a frame. When the CL560 is programmed for master mode operation (Configuration register bit 3 = 1), the **VSYNC** signal functions as an output and is asserted (LOW) when the CL560 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.

<b>BLANK</b>	<b>Blanking</b>	<b>Output</b>
--------------	-----------------	---------------

This signal is an output that indicates that there are no active pixels on the Pixel Data bus. **BLANK** changes state at the same time as the **PXADR** bus (at the beginning of the Strip buffer read cycle, when **PXPHASE** is HIGH). During compression, **BLANK** goes HIGH one **PXCLK** before the first pixel in a line





# 4 Host Interface

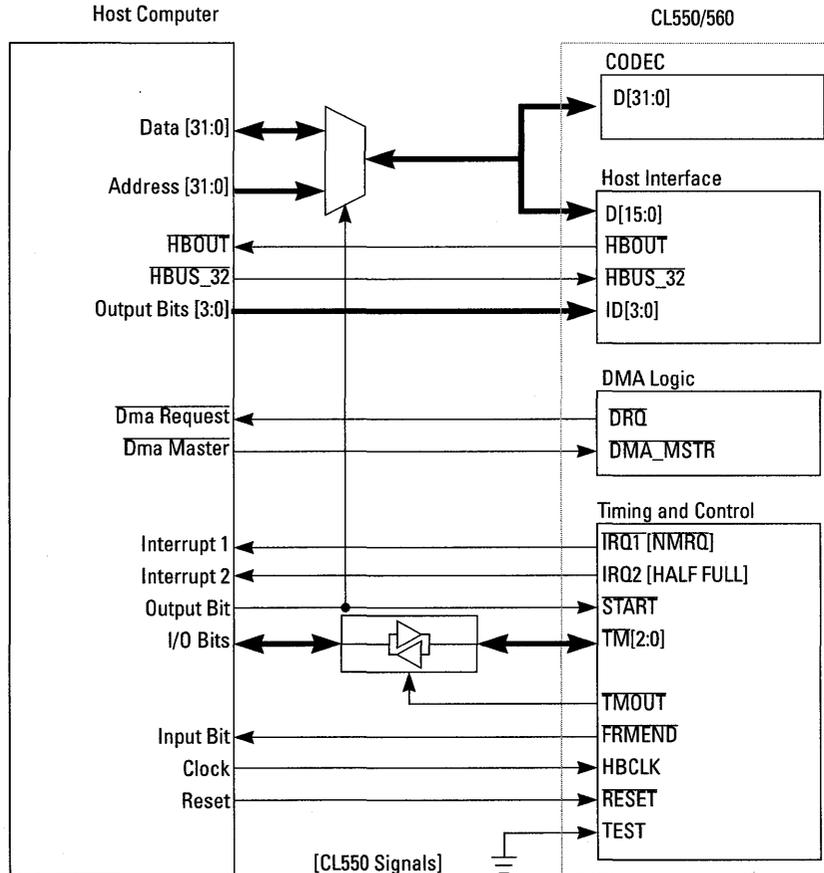
The Host interface on the CL550 and CL560 is designed to be connected to a variety of general-purpose microprocessors with a minimum of external logic. The host processor can directly access any CL550/560 register or memory address by reading or writing specific memory addresses. The host can also access CL560 data using DMA transfers. DMA transfers are provided as a fast method of transferring data to and from the CODEC FIFO. The host bus interface also includes signals for timing and control, status and interrupt processing.

This chapter is divided into sections which describe how the host interface is used. These sections are:

- 4.1, Register Access Timing
- 4.2, DMA Access Timing
- 4.3, Control Signals

Figure 4-1 shows the host bus pinout diagram of the CL550 and CL560 (CL550 signal names are enclosed in parenthesis). The host interface

signals include a 32-bit wide dual purpose data and address bus and the control signals necessary to perform data transfers and interrupt handling.



**Figure 4-1 CL550 and CL560 Host Interface Block Diagram**

The CL560 host bus interface differs from the CL550 interface in several ways:

- The CL550 signals  $\overline{\text{NMRQ}}$  and HALF\_FULL were replaced with the general-purpose interrupts  $\overline{\text{IRQ1}}$  and  $\overline{\text{IRQ2}}$  in the CL560. The function of  $\overline{\text{IRQ1}}$  and  $\overline{\text{IRQ2}}$  is programmed using the  $\overline{\text{IRQ1}}$  and

IRQ2 Interrupt Mask registers (See Chapter 7, Registers).

- The CL550 signals  $\overline{DRQ}$  (data request) and  $\overline{FRMEND}$  (frame end) operate differently in the CL560, although they retain the same name. These differences are described in Chapter 3, Signal Description.
- The CL560 samples the input data at a different point than the CL550. This difference is described in Chapter 6, Specifications.
- The signals  $\overline{TM0}$ ,  $\overline{TM1}$  and  $\overline{TM2}$  have timing differences between the CL550 and the CL560. These differences are described in Chapter 6, Specifications.
- The signals  $\overline{HBOUT}$  and  $\overline{TMOU}$  have timing differences between the CL550 and the CL560. These differences are described in Chapter 6, Specifications.

If you are designing a system that will accept both the CL550 and the CL560, the only signals that have had their external function redefined are:

**Table 4-1 CL550 and CL560 Redefined Signal Pins**

CL560 Signal	CL550 Signal	CPGA Pin #	MQUAD Pin #
TRQ	NMRQ	70	D13
IRQ2	HALF_FULL	135	L2

This section describes the timing for register accesses. It is divided into these subsections:

- 4.1.1, Signal Descriptions
- 4.1.2, Register Access Timing
- 4.1.3, Host Bus Register Access
- 4.1.4, Host Bus Register Write

#### 4.1.1 Signal Descriptions

All CL550/560 family registers and memory are accessed using register accesses except the CODEC register. The CODEC register can be accessed using either register accesses (described in this section) or DMA transfers (see Section 4.2). The following signals are used to access the CL550/560 part in register access mode:

---

## 4.1 Register Access Timing

---

- **HBCLK, Host Bus Clock:** All host bus accesses are synchronized to the Host Bus Clock. 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.
- **HBUS[31:0], Host Bus:** HBUS[31:0] is a multiplexed data and address bus. The signal **START** is used to indicate that an address is present on the bus. The width of the host bus can be either 16 or 32-bits wide depending on the signals ID[3:0] and HBUS\_32.
- **START, Transaction Start:** The **START** signal begins a data transfer. When asserted, it indicates that there is a valid address on HBUS[31:0]. **START** is sampled on the falling edge of HBCLK, and should not be asserted for more than one HBCLK period.
- **ID[3:0], Address Space Identification Signals:** ID[3:0] are inputs that select the address range of the chip. When the part receives a **START** signal, it compares the value on ID[3:0] with the address on HBUS[31:0] bits 27 through 24. If a match occurs, an internal chip select signal is generated.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	1	1	ID3	ID2	ID1	ID0	1	1	X	X	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
On-chip Register Address															

**Figure 4-2 ID[3:0] Chip Select Address Format**

To generate a valid chip select, the contents of HBUS[31:16] should be as shown in Figure 4-2, where X = Don't Care. This allows the CL550/560 part to be placed in any one of fourteen locations in the upper 1/16th of memory.

Setting ID to 0x0 overrides the decoding of the upper 16-bits of the address, putting the CL550/560 part into a 16-bit address mode. Setting ID to 0xF disables the host bus interface.

- **HBUS\_32, 32-bit Host Bus:** HBUS\_32 is a static signal used to configure the host bus data path width during CODEC accesses. When HBUS\_32 is low, the read/write path to the CODEC register is 32-bits wide. When HBUS\_32 is high, the read/write path to

the CODEC register is 16-bits wide. The data path to all on-chip registers and tables except for the CODEC register is always 16-bits wide. In the 16-bit data mode, HBUS[31:16] remain disabled (three-stated) at all times and transfers take place over HBUS[15:0].

HBUS\_32 and ID[3:0] combine to control the address and data modes as shown in Table 4-2.

**Table 4-2 Address and Data Bus Configuration**

ID[3:0]	HBUS_32	Description
0000	0	16-bit address, 32-bit data
0000	1	16-bit address, 16-bit data
0001-1110	0	32-bit address, 32-bit data
0001-1110	1	32-bit address, 16-bit data
1111	0 or 1	Chip Disabled

- **TM0, Transfer Mode 0:** In register access mode operation, TM0 is an output line that transitions to 0 (along with TMI) during the bus transaction acknowledge cycle to indicate that the transaction completed. The value of TM0 is ignored during a Start cycle.
- **TMI, Transfer Mode 1:** The CL550/560 part samples the TMI input during the Start cycle to determine whether the transaction is a read or a write. TM1 changes sense between register access cycles and DMA access cycles. During register access cycles, a low value (0) on TM1 indicates a write cycle, and a high value (1) indicates a read cycle. During DMA cycles, a high value (1) on TM1 indicates a write cycle, and a low value (0) indicates a read cycle. During a register access cycle, TMI returns to 0 during the acknowledge cycle.
- **TM2, Transfer Mode 2:** TM2 is the acknowledge signal that is driven active (0) by the CL550/560 family during a register access acknowledge cycle.
- **TMOUT, Transfer Mode Control Lines = Outputs:** The TMOUT signal provides transceiver directional control for the transfer mode control lines TM[2:0]. If this signal is low (0), the transceiver direction is from the CL550/560 part out to the host bus. If the signal is high (1), the transceiver direction is from the

host bus into the part.

The tables below show the values of the CL550/560 control signals during each of the three bus access cycles. Table 4-3 and Table 4-4 show the CL550 control signals, and Table 4-5 through Table 4-8 show the CL560 control signals. The shaded areas indicate that the signals are driven as outputs.

**Table 4-3 TM Signals During a Host Bus Register Read (CL550 to Host)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>DMA_MSTR</b>	HIGH	HIGH	HIGH
<b>TM0</b>	Don't Care	LOW	LOW
<b>TM1</b>	HIGH	LOW	LOW
<b>TM2</b>	HIGH	HIGH	LOW
<b>TMOUT</b>	HIGH	LOW	LOW

**Table 4-4 TM Signals During a Host Bus Register Write (Host to CL550)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>DMA_MSTR</b>	HIGH	HIGH	HIGH
<b>TM0</b>	Don't Care	LOW	LOW
<b>TM1</b>	LOW	LOW	LOW
<b>TM2</b>	HIGH	HIGH	LOW
<b>TMOUT</b>	HIGH	LOW	LOW

**Table 4-5 TM Signals During a Host Bus Register Read (CL560 to Host)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>DMA_MSTR</b>	HIGH	HIGH	HIGH
<b>TM0</b>	Don't Care	Don't Care	LOW
<b>TM1</b>	HIGH	Don't Care	LOW
<b>TM2</b>	HIGH	Don't Care	LOW
<b>TMOUT</b>	HIGH	HIGH	LOW

**Table 4-6 TM Signals During a Host Bus Register Write (Host to CL560)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>DMA_MSTR</b>	HIGH	HIGH	HIGH
<b>TM0</b>	Don't Care	Don't Care	LOW
<b>TM1</b>	LOW	Don't Care	LOW

**Table 4-6 TM Signals During a Host Bus Register Write (Host to CL560)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>TM2</b>	HIGH	Don't Care	LOW
<b>TMOUT</b>	HIGH	HIGH	LOW

**Table 4-7 TM Signals During a DMA Read (CL560 to Host)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>DMA_MSTR</b>	LOW	HIGH	HIGH
<b>TMO</b>	Don't Care	Don't Care	Don't Care
<b>TM1</b>	LOW	Don't Care	Don't Care
<b>TM2</b>	HIGH	HIGH	LOW
<b>TMOUT</b>	HIGH	HIGH	HIGH

**Table 4-8 TM Signals During a DMA Write (Host to CL560)**

Signal	Start Cycle	Wait State(s)	Acknowledge Cycle
<b>DMA_MSTR</b>	LOW	HIGH	HIGH
<b>TMO</b>	Don't Care	Don't Care	Don't Care
<b>TM1</b>	HIGH	Don't Care	Don't Care
<b>TM2</b>	HIGH	HIGH	LOW
<b>TMOUT</b>	HIGH	HIGH	HIGH

- **HBOUT, Host Bus = Output:** The HBOUT signal is used to control the direction of the host bus transfer. The CL550/560 does not have sufficient drive capability to be connected directly to most computer host buses. When external drivers are used, HBOUT is used to control the direction of these buffers. When this signal is low (0), the transceiver direction is from the CL550/560 to the host bus (output). When this signal is high (1), the direction is from the host bus to the part (input).

#### 4.1.2 Register Access Timing

A host bus transaction consists of two or more HBCLK clock cycles. Memory or register accesses (except accesses to the CODEC register) always take three cycles: start, wait and acknowledge. Accesses to the CODEC register can take from 2 to n cycles depending on the availability of the CODEC register.

The first cycle is called the start cycle, and is initiated by the host processor. For all register and table locations (except the CODEC), the start cycle is followed by exactly one wait state. During an access to the CL550 CODEC register, the start cycle is followed by several wait states. The last cycle is called the acknowledge cycle, and is initiated by the CL550/560 to show that it is through reading or writing data. The acknowledge cycle is indicated by  $\overline{\text{TM2}}$  being asserted low (0).

A start cycle is initiated by asserting the  $\overline{\text{START}}$  input.  $\overline{\text{START}}$  is sampled on the falling edge of HBCLK, and should never be asserted for more than one HBCLK period. It should also never be asserted twice before an acknowledge cycle occurs. When  $\overline{\text{START}}$  is sampled LOW, the CL550/560 samples HBUS for the register address,  $\overline{\text{TM1}}$  to determine the direction of the transfer, and  $\overline{\text{TM2}}$  which must be HIGH during the  $\overline{\text{START}}$  cycle. If  $\overline{\text{TM2}}$  is low during start, the cycle will be ignored and no acknowledge will be returned. Typical system designs use a pull-up resistor on  $\overline{\text{TM2}}$  for this purpose.

The CL560 has a CODEC FIFO, but the CL550 only has a CODEC register. However, the timing for accessing either is identical. Where the following section refers to the CODEC FIFO, substitute CODEC register when working with the CL550.

The CL550/560 always inserts at least one wait state between the start cycle and the acknowledge cycle during register mode accesses. If the register being addressed is any register other than the CODEC FIFO, then exactly one wait state is inserted. CODEC register or FIFO accesses can contain zero or more wait states, depending on the condition of the CODEC FIFO at the time of the access. If the host performs a register read of the CODEC FIFO when the CL560 is in the compression mode, and there is data available in the FIFO, no wait states will be inserted. Otherwise, wait states will be inserted until a data word is available. If the host performs a register write of the CODEC FIFO when the 560 is in the decompression mode, and there is an empty location available in the FIFO, no wait states will be inserted. Otherwise, wait states will be inserted until an empty location becomes available.

Note: Refer to Section 4.2.7, Operational Considerations, for more information on using the CODEC FIFO and Register.

In the CL550 only,  $\overline{TM}[2:0]$  are used during the acknowledge cycle to transmit status information. In register access mode, the CL550 outputs  $\overline{TM}[1:0] = 00$  to indicate that the transaction is complete. In DMA transfer mode, the CL550 reads  $\overline{TM}[1:0]$  looking for error information coming back from the host bus. If either  $\overline{TM1}$  or  $\overline{TM0} = 1$ , the CL550 sets the internal Bus Error Flag. The bus error flag status is determined as follows:

**Table 4-9 CL550 Bus Error Conditions**

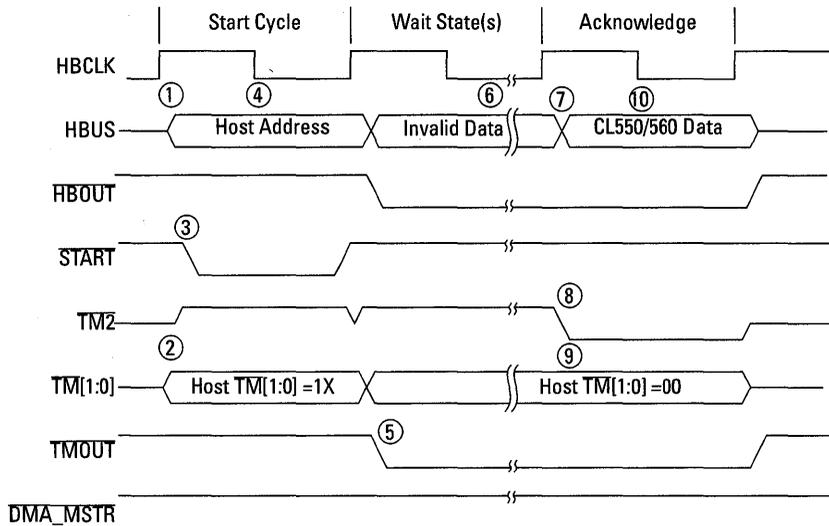
$\overline{TM}[1:0]$	$\overline{TM2}$	Operation
00	0	Transaction Complete
01	0	Bus Error
10	0	Bus Error
11	0	Bus Error

#### 4.1.3 Host Bus Register Access

Host bus register access transactions consist of a start cycle and an acknowledge cycle. The host begins a start cycle by driving the address on HBUS[31:0], driving  $\overline{TM1}$  to indicate whether the cycle is a read or a write, and asserting  $\overline{START}$ . For the CL550 write cycles only, the bus master must change HBUS[31:0] to the data to be written before the HBCLK falling edge following the  $\overline{START}$  cycle. For a CL560 write cycle, the data must be stable before the HBCLK falling edge in the acknowledge cycle. When the cycle is complete, the CL550/560 family part drives the  $\overline{TM}$  signals to indicate status.

Figure 4-3 shows a typical register read transaction. The circled numbers in the figure refer to the steps below.

1. The host places the address to be read on HBUS.
2. The host indicates that a read operation is to take place by setting  $\overline{TM1} = 1$ .  $\overline{TM2}$  must also be set high at this time.
3. The host starts the transaction by asserting  $\overline{START}$ .
4. On the falling edge of HBCLK, the CL550/560 part samples  $\overline{START}$ ,  $\overline{TM1}$ , and  $\overline{TM2}$ . If  $\overline{TM2}$  is not sampled high at this time, the part will ignore  $\overline{START}$  and not return an acknowledge.
5. Because the direction of  $\overline{TM}[2:0]$  changes at this point, the CL550/560 part must assert  $\overline{TMOU}$  to change the direction of



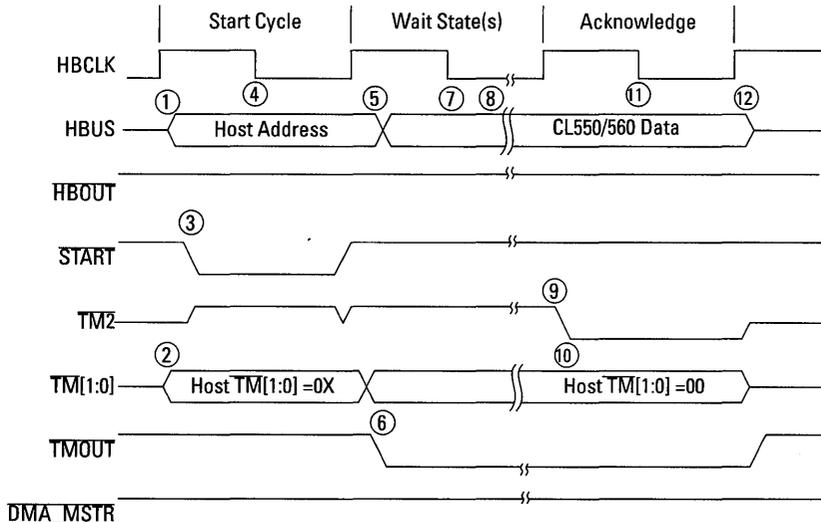
**Figure 4-3 Register Read Transaction**

the bus buffers.

6. An indefinite number of wait states can be added between the Start Cycle and the Acknowledge Cycle.
7. When the CL550/560 part has the data available, it places it on HBUS.
8. The CL550/560 part indicates to the host that the data is available by asserting **TM2**.
9. The CL550 indicates that no errors occurred by placing 00 on **TM[1:0]**.
10. The host samples the data on the falling edge of HBCLK.

#### 4.1.4 Host Bus Register Write

Figure 4-4 shows a typical register write transaction. The circled numbers in the figure refer to the steps below.



**Figure 4-4 Register Write Transaction**

1. The host places the address to be written on HBUS.
2. The host indicates that a write operation is to take place by setting  $\overline{\text{TMI}} = 0$ .  $\overline{\text{TM2}}$  must also be set HIGH at this time.
3. The host starts the transaction by asserting  $\overline{\text{START}}$ .
4. On the falling edge of HBCLK, the CL550/560 part samples  $\overline{\text{START}}$ ,  $\overline{\text{TMI}}$ , and  $\overline{\text{TM2}}$ . If  $\overline{\text{TM2}}$  is not sampled HIGH at this time, the part will ignore  $\overline{\text{START}}$  and not return an acknowledge.
5. The host places data on HBUS and waits for an acknowledgment from the CL550/560 part.
6. Because the direction of  $\overline{\text{TM}}[2:0]$  changes at this point, the CL550/560 part must assert  $\overline{\text{TMOUT}}$  to change the direction of the bus buffers.

7. If this is a CL550 write cycle, the data to be written must be stable before the falling edge of the HBCLK cycle following the start cycle.
8. An indefinite number of wait states can be added between the Start Cycle and the Acknowledge Cycle.
9. When the CL550/560 part is ready for data, it puts the acknowledge on  $\overline{TM2}$ .
10. The CL550 puts a result code on  $\overline{TM}[1:0]$ . If no errors occurred, this result code = 00h.
11. The CL560 samples the data on the falling edge of HBCLK during the acknowledge cycle.
12. The host must drive data until the end of the acknowledge cycle.

---

### 4.2 DMA Access Timing

---

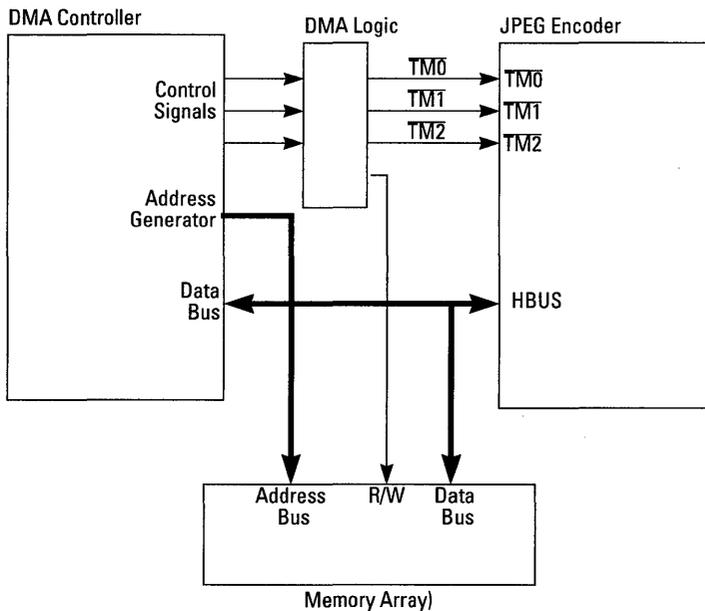
The CL560 is capable of accepting data in DMA mode in conjunction with an external DMA controller (see Figure 4-5). The CL560 relies on the DMA controller to generate the start cycle and provide the address and R/W signals.

The signals used in DMA mode transfers are identical to those used in register mode transfers with these additions:

- **DRQ, Data Request:** The DRQ signal is an output, synchronous to HBCLK, that provides CODEC status for DMA interface control. When DRQ is low, the part is ready to send or receive data.
- **DMA\_MSTR, DMA Master:** The  $\overline{DMA\_MSTR}$  signal is an input that allows the CL560 to work with a DMA controller for CODEC data transfers only. It is sampled on the falling edge of HBCLK when the  $\overline{START}$  signal is active. When  $\overline{DMA\_MSTR}$  is sampled LOW during  $\overline{START}$ , an internal chip select is generated to the CL550 CODEC Register or the CL560 CODEC FIFO. At this point, the address on HB[31:0] becomes a “don’t care”. During this time, the  $\overline{TM}[0:2]$  lines remain as inputs, and it is the responsibility of the host to drive  $\overline{TM2}$  LOW to complete the transaction. For the CL550, the host should also drive  $\overline{TM}[1:0]$  when it drives  $\overline{TM2}$  LOW, or the Bus Error flag will be set.

*Note: The CL550 has DMA capability built in, but it does not work correctly during decompression operations, and will*

*not be fixed (it does work correctly in compression only applications). Section 4.2.4, Alternative Method of CL550/560 DMA Transfers, shows an alternative method of implementing transfers that allows you to achieve DMA transfer speeds using conventional memory access techniques. DMA\_MSTR on the CL550 should always be pulled HIGH during decompression.*



**Figure 4-5 DMA Mode Operation**

The CL560 samples the signal connected to **TM1** during the start cycle to determine whether the operation being performed is a read or a write.

If the CL560 senses a write to the memory, it provides the data during the cycle following the start cycle and holds that data until it receives an acknowledge (**TM2** asserted) from the memory.

**Table 4-10 External Buffers Direction Control**

START	TMT <sup>1</sup>	DMA_MSTR	HBOUT	Buffer Direction	Operation
H	X	X	X	X	No Transaction
L	L	L	L	CL560 to Host	DMA Read Cycle
L	L	H	H	Host to CL560	Normal Write Cycle
L	H	L	H	Host to CL560	DMA Write Cycle
L	H	H	L	CL560 to Host	Normal Read Cycle

1. Note that the polarity of TMT changes sense between normal reads and writes, and DMA reads and writes.

In DMA mode, all bus transfers are to and from the CL560 CODEC FIFO. The  $\overline{\text{DMA\_MSTR}}$  signal must be asserted only when the CODEC is prepared to accept or source data. A qualified  $\overline{\text{DMA\_MSTR}}$  can be generated by using  $\overline{\text{DRQ}}$  from the CL560 to qualify  $\overline{\text{DMA\_REQ}}$  from the DMA controller.

#### 4.2.1 CL560 DMA Transfers

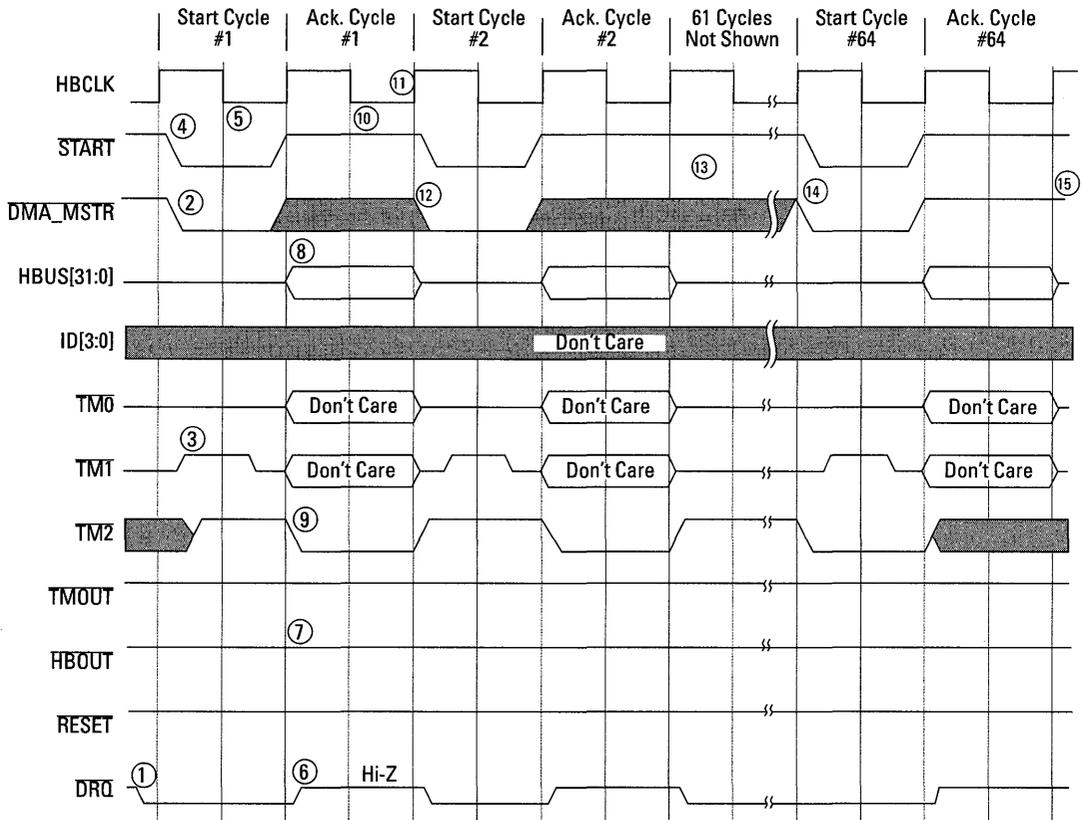
The CL560 indicates that the CODEC FIFO has room for data (or data to be read) by asserting  $\overline{\text{DRQ}}$  low. The amount of space (data) that is available is determined by which flag is set in the DMA Request Interrupt Mask Register (See Chapter 7). The CL560 can transfer as many words as is necessary to fill or empty the FIFO in a single DMA burst.

The host indicates that a DMA operation to the CL560 is going to take place by asserting  $\overline{\text{DMA\_MSTR}}$  low. The host then initiates the transfer by asserting TMT LOW for a read (CODEC to Host) or HIGH for a write (Host to CODEC).

When the CL560 samples  $\overline{\text{DMA\_MSTR}}$  and  $\overline{\text{START}}$  low, it assumes that the data is being read from or written to the CODEC register, thus the host processor does not need to supply an address.

#### 4.2.2 CL560 DMA Write Transaction Timing

Figure 4-6 shows a typical DMA mode write transaction, where the host processor (DMA controller) is writing data to the CL560 CODEC register (Decompression). In this example, the host is going to transfer 64 words of information into the CL560 CODEC. The circled numbers in the figure refer to the steps below.



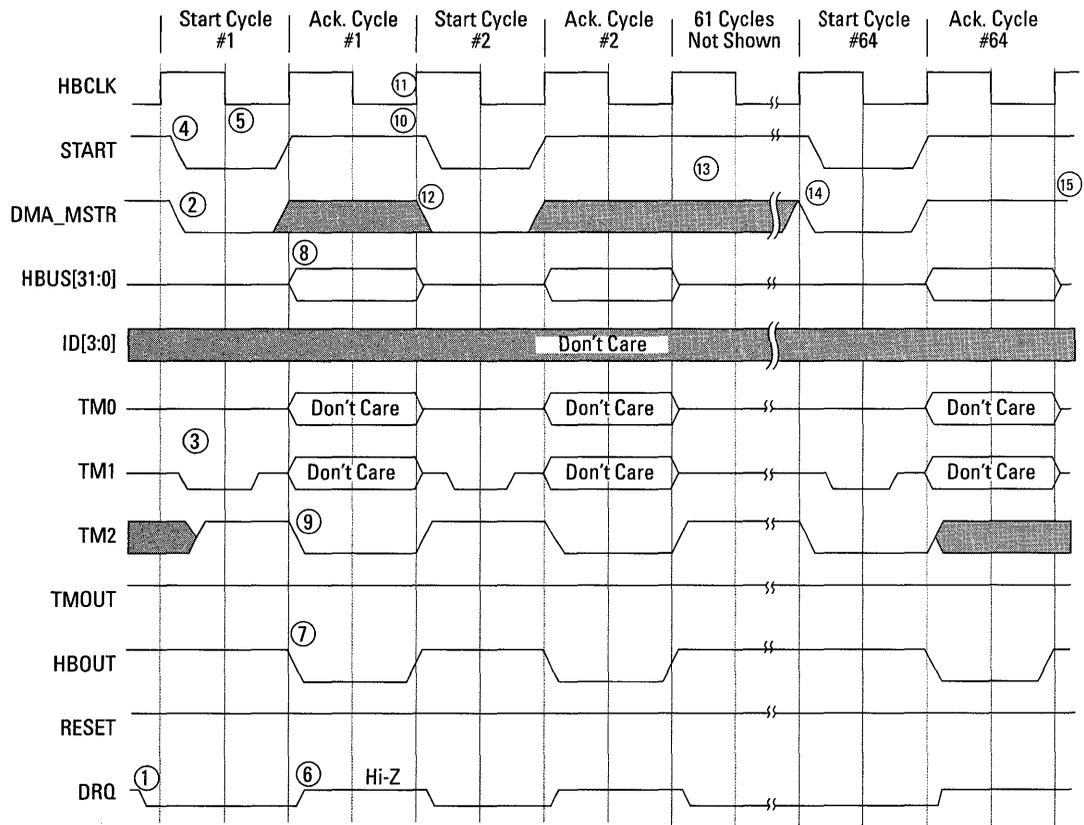
**Figure 4-6 CL560 DMA Write (Decompression - Host Write to CODEC)**

1. The CODEC FIFO is not 1/4 full, and the CL560 generates a DMA request by asserting  $\overline{DRQ}$  LOW.
2. The host processor recognizes the  $\overline{DRQ}$ , and initiates a DMA write cycle by asserting  $\overline{START}$  LOW.

3. The host informs the CL560 that the DMA operation will be a write by asserting  $\overline{\text{TMI}}$  HIGH.
4. The host starts the DMA operation by asserting  $\overline{\text{START}}$  LOW.
5. Steps 2 through 4 must take place before the falling edge of HBCLK.
6. The CL560 releases  $\overline{\text{DRQ}}$  upon recognizing the start cycle.  $\overline{\text{DRQ}}$  may change states unpredictably during the DMA transfer because the CL560 is removing data from the FIFO at the same time that the host is filling it. However,  $\overline{\text{DRQ}}$  can be ignored until the end of the transfer.
7. The CL560 drives  $\overline{\text{HBOUT}}$  HIGH to change the direction of the buffers to “write”.
8. The host (memory) puts the data to be written to the CL560 on HBUS.
9. The DMA controller terminates this cycle by asserting  $\overline{\text{TM2}}$  LOW.
10. The CL560 clocks the data into the CODEC register on the falling edge of HBCLK.
11. The data bus and the control lines must be held valid until the end of the data cycle.
12. The DMA controller starts the second transfer by asserting  $\overline{\text{DMA\_MSTR}}$  LOW ( $\overline{\text{DMA\_MSTR}}$  can remain LOW throughout the transfer, if desired).
13. The 61 intervening cycles are not shown.
14. The DMA controller starts the last (64th) transfer by asserting  $\overline{\text{DMA\_MSTR}}$  LOW.
15. The DMA controller terminates the transfer by asserting  $\overline{\text{DMA\_MSTR}}$  HIGH at the end of the start cycle. The CL560 will automatically complete the current transfer.

### 4.2.3 CL560 DMA Read Transaction Timing

Figure 4-7 shows a typical DMA mode read transaction, where the host processor (DMA controller) is reading data from the CL560 CODEC register (Compression). In this example, the CL560 is going to transfer 64 words of information. The circled numbers in the figure refer to the steps below.



**Figure 4-7 CL560 DMA Read (Compression - Host Read from CODEC)**

1. The CODEC FIFO is 3/4 full, and the CL560 generates a DMA request by asserting  $\overline{\text{DRQ}}$  LOW.
2. The host processor recognizes the  $\overline{\text{DRQ}}$ , and initiates a DMA

- read cycle by asserting  $\overline{\text{START}}$  LOW.
3. The host informs the CL560 that the DMA operation will be a read by asserting  $\overline{\text{TMI}}$  LOW.
  4. The host starts the DMA operation by asserting  $\overline{\text{START}}$  LOW.
  5. Steps 2 through 4 must take place before the falling edge of HBCLK.
  6. The CL560 releases  $\overline{\text{DRQ}}$  upon recognizing the start cycle.  $\overline{\text{DRQ}}$  may change states unpredictably during the DMA transfer because the CL560 is removing data from the FIFO at the same time that the host is filling it. However,  $\overline{\text{DRQ}}$  can be ignored until the end of the transfer.
  7. The CL560 drives  $\overline{\text{HBOUT}}$  LOW to change the direction of the data bus buffers to “read”.
  8. The CL560 outputs the requested data on HBUS.
  9. The DMA controller terminates this cycle by asserting  $\overline{\text{TM2}}$  LOW.
  10. The CL560 will hold the data stable until after the next rising edge of HBCLK.
  11. The data bus and the control lines must be held valid until the end of the data cycle.
  12. The DMA controller starts the second transfer by asserting  $\overline{\text{DMA\_MSTR}}$  LOW ( $\overline{\text{DMA\_MSTR}}$  can remain LOW throughout the transfer, if desired).
  13. The 61 intervening cycles are not shown.
  14. The DMA controller starts the last (64th) transfer by asserting  $\overline{\text{DMA\_MSTR}}$  LOW.
  15. The DMA controller terminates the transfer by asserting  $\overline{\text{DMA\_MSTR}}$  HIGH at the end of the start cycle. The CL560 will automatically complete the current transfer.

#### 4.2.4 Alternative Method of CL550/560 DMA Transfers

The DMA transfer function on the CL550 does not work correctly during decompression, and will not be fixed. This section shows an alternative method using burst mode transfers that will allow you to achieve DMA transfer speeds using conventional memory access techniques. Burst mode transfers are similar to register accesses, except that HB15 is pulled LOW to indicate that the CODEC register or FIFO is being accessed. This method can be used with either the CL550 or the CL560.

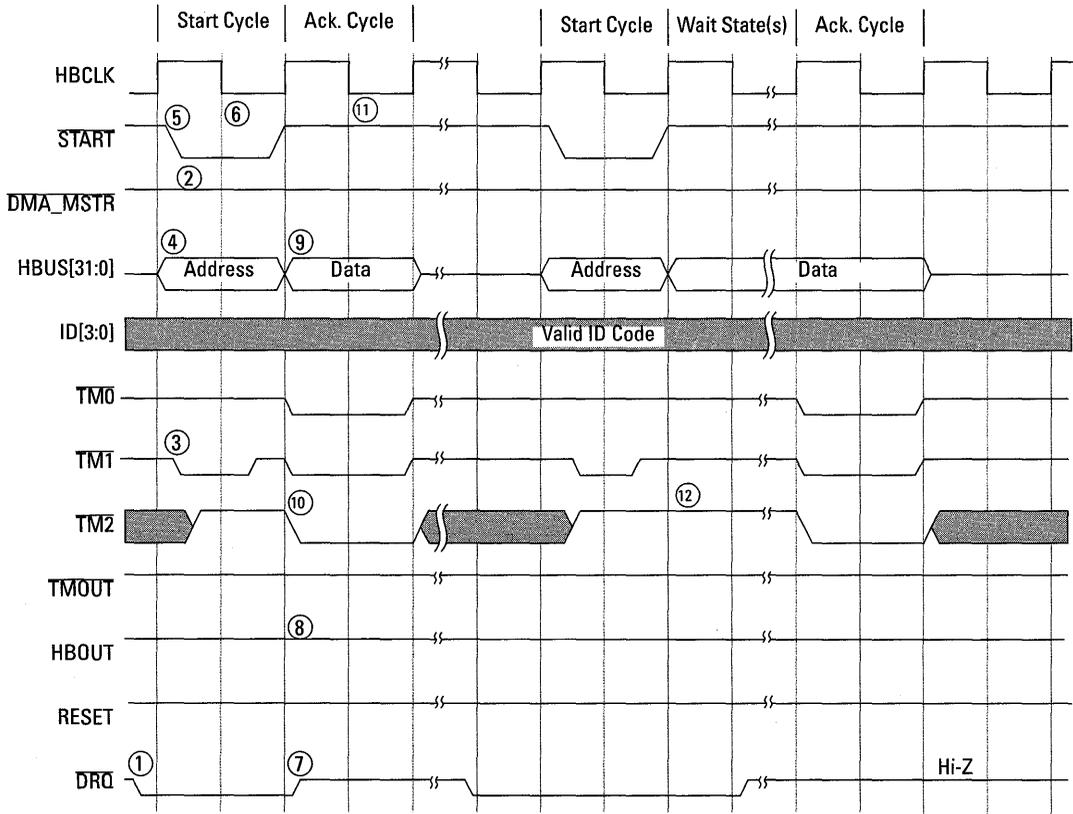
The CL550 and CL560 allow two-cycle (Start/Ack) accesses to the CODEC register. The host must indicate that a CODEC register access is about to occur by pulling HB15 LOW during the Start cycle. If the CL550/560 is in the 32-bit address mode, the host must also supply an address on HB[31:16] that meets the requirements shown in Figure 4-2. When these conditions occur, the CL550/560 will allow a no-wait-state access to the CODEC register.

The CL550/560 part indicates that the CODEC register has room for data by asserting  $\overline{DRQ}$  low (because the CL550 has a CODEC register instead of a CODEC FIFO, the assertion of  $\overline{DRQ}$  indicates that only one DMA transfer cycle can be performed). The host initiates a burst mode memory transfer by pulling HB15 LOW, providing a valid address space identification code on HB[31:16], and asserting  $\overline{START}$  LOW.

If the transfer is to be a read (CL550/560 transfer to host),  $\overline{TMT}$  should be held HIGH during the start cycle, and if the transfer is to be a write (host data transfer to CL550), then  $\overline{TMT}$  should be held LOW during the start cycle.  $\overline{DMA\_MSTR}$  must always be held HIGH during assertion of the  $\overline{START}$  signal.

**4.2.5 CL550 Burst-mode Write Transaction Timing**

Figure 4-8 shows a typical burst-mode write transaction, where the host processor (DMA controller) is writing data to the CL550 CODEC register. The circled numbers in the figure refer to the steps below.



**Figure 4-8 CL550 DMA Write (Decompression - Host Write to CODEC)**

1. The CL550 asserts DRQ to notify the host that there is space (1 empty word) in the CODEC register.
2. DMA\_MSTR must be held HIGH.
3. The host processor controller drives TM1 LOW and TM2 HIGH to indicate that a write operation is going to be per-

formed.

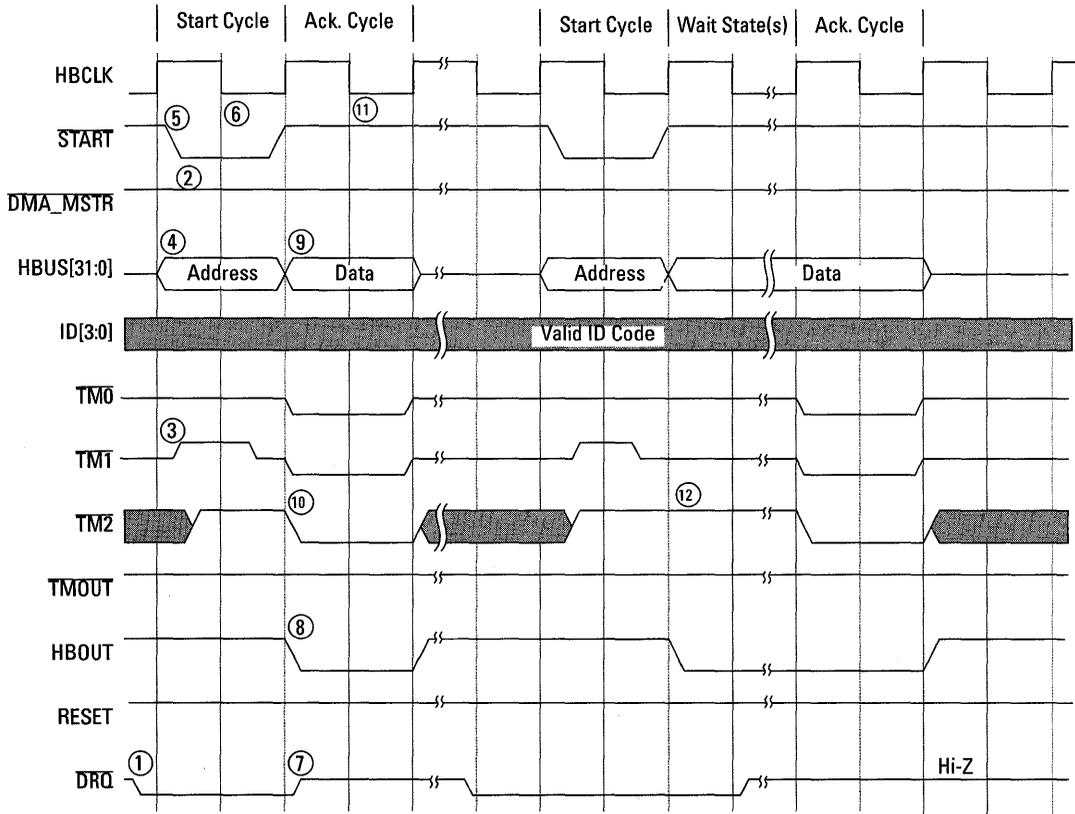
4. In 32-bit address mode, the host processor puts the following on HBUS[31:0]:
  - HBUS[31:16] = as shown in Figure 4-2
  - HBUS15 = 0 (LOW)
  - HBUS[14:0] = Don't Care

In 16-bit mode, HBUS[31:16] are not used, and are Don't Cares.
5. The host processor asserts  $\overline{\text{START}}$  to initiate the transfer.
6. Steps 2, 3 4, and 5 above must all occur before the falling edge of HBCLK.
7. The CL550 releases  $\overline{\text{DRQ}}$  upon recognizing the start cycle.
8. The CL550 drives  $\overline{\text{HBOUT}}$  HIGH to change the direction of the HBUS buffers to “write”.
9. The host (memory) puts the data to be written to the CL550 on HBUS.
10. The CL550 terminates the transfer on the next cycle by asserting  $\overline{\text{TM}}[1:0]$  and  $\overline{\text{TM}}_2$  LOW. No wait states were inserted.
11. The CL550 clocks the data into the CODEC register on the falling edge of HBCLK.
12. The data bus and the control lines must be held valid until the end of the data cycle.

The second cycle shown is identical to the first, except that the host has inserted an optional wait state at Step 12 by holding  $\overline{\text{TM}}_2$  HIGH until the data is available.

### 4.2.6 CL550 Burst-mode Read Transaction Timing

Figure 4-9 shows a typical burst-mode read transaction, where the host processor (DMA controller) is reading data from the CL550 CODEC register. The circled numbers in the figure refer to the steps below.



**Figure 4-9 CL550 DMA Read (Compression - Host Read from CODEC)**

1. The CL550 asserts  $\overline{DRQ}$  to notify the host that there is a word in the CODEC register.
2.  $\overline{DMA\_MSTR}$  must be held HIGH.
3. The host processor drives TMT HIGH to indicate that a read is going to be performed.

4. In 32-bit address mode, the host processor puts the following on HBUS[31:0]:
  - HBUS[31:16] = as shown in Figure 4-2
  - HBUS15 = 0 (LOW)
  - HBUS[14:0] = Don't Care

In 16-bit mode, HBUS[31:16] are not used, and are Don't Cares.

5. The host processor asserts  $\overline{\text{START}}$  to initiate the transfer.
6. Steps 2, 3 4, and 5 above must all occur before the falling edge of HBCLK.
7. The CL550 releases  $\overline{\text{DRQ}}$  upon recognizing the start cycle.
8. The CL550 drives  $\overline{\text{HBOUT}}$  LOW to change the direction of the HBUS buffers to "read".
9. The CL550 puts the data that is being read on HBUS.
10. The host processor terminates the transfer by asserting  $\overline{\text{TM}}[1:0]$  and  $\overline{\text{TM2}}$  LOW.
11. The data bus and the control lines must be held valid until the end of the acknowledge cycle.

The second cycle shown is identical to the first, except that the host has inserted an optional wait state at Step 11 by holding  $\overline{\text{TM2}}$  HIGH until the data has been accepted.

#### 4.2.7 Operational Considerations

The following operational considerations should be noted when designing host bus interfaces:

- **Wait States:** Access to all registers within the CL550/560 (except the CODEC) takes three HBCLK cycles (one wait state). Accesses to the CODEC take a minimum of two HBCLK cycles, but the CL550 (only) can insert wait states of up to 70 pixel clocks in length under worst case conditions. This delay can be avoided by polling the Flags register or checking the DRQ signal to determine the CODEC state prior to accessing the CODEC (see Chapter 7).
- **Drive Capability:** The HBUS and  $\overline{\text{TM}}$  signals do not have enough drive to meet the specifications of most system buses. An external transceiver must be used to buffer these signals from the

system bus. The signals  $\overline{\text{HBOUT}}$  and  $\overline{\text{TMOU}}$  are provided to control these transceivers.

- **Compression Mode:** The CODEC register is read-only in compression mode. If the host attempts to write this register, the CL550/560 will not return an acknowledge on  $\overline{\text{TM2}}$  and the HBUS will remain in a locked state until a hard reset is issued.
- **Compression Mode:** When the CL550 is in the 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 this point, 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 CL550/560 will not return an acknowledge on  $\overline{\text{TM2}}$  and the HBUS will remain in a locked state until a hard reset is issued. When in the decompression mode, if the FIFO is full and the host attempts to write data to the CODEC register, the acknowledge on  $\overline{\text{TM2}}$  will be delayed until the FIFO is not full. If the device is not actively decompressing (START register = 0), the bus will remain locked, and a reset will be needed. Therefore, the host should never fill the FIFO past 3/4 full. When, in decompression mode, the CL550/560 detects a marker code (value 0xFFXX) 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 CL550/560 will not return an acknowledge on  $\overline{\text{TM2}}$  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 (0xFFD0 through 0xFFD7). These markers are automatically detected and stripped off by the CL550/560 with no external intervention required.
- **Handshake:** When performing any access to the CL550/560 host bus, the  $\overline{\text{TM2}}$  line must be at logic level one during the assertion of START. If it is not, the CL550/560 will not recognize the access, and no acknowledge will be given on  $\overline{\text{TM2}}$ . A pull-up resistor on  $\overline{\text{TM2}}$  could be used for this purpose.
- **DMA\_MSTR Mode:** The  $\overline{\text{DMA\_MSTR}}$  input on the CL550/560

allows the CL550/560 to behave as a bus master for CODEC accesses. Once a `DMA_MSTR` transfer occurs, ACK from the addressed slave terminates the transfer. However, a subsequent ACK from a non-CL550/560 access (a “foreign ACK”) will cause the CODEC to malfunction. The workaround is to prevent foreign ACKs from reaching the CL550/560 after the `DMA_MSTR` transfer completes. Once the `DMA_MSTR` transfer ACK occurs, the ACK signal to the CL550/560 must be suppressed until one of the following occurs:

- The next `DMA_MSTR` transfer occurs (START and `DMA_MSTR`)
- A valid CL550/560 access occurs (START and valid CL550/560 Address)

This workaround is necessary only if other devices besides the CL550/560 and the CL550/560 DMA slaves can drive ACK.

- **FIFO Level Control:** In the CL560 only, it is possible to lose data if you try to either write to the CODEC FIFO when it is full, or read from the CODEC FIFO when it is empty. It is recommended that when writing to the FIFO, you set the `DRQ` trigger point so that a `DRQ` is generated when the FIFO is down to 1/4 full, and then only send enough data to bring it up to the 3/4 full point. When reading the FIFO, set the `DRQ` trigger point so that a `DRQ` is generated when the FIFO is up to 3/4 full, and then only read enough data to bring it down to the 1/4 full point. This will prevent either an overflow or underflow condition.

Timing and control signals are used to control the operation of the CL550/560, synchronize data transfers and provide information to the host processor.

#### 4.3.1 RESET

The `RESET` signal is an input that forces a hardware reset of the device. 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 part will not acknowledge any access until the third `HBCLK` cycle after `RESET` is deasserted.

---

## 4.3 Control Signals

---

### 4.3.2 $\overline{\text{NMRQ}}$

*Note:  $\overline{\text{NMRQ}}$  is a CL550 signal only. The CL560 uses  $\overline{\text{IRQ1}}$  instead.*

Interrupt Request ( $\overline{\text{NMRQ}}$ ) is an unlatched output signal, synchronous to HBCLK, that provides an indicator of both FIFO and video field status. It can be programmed to selectively indicate active status flags as specified in the Interrupt Mask Register. This signal is an open drain output and should be tied to VCC through a resistor of at least 625 ohms. On power-up, the CL550 or CL560 should be hardware reset to prevent the generation of spurious interrupts.

### 4.3.3 HALF\_FULL

*Note: HALF\_FULL is a CL550 signal only. The CL560 uses  $\overline{\text{IRQ2}}$  instead.*

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 and synchronous to PXCLK.

### 4.3.4 $\overline{\text{IRQ1}}$ , $\overline{\text{IRQ2}}$

*Note:  $\overline{\text{IRQ1}}$  and  $\overline{\text{IRQ2}}$  are CL560 signals only. The CL550 uses  $\overline{\text{NMRQ}}$  and HALF\_FULL instead.*

$\overline{\text{IRQ1}}$  and  $\overline{\text{IRQ2}}$  are general-purpose status outputs of the CL560. The assertion of these signals is programmable based on masks contained in the  $\overline{\text{IRQ1}}$  and  $\overline{\text{IRQ2}}$  mask registers described in Chapter 7, Registers.

# 5 Video Interface

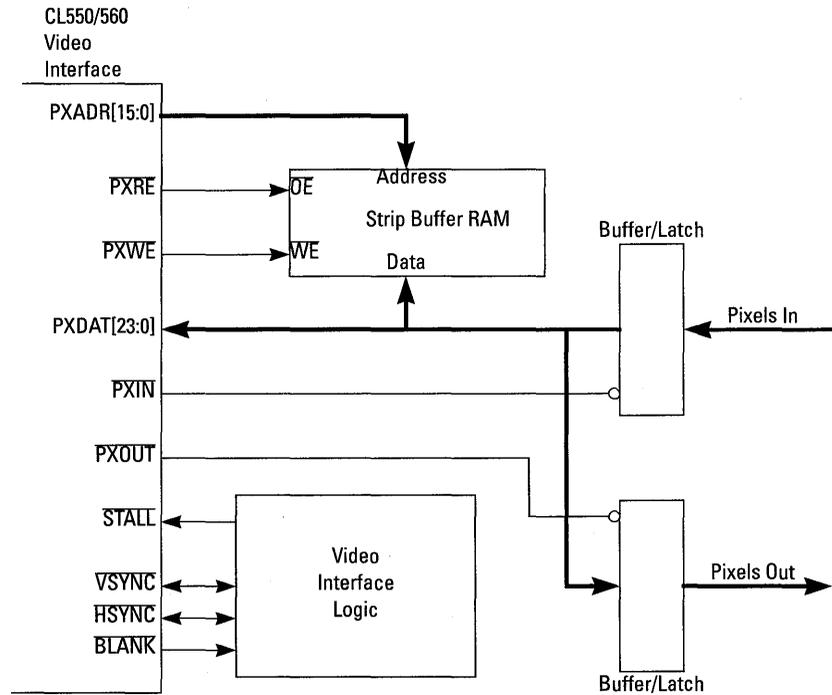
This chapter describes the Video interface to the CL550 and CL560 (referred to as CL550/560). The Video interface is used to input uncompressed video data in the compression mode, or to output decompressed video data in the decompression mode.

This chapter is divided into sections that describe how the Video interface is used. These sections are:

- 5.1, Overview
- 5.2, Video Interface Logic
- 5.3, Basic System Configurations
- 5.4, Timing Diagrams Compression Mode
- 5.5, Timing Diagrams Decompression Mode

**5.1  
Overview**

The Video interface is used to input uncompressed video data during the compression process, and output decompressed video data during the decompression process. A block diagram of the Video interface is shown in Figure 5-1.



**Figure 5-1 Video Interface Block Diagram**

During compression, the strip buffer RAM is used to store the incoming pixels until 8 complete lines of video have been received. The CL550/560 then uses the strip buffer RAM to perform a raster to 8 x 8 block conversion of the pixel data.

During decompression, the strip buffer RAM is used to store the decompressed 8 x 8 blocks until 8 complete lines of pixel data have been decompressed. The CL550/560 then uses the strip buffer RAM to perform an 8 x 8 block to raster conversion of the pixel data.

### 5.1.1 Signal Descriptions

The Video interface consists of the following signals:

- **PXDAT [23:0], Pixel Data Bus:** The Pixel data bus is a 24-bit wide bus that handles uncompressed or decompressed pixel data. It is also used to transfer data to and from the strip buffer RAM. In some modes (Grayscale, YUV 4:2:2 and CMYK), only 16 of the 24-bits are used.
- **PXADR [15:0], Pixel Address Bus:** PXADR is the address bus for the strip buffer RAM. The 16 bits of address support a strip buffer of up to 65,536 entries.
- **PXRE, Pixel Read:** PXRE is an output signal designed to directly control the Output Enable ( $\overline{OE}$ ) pin of the strip buffer RAMs. During compression, PXRE is active only when the CL550/560 is reading pixel data from the strip buffer RAM. During decompression, PXRE is active only when pixels are being read from the strip buffer RAM out to the pixel destination.
- **PXWE, Pixel Write:** PXWE is an output is designed to directly control the Write Enable ( $\overline{WE}$ ) input of the strip buffer RAMs. During compression, PXWE is active only during PXIN cycles; when pixel data is being input from the active portion of the video field into the strip buffer RAM. During decompression, PXWE is active only when active pixels are being written from the CL550/560 into the strip buffer RAM.
- **PXIN, Pixel Input Control:** PXIN is used to activate an input buffer on the Pixel Data bus, PXDAT, during input cycles. It is active only when pixel data is being input from the active portion of the video field into the strip buffer RAM.
- **PXOUT, Pixel Output Control:** PXOUT is used to load the active pixel into a register as it is read out of the strip buffer RAM. It is active only when pixels from the active region of the field are being read from the strip buffer RAM.
- **STALL, Stall:** Asserting the  $\overline{STALL}$  input signal stops all activity on the Video interface. Signals affected by STALL include PXDAT[23:0], PXADR[15:0], PXRE, PXWE, PXIN, PXOUT, BLANK, VSYNC and HSYNC.

- **HSYNC, Horizontal Synchronization:** HSYNC is a bidirectional signal used to indicate the start of a horizontal line. This signal acts as an output in Master mode operation, and as an input in Slave mode.
- **VSYNC, Vertical Synchronization:** VSYNC is a bidirectional signal used to indicate the start of a frame. This signal acts as an output in Master mode operation, and as an input in Slave mode.
- **BLANK, Blanking:** This signal is an output that indicates that there are no active pixels on the Pixel Data bus.

### 5.1.2 Video Interface Clocks

The Video interface uses three clocks to synchronize its operation:

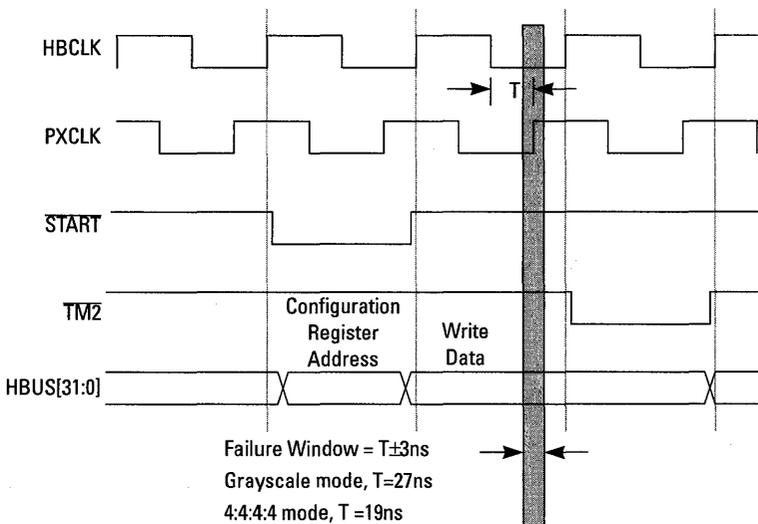
- **PXCLK, Pixel Clock:** PXCLK is the main clock for the compression processor. All circuits except those directly related to the host bus interface are driven by this clock. In single-component mode (Grayscale), this clock is equal to the pixel rate. In 4:2:2 modes, this clock is twice the actual pixel rate, and in 4:4:4 (YUV and RGB) and 4:4:4:4 modes (CMYK), this clock is four times the actual pixel rate (see Table 5-1).

*Note: For correct CL550 operation, HBCLK must be the same rate or slower than PXCLK. The CL560 will work with HBCLK timing up to 2 times faster than PXCLK.*

- **PXPHASE, Pixel Phase:** The value of the PXPHASE input signal, together with CLK3 in some modes, indicates whether a Strip buffer read or write operation is occurring on the pixel bus. This signal should be one-half the frequency of PXCLK. For all modes except 4:4:4 mode, if PXPHASE is HIGH during the rising edge of PXCLK, the ensuing cycle will be a strip buffer read operation. If PXPHASE is LOW during the rising edge of PXCLK, the cycle is a strip buffer write operation. Timing for 4:4:4 mode is discussed in the CLK3 section.
- **CLK3, Clock 3:** The CLK3 input is one-half the frequency of PXPHASE. 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 Data bus. If CLK3 is LOW during the rising edge of PX-CLK, it indicates that the second pair of components will be on the Pixel Data bus. In 4:4:4 video modes, a HIGH on CLK3 during the rising edge of PXCLK indicates that the cycle will be a strip buffer RAM read cycle, and a LOW indicates that the cycle will be a strip buffer RAM write cycle.

**Note:** When the CL550 (only) is used in single-component mode (Grayscale) or 4:4:4:4 mode, there is a restriction on the skew between PXCLK and HBCLK when setting the configuration register (see Figure 5-2). If this restriction is not satisfied, the CL550 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 and 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 within the failure window. This problem does not exist in the CL560, and there are no restrictions on CL560 clock skew.



**Figure 5-2 CL550 Host Bus Write Timing for the Configuration Register**

**Table 5-1 CL550/560 Color Modes and Pixel Data Configurations**

<b>Two Pixels per Two PXCLKs Mode</b>		<b>PXCLK 0</b>	<b>PXCLK 2</b>	<b>PXCLK 4</b>	<b>PXCLK 6</b>
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]
<b>One Pixel per Two PXCLKs Mode</b>		<b>PXCLK 0</b>	<b>PXCLK 2</b>	<b>PXCLK 4</b>	<b>PXCLK 6</b>
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]
<b>One Pixel every Fourth PXCLK Mode (Half Rate Timing)</b>		<b>PXCLK 0</b>	<b>PXCLK 4</b>	<b>PXCLK 8</b>	<b>PXCLK 12</b>
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]
<b>One Pixel per Four PXCLKs Mode</b>		<b>PXCLK 0</b>	<b>PXCLK 2</b>	<b>PXCLK 4</b>	<b>PXCLK 6</b>
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]

### 5.1.3 Master/Slave Mode Operation

The CL550/560 can be programmed to operate in either Master mode, or Slave mode. In Master mode, the CL550/560 generates  $\overline{\text{HSYNC}}$  and  $\overline{\text{VSYNC}}$ , and in Slave mode, the CL550/560 expects the external video interface logic to generate  $\overline{\text{HSYNC}}$  and  $\overline{\text{VSYNC}}$ .

Master mode is selected by programming bit 3 of the Configuration register to a 1. Slave mode is selected by programming bit 3 of the Configuration register to a 0.

### 5.1.4 STALL Operation

The  $\overline{\text{STALL}}$  input, when asserted, signal stops all activity on the Video interface. Signals affected by  $\overline{\text{STALL}}$  include  $\overline{\text{PXDAT}}[23:0]$ ,  $\overline{\text{PXA-DR}}[16:0]$ ,  $\overline{\text{PXRE}}$ ,  $\overline{\text{PXWE}}$ ,  $\overline{\text{PXIN}}$ ,  $\overline{\text{PXOUT}}$ ,  $\overline{\text{BLANK}}$ ,  $\overline{\text{VSYNC}}$  and  $\overline{\text{HSYNC}}$ . All internal logic modules in the JPEG processing pipeline between the FIFO and the Video interface are also stopped in their current state. No data transfers can take place between the FIFO and the JPEG pipeline when the device is stalled. In the CL550, the Huffman CODEC is not affected by the assertion of  $\overline{\text{STALL}}$ , so that the host processor can access the CODEC register. In the CL560, all of the modules are stalled, including the Huffman CODEC, but the CODEC FIFO is still accessible when the pipeline is stalled.

$\overline{\text{STALL}}$  is sensed on the rising edge of  $\overline{\text{PXCLK}}$ . When  $\overline{\text{STALL}}$  is negated, processing will resume when  $\overline{\text{PXCLK}}$ ,  $\overline{\text{PXPHASE}}$ , and  $\overline{\text{CLK3}}$  have the same phase relationship as when  $\overline{\text{STALL}}$  was asserted. In modes where  $\overline{\text{CLK3}}$  is not used, processing will resume when  $\overline{\text{PXCLK}}$  and  $\overline{\text{PXPHASE}}$  have the same phase relationship as when  $\overline{\text{STALL}}$  was asserted.

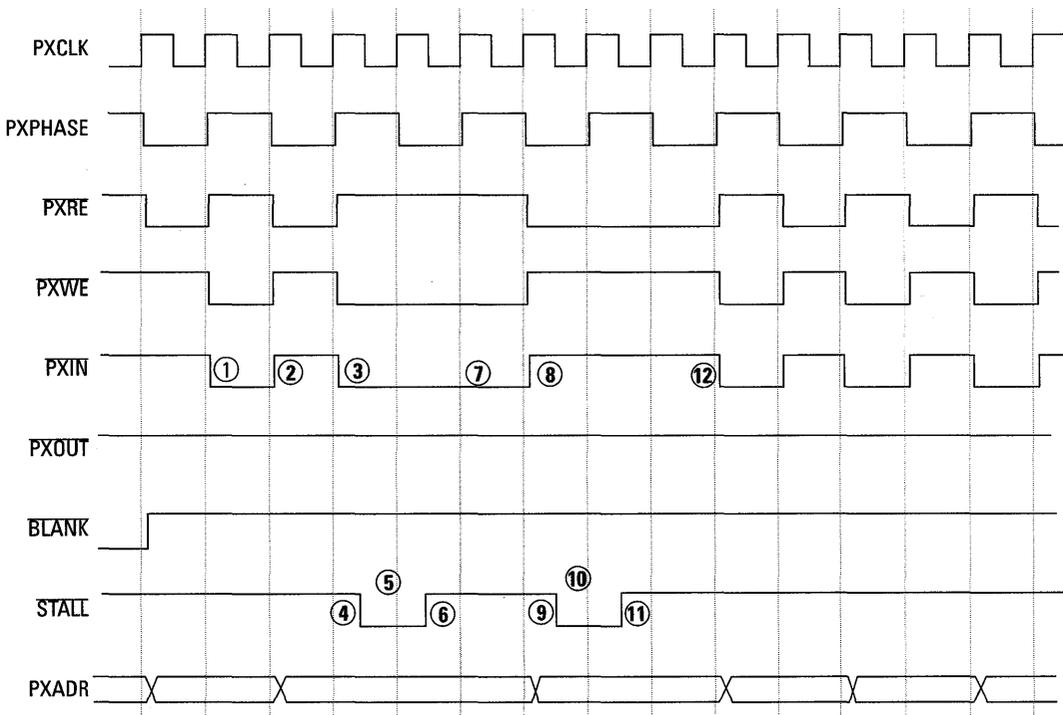
The  $\overline{\text{STALL}}$  signal should be used in the following cases:

- When the CL550/560 is operating in the compression mode,  $\overline{\text{STALL}}$  is asserted to prevent the FIFO from overflowing. One of the CL550/560's status pins,  $\overline{\text{NMRQ}}$ ,  $\overline{\text{DRQ}}$ ,  $\overline{\text{HALF\_FULL}}$ ,  $\overline{\text{IRQ1}}$  or  $\overline{\text{IRQ2}}$ , is used to generate the  $\overline{\text{STALL}}$  signal when the FIFO reaches a certain threshold level, typically 1/2 or 3/4 full. The pixel pipeline will then halt, allowing the host to drain the FIFO below the threshold.
- When the CL550/560 is operating in the decompression mode,  $\overline{\text{STALL}}$  is asserted to prevent the FIFO from underflowing. One of the CL550/560's status pins,  $\overline{\text{NMRQ}}$ ,  $\overline{\text{DRQ}}$ ,  $\overline{\text{HALF\_FULL}}$ ,

$\overline{IRQ1}$  or  $\overline{IRQ2}$ , is used to generate the  $\overline{STALL}$  signal when the FIFO reaches 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.

- During any compression or decompression operation, if the external interface is not ready to deliver a pixel to, or receive a pixel from the CL550/560, the  $\overline{STALL}$  signal should be asserted to hold off the CL550/560 processor.

Figure 5-3 shows the effect that the  $\overline{STALL}$  signal has on the Video interface control signals when the CL550/560 is stalled during a RGB-to-YUV 4:2:2 master-mode compression (Note that CLK3 is not used in this mode). The numbers in the diagram refer to the steps below.



**Figure 5-3 STALL Timing, YUV 4:2:2 Compression Example**

1. The CL560 requests a pixel for compression by asserting PXIN.

The pixel source places the data on the bus at that time. This starts a normal (not stalled) pixel write cycle.

2. The normal write cycle is completed on the next rising edge of PXCLK.
3. The CL560 requests another pixel for compression. This starts a stalled pixel write cycle.
4. The external video logic generates a  $\overline{\text{STALL}}$  signal to halt the CL550/560.  $\overline{\text{STALL}}$  could have been generated either at the request of the CL550/560 (in response to a Half-full flag), or because the video interface needed time to prepare the next pixel.
5. That  $\overline{\text{STALL}}$  signal is recognized on the rising edge of PXCLK. The CL550/560 will leave PXIN, PXOUT, PXRE and PXWE in their current state until the end of the  $\overline{\text{STALL}}$  condition. Note that PXPULSE is LOW at that time (CLK3 is not used in the mode used in this example, and therefore is not significant).
6. The external logic releases the  $\overline{\text{STALL}}$  input.
7. The CL550/560 does not recognize the fact that  $\overline{\text{STALL}}$  was released until the first rising edge of PXCLK when the state of PXPULSE (and CLK3, if used) is the same as when  $\overline{\text{STALL}}$  was first recognized.
8. The pixel write cycle is completed on the next rising edge of PXCLK.
9. The external video logic can also generate a  $\overline{\text{STALL}}$  during a CL550/560 pixel read (PXRE) cycle. It pulls  $\overline{\text{STALL}}$  LOW to start the cycle.
10. The CL550/560 recognizes the  $\overline{\text{STALL}}$  condition on the next rising edge of PXCLK. Note that PXPULSE is HIGH at this point.
11. The external video logic releases the  $\overline{\text{STALL}}$  input.
12. The CL550/560 does not recognize the fact that  $\overline{\text{STALL}}$  was released until the first rising edge of PXCLK when the state of PXPULSE (and CLK3, if used) is the same as when  $\overline{\text{STALL}}$  was first recognized. In this case,  $\overline{\text{STALL}}$  is released on the next falling edge of PXCLK when PXPULSE is HIGH after  $\overline{\text{STALL}}$  is released.

**5.2**  
**Video Interface**  
**Logic**

The logic in the Video interface performs four major functions:

- Order conversion between raster and block formats
- Blanking and active region control
- RGB-YUV conversion
- Interleaved pixel format conversion

Each of these functions is described in the sections below.

**5.2.1 Pixel Order Conversion**

Typical display systems transfer pixel data in raster format (see Figure 5-4), but the JPEG standard requires pixel data to be in 8x8 block order (see Figure 5-5). The Video interface uses an external SRAM buffer called the strip buffer, to accomplish this conversion.

Raster Line 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
Raster Line 7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...

**Figure 5-4**      **Pixels in Raster Order**

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

**Figure 5-5**      **Same Pixels in 8 x 8 Block Order**

The strip buffer should be wide enough to store the data that it is expected to handle (either 16-bits or 24-bits wide), and deep enough to store eight complete lines of data at the highest resolution. The equations for determining the memory size are:

YUV 4:2:2, RGB to 4:2:2, 4:4:4 to 4:2:2 and 4:4:4 Modes

$$\begin{aligned}\text{Line Buffer Depth} &= 8 * (\# \text{ of Pixels per Line}) \\ \text{Line Buffer Width} &= (\# \text{ of Bits / Pixel}) / 8 \text{ Bytes}\end{aligned}$$

Single Component (Grayscale)

$$\text{Line Buffer Depth} = \frac{8 * (\# \text{ of Pixels per Line})}{2}$$

$$\text{Line Buffer Width} = 16$$

4:4:4:4 Mode

$$\begin{aligned}\text{Line Buffer Depth} &= 8 * (\# \text{ of Pixels per Line}) * 2 \\ \text{Line Buffer Width} &= 16\end{aligned}$$

As an example, a system designed to use YUV 4:4:4 format pixels and 1024 pixel wide lines would require the following amount of RAM:

$$\begin{aligned}\text{Line Buffer Depth} &= 8 * 1024 \text{ Pixels per line} = 8192 \\ \text{Line Buffer Width} &= 24 \text{ bits per pixel} / 8 = 3 \text{ Bytes}\end{aligned}$$

In this example, the strip buffer would need three 8K x 8 RAMs.

The strip buffer can access RAM arrays up to 64K addresses deep. The CL550/560 always uses the lowest order address space first.

The strip buffer addressing algorithm is a complex modulo counting scheme. During the first eight lines of a frame, data is written directly from the input source to an address in the RAM. During each subsequent line, until the end of the frame, the CL550/560 reads the pixel data from an address in the strip buffer RAM (as part of the raster to block conversion) and then writes a new pixel of raster data back into the vacated address. After the end of the frame, the CL550/560 only performs reads until the buffer has been purged. Because of this addressing scheme, the SRAM array never needs to be greater than eight lines deep.

### 5.2.2 Window Management and Control

Several of the status signals on the Video interface are used in window management and control. These signals are  $\overline{\text{VSYNC}}$ ,  $\overline{\text{HSYNC}}$  and  $\overline{\text{BLANK}}$ .

The  $\overline{\text{VSYNC}}$  and  $\overline{\text{HSYNC}}$  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. They are outputs in the Master mode, and inputs in the Slave mode. The  $\overline{\text{BLANK}}$  signal is an output that is asserted when no pixels are being transferred to the external interface.

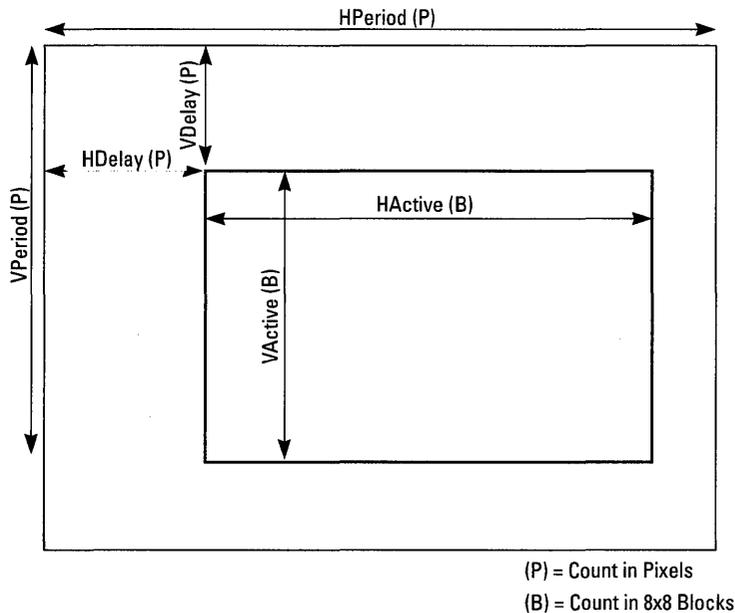
Window and frame parameters are programmed using a set of control registers. These registers are listed in Table 5-2, and defined fully in Chapter 7, Registers. VPeriod and HPeriod are used to specify the dimensions of an image frame. HDelay, VDelay, HActive and VActive are used to specify the size and position of the active image area within the frame. Figure 5-6 illustrates the function of the video field registers.

**Table 5-2 Video Field Control Registers**

Register Name	Content / Function	Units
HPeriod	Number of pixels in a line	Pixels
VPeriod	Number of lines in an image	Lines
HDelay	Horizontal delay to the first active pixel	Pixels
VDelay	Vertical delay to the first active line	Lines
HActive	Active window width	Blocks
VActive	Active window height	Blocks
HSync	Horizontal Sync pulse width	Pixels
VSyn	Vertical Sync pulse width	Lines
Vertical Line Count	Active window vertical line count	Lines

### 5.2.3 Color Conversion

The CL550/560 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 color digitizer is often presented in the 16-bit YUV 4:2:2 format. This is the format required by NTSC and PAL monitors. However, typical computer graphics monitors require data to be presented in 24-bit RGB format. The CL550/560



**Figure 5-6 Video Field Descriptions**

translates between these two formats by means of a matrix multiplier and chrominance sub-sampler. This minimizes the need for external color space conversion logic and reduces overall system complexity and cost.

Conversion between the RGB and YUV color spaces is accomplished using a matrix-multiply operation. Nine registers are provided in the CL550/560 processor to program the transform matrix.

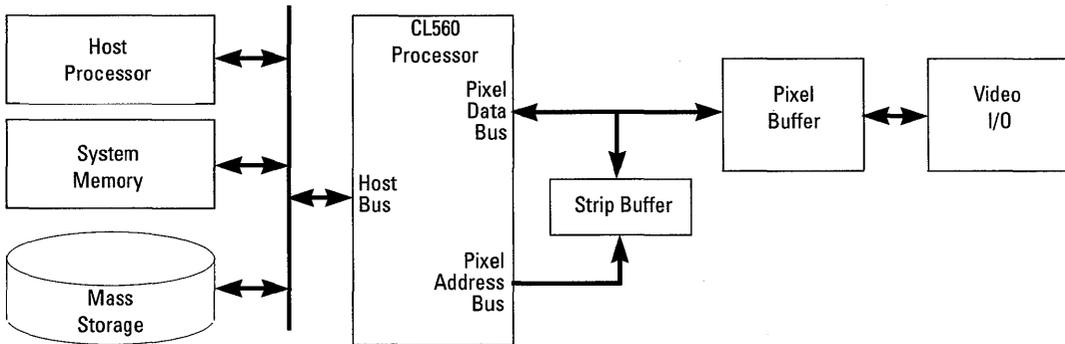
As an example, in the RGB to YUV 4:2:2 pixel conversion mode, the 24-bit RGB pixels are read into the Pixel bus. Once inside the CL550/560, the pixels are transformed into 24-bit YUV 4:4:4 pixels using the on-chip matrix multiplier. Following this operation, the U and V components are sub-sampled to obtain a 4:2:2 ratio between luminance (Y) and chrominance (U and V). The results of this operation are shown in Figure 5-7.

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 5-7 RGB to YUV Conversion Operation**

**5.3  
Basic System  
Configurations**

The CL550/560 is used in several basic system configurations. Video systems are used to perform real-time compression of video or still frames, and have the Pixel Data bus connected directly to a pixel buffer. This application is shown in a block diagram in Figure 5-8, and in greater detail in Figure 5-11. Still-frame systems are used to perform background compression of still frames, and have the Pixel Data bus connected to the processor Host bus. This application is shown in a block diagram in Figure 5-9. Multi-media systems perform real-time compression of video and still frames, but have the Pixel Data bus connected to Video Overlay and Mixer logic to allow the CL550/560 to co-exist with other video-based products in the computer. This application is shown in Figure 5-10.

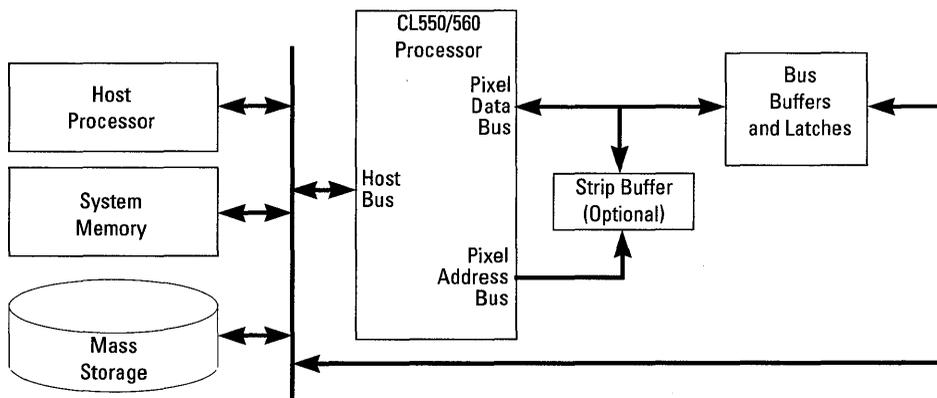


**Figure 5-8 Typical Video System Application**

In a CL550 video application, a pixel buffer is required, and is usually either a bidirectional FIFO or a VRAM frame buffer. The CL550 draws data asynchronously from this buffer at rates up to 15.0 million pixels/second @ 30 MHz or 17.5 million pixels/second @ 35 MHz. In CL560

applications, the pixel buffer is optional, and the CL560 draws data synchronously or asynchronously from this buffer at rates up to 16.5 million pixels/second. Once drawn from the pixel buffer, the data must be converted from raster format into the 8x8 block format required by the JPEG standard. The CL550/560 provides a simple mechanism for performing this conversion using an external 8-line static RAM strip buffer. During compression, the CL550/560 stores the incoming data in the strip buffer in raster format. Once eight complete lines of data have been stored, the CL550/560 reads the data back out as 8x8 blocks. These blocks are then compressed to JPEG specifications and sent to the host system over the Host bus. Addressing for the Strip buffer is designed so that only eight lines of SRAM storage are required. The Strip buffer is discussed in detail in Section 5.2.1.

The still-frame system configuration uses the CL550/560 processor as a compression/decompression co-processor in a microprocessor based system. This configuration is typically used to compress and decompress still-frame images in applications such as scanners, printers or copiers where software-based JPEG performance is too slow to do the job. An example of this design is shown in Figure 5-9.



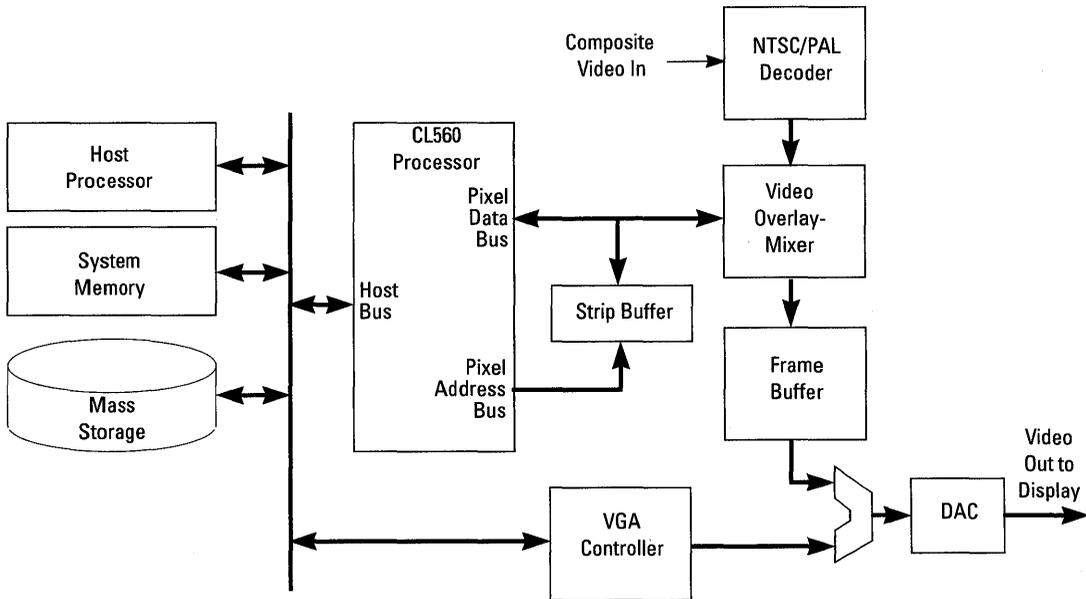
**Figure 5-9 Typical Still-frame Application**

In this configuration, a CL550/560 is used with both its Pixel data bus and its Host data bus connected to the system data bus. In a typical compression operation, the host writes pixel data to a pixel data latch, where

it is read by the CL550/560. The host processor then reads the compressed data from the Host bus interface. Because the CL550/560 host bus and the pixel bus are connected to the same system bus the CL550/560 must be put into a stalled state between pixel accesses and during host accesses.

The strip buffer is optional in this application because the host processor can perform the pixel reordering in software. This further reduces the hardware requirements of an already simple design.

Connections in multimedia applications are similar to those in a video system application, but the CL560 must co-exist with other video peripherals in the system. In this application, the CL560 is connected to Video overlay and mixer logic.



**Figure 5-10 Typical Multimedia System Application**

The strip buffer is required in this application to ensure that the CL560 can meet the speed requirements of real-time video compression and decompression system.

This section describes the timing waveforms seen in a CL550/560 based system. The example system uses master-mode compression in the 4:4:4-to-4:2:2 mode. The video parameters are set to the following:

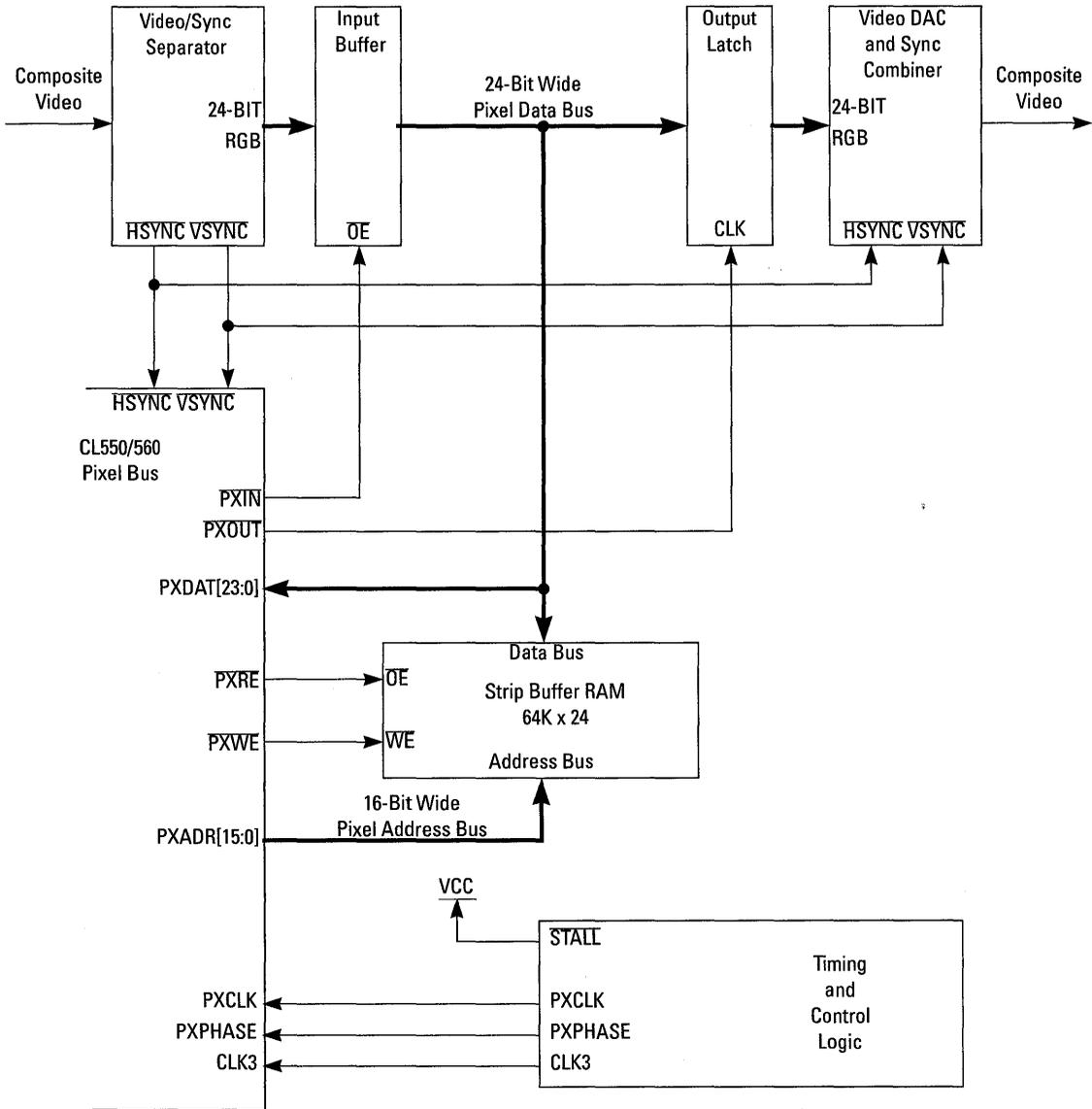
**Table 5-3 Compression Timing Example Register Values**

Register	Value	Comments
HPeriod	56	57 pixels per horizontal line
HSync	9	HSYNC pulse is 10 pixels wide
HDelay	6	6-pixel delay from falling edge of HSYNC to the first active pixel
HActive	11	48 pixels per active line
VPeriod	53	53 active 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

Figure 5-11 shows the design of the logic used in this example. Figure 5-12 shows the timing diagram for one complete vertical period, and Figure 5-13 through Figure 5-16 show details of important events during that vertical period.

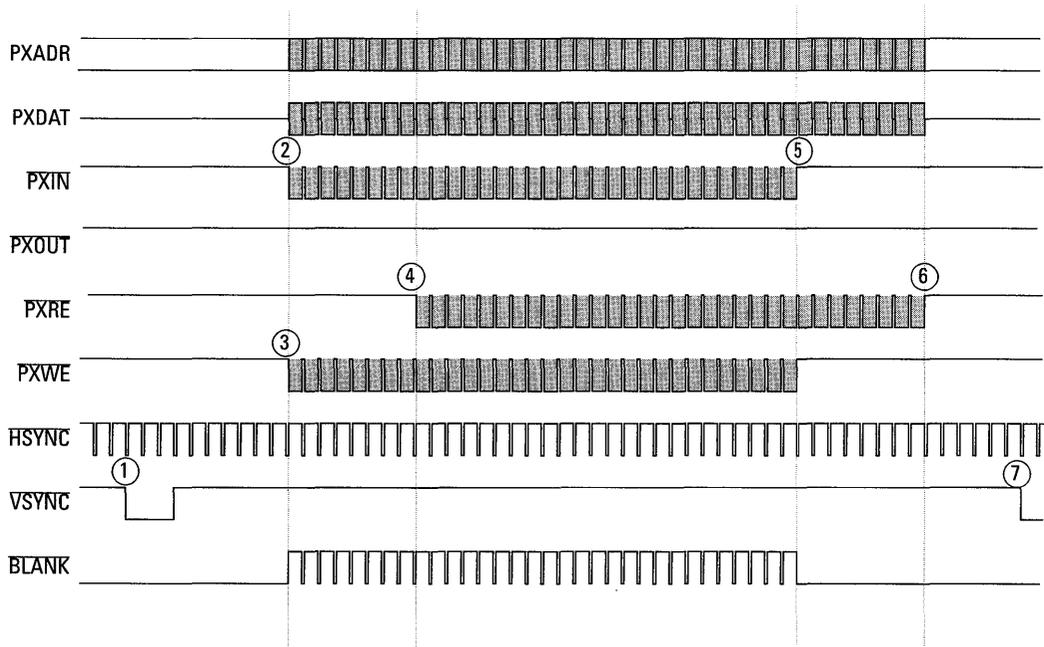
## 5.4 Timing Diagrams Compression Mode

# Timing Diagrams Compression Mode



**Figure 5-11 Typical CL560 Synchronous Interface**

Figure 5-12 shows the timing diagram for a typical compression cycle. The circled numbers in the figure refer to the steps below. The grayed out areas of the timing diagram indicate times where there are too many transitions to show in the limited space available.



**Figure 5-12 Compression Overview**

1. The CL550/560 starts the compression cycle by generating a pulse on the  $\overline{\text{VSYNC}}$  output. This pulse is three lines wide (the width is determined by the value programmed into the VSync register).
2. Ten lines after  $\overline{\text{VSYNC}}$  (the value programmed into VDelay), the CL550/560 starts to input pixel data. It asserts  $\overline{\text{PXIN}}$  and deasserts  $\overline{\text{BLANK}}$  to enable the data from the external source onto the  $\overline{\text{PXDAT}}$  bus, and  $\overline{\text{PXADR}}$  to select the address in the Strip buffer where the data will be written.
3. The CL550/560 asserts  $\overline{\text{PXWE}}$  to allow the pixel data to be written directly into the strip buffer address pointed to by  $\overline{\text{PXA-}}$

DR. For the first eight lines, the CL550/560 will only write data into the strip buffer. Expanded timing for the first line is shown in detail in Figure 5-13.

4. The CL550/560 changes modes after the first eight lines of video have been stored in the strip buffer. The CL550/560 now alternates between reading data from the strip buffer for raster to block conversion of the first eight lines, and writing data to the strip buffer for raster to block conversion of the next eight lines. This process continues until the last visible line of the frame. Expanded timing for the eight lines is shown in detail in Figure 5-14.
5. The CL550/560 stops inputting data after 32 lines (the value programmed into VActive). It stops generating  $\overline{\text{PXWE}}$  and  $\overline{\text{PXIN}}$  and asserts  $\overline{\text{BLANK}}$  to stop the flow of data from the video source onto the PXDAT bus. Note that the PXDAT and PXADR busses and  $\overline{\text{PXRE}}$  remain active because the CL550/560 is still performing raster-to-block conversion. Expanded timing for the last visible line is shown in detail in Figure 5-15.
6. The CL550/560 has completed the raster-to-block conversion eight lines after the end of visible video. It stops generating  $\overline{\text{PXRE}}$  signals and releases the PXADR and PXDAT busses. Timing for the last line with  $\overline{\text{PXRE}}$  is shown in Figure 5-16.
7. The CL550/560 continues generating  $\overline{\text{HSYNC}}$  pulses until the end of the frame.  $\overline{\text{VSYNC}}$  begins after the 53rd  $\overline{\text{HSYNC}}$  pulse (The value programmed into HPeriod).

Figure 5-13 shows the beginning of the first active line in the frame. This diagram is an expansion of the area shown at step 3 of Figure 5-12. The circled numbers in the figure refer to the steps below:

1. The CL550/560 outputs a  $\overline{\text{HSYNC}}$  pulse to begin the horizontal line. The width of the  $\overline{\text{HSYNC}}$  pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to input data. The CL550/560 indicates the start of the active line by negating  $\overline{\text{BLANK}}$ .
3. The CL550/560 writes the first pixel into the strip buffer by enabling the data onto the PXDAT bus with the  $\overline{\text{PXIN}}$  signal and asserting  $\overline{\text{PXWE}}$  to write the data into the RAM. The address that the data will be written to is determined by the contents of the PXADR bus.
4. During the first eight lines, there is no activity on  $\overline{\text{PXRE}}$ , and the PXDAT bus is allowed to float when  $\overline{\text{PXIN}}$  is HIGH.
5.  $\overline{\text{HSYNC}}$  goes HIGH at the end of the time determined by the value written into the HSync register.
6. The CL550/560 continues to write pixels into the strip buffer until the end of the horizontal line (not shown).

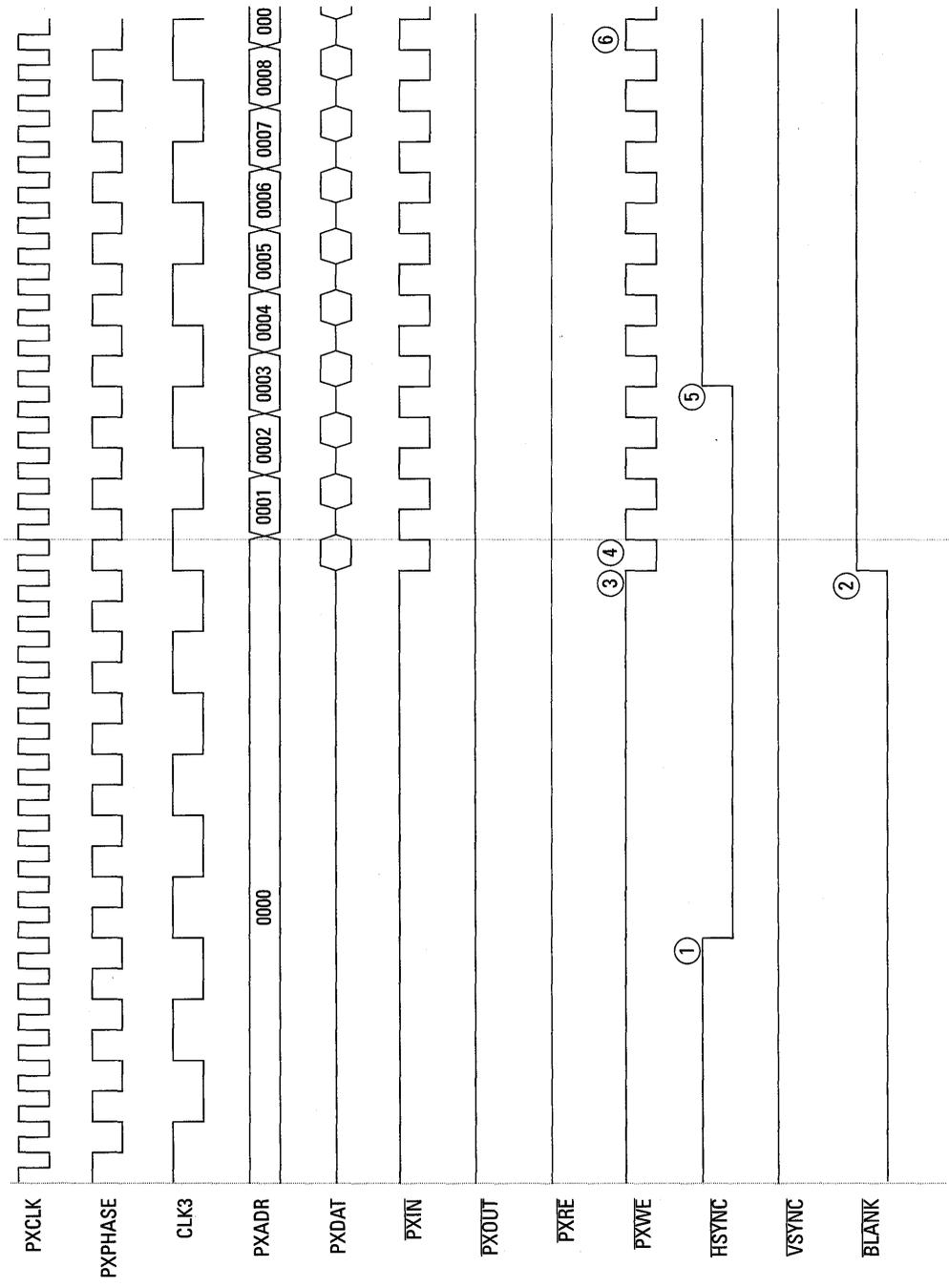


Figure 5-13 First Active Line (Compression)

Figure 5-14 shows the ninth active line in the frame. At this point, the CL550/560 starts to alternate between performing a raster-to-block conversion of the first eight lines and inputting data for the next eight lines. The strip buffer addressing counter is designed so that the pixel that is going to be input will occupy the address that was just vacated by the pixel that was read for the raster-to-block conversion.

This diagram is an expansion of the area shown at step 4 of Figure 5-12. The circled numbers in the figure refer to the steps below:

1. The CL550/560 outputs a  $\overline{\text{HSYNC}}$  pulse to begin the horizontal line. The width of the  $\overline{\text{HSYNC}}$  pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to input data. The CL550/560 indicates the start of the active line by negating  $\overline{\text{BLANK}}$ .
3. The CL550/560 reads the first pixel of the first block from the strip buffer RAM to start the raster to block conversion of the first eight lines. The address that the data will be read from is determined by the contents of the PXADR bus.
4. The CL550/560 writes the first pixel of the ninth line into the strip buffer by enabling the data onto the PXDAT bus with the  $\overline{\text{PXIN}}$  signal and asserting  $\overline{\text{PXWE}}$  to write the data into the RAM. The address that the data will be written to is the same address that was just read.
5.  $\overline{\text{HSYNC}}$  goes HIGH at the end of the time determined by the value written into the HSync register.
6. The process of reading an old pixel for the raster to block conversion, and writing a new pixel into the same location continues until the end of the horizontal line. In this example, it continues until 48 pixels have been processed.

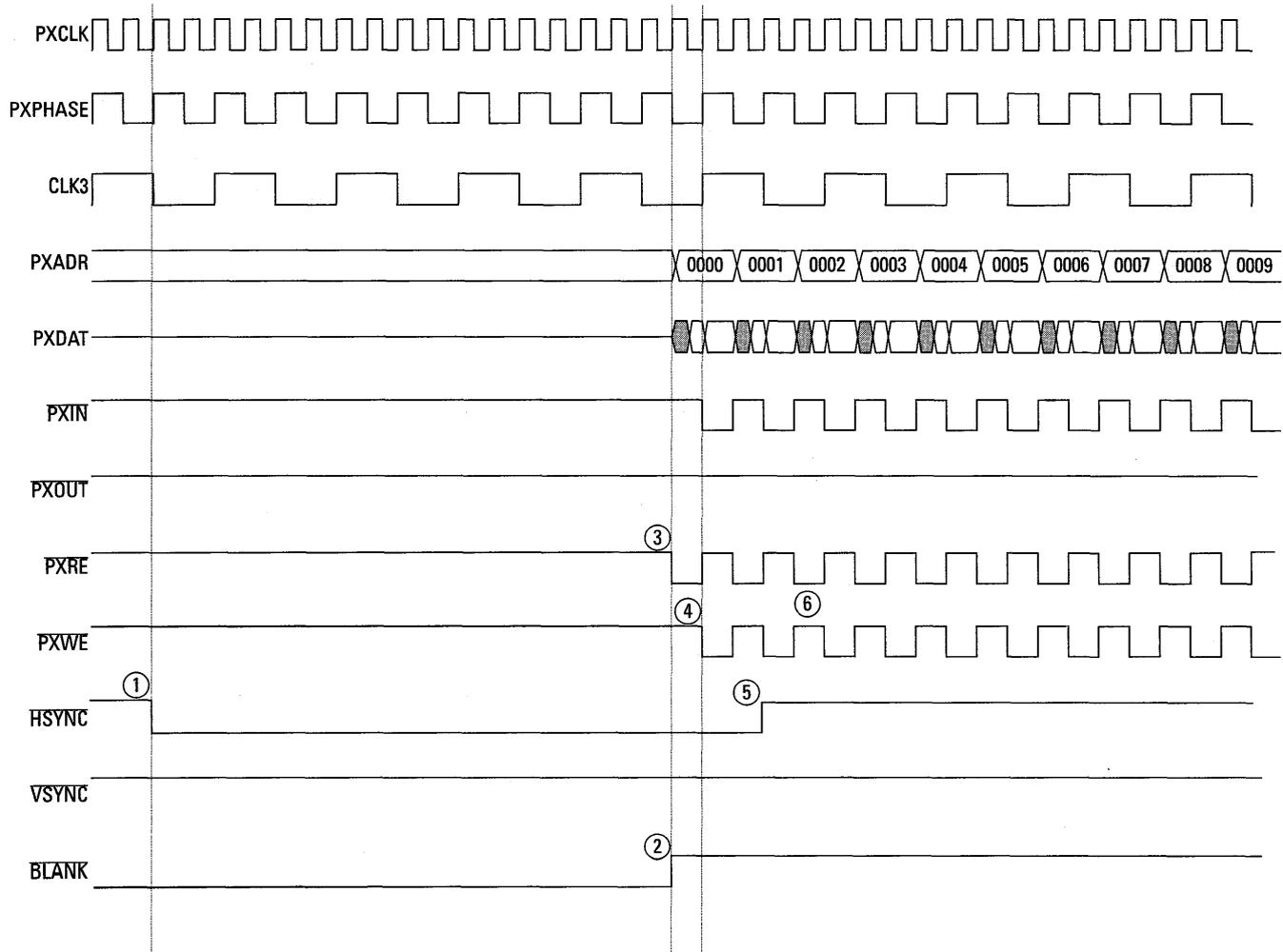


Figure 5-14 Beginning of the First Active Line with Active PXRE (Compression)

Figure 5-15 shows both the last line with active **PXIN** and **PXWE**, and the beginning of the line following. At this point, the CL550/560 is through accepting new data, but still needs to perform a raster to block conversion of the last eight lines.

This diagram is an expansion of the area shown at step 5 of Figure 5-12. The circled numbers in the diagram refer to the steps below:

1. The CL550/560 outputs a **HSYNC** pulse to begin the horizontal line. The width of the **HSYNC** pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to input data. The CL550/560 indicates the start of the active line by asserting **BLANK**.
3. The CL550/560 reads the first pixel of the first block from the strip buffer RAM to start the raster to block conversion of the first eight lines. The address that the data will be read from is determined by the contents of the **PXADR** bus.
4. The CL550/560 writes the first pixel of the last visible line into the strip buffer by enabling the data onto the **PXDAT** bus with the **PXIN** signal and asserting **PXWE** to write the data into the RAM. The address that the data will be written to is the same address that was just read.
5. **HSYNC** goes HIGH at the end of the time determined by the value written into the HSync register.
6. The process of reading an old pixel for the raster to block conversion, and writing a new pixel into the same location continues until the end on the horizontal line. In this example, it continues until 48 pixels have been processed.
7. At the end of the last active line, **BLANK** goes LOW to indicate that the CL550/560 will no longer input data.
8. The raster to block conversion must continue for eight more lines to allow the last eight lines of the active frame to be compressed. Note at this point that data is being read from the strip buffer, but no new data is being written.

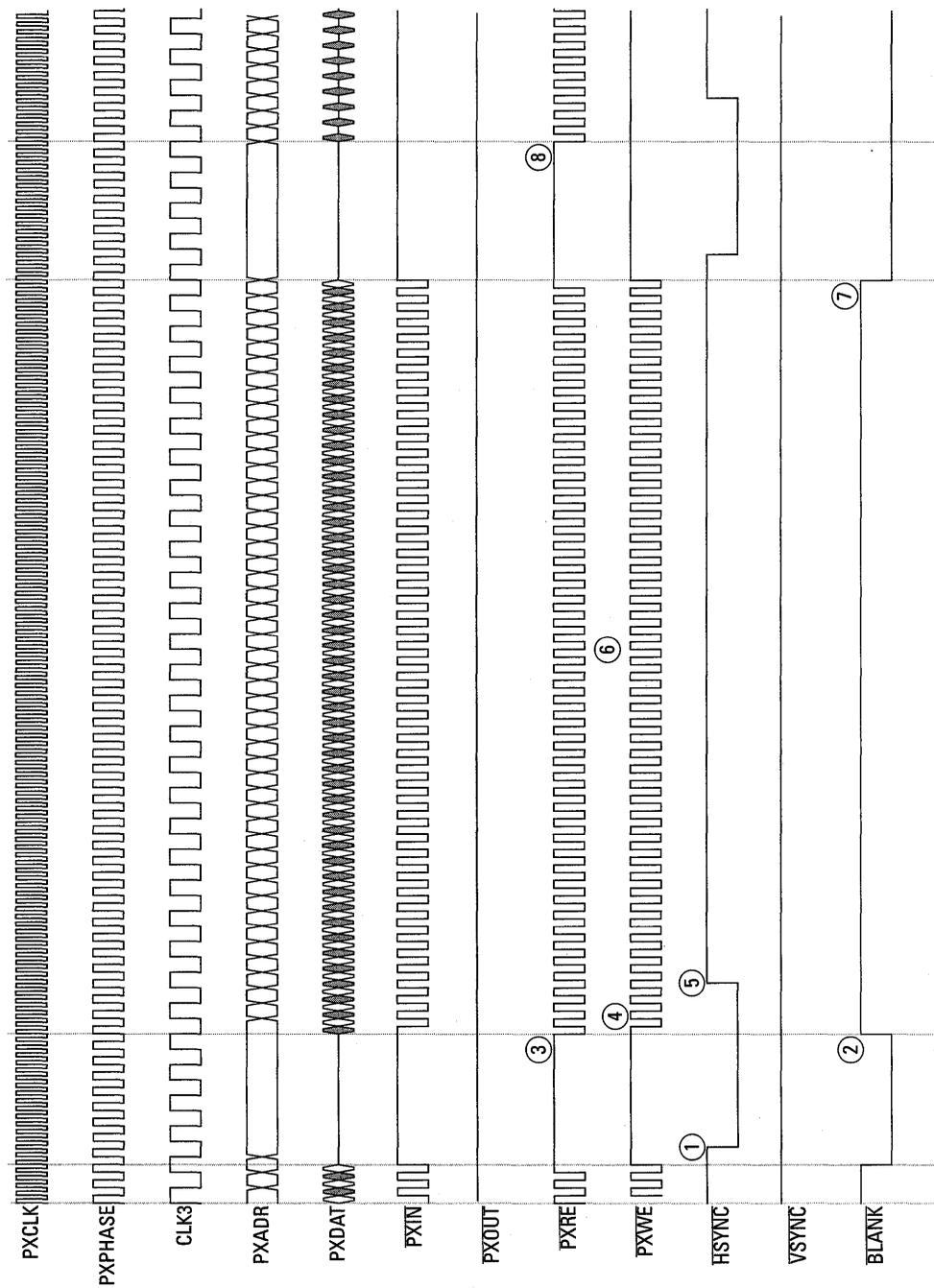


Figure 5-15 Last Line with Active PXIN (Compression)

Figure 5-16 shows the last line with active  $\overline{\text{PXRE}}$ . At the end of this line, the CL550/560 has completed raster to block conversion of the last eight lines of the active frame. Beyond this point  $\overline{\text{PXIN}}$ ,  $\overline{\text{PXRE}}$ , and  $\overline{\text{PXWE}}$  remain inactive until the first line of the next vertical frame.

This diagram is an expansion of the area shown at step 6 of Figure 5-12. The circled numbers in the diagram refer to the steps below:

1. The CL550/560 outputs a  $\overline{\text{HSYNC}}$  pulse to begin the horizontal line. The width of the  $\overline{\text{HSYNC}}$  pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to input data.
3. The CL550/560 asserts  $\overline{\text{PXRE}}$  to read the first pixel of the last line from the Strip buffer RAM. This starts the raster-to-block conversion of the last line. The address that the data will be read from is determined by the contents of the PXADR bus.
4. The raster to block conversion process continues until the end of the horizontal line. At this point  $\overline{\text{PXIN}}$ ,  $\overline{\text{PXRE}}$ , and  $\overline{\text{PXWE}}$  go inactive and will remain that way until the end of the vertical frame.
5. After the last pixel is read, compression continues until the compression pipeline has flushed all of the remaining data to the FIFO. You should be aware that the possibility of FIFO overflow exists during this time, even though no video bus signals are active. The FIFO level can be monitored using  $\overline{\text{NMRQ}}$ ,  $\text{HALF\_FULL}$ ,  $\overline{\text{IRQ1}}$  or  $\overline{\text{IRQ2}}$ .

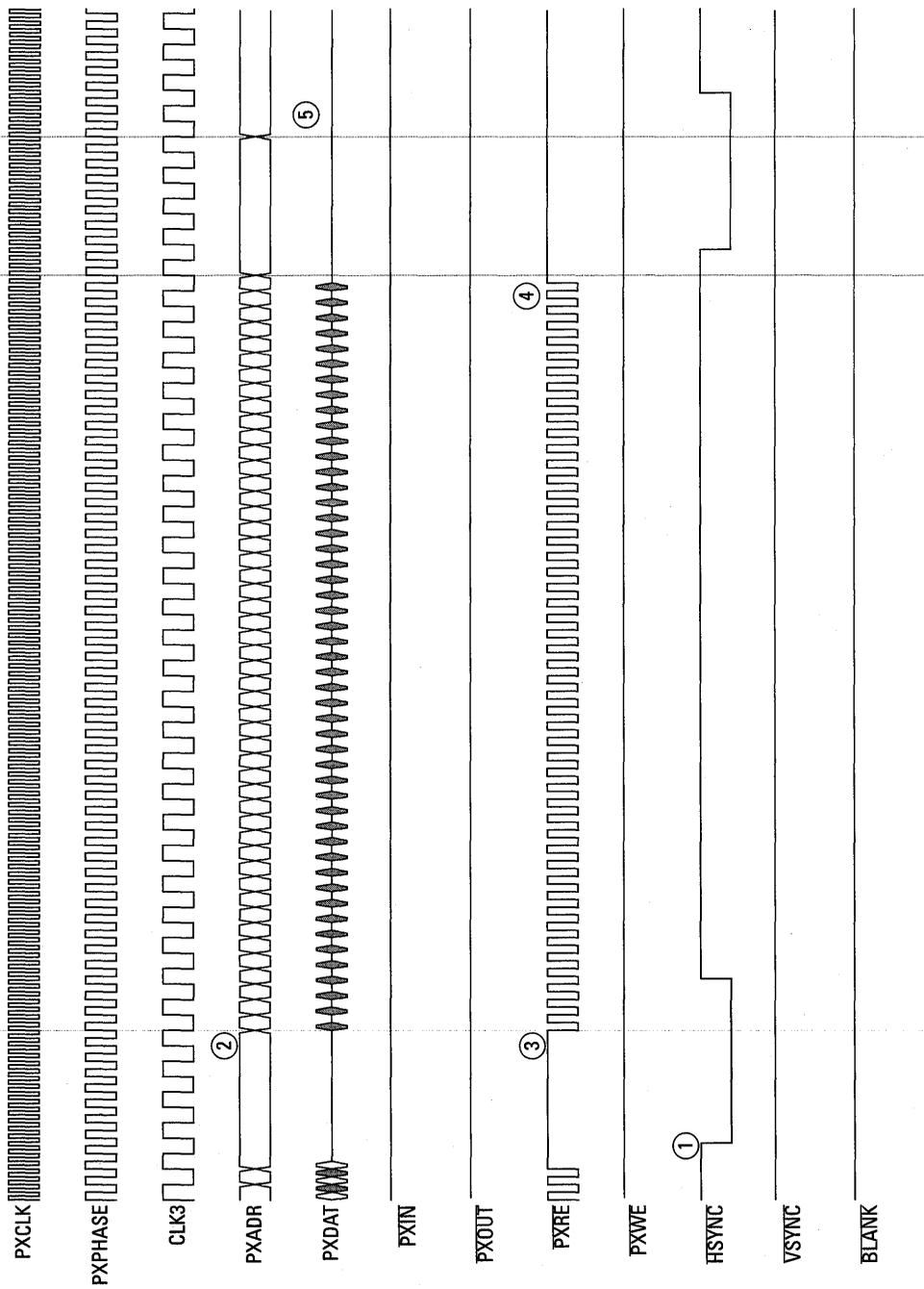


Figure 5-16 Last Line with Active PXRE (Compression)

This section describes the timing waveforms seen in a CL550/560 based system. The example system uses master-mode decompression in the 4:4:4 to 4:2:2 mode. The video parameters are set to the following:

---

## 5.5 Timing Diagrams Decompression Mode

---

**Table 5-4**      **Timing Example Video Parameters**

Register	Value	Comments
HPeriod	56	57 pixels per horizontal line
HSync	9	HSYNC pulse is 10 pixels wide
HDelay	6	6-pixel delay from falling edge of HSYNC to the first active pixel
HActive	11	48 pixels per active line
VPeriod	53	53 active lines
VSync	3	VSYNC pulse is 3 lines wide
VDelay	5	10-line delay from the falling edge of VSYNC to the first active line
VActive	4	32 active lines

Figure 5-17 shows the timing diagram for one complete vertical period, and Figure 5-18 through Figure 5-21 show details of important events during that vertical period.

Figure 5-17 shows the timing cycle for a typical decompression cycle. The grayed out areas of the timing diagram indicate times where there are too many transitions to show in the limited space available.



4. The CL550/560 stops writing blocks to be converted to raster format eight lines before the end of visible video. Expanded timing for this area is shown in Figure 5-20.
5. The CL550/560 stops reading pixels from the strip buffer RAM at the end of the visible frame. Expanded timing for the last visible line is shown in Figure 5-21.
6. The CL550/560 continues to generate  $\overline{\text{HSYNC}}$  pulses until the end of the frame.  $\overline{\text{VSYNC}}$  begins again after the 53rd  $\overline{\text{HSYNC}}$  pulse (the value programmed into HPeriod) has been generated.

Figure 5-18 shows the signal activity around the first data written to the strip buffer RAM. This data is the pixel data for the first eight lines of visible video. It will be written into the strip buffer RAM in block format, and read back out in raster format.

This diagram is an expansion of the area shown at step 2 of Figure 5-17. The circled numbers in the figure refer to the steps below:

1. The CL550/560 writes the first group of pixels into the strip buffer RAM. Note that this group of pixels is not bounded by a  $\overline{\text{HSYNC}}$  interval, because decompression from the FIFO starts on a  $\overline{\text{HSYNC}}$  and the data is written to the strip buffer after a delay of HDelay plus the CL550/560 pipeline latency. On all subsequent lines, the CL550/560 waits until the start of the horizontal line to process data.
2. The CL550/560 generates a pulse on  $\overline{\text{HSYNC}}$  to synchronize the system.
3.  $\overline{\text{HSYNC}}$  goes high after ten pixel clocks (the value programmed into the HSync register).
4. The CL550/560 continues writing pixels into the strip buffer RAM. This process continues until all 48 (HActive) pixels have been written.

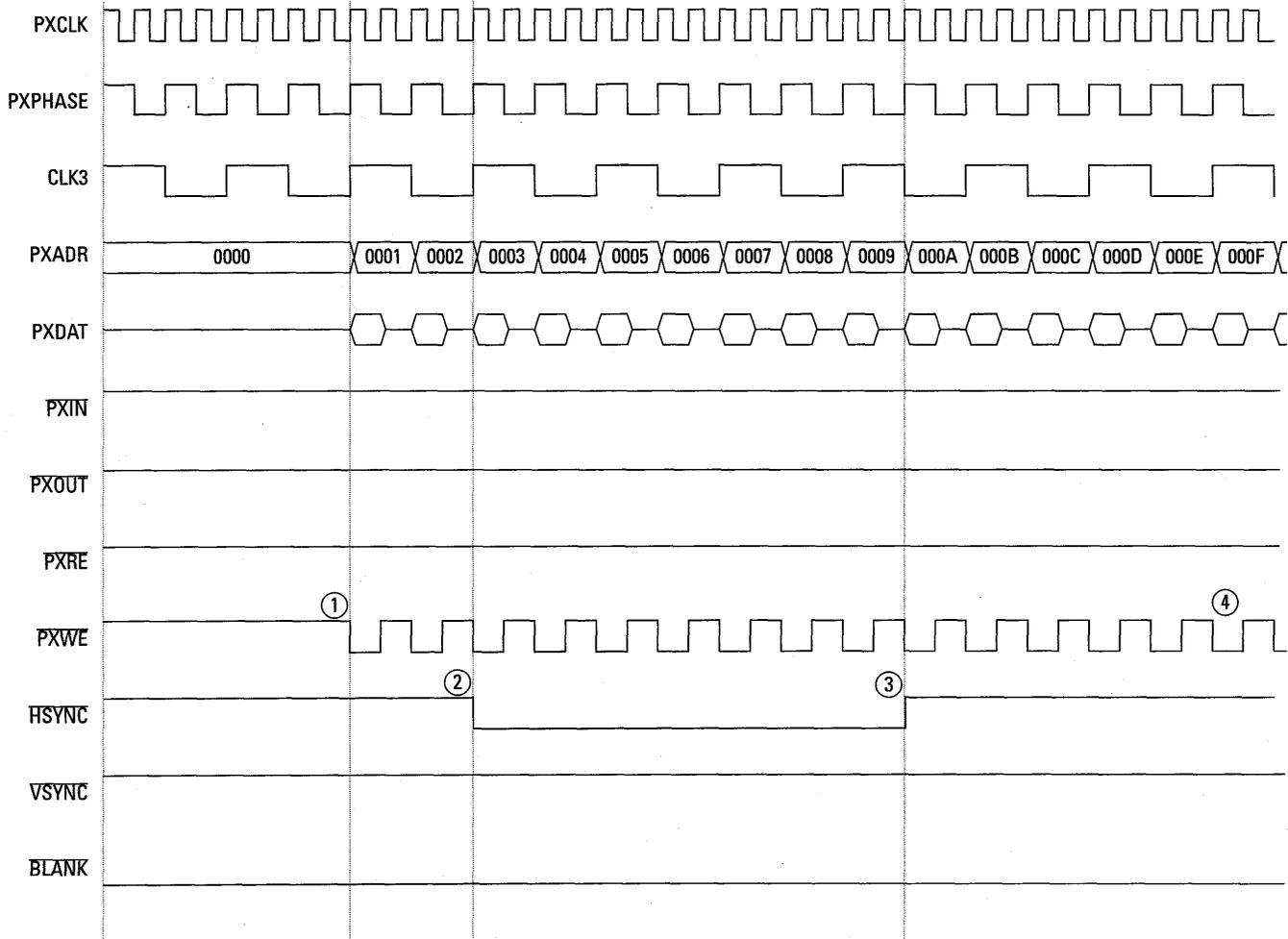


Figure 5-18 First Line with Active PXWE (Decompression)

Figure 5-19 shows the beginning of the ninth line in the frame (the first line of active video). At this point, the CL550/560 starts to alternate between reading the pixel data out of the strip buffer in raster order and writing new pixel data in block order.

This diagram is an expansion of the area shown at step 3 of Figure 5-17. The circled numbers in the diagram refer to the steps below:

1. The CL550/560 outputs a  $\overline{\text{HSYNC}}$  pulse to start the horizontal line. The width of the  $\overline{\text{HSYNC}}$  pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to output the first pixel. The CL550/560 indicates the start of active video by deasserting  $\overline{\text{BLANK}}$ .
3. The CL550/560 outputs the first raster format pixel from the Strip buffer RAM on PXDAT. This pixel was written into the RAM in block format eight lines earlier. The address that the data will be read from is determined by the contents of the PX-ADR bus.
4. The CL550/560 writes a new block format pixel into the Strip buffer address just vacated. This block will be converted into raster format in eight lines.
5. The CL550/560 drives  $\overline{\text{HSYNC}}$  HIGH at the end of the time determined by the value written into the HSync register.
6. The CL550/560 continues the process of reading a pixel that has been converted and sending it out to the display device, and writing a pixel to be converted back into the same address. This process continues until the end of the horizontal line. In this example, it continues until 48 (determined by HActive) pixels have been processed.

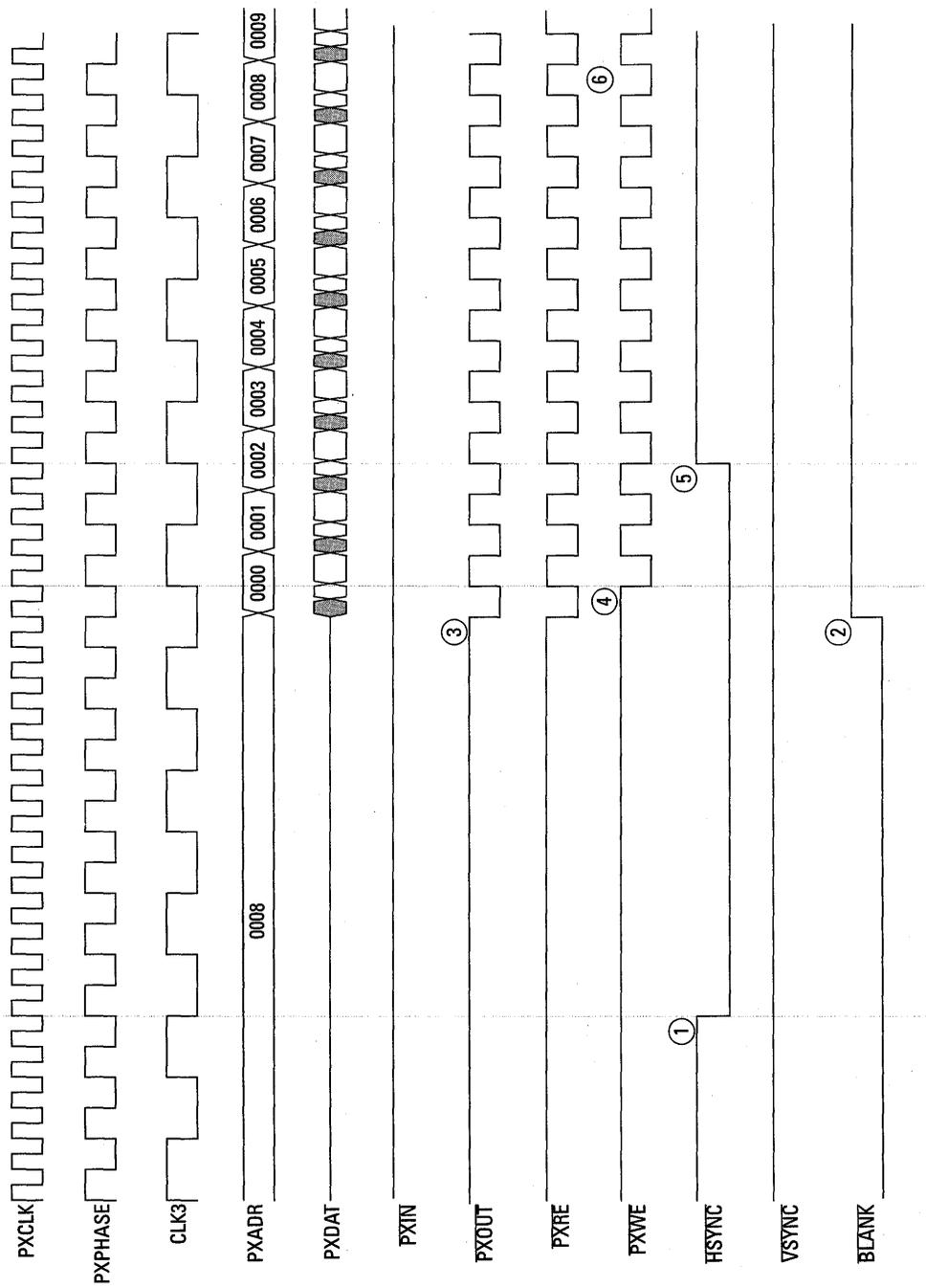


Figure 5-19 First Active Line with Active PXOUT

Figure 5-20 shows the signal activity around the last line with **PXOUT**, **PXRE** and **PXWE** active. At this point, the CL550/560 has written the last eight lines of pixel blocks into the strip buffer, and is starting to output them in raster format.

This diagram is an expansion of the area shown at step 4 of Figure 5-17. The circled numbers in the diagram refer to the steps below:

1. The CL550/560 outputs a **HSYNC** pulse to start the horizontal line. The width of the **HSYNC** pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to output the first pixel. The CL550/560 indicates the start of active video by deasserting **BLANK**.
3. The CL550/560 outputs the first raster format pixel from the strip buffer RAM. This pixel was written into the RAM in block format eight lines earlier. The address that the data will be read from is determined by the contents of the **PXADR** bus.
4. The CL550/560 writes a new block format pixel into the strip buffer address just vacated. This block will be converted into raster format in eight lines.
5. The CL550/560 drives **HSYNC** HIGH at the end of the time determined by the value written into the HSync register.
6. The CL550/560 writes the last pixel of the last block to be converted into the strip buffer RAM.
7. The CL550/560 reads the first pixel of the eighth from the last line out. At this point, the CL550/560 is reading the pixels from the strip buffer without a corresponding write, because there are no more blocks to be converted.

Timing Diagrams Decompression Mode

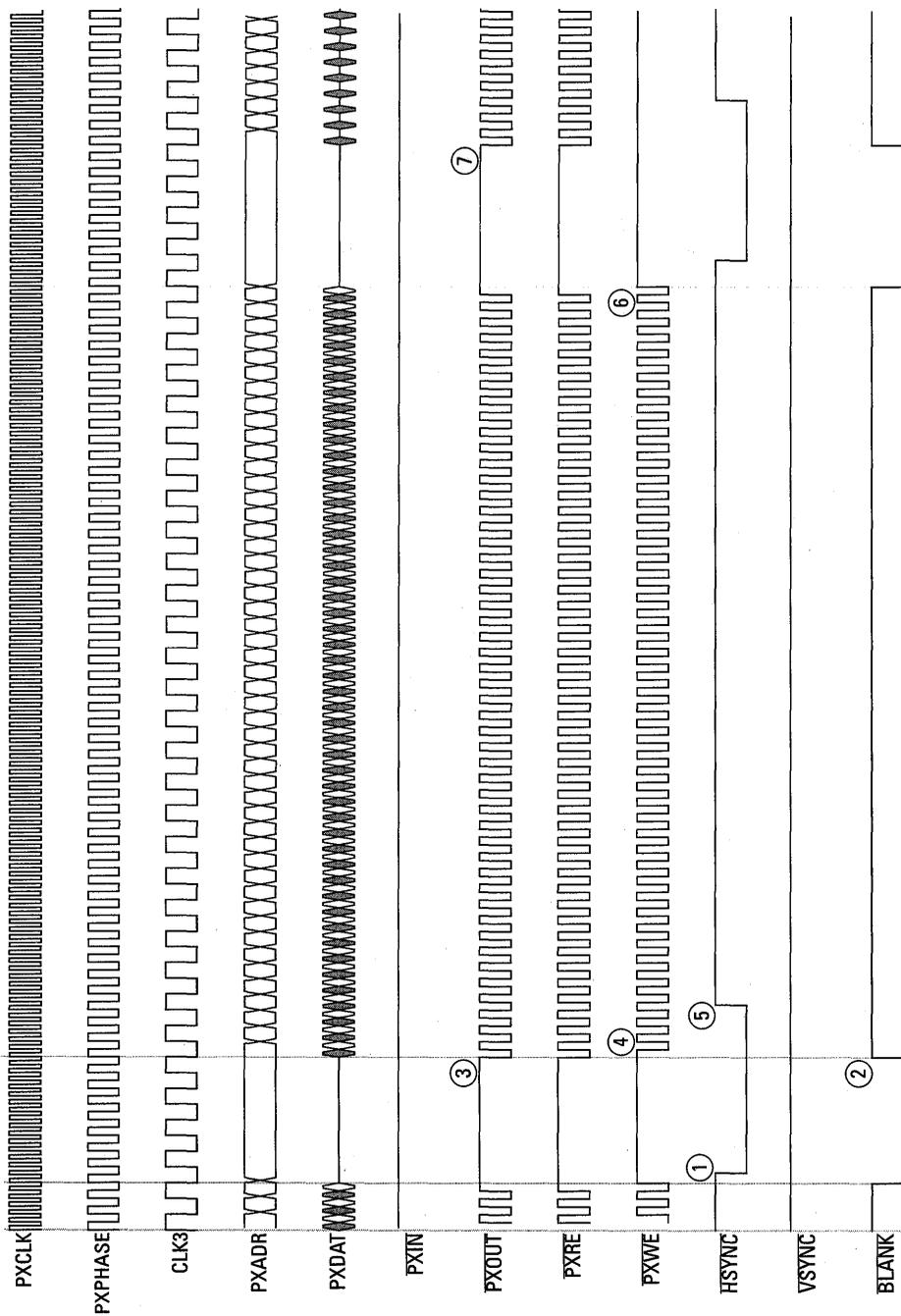


Figure 5-20 Last Active Line with PXWE

Figure 5-21 shows the signal activity around the last visible line of the frame. The CL550/560 is reading the last line of the last block at this point.

1. The CL550/560 outputs a  $\overline{\text{HSYNC}}$  pulse to start the horizontal line. The width of the  $\overline{\text{HSYNC}}$  pulse is determined by the value programmed into the HSync register (10 pixels in this example).
2. The CL550/560 waits until the delay amount programmed into the HDelay register (6 pixels in this example) before it starts to output the first pixel. The CL550/560 indicates the start of active video by deasserting **BLANK**.
3. The CL550/560 outputs the first pixel of the last line. This pixel was written into the RAM in block format eight lines earlier. The address that the data will be read from is determined by the contents of the PXADR bus.
4. The CL550/560 drives  $\overline{\text{HSYNC}}$  HIGH at the end of the time determined by the value written into the HSync register.
5. The CL550/560 outputs the last pixel of the last line. At this time, the CL550/560 is through converting the last eight lines, and **PXOUT** and **PXRE** become inactive.

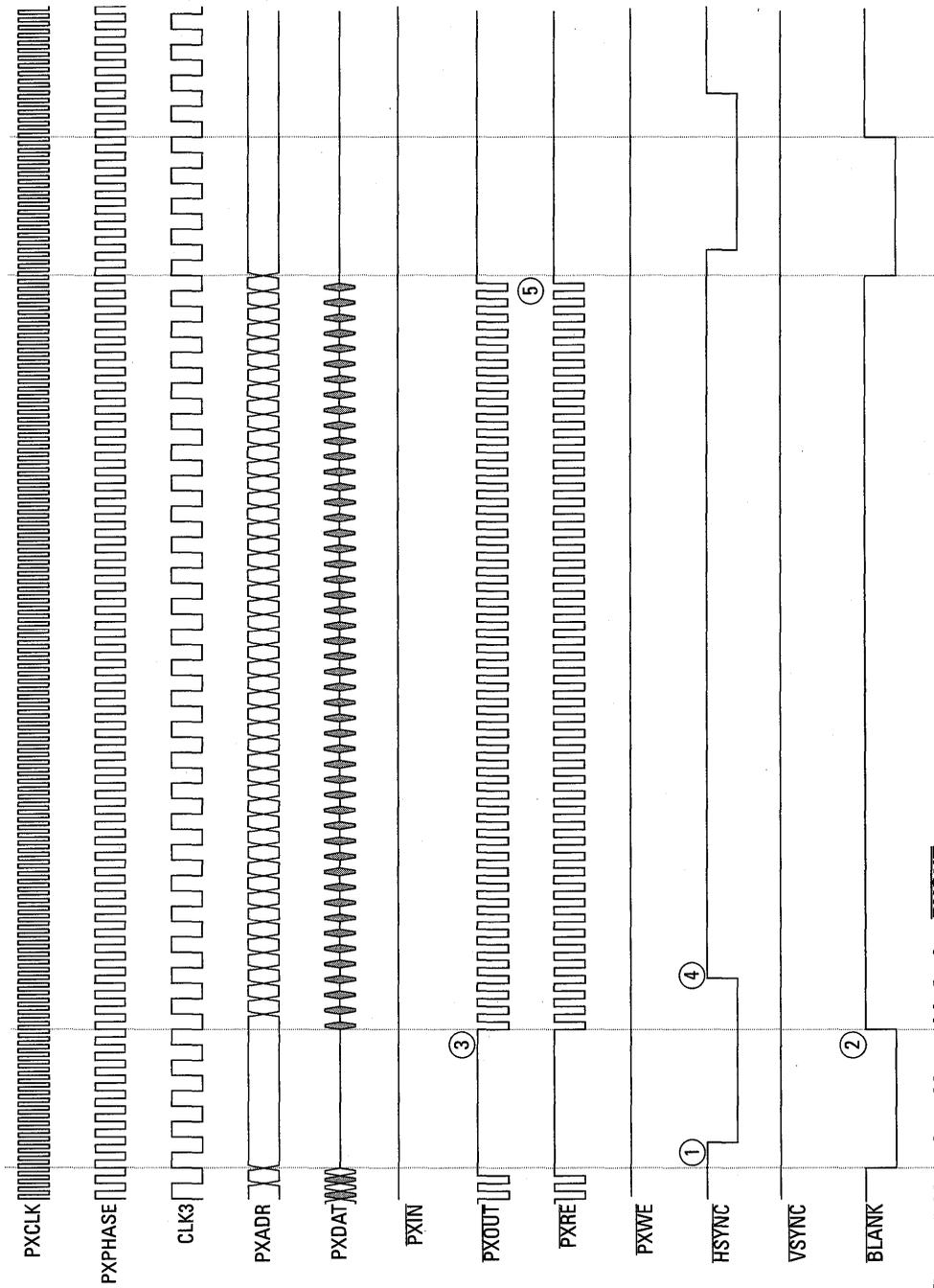


Figure 5-21 Last Line with Active PXOUT

# 6 Specifications

This chapter describes the electrical and mechanical characteristics of the CL550 and CL560. The chapter is divided into three sections:

- 6.1, Operating Conditions
- 6.2, AC Characteristics
- 6.3, Package Specifications

The AC and DC electrical parameters for the CL560 are based on characterization of initial silicon, but should be considered preliminary. A complete characterization of the silicon over all process corners had not been completed by the time that this manual was sent to the printers. Margins have been added to the parameters wherever possible. The final values of the electrical parameters are expected to be better than the preliminary values listed in this chapter.

**6.1  
Operating  
Conditions**

This section specifies the electrical characteristics of the CL550 and CL560. The CL560 numbers are preliminary and subject to change.

**Table 6-1 Absolute Maximum Ratings**

Parameter	Value
<b>Supply Voltage</b>	<b>-0.5 to 6.5 V</b>
Input Voltage	-1.0 to $V_{DD}$
Output Voltage	-0.5 to $V_{DD}$
Storage temperature range	-65 °C to 150 °C
Operating temperature range (case)	0°C to 90 °C

**Table 6-2 Operating Conditions**

Parameters	Test Conditions	Commercial		Unit
		Min	Max	
V <sub>DD</sub> Supply Voltage		4.75	5.25	V
t <sub>CASE</sub> Operating Temperature		0	85	°C

**Table 6-3 DC Characteristics**

Parameters	Test Conditions	Commercial			Unit
		Min	Typ	Max	
V <sub>IH</sub> High-level input voltage <sup>1</sup>	V <sub>DD</sub> = MAX	2.4	1.3		V
V <sub>IL</sub> Low-level input voltage <sup>1</sup>	V <sub>DD</sub> = MIN		1.3	0.8	V
V <sub>OH</sub> High-level output voltage	V <sub>DD</sub> = MIN, I <sub>OH</sub> = -8.0 mA	2.4	4.3		V
V <sub>OL</sub> Low-level output voltage	V <sub>DD</sub> = MIN, I <sub>OL</sub> = 12.0 mA		0.3	0.5	V
I <sub>IH</sub> High-level input current	V <sub>DD</sub> = MAX, V <sub>IN</sub> = V <sub>DD</sub>		0.2	10	μA
I <sub>IL</sub> Low-level input current	V <sub>DD</sub> = MAX, V <sub>IN</sub> = 0 V	-10	-0.2		μA
I <sub>OZ</sub> Output leakage current	Hi-Z output driven to 0V and 5.25 V		±0.2	±10	μA
I <sub>DD0</sub> CL550 Supply Current V <sub>IN</sub> = 0 or V <sub>DD</sub> , C <sub>L</sub> = 50pF	V <sub>DD</sub> = MAX, PXCLK = 0 MHz			260	mA
	V <sub>DD</sub> = MAX, PXCLK = 10 MHz			375	mA
	V <sub>DD</sub> = MAX, PXCLK = 30 MHz			590	mA
	V <sub>DD</sub> = MAX, PXCLK = 35 MHz			670	mA
I <sub>DD0</sub> CL560 Supply Current V <sub>IN</sub> = 0 or V <sub>DD</sub> , C <sub>L</sub> = 50pF	V <sub>DD</sub> = MAX, PXCLK = 0 MHz			300	mA
	V <sub>DD</sub> = MAX, PXCLK = 15 MHz			500	mA
	V <sub>DD</sub> = MAX, PXCLK = 30 MHz			650	mA
C <sub>IN</sub> Input Capacitance <sup>1</sup>			10		pF
C <sub>OUT</sub> Output Capacitance <sup>1</sup>			12		pF

1. Not 100% tested, guaranteed by design characterization

This section describes the AC timing characteristics of the CL550 and CL560. The timing characteristics are divided into related groups and depicted with one or more timing diagrams and a table of the timing values. The groups are:

- Host Interface Control Signal Timing
  - HBCLK and  $\overline{\text{RESET}}$  Timing
  - $\overline{\text{DRQ}}$  Timing
  - $\overline{\text{NMRQ}}$  ,  $\overline{\text{IRQ1}}$  Timing
  - HALF\_FULL,  $\overline{\text{IRQ2}}$ ,  $\overline{\text{FRMEND}}$  Timing
- Host Interface Memory and Register Timing
  - Host Bus Timing, Memory and Register Write
  - Host Bus Timing, Memory and Register Read
  - Host Bus Timing, Burst Mode Write
  - Host Bus Timing, Burst Mode Read
- Video Interface Timing
  - Video Interface Clock Timing
  - Video Interface Timing: Compression, Full-Rate Mode
  - Video Interface Timing: Decompression, Full-Rate Mode
  - Video Interface Timing: Compression, Half-Rate Mode
  - Video Interface Timing: Decompression, Half-Rate Mode

---

## 6.2 AC Characteristics

---

## 6.2.1 Host Interface Control Signal Timing

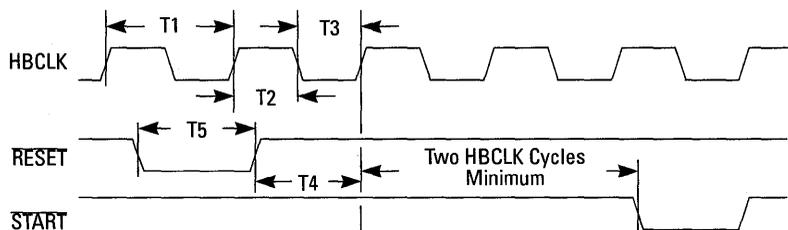


Figure 6-1 HBCLK and RESET Timing

Table 6-4 HBCLK and RESET Timing Parameters, CPGA Package

Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T1	HBCLK Clock Period	84		100		33		ns
T2	HBCLK Pulse Width HIGH	50		50		15		ns
T3	HBCLK Pulse Width LOW <sup>1</sup>	23		23		15		ns
T4	RESET Setup Period <sup>2,3</sup>	10		10		10		ns
T5	RESET Pulse Width LOW <sup>2</sup>	170		200		70		ns

Table 6-5 HBCLK and RESET Timing Parameters, MQUAD Package

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T1	HBCLK Clock Period	100		100		66		33		ns
T2	HBCLK Pulse Width HIGH	50		50		33		15		ns
T3	HBCLK Pulse Width LOW <sup>1</sup>	30 (23)		23		23		15		ns
T4	RESET Setup Period <sup>2,3</sup>	15		10		10		10		ns
T5	RESET Pulse Width LOW <sup>2</sup>	200		200		140		70		ns

1. Characteristics in parenthesis apply to part number CL550-10N. These are required for NuBus designs.

2. RESET is seen immediately when going low, but is only removed on a positive edge of HBCLK.

3. Two HBCLK cycles are required for internal reset release.

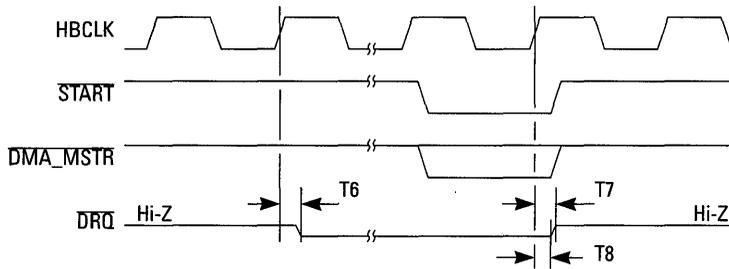


Figure 6-2 DRQ Timing

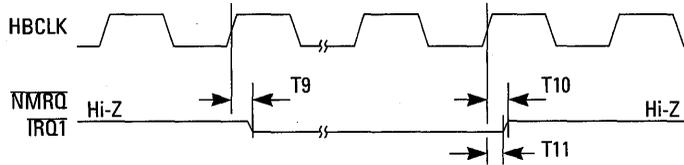
Table 6-6 DRQ Timing, CPGA Package

Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T6	DRQ Hi-Z to LOW Delay <sup>1</sup>		18		18		18	ns
T7	DRQ LOW to Hi-Z Delay <sup>2</sup>		18		18		18	ns
T8	DRQ Delay Hold Time <sup>2</sup>	5		5		5		ns

Table 6-7 DRQ Timing, MQUAD Package

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T6	DRQ Hi-Z to LOW Delay <sup>1</sup>		25		20		22		20	ns
T7	DRQ LOW to Hi-Z Delay <sup>2</sup>		25		20		22		20	ns
T8	DRQ Delay Hold Time <sup>2</sup>	3		5		5		5		ns

1. DRQ is an open-drain signal.
2. Not 100% tested, guaranteed by design characterization.



**Figure 6-3 NMRO, IRQ1 Timing**

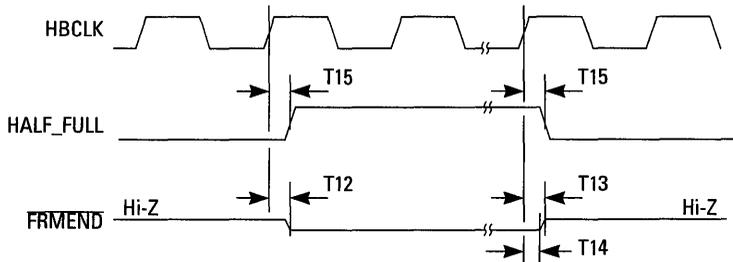
**Table 6-8 NMRO, IRQ1 Timing, CPGA Package**

Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T9	NMRO, IRQ1 Hi-Z to LOW Delay <sup>1,2</sup>		18		18		18	ns
T10	NMRO, IRQ1 LOW to Hi-Z Delay <sup>3</sup>		18		18		18	ns
T11	NMRO, IRQ1 Delay Hold Time	5		5		5		ns

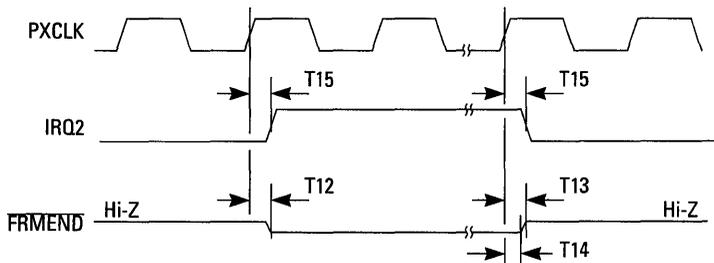
**Table 6-9 NMRO, IRQ1 Timing, MQUAD Package**

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T9	NMRO, IRQ1 Hi-Z to LOW Delay <sup>1,2</sup>		25		20		22		20	ns
T10	NMRO, IRQ1 LOW to Hi-Z Delay <sup>3</sup>		25		20		22		20	ns
T11	NMRO, IRQ1 Delay Hold Time	3		3		3		3		ns

1. NMRO and IRQ1 are open-drain signals.
2. NMRO and IRQ1 change on the positive edge of HBCLK, and are not related to any specific transaction phase.
3. Not 100% tested. Guaranteed by design characteristics.



**Figure 6-4 CL550 HALF\_FULL, FRMEND Timing**



**Figure 6-5 CL560 IRQ2, FRMEND Timing**

**Table 6-10 HALF\_FULL, FRMEND Timing, CPGA Package**

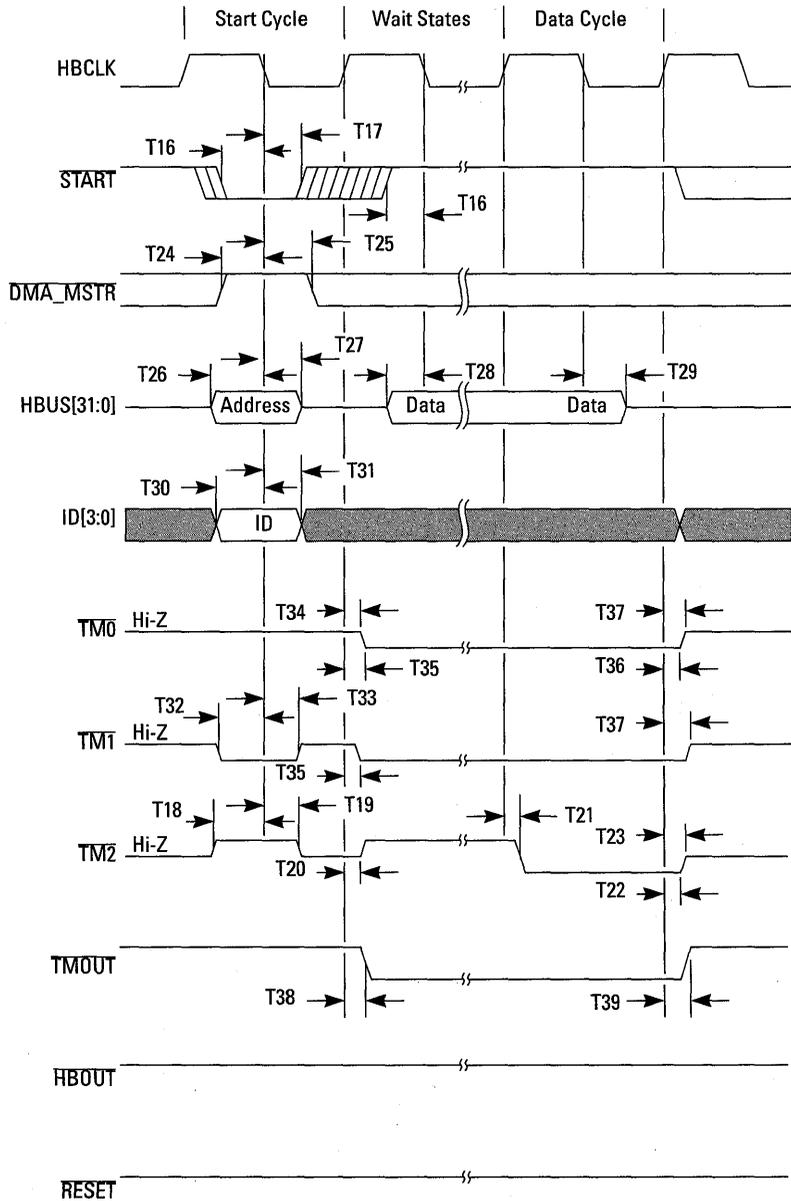
Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T12	FRMEND Hi-Z to LOW Delay <sup>1</sup>		23		25		18	ns
T13	FRMEND LOW to Hi-Z Delay <sup>2</sup>		23		25		18	ns
T14	FRMEND Delay Hold Time <sup>2</sup>	5		5		5		ns
T15	HALF_FULL / IRQ2 Delay		20		22		18	ns

**Table 6-11 HALF\_FULL, FRMEND Timing, MQUAD Package**

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T12	FRMEND Hi-Z to LOW Delay <sup>1</sup>		28		25		25		20	ns
T13	FRMEND LOW to Hi-Z Delay <sup>2</sup>		28		25		25		20	ns
T14	FRMEND Delay Hold Time <sup>2</sup>	3		5		5		5		ns
T15	HALF_FULL / IRQ2 Delay		25		22		22		20	ns

1. FRMEND is an open-drain signal.
2. Not 100% tested. Guaranteed by design characteristics.

# AC Characteristics



**Figure 6-6 CL550 Host Interface Timing: Register and Memory Write**

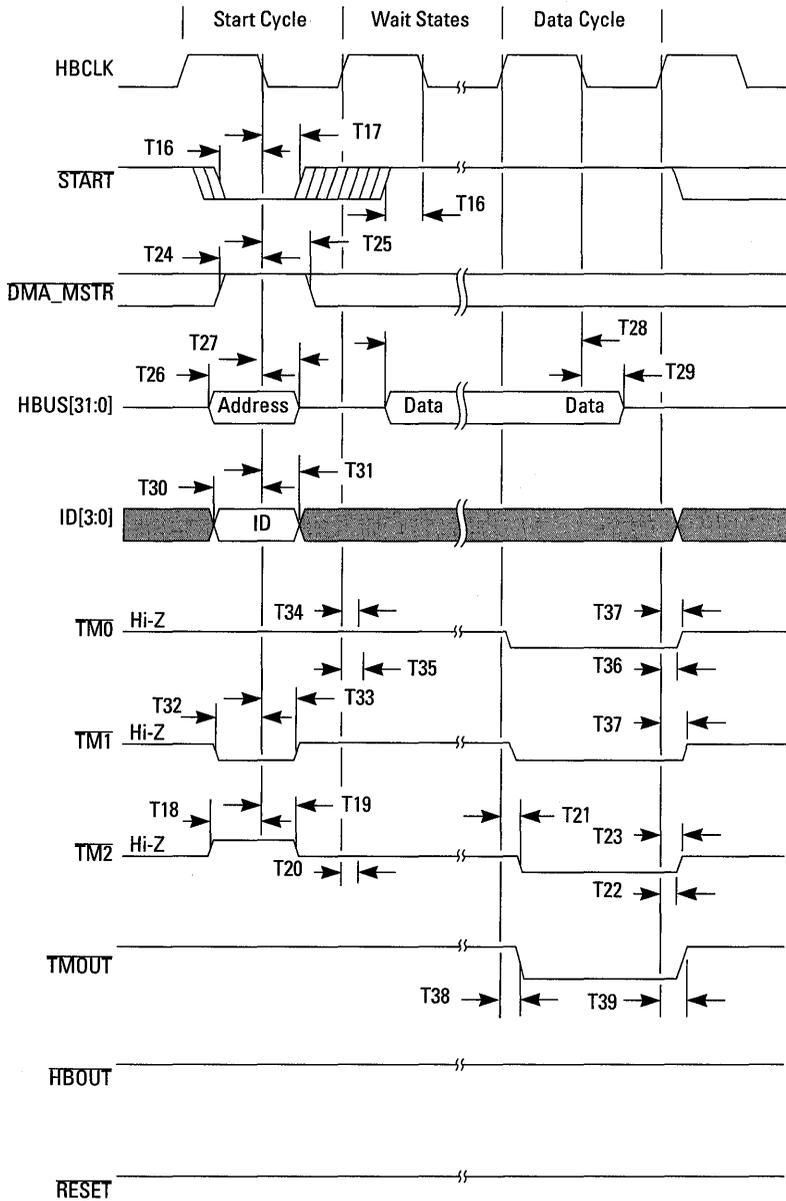


Figure 6-7 CL560 Host Interface Timing: Register and Memory Write

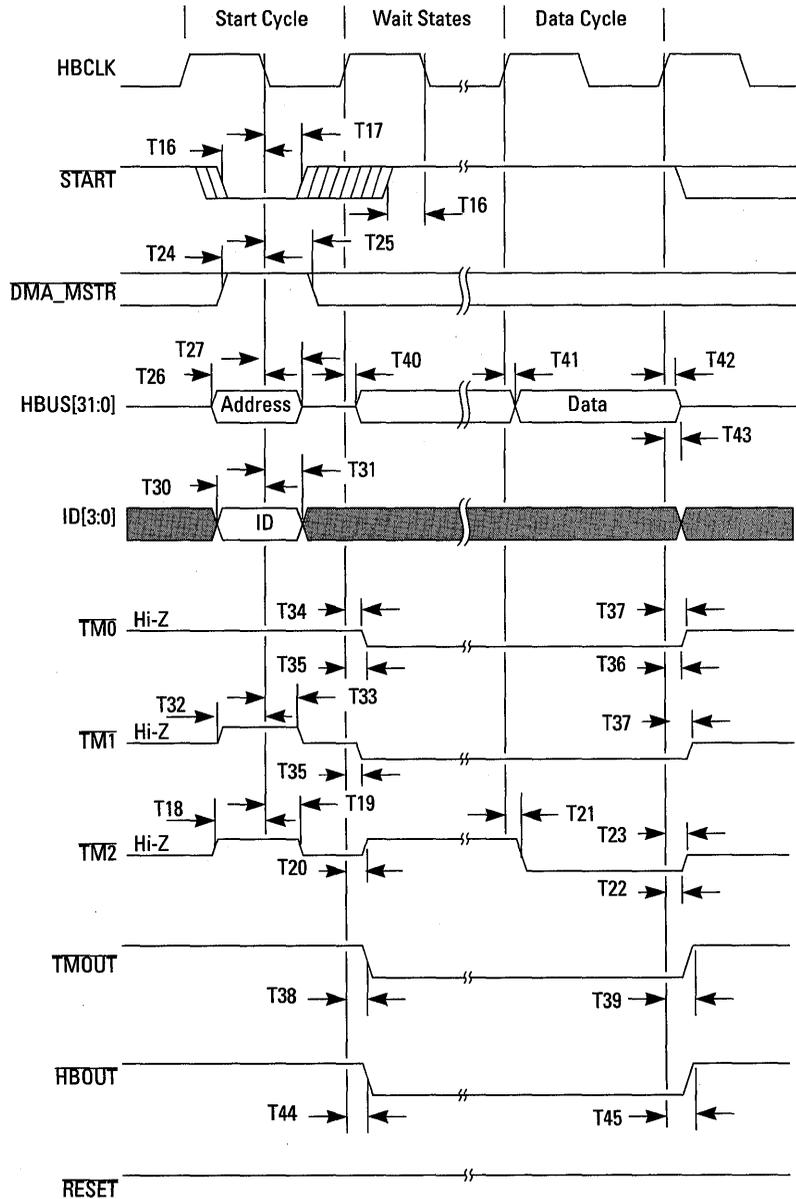


Figure 6-8 CL550 Host Interface Timing: Register and Memory Read

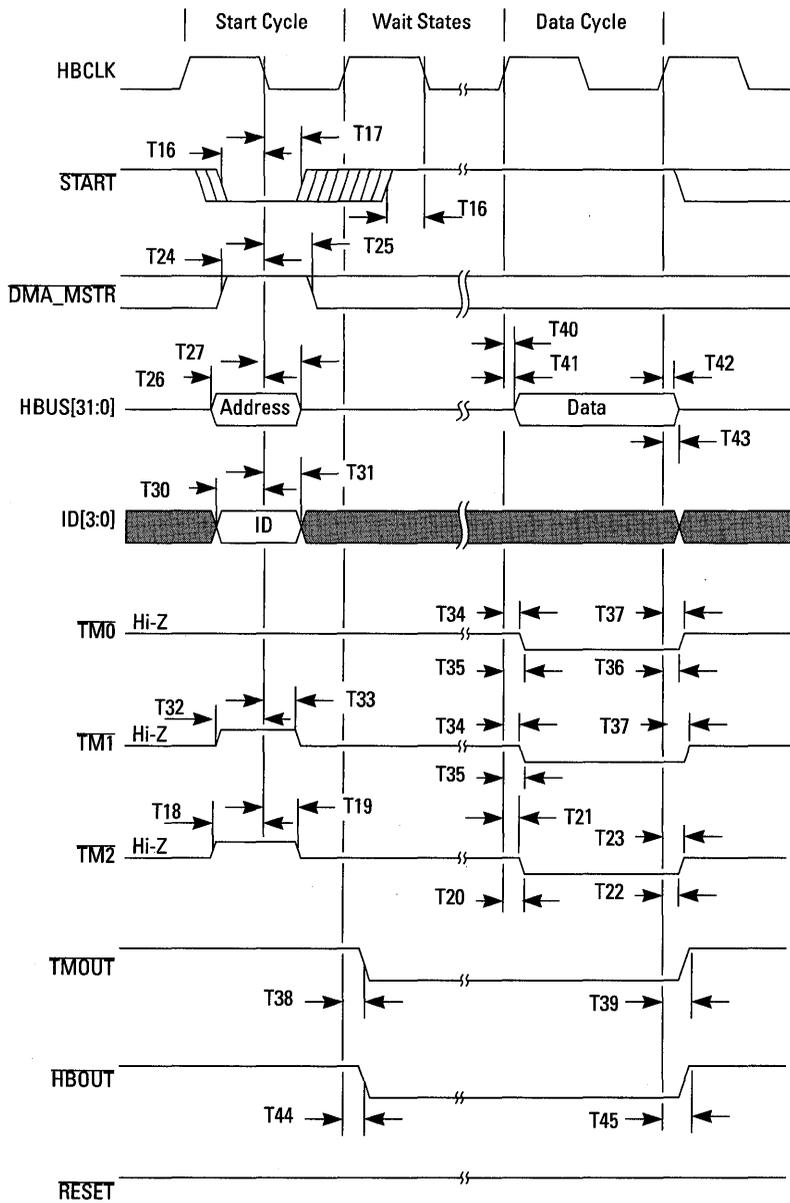


Figure 6-9 CL560 Host Interface Timing: Register and Memory Read

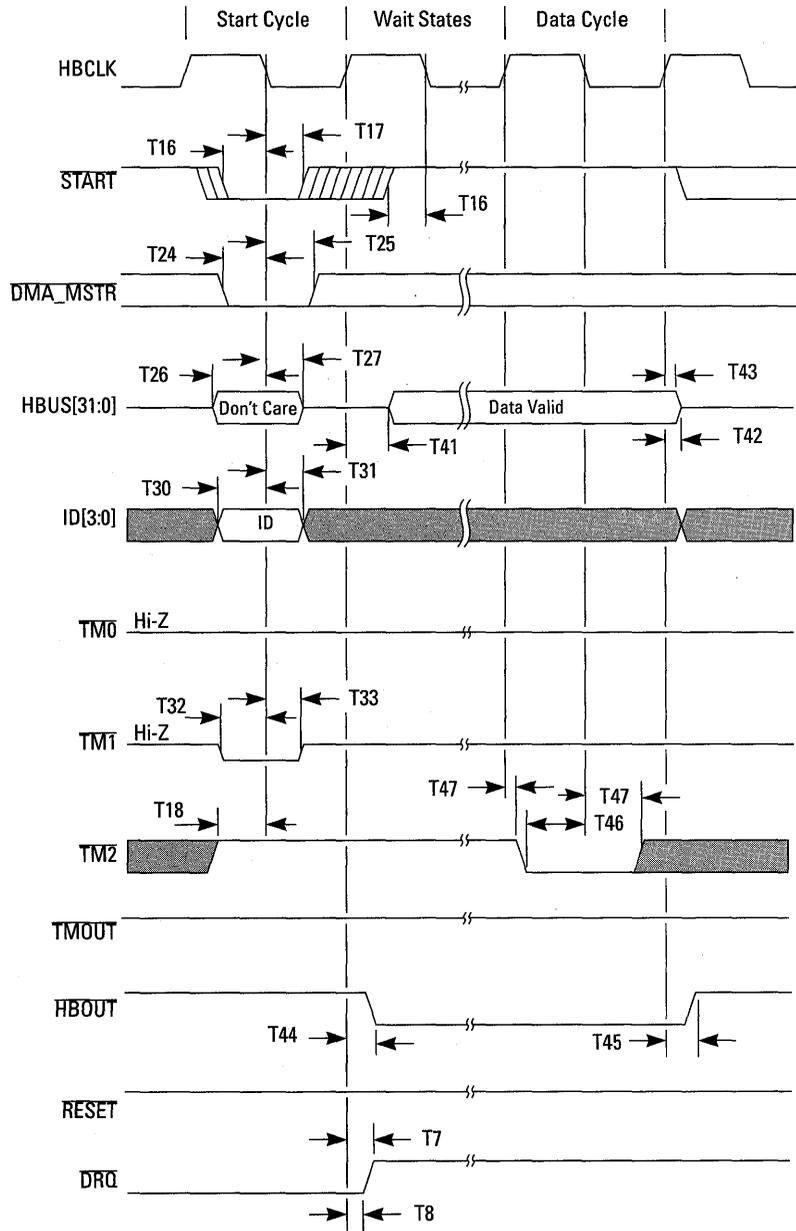


Figure 6-10 CL560 Host Interface Timing: Burst Mode Read

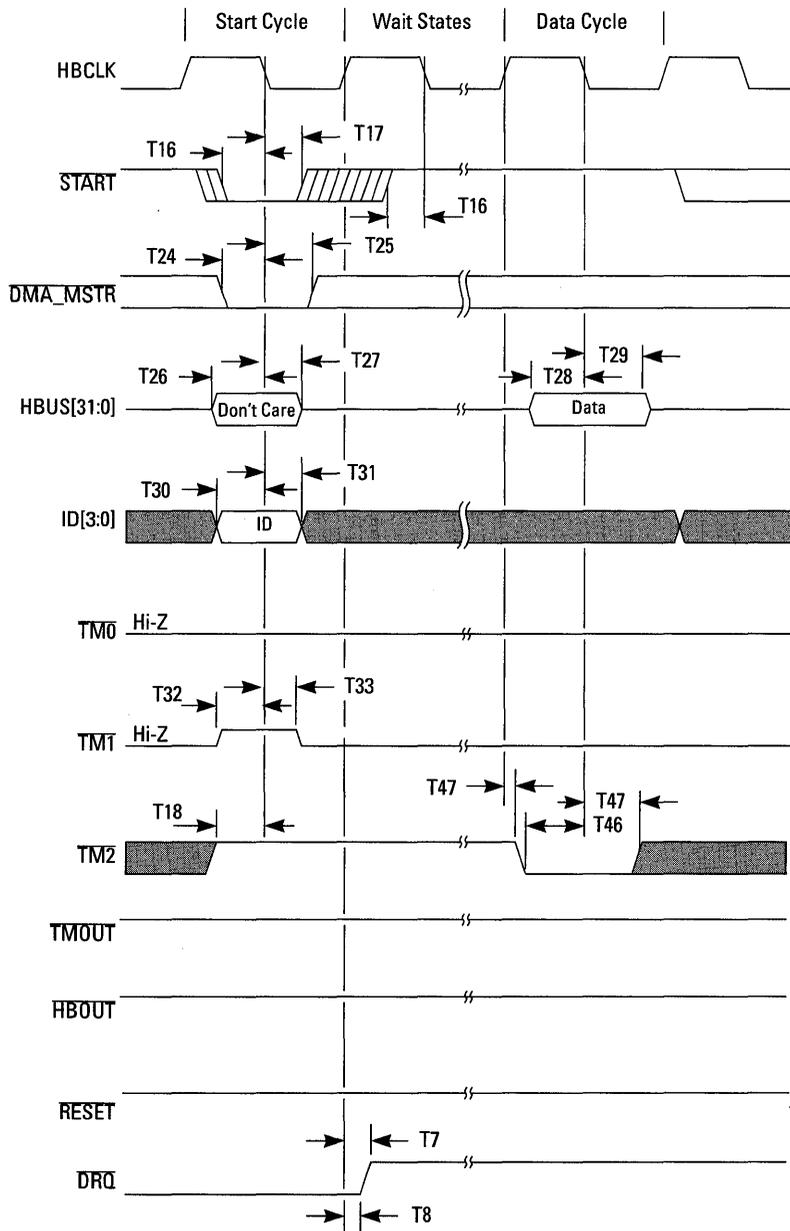


Figure 6-11 CL560 Host Interface Timing: Burst Mode Write

## AC Characteristics

**Table 6-12 Host Interface Timing, CPGA Package**

Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T16	START Setup Time <sup>1,2</sup>	8		10		8		ns
T17	START Hold Time	12		15		12		ns
T18	TM2 Start Cycle Setup Time <sup>3</sup>	8		10		8		ns
T19	TM2 Start Cycle Hold Time	12		15		12		ns
T20	TM2 Delay Hi-Z to HIGH		18		18		18	ns
T21	TM2 Delay to LOW		18		18		18	ns
T22	TM2 Delay Hold Time <sup>4</sup>	5		5		5		ns
T23	TM2 Delay to Hi-Z <sup>4</sup>		18		18		18	ns
T24	DMA_MSTR Setup Time	8		10		8		ns
T25	DMA_MSTR Hold Time	12		15		12		ns
T26	HBUS[31:0] Address Setup Time	8		10		8		ns
T27	HBUS[31:0] Address Hold Time	12		15		12		ns
T28	HBUS[31:0] Data Setup Time	8		10		8		ns
T29	HBUS[31:0] Data Setup Time	12		15		12		ns
T30	ID Setup Time	8		10		8		ns
T31	ID Hold Time	12		15		12		ns
T32	TM0, TM1 Start Cycle Setup Time	8		10		8		ns
T33	TM0, TM1 Start Cycle Hold Time	12		15		12		ns
T34	TM0, TM1 Hi-Z Hold Time <sup>4</sup>	5		5		5		ns
T35	TM0, TM1 Hi-Z to LOW Delay		18		18		18	ns
T36	TM0, TM1 Delay Hold Time <sup>4</sup>	5		5		5		ns
T37	TM0, TM1 Delay to Hi-Z <sup>4</sup>		18		18		18	ns
T38	TMOUT Delay to LOW <sup>5</sup>		22		22		22	ns
T39	TMOUT Delay to HIGH		22		22		22	ns
T40	HBUS[31:0] Hi-Z Hold Time <sup>4</sup>	2		2		2		ns
T41	HBUS[31:0] Delay Time		22		23		22	ns
T42	HBUS[31:0] Delay to Hi-Z <sup>4</sup>		22		23		22	ns
T43	HBUS[31:0] Hi-Z Delay Hold Time <sup>4</sup>	5		5		5		ns
T44	HBOUT Delay to LOW		22		22		22	ns
T45	HBOUT Delay to HIGH <sup>6</sup>		22		22		22	ns
T46	TM2 Acknowledge Cycle Setup	8		10		8		ns
T47	TM2 Acknowledge Cycle Hold	5		5		5		ns
T48	TM0, TM1 Status Cycle Setup Time	8		10		–	–	ns
T49	TM0, TM1 Status Cycle Hold Time	5		15		–	–	ns

1. START should remain HIGH until the transaction is complete.
2. A valid start cycle implies START LOW, TM2 HIGH, and that the CS fields of HBUS match the ID inputs.
3. TM2 is sometimes called ACK in design documentation. Reference part number CL550-10x for NuBus specifications.
4. Not 100% Tested, guaranteed by design.
5. TMOUT is the direction control for TM2, TM1, and TM0. When HIGH, these pins are expected to be inputs.
6. HBOUT is the direction control for the Host bus, When HIGH, the direction is from the host to the CL5XX (input).

**Table 6-13 Host Interface Timing, MQUAD Package**

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T16	START Setup Time <sup>1,2</sup>	10		10		10		8		ns
T17	START Hold Time	15		15		15		12		ns
T18	TM2 Start Cycle Setup Time <sup>3</sup>	10		10		10		8		ns
T19	TM2 Start Cycle Hold Time	15		15		15		12		ns
T20	TM2 Delay Hi-Z to HIGH		20		18		20		18	ns
T21	TM2 Delay to LOW		20		18		20		18	ns
T22	TM2 Delay Hold Time <sup>4</sup>	4		5		4		5		ns
T23	TM2 Delay to Hi-Z <sup>4</sup>		20		18		20		18	ns
T24	DMA_MSTR Setup Time	10		10		10		8		ns
T25	DMA_MSTR Hold Time	15		15		15		12		ns
T26	HBUS[31:0] Address Setup Time	10		10		10		8		ns
T27	HBUS[31:0] Address Hold Time	15		15		15		12		ns
T28	HBUS[31:0] Data Setup Time	10		10		10		8		ns
T29	HBUS[31:0] Data Setup Time	15		15		15		12		ns
T30	ID Setup Time	10		10		10		8		ns
T31	ID Hold Time	15		15		15		12		ns
T32	TM0, TMT Start Cycle Setup Time	10		10		10		8		ns
T33	TM0, TMT Start Cycle Hold Time	15		15		15		12		ns
T34	TM0, TMT Hi-Z Hold Time <sup>4</sup>	5		5		5		5		ns
T35	TM0, TMT Hi-Z to LOW Delay		20		18		20		18	ns
T36	TM0, TMT Delay Hold Time <sup>4</sup>	5		5		5		5		ns
T37	TM0, TMT Delay to Hi-Z <sup>4</sup>		20		18		20		18	ns
T38	TMOUT Delay to LOW <sup>5</sup>		26		22		22		20	ns
T39	TMOUT Delay to HIGH		26		22		22		20	ns
T40	HBUS[31:0] Hi-Z Hold Time <sup>4</sup>	2		2		2		2		ns
T41	HBUS[31:0] Delay Time		26		23		22		20	ns
T42	HBUS[31:0] Delay to Hi-Z <sup>4</sup>		26		23		22		20	ns
T43	HBUS[31:0] Hi-Z Delay Hold Time <sup>4</sup>	3		5		3		5		ns
T44	HBOUT Delay to LOW		26		22		22		20	ns
T45	HBOUT Delay to HIGH <sup>6</sup>		26		22		22		20	ns
T46	TM2 Acknowledge Cycle Setup	10		10		10		10		ns
T47	TM2 Acknowledge Cycle Hold	5		5		5		5		ns
T48	TM0, TMT Status Cycle Setup Time	10		10		—		—		ns
T49	TM0, TMT Status Cycle Hold Time	15		15		—		—		ns

1. START should remain HIGH until the transaction is complete.

2. A valid start cycle implies START LOW, TM2 HIGH, and that the CS fields of HBUS match the ID inputs.

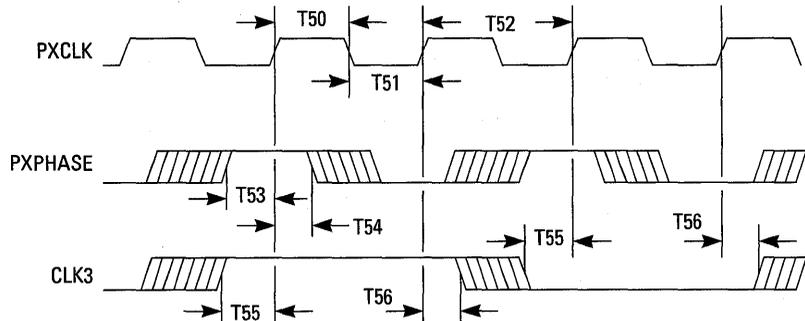
3. TM2 is sometimes called ACK in design documentation. Reference part number CL550-10x for NuBus specifications.

4. Not 100% Tested, guaranteed by design.

5. TMOUT is the direction control for TM2, TM1, and TM0. When HIGH, these pins are expected to be inputs.

6. HBOUT is the direction control for the Host bus, When HIGH, the direction is from the host to the CL5XX (input).

**6.2.2 Video Interface Signal Timing**



**Figure 6-12 Video Interface Clock Timing**

**Table 6-14 Video Interface Clock Timing, CPGA Package**

Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T50	PXCLK Pulse Width HIGH	13		15		15		ns
T51	PXCLK Pulse Width LOW <sup>1</sup>	13		15		15		ns
T52	PXCLK Clock Period	29		34		33		ns
T53	PXPHASE Setup Time	16		18		16		ns
T54	PXPHASE Hold Time	5		5		5		ns
T55	CLK3 Setup Time	16		18		16		ns
T56	CLK3 Hold Time	5		5		5		ns

**Table 6-15 Video Interface Clock Timing, MQUAD Package**

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T50	PXCLK Pulse Width HIGH	30		15		15		15		ns
T51	PXCLK Pulse Width LOW <sup>1</sup>	30(23)		15		15		15		ns
T52	PXCLK Clock Period	100		34		33		33		ns
T53	PXPHASE Setup Time	20		18		18		16		ns
T54	PXPHASE Hold Time	7		5		5		5		ns
T55	CLK3 Setup Time	20		18		18		16		ns
T56	CLK3 Hold Time	7		5		5		5		ns

1. Characteristics in parenthesis apply to part number CL550-10N, required for NuBus applications.

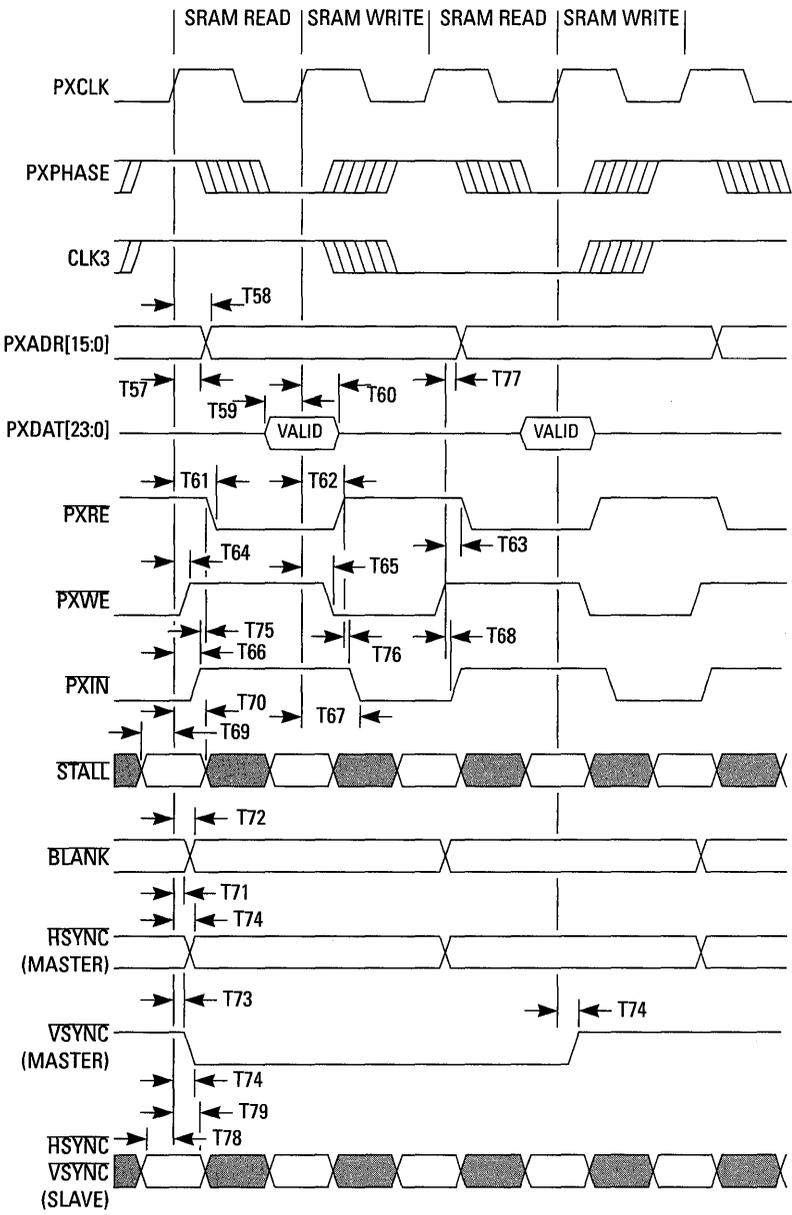


Figure 6-13 Video Interface Timing: Compression, Full Rate Mode

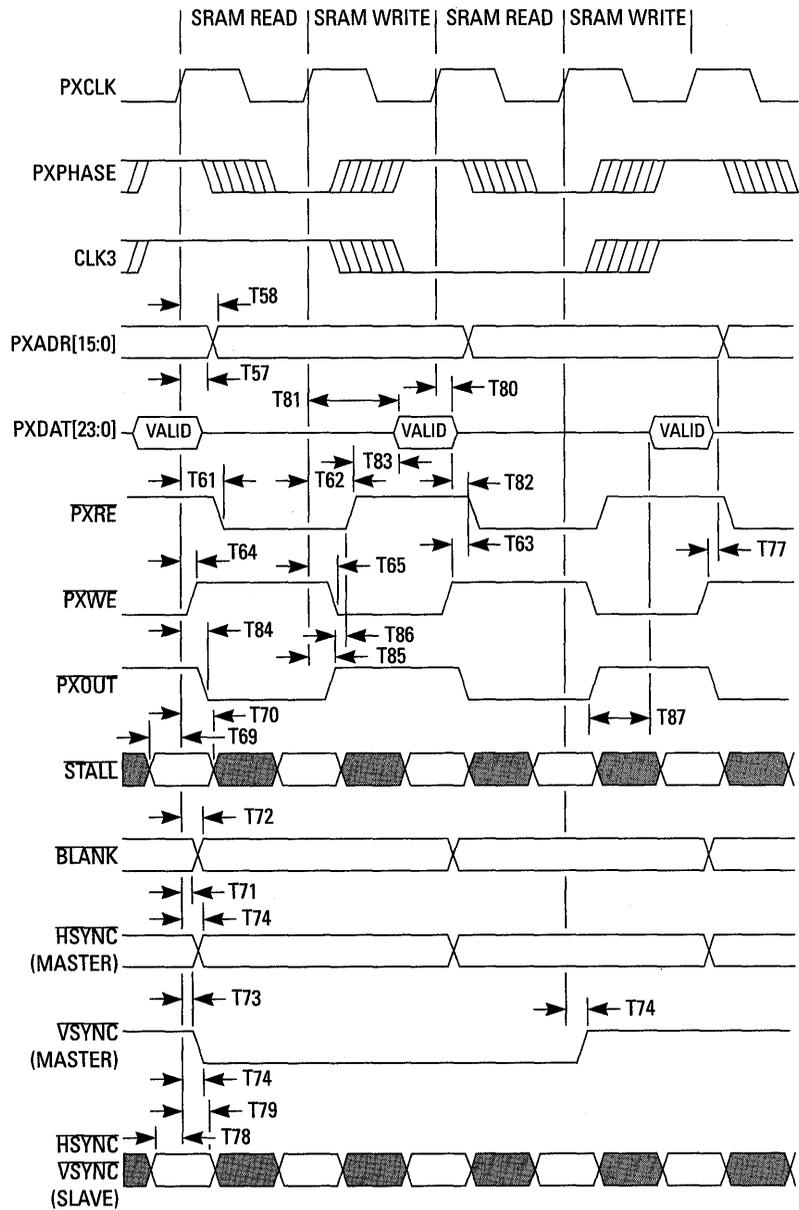


Figure 6-14 Video Interface Timing: Decompression, Full Rate Mode

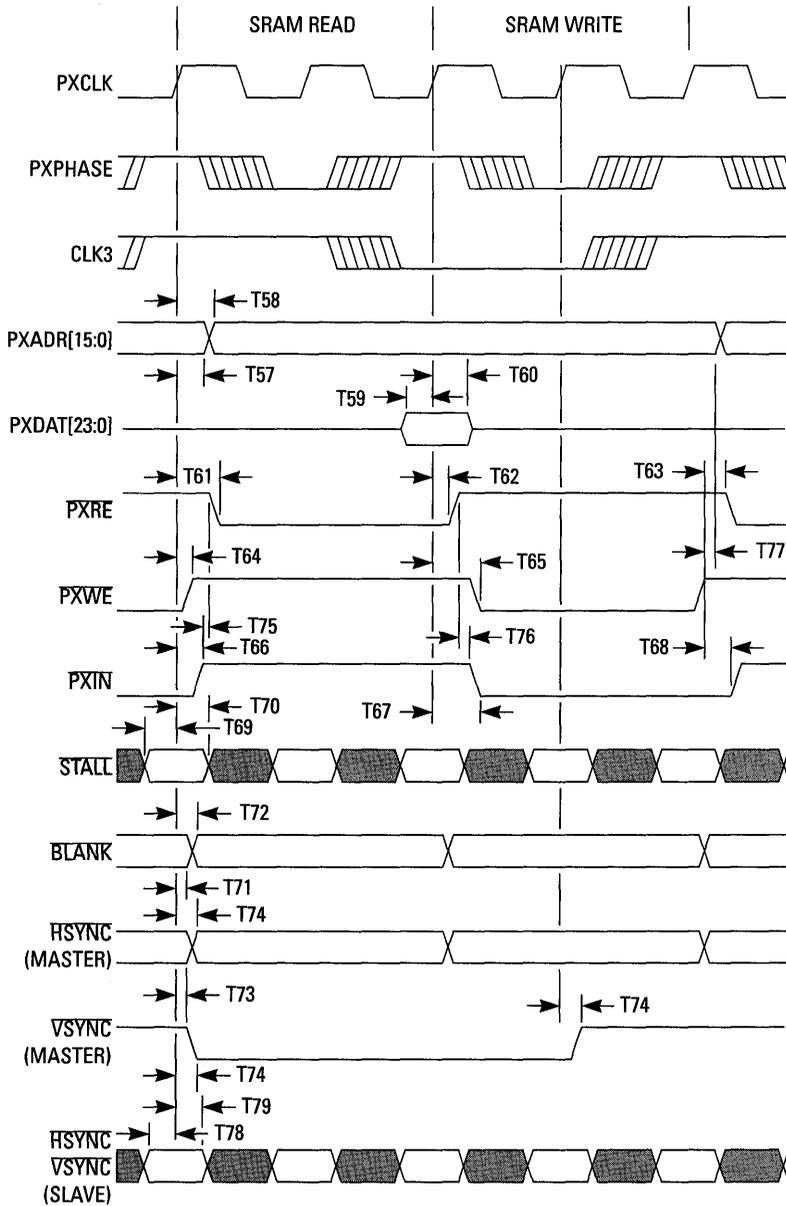


Figure 6-15 Video Interface Timing: Compression, Half-Rate Mode

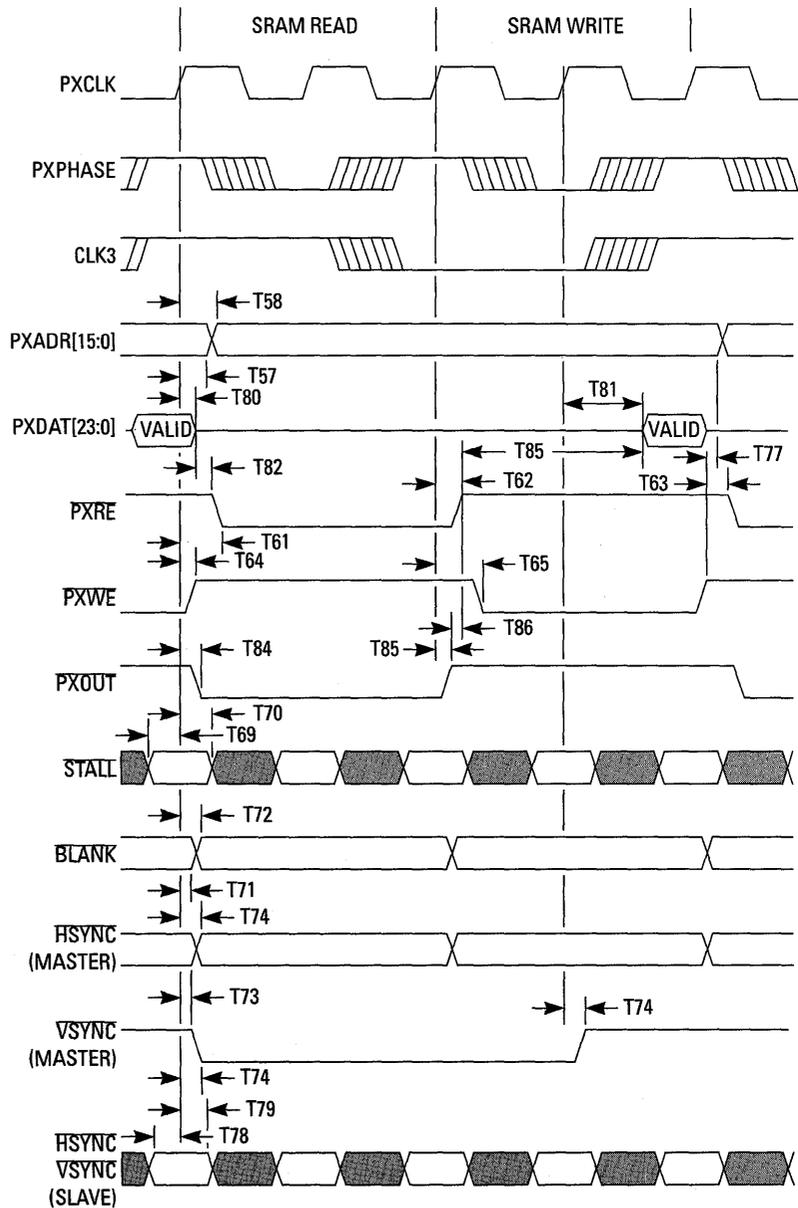


Figure 6-16 Video Interface Timing: Decompression, Half-Rate Mode

**Table 6-16 Video Interface Timing Table, CPGA Package**

Time	Description	CL550-35		CL550-30		CL560-30		Units
		Min	Max	Min	Max	Min	Max	
T57	PXADR [15:0] Hold Delay <sup>1</sup>	4		4		4		ns
T58	PXADR [15:0] Delay Time		14		15		14	ns
T59	PXDAT [23:0] Setup Time <sup>2,3</sup>	3		4		3		ns
T60	PXDAT[23:0] Hold Time	4		5		4		ns
T61	PXRE Delay to LOW		12		13		12	ns
T62	PXRE Delay to HIGH		12		13		12	ns
T63	PXWE, PXRE Overlap <sup>1</sup>	-3		-3		-3		ns
T64	PXWE Delay to HIGH		12		13		12	ns
T65	PXWE Delay to LOW		12		13		12	ns
T66	PXIN Delay to HIGH		12		13		12	ns
T67	PXIN Delay to LOW		12		13		12	ns
T68	PXWE HIGH to Overlap <sup>1</sup>	-2	2	-2	2	-2	2	ns
T69	STALL Setup Time	14		15		14		ns
T70	STALL Hold Time	7		8		7		ns
T71	BLANK Delay Hold Time <sup>1</sup>	4		4		4		ns
T72	BLANK Delay Time		13		15		13	ns
T73	HSYNC, VSYNC Delay Hold Time <sup>1</sup>	4		4		4		ns
T74	HSYNC, VSYNC Delay Time		24		26		24	ns
T75	PXIN, PXRE Overlap <sup>1</sup>	-2	2	-2	2	-2	2	ns
T76	PXRE, PXIN Overlap <sup>1</sup>	-2	2	-2	2	-2	2	ns
T77	PXWE HIGH, PXADR Overlap <sup>1</sup>	0		0		0		ns
T78	HSYNC, VSYNC Setup	7		8		7		ns
T79	HSYNC, VSYNC Hold	7		8		7		ns
T80	PXDAT Delay to Hi-Z <sup>1,4</sup>		23		24		23	ns
T81	PXDAT [23:0] Delay Time		23		24		23	ns
T82	PXDAT, PXRE Overlap <sup>1</sup>	-3		-3		-3		ns
T83	PXRE, PXDAT Overlap <sup>1</sup>	3	12	3	12	3	12	ns
T84	PXOUT to LOW Delay		12		13		12	ns
T85	PXOUT to HIGH Delay	4	12	4	13	4	15	ns
T86	PXOUT, PXRE Overlap <sup>1</sup>	-2		-2		-2		ns
T87	PXOUT, PXDAT Overlap <sup>1</sup>	3	12	3	12	3	12	ns

1. Not 100% tested, guaranteed by design.
2. PXDAT [23:0] are inputs only during compression.
3. SRAM access time  $\leq T_{52} - T_{58} - T_{59}$
4. Decompression parameter

## AC Characteristics

**Table 6-17 Video Bus Timing Table, MQUAD Package**

Time	Description	CL550-10		CL550-30		CL560-15		CL560-30		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
T57	PXADR [15:0] Hold Delay <sup>1</sup>	3		4		4		4		ns
T58	PXADR [15:0] Delay Time		22		15		18		14	ns
T59	PXDAT [23:0] Setup Time <sup>2,3</sup>	6		4		5		4		ns
T60	PXDAT[23:0] Hold Time	7		5		6		5		ns
T61	PXRE Delay to LOW		18		13		15		12	ns
T62	PXRE Delay to HIGH		18		13		15		12	ns
T63	PXWE, PXRE Overlap <sup>1</sup>	-4		-3		-4		-3		ns
T64	PXWE Delay to HIGH		18		13		15		12	ns
T65	PXWE Delay to LOW		18		13		15		12	ns
T66	PXIN Delay to HIGH		18		13		15		12	ns
T67	PXIN Delay to LOW		18		13		15		12	ns
T68	PXWE HIGH to Overlap <sup>1</sup>	-4	4	-2	2	-3	3	-2	2	ns
T69	STALL Setup Time	20		15		18		14		ns
T70	STALL Hold Time	12		8		10		7		ns
T71	BLANK Delay Hold Time <sup>1</sup>	3		4		4		4		ns
T72	BLANK Delay Time		18		15		17		14	ns
T73	HSYNC, VSYNC Delay Hold Time <sup>1</sup>	3		4		4		4		ns
T74	HSYNC, VSYNC Delay Time		30		26		28		25	ns
T75	PXIN, PXRE Overlap <sup>1</sup>	-4	4	-2	2	-3	3	-2	2	ns
T76	PXRE, PXIN Overlap <sup>1</sup>	-4	4	-2	2	-3	3	-2	2	ns
T77	PXWE HIGH, PXADR Overlap <sup>1</sup>	0		0		0		0		ns
T78	HSYNC, VSYNC Setup	12		8		10		7		ns
T79	HSYNC, VSYNC Hold	12		8		10		7		ns
T80	PXDAT Delay to Hi-Z <sup>1,4</sup>		28		24		26		23	ns
T81	PXDAT [23:0] Delay Time		28		24		26		23	ns
T82	PXDAT, PXRE Overlap <sup>1</sup>	-4		-3		-4		-3		ns
T83	PXRE, PXDAT Overlap <sup>1</sup>	2	13	3	12	2	13	3	12	ns
T84	PXOUT to LOW Delay		18		13		15		12	ns
T85	PXOUT to HIGH Delay	3	18	4	13	4	15	4	12	ns
T86	PXOUT, PXRE Overlap <sup>1</sup>	-4		-2		-3		-2		ns
T87	PXOUT, PXDAT Overlap <sup>1</sup>	2	13	3	12	2	13	3	12	ns

1. Not 100% tested, guaranteed by design.
2. PXDAT [23:0] are inputs only during compression.
3. SRAM access time  $\leq T_{52} - T_{58} - T_{59}$
4. Decompression parameter

The CL550 and CL560 are packaged in two packages:

- 144 Pin Ceramic Pin Grid Array (CPGA)
- 144 Pin Metal Quad Flat Pack (MQUAD)

This chapter is divided into three sections, one for each of the three package types. Contained in each section is:

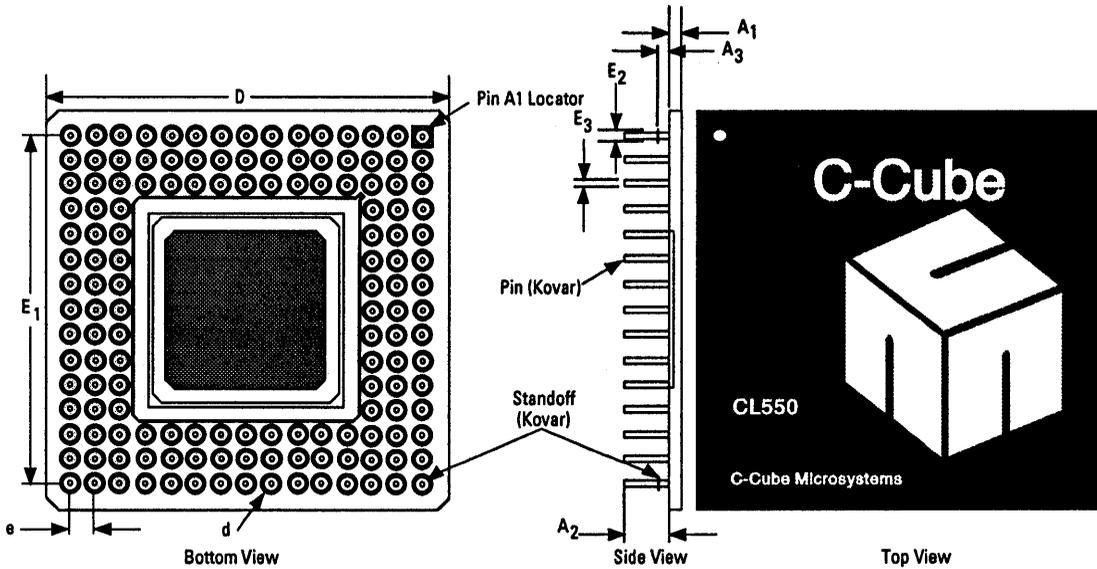
- The package physical dimensions
- The package pinout diagram
- Tables of the pin connections for that package sorted by:
  - Pin Number
  - Signal Name

---

### **6.3 Package Specifications**

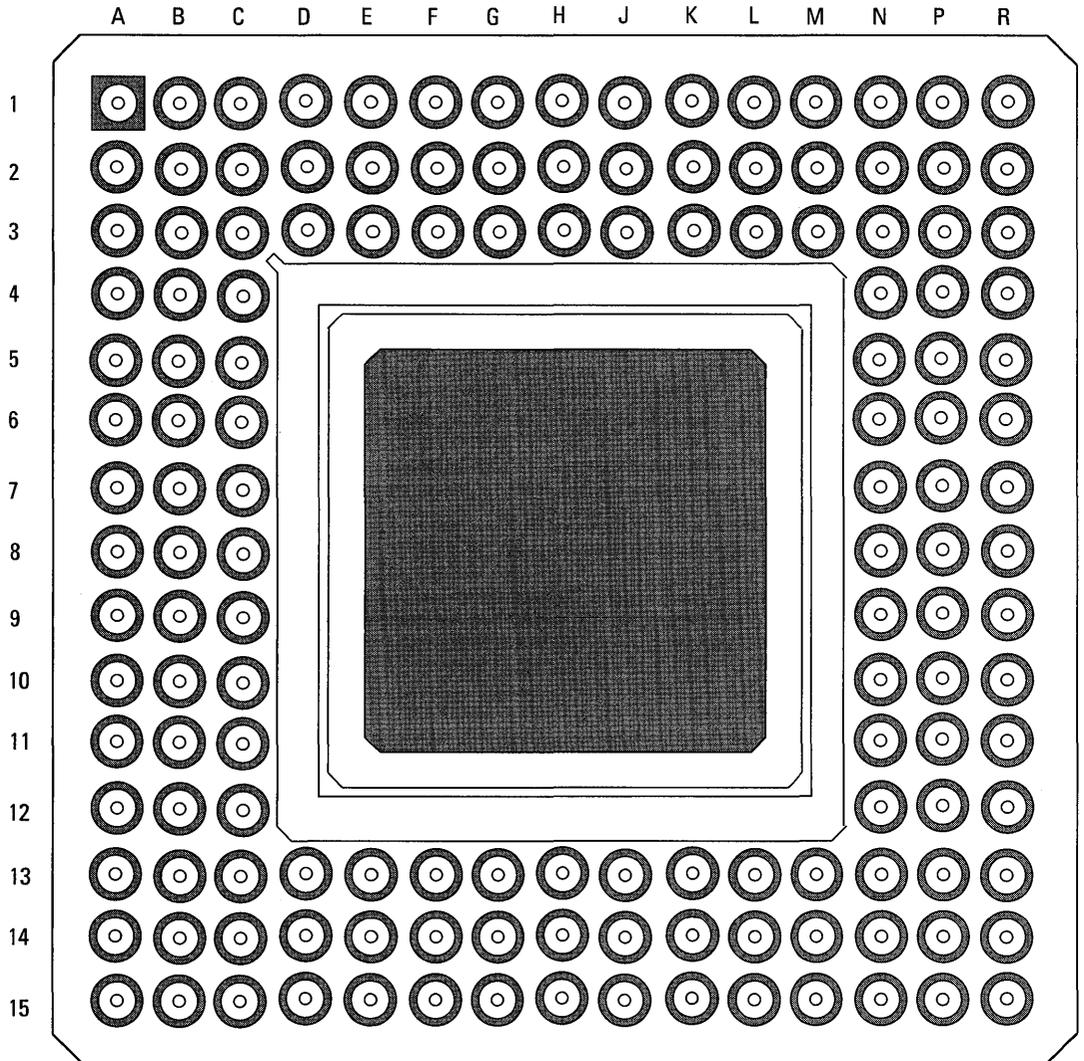
---

6.3.1 144-Pin Ceramic Pin-Grid Array



Dimensions		
Symbol	Inches	MM
A1	.0100 ±0.10	2.54 ±0.25
A2	0.180 typ. ±0.010	4.57 typ. ±0.25
A3	0.050 typ. ±0.010	1.27 typ. ±0.25
D	1.575 sq. ±0.030	40.0 sq. ±0.80
E1	1.400 typ. ±0.014	35.56 typ. ±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.

Figure 6-17 CL550 and CL560 CPGA Physical Dimensions



**Figure 6-18 CL550 and CL560 CPGA Pin Layout (Bottom View)**

# Package Specifications

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	VSS	PXADR5	PXADR2	PXDAT23	PXDAT21	PXDAT19	PXDAT16	VDD	PXDAT14	PXDAT11	PXDAT9	PXDAT7	PXDAT4	PXDAT1	TEST
B	PXADR7	PXADR6	PXADR3	PXADR0	PXDAT22	PXDAT20	PXDAT17	PXDAT15	PXDAT13	PXDAT10	PXDAT8	PXDAT5	PXDAT2	PXDAT0	$\overline{\text{TMOUT}}$
C	PXADR9	PXADR8	VSS	PXADR4	PXADR1	VSS	PXDAT18	VDD	PXDAT12	VSS	PXDAT6	PXDAT3	VSS	$\overline{\text{RESET}}$	$\overline{\text{TM1}}$
D	PXADR12	PXADR11	PXADR10	<b>Top View</b>									$\overline{\text{NMRQ}}$	$\overline{\text{TM2}}$	$\overline{\text{START}}$
E	PXCLK	$\overline{\text{PXOUT}}$	$\overline{\text{PXIN}}$										$\overline{\text{TM0}}$	HBCLK	$\overline{\text{HBOUT}}$
F	$\overline{\text{PXRE}}$	$\overline{\text{PXWE}}$	VDD										VDD	HBUS0	HBUS1
G	$\overline{\text{VSYNC}}$	$\overline{\text{HSYNC}}$	PXPHSE										HBUS2	HBUS3	HBUS4
H	VDD	PXADR13	VSS										VSS	HBUS5	HBUS6
J	PXADR14	PXADR15	CLK3										HBUS9	HBUS8	HBUS7
K	$\overline{\text{STALL}}$	$\overline{\text{BLANK}}$	VDD										VDD	HBUS11	HBUS10
L	$\overline{\text{FRMEND}}$	HALF-FULL	NC										HBUS14	HBUS13	HBUS12
M	NC	NC	NC										HBUS17	HBUS16	HBUS15
N	NC	NC	VSS										NC	NC	VSS
P	NC	NC	NC	NC	NC	NC	$\overline{\text{HBUS\_32}}$	$\overline{\text{DRQ}}$	ID2	HBUS31	HBUS29	HBUS26	HBUS24	HBUS21	HBUS20
R	VSS	NC	NC	NC	NC	NC	$\overline{\text{DMA\_MSTR}}$	VDD	ID3	ID[0]	HBUS30	HBUS28	HBUS25	HBUS22	VSS

**Figure 6-19 CL550 and CL560 CPGA Pinout Diagram (Top View Through Chip)**

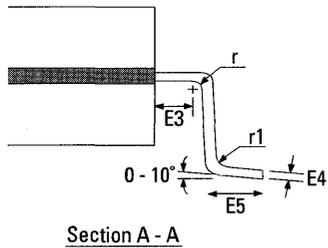
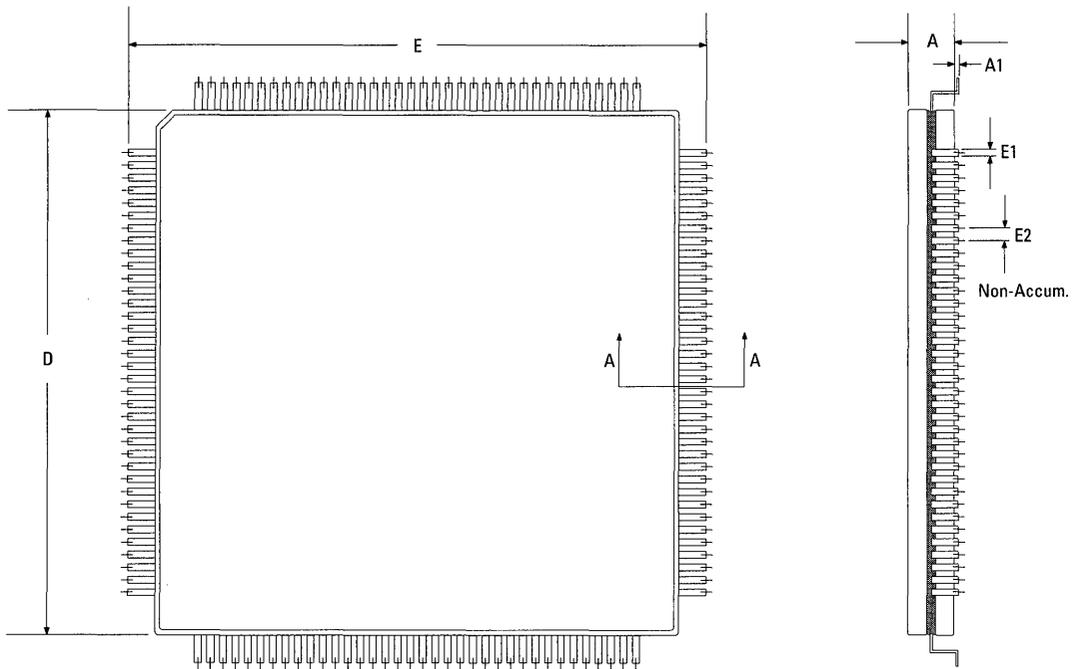
**Table 6-18 CPGA Pin List Sorted by Pin Number**

Pin #	Signal Name						
A1	VSS	C7	PXDAT18	H13	VSS	N10	VSS
A2	PXADR5	C8	VDD	H14	HBUS5	N11	HBUS27
A3	PXADR2	C9	PXDAT12	H15	HBUS6	N12	HBUS23
A4	PXDAT23	C10	VSS	J1	PXADR14	N13	VSS
A5	PXDAT21	C11	PXDAT6	J2	PXADR15	N14	HBUS19
A6	PXDAT19	C12	PXDAT3	J3	CLK3	N15	HBUS18
A7	PXDAT16	C13	VSS	J13	HBUS9	P1	NC
A8	VDD	C14	RESET	J14	HBUS8	P2	NC
A9	PXDAT14	C15	TMT	J15	HBUS7	P3	NC
A10	PXDAT11	D1	PXADR12	K1	STALL	P4	NC
A11	PXDAT9	D2	PXADR11	K2	BLANK	P5	NC
A12	PXDAT7	D3	PXADR10	K3	VDD	P6	NC
A13	PXDAT4	D13	NMRQ	K13	VDD	P7	HBUS_32
A14	PXDAT1	D14	TMZ	K14	HBUS11	P8	DRQ
A15	TEST	D15	START	K15	HBUS10	P9	ID2
B1	PXADR7	E1	PXCLK	L1	FRMEND	P10	HBUS31
B2	PXADR6	E2	PXOUT	L2	HALF_FULL	P11	HBUS29
B3	PXADR3	E3	PXIN	L3	NC	P12	HBUS26
B4	PXADR0	E13	TMO	L13	HBUS14	P13	HBUS24
B5	PXDAT22	E14	HBCLK	L14	HBUS13	P14	HBUS21
B6	PXDAT20	E15	HBOUT	L15	HBUS12	P15	HBUS20
B7	PXDAT17	F1	PXRE	M1	NC	R1	VSS
B8	PXDAT15	F2	PXWE	M2	NC	R2	NC
B9	PXDAT13	F3	VDD	M3	NC	R3	NC
B10	PXDAT10	F13	VDD	M13	HBUS17	R4	NC
B11	PXDAT8	F14	HBUS0	M14	HBUS16	R5	NC
B12	PXDAT5	F15	HBUS1	M15	HBUS15	R6	NC
B13	PXDAT2	G1	VSYNC	N1	NC	R7	DMA_MSTR
B14	PXDAT0	G2	HSYNC	N2	NC	R8	VDD
B15	TMOUT	G3	PXPHASE	N3	VSS	R9	ID3
C1	PXADR9	G13	HBUS2	N4	NC	R10	ID0
C2	PXADR8	G14	HBUS3	N5	NC	R11	HBUS30
C3	VSS	G15	HBUS4	N6	VSS	R12	HBUS28
C4	PXADR4	H1	VDD	N7	NC	R13	HBUS25
C5	PXADR1	H2	PXADR13	N8	VDD	R14	HBUS22
C6	VSS	H3	VSS	N9	ID1	R15	VSS

**Table 6-19 CPGA Pin List Sorted by Pin Name**

Signal Name	Pin #						
BLANK	K2	HBUS6	H15	PXADR10	D3	PXDAT7	A12
CLK3	J3	HBUS7	J15	PXADR11	D2	PXDAT8	B11
DMA_MSTR	R7	HBUS8	J14	PXADR12	D1	PXDAT9	A11
DRQ	P8	HBUS9	J13	PXADR13	H2	PXIN	E3
FRMEND	L1	HBUS_32	P7	PXADR14	J1	PXOUT	E2
HALF-FULL	L2	HSYNC	G2	PXADR15	J2	PXPHASE	G3
HBCLK	E14	ID0	R10	PXADR2	A3	PXRE	F1
HBOUT	E15	ID1	N9	PXADR3	B3	PXWE	F2
HBUS0	F14	ID2	P9	PXADR4	C4	RESET	C14
HBUS1	F15	ID3	R9	PXADR5	A2	STALL	K1
HBUS10	K15	NC	L3	PXADR6	B2	START	D15
HBUS11	K14	NC	M1	PXADR7	B1	TEST	A15
HBUS12	L15	NC	M2	PXADR8	C2	TMO	E13
HBUS13	L14	NC	M3	PXADR9	C1	TMT	C15
HBUS14	L13	NC	N1	PXCLK	E1	TMZ	D14
HBUS15	M15	NC	N2	PXDAT0	B14	TMOUT	B15
HBUS16	M14	NC	N4	PXDAT1	A14	VDD	A8
HBUS17	M13	NC	N5	PXDAT10	B10	VDD	C8
HBUS18	N15	NC	N7	PXDAT11	A10	VDD	F3
HBUS19	N14	NC	P1	PXDAT12	C9	VDD	F13
HBUS2	G13	NC	P2	PXDAT13	B9	VDD	H1
HBUS20	P15	NC	P3	PXDAT14	A9	VDD	K3
HBUS21	P14	NC	P4	PXDAT15	B8	VDD	K13
HBUS22	R14	NC	P5	PXDAT16	A7	VDD	N8
HBUS23	N12	NC	P6	PXDAT17	B7	VDD	R8
HBUS24	P13	NC	R2	PXDAT18	C7	VSS	A1
HBUS25	R13	NC	R3	PXDAT19	A6	VSS	C3
HBUS26	P12	NC	R4	PXDAT2	B13	VSS	C6
HBUS27	N11	NC	R5	PXDAT20	B6	VSS	C10
HBUS28	R12	NC	R6	PXDAT21	A5	VSS	C13
HBUS29	P11	PXDAT23	A4	PXDAT22	B5	VSS	H3
HBUS3	G14	PXDAT3	C12	VSS	N3	VSS	H13
HBUS30	R11	PXDAT4	A13	VSS	N6	VSS	N13
HBUS31	P10	NMRQ	D13	VSS	N10	VSS	R1
HBUS4	G15	PXADR0	B4	PXDAT5	B12	VSS	R15
HBUS5	H14	PXADR1	C5	PXDAT6	C11	VSYNC	G1

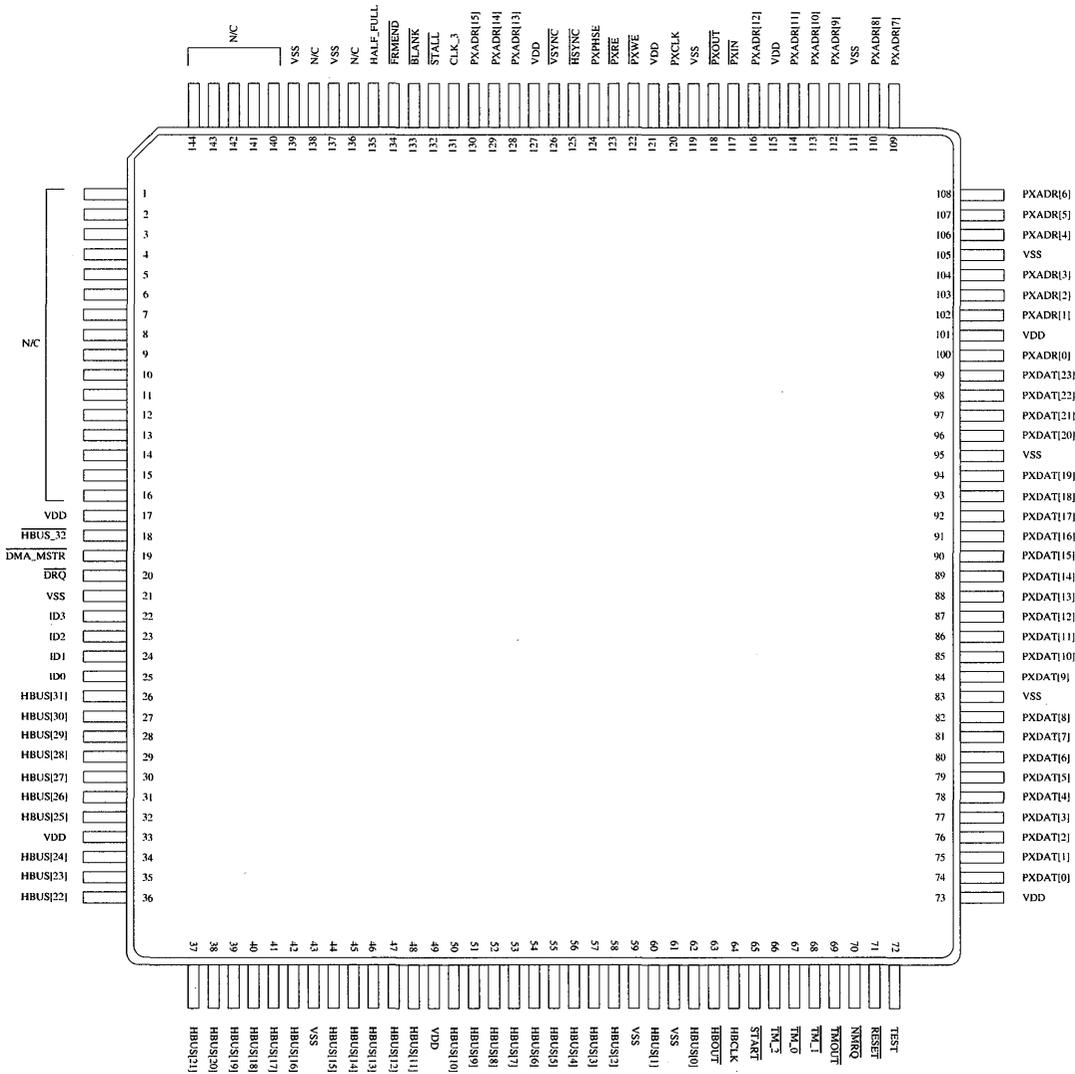
6.3.2 144-Pin MQUAD Package



Symbol	Dimensions	
	Inches	MM
A	0.130 ±0.012	3.30 ±0.30
A1	0.015 ±0.008	0.38 ±0.20
D	1.088 ±0.016	27.60 ±0.40
E	1.256 ±0.016	31.90 ±0.40
E1	0.012 typ.	0.030 typ.
E2	0.0256 typ.	0.65 typ.
E3	0.020 min.	0.50 min.
E4	0.006 typ.	0.15 typ.
E5	0.031 ±0.008	0.80 ±0.2
r	0.010 typ. R	0.25 typ. R
r1	0.010 typ. R	0.25 typ. R

Figure 6-20 MQUAD Physical Dimensions

# Package Specifications



**Figure 6-21 CL550 MQUAD Pinout Diagram**

**Table 6-20 CL550/CL560 MQUAD Pinout Differences**

Pin #	CL550 Name	CL560 Name	Pin #	CL550 Name	CL560 Name
2	No Connect	VSS	12	No Connect	VSS
4	No Connect	VDD	70	NMRQ	TRQ1
9	No Connect	VDD	135	HALF_FULL	TRQ2

**Table 6-21 CL550(CL560) MQUAD Pin List Sorted by Pin Number**

Pin #	Signal Name						
1	NC	37	HBUS21	73	VDD	109	PXADR7
2	NC (VSS)	38	HBUS20	74	PXDAT0	110	PXADR8
3	NC	39	HBUS19	75	PXDAT1	111	VSS
4	NC (VDD)	40	HBUS18	76	PXDAT2	112	PXADR9
5	NC	41	HBUS17	77	PXDAT3	113	PXADR10
6	NC	42	HBUS16	78	PXDAT4	114	PXADR11
7	NC	43	VSS	79	PXDAT5	115	VDD
8	NC	44	HBUS15	80	PXDAT6	116	PXADR12
9	NC (VDD)	45	HBUS14	81	PXDAT7	117	PXIN
10	PXADR5	46	HBUS13	82	PXDAT8	118	PXOUT
11	NC	47	HBUS12	83	VSS	119	VSS
12	NC (VSS)	48	HBUS11	84	PXDAT9	120	PXCLK
13	NC	49	VDD	85	PXDAT10	121	VDD
14	NC	50	HBUS10	86	PXDAT11	122	PXWE
15	NC	51	HBUS9	87	PXDAT12	123	PXRE
16	NC	52	HBUS8	88	PXDAT13	124	PXPHASE
17	VDD	53	HBUS7	89	PXDAT14	125	HSYNC
18	HBUS_32	54	HBUS6	90	PXDAT15	126	VSYNC
19	DMA_MSTR	55	HBUS5	91	PXDAT16	127	VDD
20	DRQ	56	HBUS4	92	PXDAT17	128	PXADR13
21	VSS	57	HBUS3	93	PXDAT18	129	PXADR14
22	ID3	58	HBUS2	94	PXDAT19	130	PXADR15
23	ID2	59	VSS	95	VSS	131	CLK3
24	ID1	60	HBUS1	96	PXDAT20	132	STALL
25	ID0	61	VSS	97	PXDAT21	133	BLANK
26	HBUS31	62	HBUS0	98	PXDAT22	134	FRMEND
27	HBUS30	63	HBOUT	99	PXDAT23	135	HALF-FULL (IRQ2)
28	HBUS29	64	HBCLK	100	PXADR0	136	NC
29	HBUS28	65	START	101	VDD	137	VSS
30	HBUS27	66	TMZ	102	PXADR1	138	NC
31	HBUS26	67	TMO	103	PXADR2	139	VSS
32	HBUS25	68	TMT	104	PXADR3	140	NC
33	VDD	69	TMOUT	105	VSS	141	NC
34	HBUS24	70	NMRQ (IRQ1)	106	PXADR4	142	NC
35	HBUS23	71	RESET	107	PXADR5	143	NC
36	HBUS22	72	TEST	108	PXADR6	144	NC

**Table 6-22 CL560(CL550) MQUAD Pin List Sorted by Pin Name**

Signal Name	Pin #	Signal Name	Pin #	Signal Name	Pin #	Signal Name	Pin #
BLANK	133	HBUS(6)	54	PXADR(14)	129	PXOUT	118
CLK3	131	HBUS(7)	53	PXADR(15)	130	PXPHASE	124
DMA_MSTR	19	HBUS(8)	52	PXADR(2)	103	PXRE	123
DRQ	20	HBUS(9)	51	PXADR(3)	104	PXWE	122
FRMEND	134	HBUS_32	18	PXADR(4)	106	RESET	71
HALF-FULL (IRQ2)	135	HSYNC	125	PXADR(5)	10	STALL	132
HBCLK	64	ID(0)	25	PXADR(6)	108	START	65
HBOOT	63	ID(1)	24	PXADR(7)	109	TEST	72
HBUS(0)	62	ID(2)	23	PXADR(8)	110	TMO	67
HBUS(1)	60	ID(3)	22	PXADR(9)	112	TMT	68
HBUS(10)	50	NC	140	PXCLK	120	TMZ	66
HBUS(11)	48	NC	1	PXDAT(0)	74	TMOUT	69
HBUS(12)	47	NC	3	PXDAT(1)	75	VDD (NC)	4
HBUS(13)	46	NC	5	PXDAT(10)	85	VDD (NC)	9
HBUS(14)	45	NC	6	PXDAT(11)	86	VDD	73
HBUS(15)	44	NC	7	PXDAT(12)	87	VDD	101
HBUS(16)	42	NC	8	PXDAT(13)	88	VDD	115
HBUS(17)	41	NC	10	PXDAT(14)	89	VDD	121
HBUS(18)	40	NC	11	PXDAT(15)	90	VDD	127
HBUS(19)	39	NC	13	PXDAT(16)	91	VDD	49
HBUS(2)	58	NC	14	PXDAT(17)	92	VDD	33
HBUS(20)	38	NC	15	PXDAT(18)	93	VDD	17
HBUS(21)	37	NC	16	PXDAT(19)	94	VSS (NC)	2
HBUS(22)	36	NC	136	PXDAT(2)	76	VSS (NC)	12
HBUS(23)	35	NC	138	PXDAT(20)	96	VSS	95
HBUS(24)	34	NC	141	PXDAT(21)	97	VSS	83
HBUS(25)	32	NC	142	PXDAT(22)	98	VSS	105
HBUS(26)	31	NC	143	PXDAT(23)	99	VSS	111
HBUS(27)	30	NC	144	PXDAT(3)	77	VSS	119
HBUS(28)	29	NMRQ (IRQ1)	70	PXDAT(4)	78	VSS	59
HBUS(29)	28	PXADR(0)	100	PXDAT(5)	79	VSS	61
HBUS(3)	57	PXADR(1)	102	PXDAT(6)	80	VSS	43
HBUS(30)	27	PXADR(10)	113	PXDAT(7)	81	VSS	137
HBUS(31)	26	PXADR(11)	114	PXDAT(8)	82	VSS	139
HBUS(4)	56	PXADR(12)	116	PXDAT(9)	84	VSS	21
HBUS(5)	55	PXADR(13)	128	PXIN	117	VSYNC	126

# 7 Registers

This chapter describes each of the registers used by programmers. You should be familiar with the CL550 and CL560's external signals as described in Chapter 3. The sections in this chapter are:

- 7.1, Video Interface Registers
- 7.2, Compression and Decompression Registers

This chapter includes detailed register definitions for each programmable register. The following information applies to all registers:

- Bits marked *Res* are reserved. Reserved bits should be written as zeros. Reading from reserved bits gives undefined data.
- The CL550 and the CL560 register sets have slight differences. If a register exists in one part, and does not exist in the other, then the part number is included in **bold** in the register name. In the register definitions that follow, these differences are noted with a *warning in italics*.
- All register addresses are given in hexadecimal.

**Table 7-1 CL550 Register and Table Summary**

<b>Register/Table Name</b>	<b>Group</b>	<b>R/W</b>	<b>Addresses</b>	<b>Size</b>	<b>Page</b>
CODEC Register	FIFO	R/W	0000-7FFC	32/16	7-39
HPeriod Register	Video	R/W	8000	14	7-7
HSync Register	Video	R/W	8004	14	7-8
HDelay Register	Video	R/W	8008	14	7-8
HActive Register	Video	R/W	800C	12	7-9
VPeriod Register	Video	R/W	8010	14	7-9
VSync Register	Video	R/W	8014	11	7-8
VDelay Register	Video	R/W	8018	14	7-10
VActive Register	Video	R/W	801C	11	7-11
DCT; coef0 of multi1	DCT	W	8800	16	7-15
DCT; coef1 of multi1	DCT	W	8804	16	7-15
DCT; coef2 of multi1	DCT	W	8808	16	7-15
DCT; coef3 of multi1	DCT	W	880C	16	7-15
DCT; coef0 of multi2	DCT	W	8810	16	7-15
DCT; coef1 of multi2	DCT	W	8814	16	7-15
DCT; coef2 of multi2	DCT	W	8818	16	7-15
DCT; coef3 of multi2	DCT	W	881C	16	7-15
Init Register 5	DCT	W	8820	16	7-30
Init Register 6	DCT	W	8824	16	7-15
Configuration Register	Host Bus	R/W	9000	9	7-28
Huffman Table Load Enable Register	Host Bus	W	9004	1	7-20
S-Reset Register	Host Bus	W	9008	1	7-29
Start Register	Host Bus	R/W	900C	1	7-29
HV Enable	Host Bus	R/W	9010	1	7-13
Flags Register	Host Bus	R/W	9014	16	7-31
Interrupt Mask Register	Host Bus	R/W	9018	16	7-33
DMA Request Interrupt Mask Register	Host Bus	R/W	901C	16	7-35
Start of Frame Register	Host Bus	R/W	9020	1	7-26
Version Register	Host Bus	R	9024	3	7-30
Init Register 1	Host Bus	W	9800	7	7-15
Init Register 2	Host Bus	W	9804	7	7-15
Huffman Table Sequence Register	Huffman Unit	W	A000	10	7-21
DPCM Register Sequence High	Huffman Unit	W	A004	10	7-19
DPCM Register Sequence Low	Huffman Unit	W	A008	10	7-19
Coder Attributes Register	Huffman Unit	W	A00C	7	7-21

**Table 7-1 CL550 Register and Table Summary**

<b>Register/Table Name</b>	<b>Group</b>	<b>R/W</b>	<b>Addresses</b>	<b>Size</b>	<b>Page</b>
Coding Interval Register H	Huffman Unit	W	A010	8	7-22
Coding Interval Register L	Huffman Unit	W	A014	8	7-22
Decoder Table Sequence Length Register	Huffman Unit	R/W	A80C	4	7-21
Decoder Marker Register	Huffman Unit	R	A810	8	7-26
Decoder Resume Flag Register	Huffman Unit	W	A814	1	7-26
Decoder DPCM Reset Register	Huffman Unit	W	A818	1	7-20
Decoder Code Order Register	Huffman Unit	R/W	A81C	1	7-27
Init Register 3	Huffman Unit	W	B600	11	7-15
Quantizer Table (Double-Buffer Mode)	Quantizer	R/W	B800-B9FC	16	
Quantizer Table (Four-Table Mode)	Quantizer	R/W	B800-BBFC	16	
Quantizer A/B Table Select Register	Quantizer	W	BC00	1	7-16
Quantizer Sync Register	Quantizer	W	BE00	14	7-18
Quantizer Y/C Table Sequence Register	Quantizer	W	BE08	14	7-17
Quantizer A/B Table Sequence Register	Quantizer	W	BE0C	10	7-18
Color Transform Matrix	Pipeline	W	C000-C020	12	7-13
Video Latency Register	Pipeline	R/W	C030	14	7-11
HControl Register	Pipeline	R/W	C034	14	7-12
VControl Register	Pipeline	R/W	C038	14	7-12
Vertical Line Count Register	Pipeline	R	C03C	14	7-12
Init Register 4	Pipeline	W	CF00	11	7-15
Init Register 7	Pipeline	W	D400	16	7-15
Huffman Y-AC	Pipeline	R/W	E000-EAFC	9	7-20
FIFO Memory	Pipeline	R/W	D800-D9FC	13	
Huffman Y-DC	Pipeline	R/W	EC00-EC7C	9	7-20
Huffman C-AC	Pipeline	R/W	F000-FAFC	9	7-20
Huffman C-DC	Pipeline	R/W	FC00-FC7C	9	7-20

**Table 7-2 CL560 Register and Table Summary**

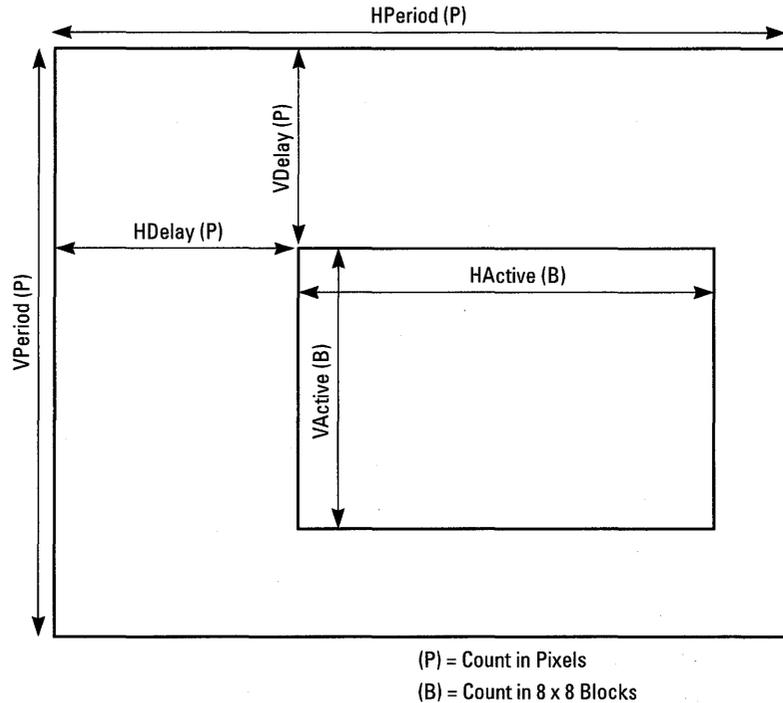
<b>Register/Table Name</b>	<b>Group</b>	<b>R/W</b>	<b>Addresses</b>	<b>Size</b>	<b>Page</b>
CODEC FIFO	Video	R/W	0000-7FFC	32/16	7-39
HPeriod Register	Video	R/W	8000	14	7-7
HSync Register	Video	R/W	8004	14	7-8
HDelay Register	Video	R/W	8008	14	7-8
HActive Register	Video	R/W	800C	12	7-9
VPeriod Register	Video	R/W	8010	14	7-9
VSynC Register	Video	R/W	8014	11	7-8
VDelay Register	Video	R/W	8018	14	7-10
VActive Register	Video	R/W	801C	11	7-11
DCT; coef0 of multi1	DCT	W	8800	16	7-15
DCT; coef1 of multi1	DCT	W	8804	16	7-15
DCT; coef2 of multi1	DCT	W	8808	16	7-15
DCT; coef3 of multi1	DCT	W	880C	16	7-15
DCT; coef0 of multi2	DCT	W	8810	16	7-15
DCT; coef1 of multi2	DCT	W	8814	16	7-15
DCT; coef2 of multi2	DCT	W	8818	16	7-15
DCT; coef3 of multi2	DCT	W	881C	16	7-15
Init Register 5	DCT	W	8820	16	7-30
Init Register 6	DCT	W	8824	16	7-15
Configuration Register	Host Bus	R/W	9000	9	7-28
Huffman Table Load Enable Register	Host Bus	W	9004	1	7-20
S-Reset Register	Host Bus	W	9008	1	7-29
Start Register	Host Bus	R/W	900C	1	7-29
HV Enable	Host Bus	R/W	9010	1	7-13
Flags Register	Host Bus	R/W	9014	16	7-31
IRQ1 Mask Register	Host Bus	R/W	9018	16	7-33
DMA Request Interrupt Mask Register	Host Bus	R/W	901C	16	7-35
Start of Frame Register	Host Bus	R/W	9020	1	7-26
Version Register	Host Bus	R	9024	3	7-30
IRQ2 Interrupt Mask Register	Host Bus	R/W	9028	16	7-37
FRMEND Enable Register	Host Bus	R/W	902C	3	7-38
Init Register 1	Host Bus	W	9800	7	7-15
Init Register 2	Host Bus	W	9804	7	7-15
Huffman Table Sequence Register	Huffman Unit	W	A000	10	7-21
DPCM Register Sequence High	Huffman Unit	W	A004	10	7-19
DPCM Register Sequence Low	Huffman Unit	W	A008	10	7-19

**Table 7-2 CL560 Register and Table Summary**

<b>Register/Table Name</b>	<b>Group</b>	<b>R/W</b>	<b>Addresses</b>	<b>Size</b>	<b>Page</b>
Coder Attributes Register	Huffman Unit	W	A00C	7	7-21
Coding Interval Register H	Huffman Unit	W	A010	8	7-22
Coding Interval Register L	Huffman Unit	W	A014	8	7-22
Coder Sync Register	Huffman Unit	W	A020	10	7-22
Compressed Word Count Register (High)	Huffman Unit	R/W	A024	16	7-23
Compressed Word Count Register (Low)	Huffman Unit	R/W	A028	16	7-23
Coder Rate Control Active Register	Huffman Unit	R/W	A02C	1	7-24
Coder Rate Control Enable Register	Huffman Unit	R/W	A030	5	7-24
Coder Robustness Active Register	Huffman Unit	R/W	A034	1	7-25
Coder RST Padding Control Register	Huffman Unit	R/W	A038	16	7-25
Decoder Table Sequence Length Register	Huffman Unit	R/W	A80C	4	7-21
Decoder Marker Register	Huffman Unit	R	A810	8	7-26
Decoder DPCM Reset Register	Huffman Unit	W	A818	1	7-20
Decoder Code Order Register	Huffman Unit	R/W	A81C	1	7-27
Decoder Start Register	Huffman Unit	W	A820	1	7-27
Decoder Mismatch Register	Huffman Unit	R/W	A824	1	7-27
Decoder Mismatch Error Code Register	Huffman Unit	R	A828	16	7-28
Init Register 3	Huffman Unit	W	B600	11	7-15
Quantizer Table (Double-Buffer Mode)	Quantizer	R/W	B800-B9FC	16	
Quantizer Table (Four-Table Mode)	Quantizer	R/W	B800-BBFC	16	
Quantizer A/B Table Select Register	Quantizer	W	BC00	1	7-16
Quantizer Sync Register	Quantizer	W	BE00	14	7-18
Quantizer Y/C Table Sequence Register	Quantizer	W	BE08	14	7-17
Quantizer A/B Table Sequence Register	Quantizer	W	BE0C	10	7-18
Color Transform Matrix	Pipeline	W	C000-C020	12	7-13
Video Latency Register	Pipeline	R/W	C030	14	7-11
HControl Register	Pipeline	R/W	C034	14	7-12
VControl Register	Pipeline	R/W	C038	14	7-12
Vertical Line Count Register	Pipeline	R	C03C	14	7-12
Init Register 4	Pipeline	W	CF00	11	7-15
Init Register 7	Pipeline	W	D400	16	7-15
FIFO Level Register	Pipeline	R	DA04	16	7-40
Huffman Y-AC	Pipeline	R/W	E000-E5F	9	7-20
Huffman Y-DC	Pipeline	R/W	E600-E65C	9	7-20
Huffman C-AC	Pipeline	R/W	E800-EDFC	9	7-20
Huffman C-DC	Pipeline	R/W	EE00-EE5C	9	7-20
Ping-pong Buffer	Pipeline	R/W	F000-F1FC	13	

## 7.1 Video Interface Registers

Six registers are used to control the horizontal and vertical synchronization pulse widths, frame size and active video period. These values are shown graphically in Figure 7-1.



**Figure 7-1 Video Field Registers**

Some of the video interface registers have a master/slave arrangement. When you write to the register, you load the master, and when you read from the register, you read the contents of the slave. Slave registers only get updated after VSYNC starts the compression or decompression process. The registers that have this property are:

- HPeriod, VPeriod
- HSync, VSync
- HDelay, VDelay
- HActive, VActive

**HPeriod Register****0x8000**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res		Horizontal Sync Period													

The HPeriod register serves two different functions depending on whether the part is operating in the master mode or the slave mode. In master mode, it controls the number of pixels between consecutive HSYNC pulses. The value used to set HPeriod varies depending on the video mode selected. The formula used to compute this value is shown in Table 7-3.

**Table 7-3 HPeriod Register Value Calculation**

Video Mode	HPeriod Value <sup>1</sup>
Single Component (Grayscale)	(NumPixel / 2) - 1
RGB to YUV 4:2:2	NumPixel - 1
YUV 4:2:2	NumPixel - 1
4:4:4 to 4:2:2	NumPixel - 1
4:4:4	(2 * NumPixel) - 1
4:4:4:4	(2 * NumPixel) - 1
CMYK	(2 * NumPixel) - 1

1. NumPixel = The number of pixels between HSYNC assertions.

In slave mode, the value programmed into the HPeriod register is the time from the falling edge of a horizontal sync pulse until the next HSYNC pulse is recognized. It is used to reject the composite component of HSYNC during the vertical blanking time (serrated VSYNC). A typical value stored in the HPeriod register is 90% of the number of pixels contained in one horizontal line.

*Note: During slave mode operations, if STALL is asserted during an active compression or decompression operation, the HSYNC input should be delayed by the same amount of time that STALL is asserted. This is necessary because STALL stops the internal HPERIOD counter.*

**HSync Register****0x8004**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<i>Res</i>		HSync													

The HSync register is only active in the master mode. In master mode, it is used to program the width of the HSYNC pulse. HSync should be set to the number of pixels contained in one horizontal sync pulse, minus one.

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

**HDelay Register****0x8008**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<i>Res</i>		HDelay													

The HDelay register controls the delay from the falling edge of HSYNC to the first active pixel. The value programmed into HDelay depends on the video mode. The formula used to compute this value is shown in Table 7-4.

**Table 7-4****HDelay Register Value Calculation**

<b>Video Mode</b>	<b>HDelay Value<sup>1</sup></b>
Single Component (Grayscale)	$1/2 * \text{Pixel Delay}$
RGB to YUV 4:2:2	Pixel Delay
YUV 4:2:2	Pixel Delay
4:4:4 to 4:2:2	Pixel Delay
4:4:4	$2 * \text{Pixel Delay}$
4:4:4:4	$2 * \text{Pixel Delay}$
CMYK	$2 * \text{Pixel Delay}$

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

**HActive Register****0x800C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>				HActive											

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 8 x 8 blocks). The formula used to compute this value is shown in Table 7-5.

**Table 7-5 HActive Register Value Calculation**

Video Mode	HActive Value <sup>1</sup>
Single Component (Grayscale)	NumActiveBlocks - 1
RGB to YUV 4:2:2	(2 * NumActiveBlocks) - 1
YUV 4:2:2	(2 * NumActiveBlocks) - 1
4:4:4 to 4:2:2	(2 * NumActiveBlocks) - 1
4:4:4	(4 * NumActiveBlocks) - 1
4:4:4:4	(4 * NumActiveBlocks) - 1
CMYK	(4 * NumActiveBlocks) - 1

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

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

**VPeriod Register****0x8010**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>		VPeriod													

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

**VSync Register****0x8014**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>						VSync									

VSync is used to control the width of the vertical sync pulse in the master mode. It should be set to the vertical sync pulse width in lines. The VSync register is ignored in slave mode.

**VDelay Register****0x8018**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>		VDelay													

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

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

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

$$\text{Minimum VDelay} = 9 + \text{trunc}(\frac{\text{Video Latency} + \text{HActive Clocks}}{\text{HPeriod Clocks}})$$

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

and HPeriod Clocks = 2 \* HPeriod

The value that is loaded into the VDelay register is:

$$\text{VDelay Register Value} = \text{Actual VDelay} - \text{Minimum VDelay}$$

Note: The minimum VDelay is 9 lines. This minimum is because 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 part. For horizontal periods less than 200 pixels, internal pipeline latencies may be greater than one line, and the minimum Vertical Delay must be adjusted upwards.

**VActive Register****0x801C**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<i>Res</i>					VActive										

The VActive register controls the number of active vertical lines. It should be set to the vertical block count of the active window (one eighth the number of active lines).

**Video Latency Register****0xC030**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<i>Res</i>		Video Latency													

The video latency register controls internal pipelining timings. The value programmed into the video latency register depends on the video mode and are shown in Table 7-6.

**Table 7-6 Video Latency Register Values**

<b>Video Mode</b>	<b>Value</b>
MONO	00BF
4:2:2	017F
1 Table 4:4:4	0181
2 Table 4:4:4	0181
3 Table 4:4:4	0181
4:4:4	017F
RGB to YUV 4:2:2	0185
4:4:4 to 4:2:2	017F

**Vertical Line Count Register****0xC03C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>		Vertical Line Count													

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.

**HControl Register****0xC034**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>		HControl													

HControl is a decompression parameter that determines the horizontal position at which the part should stop removing data from the internal FIFO for the current frame. If the part is being reset between frames, as in most still and many video systems, the HControl value should be set to 0x3FFF. During streaming operations (frames that are written back-to-back, with no reset between frames) the value for Hcontrol must be computed. Routines for calculating this value are provided on the CL550/CL560 programming examples diskette. The HControl register is ignored in the compression mode.

**VControl Register****0xC038**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>		VControl													

VControl is a decompression parameter that determines the vertical position at which the part should stop removing data from the internal FIFO for the current frame. If the part is being reset between frames, as in most still and many video systems, the VControl value should be set to 0x3FFF. During streaming operations (frames that are written back-to-back, with no reset be-

tween frames) the value for Vcontrol must be computed. Routines for calculating this value are provided on the CL550/CL560 programming examples diskette. The VControl register is ignored in the compression mode.

**HV Enable Register****0x9010**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															HVEn

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  $\overline{\text{VSYNC}}$  pulse occurs during the update. There are two sets of video parameter registers; a shadow register written from the host bus, and an active register that is used for the current frame. The shadow register set can always be updated by the host; however the active register set can only be updated on the falling edge of  $\overline{\text{VSYNC}}$ .

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).

Notes:

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

**Color Transformation Matrix****0xC000 - 0xC020**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>				Color Transformation Matrix (One of Nine)											

There are nine 12-bit registers for loading the coefficients of the RGB to YUV transformation matrix. The nine register address-

es are shown in Table 7-7.

$M_{ij}$  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 as shown in Table 7-7.

**Table 7-7 RGB - YUV Color Transformation Coefficient Addresses**

Address	Compression Mode	Decompression Mode
	RGB to YUV	YUV to RGB
0xC000	$M_{11}$ (0x0133)	$M'_{11}$ (0x0400)
0xC004	$M_{12}$ (0x0259)	$M'_{12}$ (0x0FFE)
0xC008	$M_{13}$ (0x0074)	$M'_{13}$ (0x059C)
0xC00C	$M_{21}$ (0x0F54)	$M'_{21}$ (0x0400)
0xC010	$M_{22}$ (0x0EAD)	$M'_{22}$ (0x0EA2)
0xC014	$M_{23}$ (0x01FF)	$M'_{23}$ (0x0D23)
0xC018	$M_{31}$ (0x01FF)	$M'_{31}$ (0x0400)
0xC01C	$M_{32}$ (0x0E53)	$M'_{32}$ (0x071B)
0xC020	$M_{33}$ (0x0FAE)	$M'_{33}$ (0x0FFD)

CL550/560 matrix values are computed by multiplying the actual decimal coefficient by 1024. The result is then rounded up to the next integer. For example:

$$Y = .299 R$$

$$M_{11} = .299 \times 1024 = 307_{10} \text{ or } 0x133$$

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

- The matrices should be approximate inverses of one another in order to minimize round-up errors.
- The forward matrix should produce pixel values in the range of [0,255.37] for unsigned values or [-128,127.37] for signed values.

Calculating matrix values is outlined in Chapter 2.

**7.2  
Compression and  
Decompression  
Registers**

**DCT Tables**

**0x8800 - 0x881C**

The DCT tables consist of eight 16-bit entries. The host bus interface can write directly to each entry. DCT values are used as multiplication values for the DCT/IDCT operation during the video compression/decompression operation. The values loaded into the DCT tables are constant for both compression and decompression, and are only loaded once during initialization. The values that should be programmed into the registers are shown in Table 7-8.

**Table 7-8**

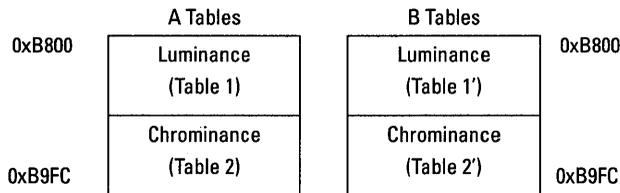
**DCT Table Values**

Address	DCT Data	Address	DCT Data
0x8800	0x5A82	0x8810	0x5A82
0x8804	0x7FFF	0x8814	0x7FFF
0x8808	0x30FC	0x8818	0x30FC
0x880C	0x7642	0x881C	0x7642

**Quantizer Tables**

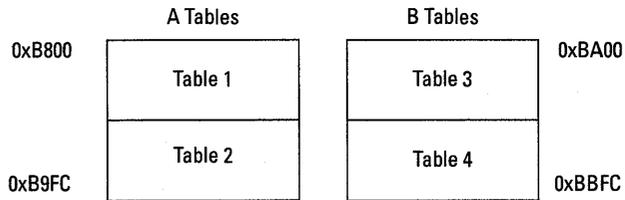
**0xB800 - 0xBBFC**

The Quantizer tables consist of two sets of 128 16-bit entries that the host bus interface can read or write directly. These values are used during the compression/decompression process. Table entries can be used in double-buffer mode or four-table mode as shown in Figures 7-2 and 7-3. When the part operates in double-buffer mode, one set of tables can be loaded while the other set is active.



**Figure 7-2**

**Quantizer Tables Configuration (Double-buffer Mode)**



**Figure 7-3 Quantizer Tables Configuration (Four-table Mode)**

The four-table mode is useful for most still-image and video applications. Two-table (double buffer) mode is used in applications that need to change the Q-factor every frame. Users should use the four-table mode to load the registers and then switch to two-table mode to do double-buffering.

In double-buffering mode, the quantizer table RAM is split into two parts. One half of the RAM is used by the host interface while the other half is being used by the quantizer unit.

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 0xBC00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															A/B

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 part is reset, the Quantizer A/B Table Select register points to the B tables, allowing the A tables to be loaded. This register is only used in the double-buffer mode.

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 that 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 a VSYNC falling edge.

This limitation can be overcome when the part is not in active operation. Quantizer Sync Register (Address BE00) bits 8 and 9 control an internal test sync signal that, when set to 1, and then 0, will toggle the table select switch in the same fashion as VSYNC. An alternative method is to use four-table mode to load each of the four tables (0xB800, 0xB900, 0xBA00 and 0xBB00), and then switch to two-table mode.

**Quantizer Y/C Table Sequence Register                      0xBE08**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
<i>Res</i>		Quantizer Y/C Table Select Sequence Register													

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 the data formats being processed. The values for different operating modes and data formats are shown in Table 7-9.

**Table 7-9                      Quantizer Y/C Table Values**

<b>Mode</b>	<b>Compression Value</b>	<b>Decompression Value</b>
MONO	0x2000	0x2000
4:2:2	0x2099	0x2033
4:4:4	0x2044	0x2088
4:4:4:4	0x2055	0x20AA
RGB to YUV 4:2:2	0x2099	0x2033
4:4:4 to 4:2:2	0x2099	0x2033

**Quantizer A/B Table Sequence Register** **0xBE0C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>						Quantizer A/B Table Sequence									

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 part operates in 4:4:4:4 and 4:4:4 mode. Register values for different operating modes and data formats are provided in Table 7-10.

**Table 7-10**      **Quantizer A/B Table Values**

Mode	Compression Value	Decompression Value
MONO	0x0000	0x0000
4:2:2	0x0000	0x0000
4:4:4	0x0088	0x0011
4:4:4:4	0x0099	0x0033
RGB to YUV 4:2:2	0x0000	0x0000
4:4:4 to 4:2:2	0x0000	0x0000

**Quantizer Sync Register** **0xBE00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>										Data Sync Field					

The Quantizer Sync register supports 4:4:4:4 mode (four-table mode) and data synchronization. All bits marked Res should be set to 0 during normal operation. The function of the remaining bits is:

- **Bit 10, Double-Buffer:** This bit selects double-buffer mode when set to 0. When set to 1, this bit selects four-table mode.
- **Bits 5:0, Data Sync Field:** The data stored in this field depends on the video mode selected. The proper value for this field is shown in Table 7-11, Quantizer Sync Register Data Sync Field Values.

**Table 7-11 Quantizer Sync Register Data Sync Field Values**

<b>Mode</b>	<b>Compression Value</b>	<b>Decompression Value</b>
MONO	0x0406	0x043E
4:2:2	0x0406	0x043E
4:4:4	0x0404	0x043E
4:4:4:4	0x0406	0x043E
RGB to YUV 4:2:2	0x0400	0x043E
4:4:4 to 4:2:2	0x0406	0x043E

**Coder/Decoder DPCM Reg. Seq. Registers, RH 0xA004**

**Coder/Decoder DPCM Reg. Seq. Registers, RL 0xA008**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>						Coder/Decoder DPCM Register Sequence									

The CL550 and CL560 have two registers (RH and RL) to select the active DPCM registers. There are four DPCM registers specified by a two-bit address. The values programmed into these registers are shown in Table 7-12.

**Table 7-12 DPCM Sequence Register Values**

<b>Mode</b>	<b>DPCM_RL</b>	<b>DPCM_RH</b>
MONO	0x000	0x000
4:2:2	0x044	0x088
4:4:4	0x012	0x044
4:4:4:4	0x0AA	0x0CC
RGB to YUV 4:2:2	0x044	0x088
4:4:4 to 4:2:2	0x044	0x088

The values programmed into the RL and RH registers are identical during either compression or decompression. See Chapter 8 for details on how to program these registers.

**Decoder DPCM Reset Register**

**0xA818**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															DDRR

The host processor resets the four DPCM registers by writing to bit zero of the Decoder DPCM Reset register. In decompression mode, after the decoder has flagged the End of Image (EOI) marker code, the host should reset the DPCM registers before restarting the decoder.

**Huffman Code Tables**

**0xE000 - 0xFC7C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>							Huffman Code Table Value								

The CL550 uses two Huffman table layouts: one for coding and another for decoding. The CL560 uses a single Huffman table for both coding and decoding. The default Huffman code tables for the baseline JPEG algorithm are defined in the JPEG Draft Proposal. The tables loaded into the part are generated by processing the JPEG code table with the initialization software. Unused Huffman table entries must be initialized to zero. See Chapter 8 for details on programming this register.

**Huffman Table Load Enable Register**

**0x9004**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															LEn

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

**Huffman Table Sequence Register**

**0xA000**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>						Huffman Table Sequence Register									

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 the different operating modes and formats are shown in Table 7-13.

**Table 7-13 Huffman Table Sequence Register Values**

Mode	Value
MONO	0x0000
4:2:2	0x00CC
4:4:4	0x0000
4:4:4:4	0x0000
RGB to YUV 4:2:2	0x00CC
4:4:4 to 4:2:2	0x00CC

**Coder Attributes Register**

**0xA00C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>									EOB	LSB	RSTEN	MCU Block #			

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

- **Bit 6, EOB:** When EOB (End of Block) 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, RSTEN:** When RSTEN is set to 1, the coder will set to 1 the restart marker codes in the compressed data stream at the end of each new coding interval.
- **Bits 3-0, MCU Block #:** These bits specify the number of

blocks are Minimum Coded Unit (MCU). Suggested values for several different modes are listed in Table 7-14.

**Table 7-14 MCU Block Number Values**

Mode	Value
MONO	0x01
422	0x04
444	0x03
RGB-422	0x04
444-422	0x04
4444	0x04

**Coder Coding Interval Registers 0xA010 and 0xA014**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>								RH Coding Interval							
<i>Res</i>								RL Coding Interval							

These two registers (RH and RL) define the number of MCUs per coding interval. The largest number of MCUs supported by the CL550 or CL560 is  $2^{16}-1$ , in accordance with the JPEG standard specifications. Assuming a coding interval of 15 MCUs, for example, then the value of RH and RL must be set to 0x00 and 0x0F respectively.

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

Note that for the 4:2:2, RGB -> 4:2:2 and 4:4:4 -> 4:2:2 modes, one MCU corresponds to a block size of 16 x 8 pixels, and for all other modes a MCU corresponds to a block size of 8 x 8 pixels.

**Coder Sync Register (CL560) 0xA020**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>					CSB	Coder Sync Register									

- **Bit 10: CSB:** The Coder Sync Bit is used to select whether an EOI (0xFFD9) or an RST (0xFFD0 - 0xFFD7) is generated at the end of an image. Setting this bit to 0 causes an EOI (0xFFD9) to be generated, and setting this bit to 1 causes an RST (0xFFD0 - 0xFFD7) to be generated.
- **Bits 9:0 Coder Sync Register:** The Coder Sync register is used to synchronize the data stream in the Huffman Coder Unit. This register is loaded at initialization with a value specific to the video mode. The required values are shown in Table 7-15.

**Table 7-15 Coder Sync Register Initialization Values**

Mode	Value
MONO	0x0100
4:2:2	0x01C0
4:4:4	0x01C2
4:4:4:4	0x01C0
RGB to YUV 4:2:2	0x01C6
4:4:4 to 4:2:2	0x01C0

**Compressed Word Count Register, High (CL560) 0xA024**  
**Compressed Word Count Register, Low (CL560) 0xA028**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Compressed Word Count, High and Low															

At the end of the compression of an image, the combined 32-bit value in these registers represents the total number of 32-bit words in the compressed frame. The count is updated once each frame at the end of compression. This value can be read in response to the FRMEND interrupt to determine the size of each compressed video frame. Reading either of these registers resets the FRMEND interrupt. The host can reset either register to zero by writing any data to that register or by resetting the device.

**Coder Rate Control Active Register (CL560) 0xA02C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															CRCA

The CL560 sets the Coder Rate Control Active register to 1 if the rate control mechanism was activated at any point in the compression of a frame. It is cleared to 0 by a host write or by a reset.

**Coder Rate Control Enable Register (CL560) 0xA030**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>											Coder Rate Control Enable				

This register enables the CL560's internal data rate limiting mechanism during compression (not used during decompression). The rate control mechanism provides a means of recovery when the host system cannot keep up with the CL560's output data rate during compression operations. Rate control becomes active when the FIFO level exceeds the threshold level set in this register. When rate control is active, the Huffman coding unit automatically drops all remaining AC terms in the current block and inserts an end-of-block (EOB) code. It also drops all AC terms for all of the blocks in the current frame. This action results in significant image artifacts but slows the output data rate so that the host may be able to read down the FIFO data before an overflow occurs. If an overflow occurs, the LATE bit in the Flags register is set and the rest of the compressed data is invalid. Table 7-16 shows the threshold level settings of the register (X = don't care). When the register value is 0, the rate control feature is disabled. This register is cleared to 0 at reset.

**Table 7-16 FIFO Threshold Levels**

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	FIFO Threshold Level
X	X	X	X	1	1/4 Full
X	X	X	1	0	1/2 Full
X	X	1	0	0	3/4 Full
X	1	0	0	0	7/8 Full
1	0	0	0	0	Full

**Coder Robustness Active Register (CL560) 0xA034**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															CRAR

The Coder Robustness Active Register is set to 1 when AC terms are lost when the rate control mechanism is enabled during compression. The register is cleared to 0 either by an explicit host write or by a device reset.

**Coder RST Padding Control Register (CL560) 0xA038**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Coder RST Padding Control															

The Coder RST (Restart) Padding Control register controls the word-padding mechanism of the CL560. If this register is set to 0, the CL560 applies word padding only to the End-Of-Image marker (EOI, 0xFFD9). If this register is set to any non-zero value *n*, every *n*th RST marker word is padded.

**Start of Frame Register**

**0x9020**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															SOF

The Huffman coder is reset by writing either 0 or 1 to bit 0 the Start of Frame register. The host can use this register to reset the coder when the host starts reading “FFFF...” data (end of frame) out of the coder. This is used in frame buffer applications.

**Decoder Table Sequence Length Register**

**0xA80C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>												Sequence Length			

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

**Decoder Marker Register**

**0xA810**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>								Marker Code							

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 (CL550)**

**0xA814**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															DRF

When the decoder detects a marker code (except the RST marker code), it writes the code 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** **0xA81C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															DCO

This register must be set to 1.

**Decoder Start Register (CL560)** **0xA820**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decoder Start															

The CL560 starts decoding when either the FIFO contains at least 32 entries or the host processor writes to this register. The value written is ignored. For small images whose coded files contain less than 32 coded words, the host processor should pre-fill the FIFO then write to this register. A write to this register has no effect after the decoding operation has started.

**Decoding Mismatch Register (CL560)** **0xA824**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															DM

The CL560 sets this register to 1 when a mismatch occurs during Huffman decoding. The host processor must explicitly clear this register by writing a 0. This register is cleared to 0 at reset.

**Decoding Mismatch Error Code Register (CL560) 0xA828**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decoding Mismatch Error Code															

When a decoding mismatch occurs, the CL560 loads this register with the first mismatched 16-bit bitstream. If the decoding Mismatch register is set to 0, the contents of this register are not meaningful.

**Configuration Register 0x9000**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>						Zero	Dir	Video Mode Select				Mstr	Zero	EOF	<i>Res</i>

The values loaded into the Configuration register determine the device’s operating mode and environment. The function of each bit is shown below:

- Bit 9, Zero:** This bit must be set to 0.
- Bit 8, Dir:** This bit defines the direction of the processing path. When set to 0, the part is in the compression mode. When set to 12, the part is in the decompression mode.
- Bits 7 - 4, Video Mode Select:** These four bits are used to select the video mode format as shown in Table 7-17.

**Table 7-17 Video Mode Select Bits**

Video Mode	Select Bits
Invalid	0000
YUV 4:2:2 Mode	0001
4:4:4 to 4:2:2 Mode	0010
RGB to YUV 4:2:2 Mode	0011
4:4:4 Mode	0100
Invalid	0101
4:4:4:4 Mode	0110
Invalid	0111
Single Component Mode (Grayscale)	1000
Invalid	1001 - 1111

- **Bit 3, Master Mode:** When set to 0, this bit puts the device into the slave mode. When set to 1, this bit puts the device into the master mode (driving HSYNC and VSYNC).
- **Bit 2, Zero:** This bit must be set to 0.
- **Bit 1, End of Frame Enable:** When this bit is set to 0, the FRMEND output is never asserted. When set to 1, the FRMEND output is equal to the logical “nand” of Video Inactive (Vnac) and FIFO-empty (fi0e) bits in the Flags register.

**S-Reset Register**

**0x9008**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															S-Res

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

**Start Register**

**0x900C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>															Start

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

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

**Version Register**

**0x9024**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>													Version		

The version register contains the version number of the part. This is important information to have when calling for technical support. Version numbers of the parts at the time this manual was printed were:

**Table 7-18 Version Number Register Contents**

Part Type	Version
CL550	011
CL560 (Revision ES3)	100

**Init Registers**

**See Table**

The Initialization registers configure the compression/decompression unit pipeline for each video mode. The values that should be loaded are listed in Table 7-19.

**Table 7-19 Initialization Registers**

Register	Type	Address	Size <sup>1</sup>
Init Register 1	W	0x9800	7
Init Register 2	W	0x9804	7
Init Register 3	W	0xB600	11
Init Register 4	W	0xCF00	11
Init Register 5	W	0x8820	16
Init Register 6	W	0x8824	16
Init Register 7	W	0xD400	16

1. Valid bits start at the Least Significant Bit and work up. All bits not listed should be set to 0. The values listed below take this into account.

The values that should be programmed into these registers for each of the modes are shown in Table 7-20.

**Table 7-20 Initialization Register Values by Mode**

Mode	Mono	4:2:2	4:4:4	4:4:4:4	RGB to YUV 4:2:2	4:4:4 to 4:2:2
Init Reg 1, Compression	0x02	0x42	0x40	0x42	0x3C	0x3C
Init Reg 2, Compression	0x01	0x01	0x00	0x01	0x3D	0x3D
Init Reg 3, Compression	0x141	0x081	0x07F	0x081	0x07B	0x07B
Init Reg 4, Compression	0xF7	0xF7	0xF5	0xF7	0xF1	0xF1
Init Reg 5, Compression	0x0	0x0	0xE	0x0	0xA	0xA
Init Reg 6, Compression	0x0	0x0	0x0	0x0	0x0	0x0
Init Reg 7, Compression	0x5D	0x1D	0x1B	0x1D	0x17	0x17
Init Reg 1, Decompression	0x3E	0x3E	0x3E	0x3E	0x3E	0x3E
Init Reg 2, Decompression	0x37	0x37	0x37	0x37	0x37	0x37
Init Reg 3, Decompression	0x1FF	0x1FF	0x1FF	0x1FF	0x1FF	0x1FF
Init Reg 4, Decompression	0x49	0x49	0x49	0x49	0x49	0x49
Init Reg 5, Decompression	0x5	0x5	0x5	0x5	0x5	0x5
Init Reg 6, Decompression	0x0	0x0	0x0	0x0	0x0	0x0
Init Reg 7, Decompression	0x22	0x22	0x22	0x22	0x22	0x22

**Flags Register**

**0x9014**

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

The Flags register provides status information. Each flag indicates the correct state of the part except the Late bit, which is latched. The function of each bit is defined as follows:

- **Bit 15, fi0nf:** When asserted, this bit indicates that the FIFO is not full.
- **Bit 14, fi0ne:** When asserted, this bit indicates that the FIFO is not empty.
- **Bit 13, CodecNB (CL550):** When equal to 1, this bit indicates that the CODEC register is not busy.

*Bit 12 is defined as Buser for the CL550, and FRMEND for the CL560. Both are explained below.*

- **Bit 12, Buser (CL550):** When equal to 1, this bit indicates that a bus error has occurred during a CODEC DMA operation.
- **Bit 12, FRMEND (CL560):** This bit is the status of the  $\overline{\text{FRMEND}}$  external signal. Bit 12 is set to 0 when  $\overline{\text{FRMEND}}$  is High (Deasserted), and is set to 1 when  $\overline{\text{FRMEND}}$  is low (asserted).
- **Bit 11, Mark:** When equal to 1, this bit indicates that a marker code has been detected in the data stream being decompressed.
- **Bit 10, Vsyn:** This field reflects the state of the  $\overline{\text{VSYNC}}$  pin. If it is a 1, the  $\overline{\text{VSYNC}}$  pin is a 0.
- **Bit 9, Vnac:** This bit defines the state of the part 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). 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 internal pipeline latency of the part. 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 last active line of the pixel bus.
- **Bit 7, fi0e:** When equal to 1, this bit indicates that the FIFO is empty.
- **Bit 6, fi1q:** When equal to 1, this bit indicates that the FIFO is one-quarter full.
- **Bit 5, fi1h:** When equal to 1, this bit indicates that the FIFO is one-half full.
- **Bit 4, fi3q:** When equal to 1, this bit indicates that the FIFO is three-quarters full.
- **Bit 3, fin1q:** When equal to 1, this bit indicates that the FIFO is not one-quarter full.
- **Bit 2, fin1h:** When equal to 1, this bit indicates that the FIFO is not one-half full.
- **Bit 1, fin3q:** When equal to 1, this bit indicates that the FIFO is not three-quarters full.
- **Bit 0, Late:** This bit 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 bit.

**NMRQ Interrupt Mask Register (CL550)                      0x9018**

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

The Interrupt Mask register defines the interrupt mask for the  $\overline{\text{NMRQ}}$  output pin. The condition of the output pin is determined by the following equation:

$$\begin{aligned} \overline{\text{NMRQ}} = & \text{!(flag[13] \cdot mask[13]) \cdot !(flag[12] \cdot mask[12]) \cdot} \\ & ((\text{flag}[0] \cdot \text{mask}[0]) \\ & + (\text{flag}[1] \cdot \text{mask}[1]) \\ & + (\text{flag}[2] \cdot \text{mask}[2]) \\ & + (\text{flag}[3] \cdot \text{mask}[3]) \\ & + (\text{flag}[4] \cdot \text{mask}[4]) \\ & + (\text{flag}[5] \cdot \text{mask}[5]) \\ & + (\text{flag}[6] \cdot \text{mask}[6]) \\ & + (\text{flag}[7] \cdot \text{mask}[7]) \\ & + (\text{flag}[8] \cdot \text{mask}[8]) \\ & + (\text{flag}[9] \cdot \text{mask}[9]) \\ & + (\text{flag}[10] \cdot \text{mask}[10]) \\ & + (\text{flag}[11] \cdot \text{mask}[11]) \\ & + (\text{flag}[14] \cdot \text{mask}[14]) \\ & + (\text{flag}[15] \cdot \text{mask}[15])) \end{aligned}$$

To mask (disable) an interrupt, that interrupts mask bit must be set to 0. If an interrupt occurs and its corresponding interrupt mask bit is set to 1, the part asserts the  $\overline{\text{NMRQ}}$  bit. Asserting the RESET input of the S-Reset Flag register sets all fields in the Interrupt Mask register to 0. The function of each bit is defined as follows:

- **Bit 15, fi0nf:** This field enables the FIFO Not Full Interrupt.
- **Bit 14, fi0ne:** This field enables the FIFO Not Empty Interrupt.
- **Bit 13, CodecNB:** This field enables the CODEC Register

Not Busy Interrupt.

- **Bit 12, Buser:** This field enables the Bus Error Interrupt.
- **Bit 11, Mark:** This field enables the Marker Code Interrupt.
- **Bit 10, Vsyn:** This field enables the Vertical Sync Interrupt.
- **Bit 9, Vnac:** In slave mode, this field enables the Vertical In-active Interrupt.
- **Bit 8, fend:** In slave mode, this field enables the FRMEND Interrupt.
- **Bit 7, fi0E:** This field enables the FIFO Empty Interrupt.
- **Bit 6, fi1q:** This field enables the FIFO One-quarter Full Interrupt.
- **Bit 5, fi1h:** This field enables the FIFO One-half Full Interrupt.
- **Bit 4, fi3q:** This field enables the FIFO Three-quarters Full Interrupt.
- **Bit 3, fin1q:** This field enables the FIFO Not One-quarter Full Interrupt.
- **Bit 2, fin1h:** This field enables the FIFO Not One-half Full Interrupt.
- **Bit 1, fin3q:** This field enables the FIFO Not Three-quarters Full Interrupt.
- **Bit 0, Late:** This interrupt enables the Data Late Interrupt.

**IRQ1 Mask Register (CL560)**

**0x9018**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
fi0nf	fi0ne	<i>Res</i>		Mark	Vsyn	Vnac	fend	fi0e	fi1q	fi1h	fi3q	fin1q	fin1h	fin3q	Late

This register has the same function as the Interrupt Mask Register in the CL550, with the exception of Bits 13 and 12, which are now reserved and must be set to zero.

**DMA Request Interrupt Mask Register**

**0x901C**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
fi0nf	fi0ne	CodecNB (Res)	Buser (Res)	Mark	Vsyn	Vnac	fend (Res)	fi0e	fi1q	fi1h	fi3q	fin1q	fin1h	fin3q	Late

The DMA Request Interrupt Mask register defines the interrupt mask for the  $\overline{DRQ}$  output pin. It can also be used as a general purpose interrupt signal. The DMA Request Interrupt Mask register is identical to the Interrupt Mask Register. For the CL560, the condition of the  $\overline{DRQ}$  pin is determined by the equation:

$$\begin{aligned} \overline{DRQ} = & ((\text{flag}[0] \cdot \text{mask}[0]) \\ & + (\text{flag}[1] \cdot \text{mask}[1]) \\ & + (\text{flag}[2] \cdot \text{mask}[2]) \\ & + (\text{flag}[3] \cdot \text{mask}[3]) \\ & + (\text{flag}[4] \cdot \text{mask}[4]) \\ & + (\text{flag}[5] \cdot \text{mask}[5]) \\ & + (\text{flag}[6] \cdot \text{mask}[6]) \\ & + (\text{flag}[7] \cdot \text{mask}[7]) \\ & + (\text{flag}[8] \cdot \text{mask}[8]) \\ & + (\text{flag}[9] \cdot \text{mask}[9]) \\ & + (\text{flag}[10] \cdot \text{mask}[10]) \\ & + (\text{flag}[11] \cdot \text{mask}[11]) \\ & + (\text{flag}[14] \cdot \text{mask}[14]) \\ & + (\text{flag}[15] \cdot \text{mask}[15])) \end{aligned}$$

For the CL550, the condition of the  $\overline{DRQ}$  pin is determined by the equation:

$$\begin{aligned} \overline{DRQ} = & /((\text{flag}[13] \cdot \text{mask}[13])) \cdot /((\text{flag}[12] \cdot \text{mask}[12]) \cdot \\ & ((\text{flag}[0] \cdot \text{mask}[0]) \\ & + (\text{flag}[1] \cdot \text{mask}[1]) \\ & + (\text{flag}[2] \cdot \text{mask}[2]) \\ & + (\text{flag}[3] \cdot \text{mask}[3]) \\ & + (\text{flag}[4] \cdot \text{mask}[4]) \\ & + (\text{flag}[5] \cdot \text{mask}[5]) \\ & + (\text{flag}[6] \cdot \text{mask}[6]) \\ & + (\text{flag}[7] \cdot \text{mask}[7]) \\ & + (\text{flag}[8] \cdot \text{mask}[8]) \\ & + (\text{flag}[9] \cdot \text{mask}[9]) \\ & + (\text{flag}[10] \cdot \text{mask}[10]) \\ & + (\text{flag}[11] \cdot \text{mask}[11]) \\ & + (\text{flag}[14] \cdot \text{mask}[14]) \\ & + (\text{flag}[15] \cdot \text{mask}[15])) \end{aligned}$$

To mask (disable) an interrupt, that interrupts mask bit must be set to 0. If an interrupt occurs and its corresponding interrupt mask bit is set to 1, the part asserts the NMRQ bit. Asserting the RESET input of the S-Reset Flag register sets all fields in the Interrupt Mask register to 0.

The  $\overline{\text{DRQ}}$  output pin is always deasserted in the cycle following the assertion of the START signal.

Note: The  $\overline{\text{DRQ}}$  signal is suppressed during a CODEC access.

The function of each bit is defined as follows:

- **Bit 15, fif0nf:** This field enables the FIFO Not Full Interrupt.
- **Bit 14, fif0ne:** This bit enables the FIFO Not Empty Interrupt.

*Note: Bit 13 in the CL560 DMA Request Interrupt Mask register is reserved, and should always be set to 0.*

- **Bit 13, CodecNB:** This field enables the CODEC Register Not Busy Interrupt.

*Note: Bit 12 in the CL560 DMA Request Interrupt Mask register is reserved, and should always be set to 0.*

- **Bit 12, Buser:** This field enables the Bus Error Interrupt.
- **Bit 11, Mark:** This field enables the Marker Code Interrupt.
- **Bit 10, Vsyn:** This field enables the Vertical Sync Interrupt.
- **Bit 9, Vnac:** In slave mode, this field enables the Vertical Inactive Interrupt.

*Note: Bit 8 in the CL560 DMA Request Interrupt Mask register is reserved, and should always be set to 0.*

- **Bit 8, fend:** In slave mode, this field enables the FRMEND Interrupt.
- **Bit 7, fif0E:** This field enables the FIFO Empty Interrupt.
- **Bit 6, fi1q:** This field enables the FIFO One-quarter Full Interrupt.
- **Bit 5, fi1h:** This field enables the FIFO One-half Full Interrupt.
- **Bit 4, fi3q:** This field enables the FIFO Three-quarters Full Interrupt.
- **Bit 3, fin1q:** This field enables the FIFO Not One-quarter

Full Interrupt.

- **Bit 2, fin1h:** This field enables the FIFO Not One-half Full Interrupt.
- **Bit 1, fin3q:** This field enables the FIFO Not Three-quarters Full Interrupt.
- **Bit 0, Late:** This interrupt enables the Data Late Interrupt.

**IRQ2 Interrupt Mask Register (CL560)**

**0x9028**

<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
fi0nf	fi0ne	Res		Mark	Vsyn	Vnac	fend	fi0e	fi1q	fi1h	fi3q	fin1q	fin1h	fin3q	Late

The Interrupt Mask register defines the interrupt mask for the **IRQ2** output pin. The condition of the output pin is determined by the equation:

$$\begin{aligned} \overline{\text{IRQ2}} = & ((\text{flag}[0] \cdot \text{mask}[0]) \\ & + (\text{flag}[1] \cdot \text{mask}[1]) \\ & + (\text{flag}[2] \cdot \text{mask}[2]) \\ & + (\text{flag}[3] \cdot \text{mask}[3]) \\ & + (\text{flag}[4] \cdot \text{mask}[4]) \\ & + (\text{flag}[5] \cdot \text{mask}[5]) \\ & + (\text{flag}[6] \cdot \text{mask}[6]) \\ & + (\text{flag}[7] \cdot \text{mask}[7]) \\ & + (\text{flag}[8] \cdot \text{mask}[8]) \\ & + (\text{flag}[9] \cdot \text{mask}[9]) \\ & + (\text{flag}[10] \cdot \text{mask}[10]) \\ & + (\text{flag}[11] \cdot \text{mask}[11]) \\ & + (\text{flag}[14] \cdot \text{mask}[14]) \\ & + (\text{flag}[15] \cdot \text{mask}[15])) \end{aligned}$$

To mask (disable) an interrupt, that interrupts mask bit must be set to 0. If an interrupt occurs and its corresponding interrupt mask bit is set to 1, the part asserts the **NMRQ** bit. Asserting the **RESET** input of the S-Reset Flag register sets all fields in the Interrupt Mask register to 0. The function of each bit is defined as follows:

- **Bit 15, fi0nf:** This field enables the FIFO Not Full Interrupt.
- **Bit 14, fi0ne:** This field enables the FIFO Not Empty Inter-

- rupt.
- **Bit 13, Res:** This bit is reserved and should always be set to 0.
  - **Bit 12, Res:** This bit is reserved and should always be set to 0.
  - **Bit 11, Mark:** This field enables the Marker Code Interrupt.
  - **Bit 10, Vsyn:** This field enables the Vertical Sync Interrupt.
  - **Bit 9, Vnac:** In slave mode, this field enables the Vertical In-active Interrupt.
  - **Bit 8, fend:** In slave mode, this field enables the FRMEND Interrupt.
  - **Bit 7, fi0E:** This field enables the FIFO Empty Interrupt.
  - **Bit 6, fi1q:** This field enables the FIFO One-quarter Full Inter-rupt.
  - **Bit 5, fi1h:** This field enables the FIFO One-half Full Inter-rupt.
  - **Bit 4, fi3q:** This field enables the FIFO Three-quarters Full Interrupt.
  - **Bit 3, fin1q:** This field enables the FIFO Not One-quarter Full Interrupt.
  - **Bit 2, fin1h:** This field enables the FIFO Not One-half Full Interrupt.
  - **Bit 1, fin3q:** This field enables the FIFO Not Three-quarters Full Interrupt.
  - **Bit 0, Late:** This interrupt enables the Data Late Interrupt.

**FRMEND Enable Register (CL560)**

**0x902C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Res</i>													EOI@HBI	EOI@Coder	RST@HBI

- **Bit 0, RST@HBI:** When this bit is set to 1, the CL560 as-serts FRMEND when the RST marker passes to the host bus interface from the internal FIFO.
- **Bit 1, EOI@Coder:** When this bit is set to 1, the CL560 as-serts FRMEND when the coder generated EOI and passes it

into the internal FIFO.

- **Bit 2, EOI@HBI:** When this bit is set to one, the CL560 asserts FRMEND when the EOI passes to the host bus interface from the internal FIFO.

This register is set to zero at reset. When asserted (Low), the FRMEND output register remains asserted until the host processor does one of the following:

- Reads either Compressed Word Count Register
- Writes zero to the FRMEND Enable register
- Resets the CL560

Note: Operation of FRMEND requires that the EOF (End of Frame Enable, Bit 1 of the Configuration Register, page 28) bit be set to 1.

#### CODEC Register (CL550)

0x0000 - 0x7FFC

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 CODEC register and the Huffman decoder 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 take place. During compression operations, 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 decoder 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 part. Writes to the registers (other than the CODEC register) are prohibited while the CodecNB flag is not active (equal to zero). Writes to the 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*

## Compression and Decompression Registers

*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 fulfill this requirement. This only applies to the CL550.*

### **CODEC FIFO (CL560)**

**0x0000 - 0x7FFC**

The CODEC FIFO is a 128 x 32 FIFO used to buffer data between the Host interface and the Huffman CODEC.

### **FIFO Level Register (CL560)**

**0xDA04**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIFO Level Control															

The value in this register is the number of 32-bit words currently in the FIFO.

# 8 System Designer's Guide

This chapter is intended to be used by both hardware and software designers. It includes an overview of general concepts relating to JPEG hardware systems, system models for the CL550 and CL560, and programming guidelines for both still-image and video system designs. The programming guidelines include initialization of the device, operation of the device, and Huffman and Quantizer table programming.

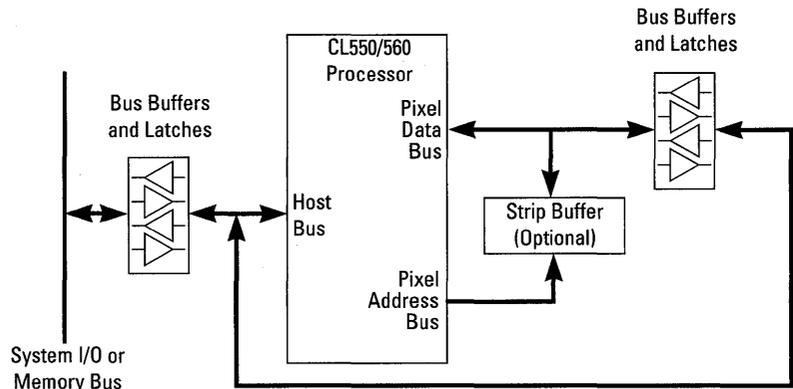
There are two-basic system configurations for CL5xx family devices. In the first basic configuration, the CL5xx is used as a simple JPEG co-processor (Figure 8-1). In this model both the Host data bus and the Pixel data bus are connected to the system memory or I/O bus. Pixel data and compressed data are both cycled through the CL5xx device using the system bus. This type of model is useful in systems that require hardware acceleration of JPEG image compression or decompression. Either programmed I/O or DMA-assisted transfer techniques can be used to move the data into and out of the CL5xx device. In the copro-

---

## 8.1 Typical System Configurations

---

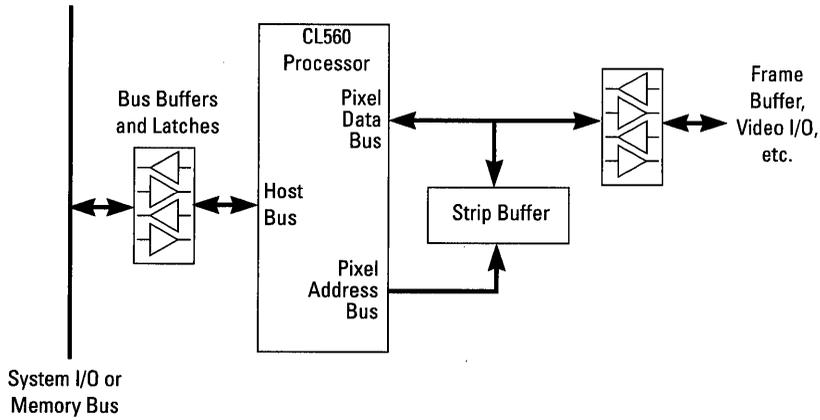
processor configuration the system host only passes data, and is relieved of the intensive calculations necessary in the JPEG image compression/de-compression process.



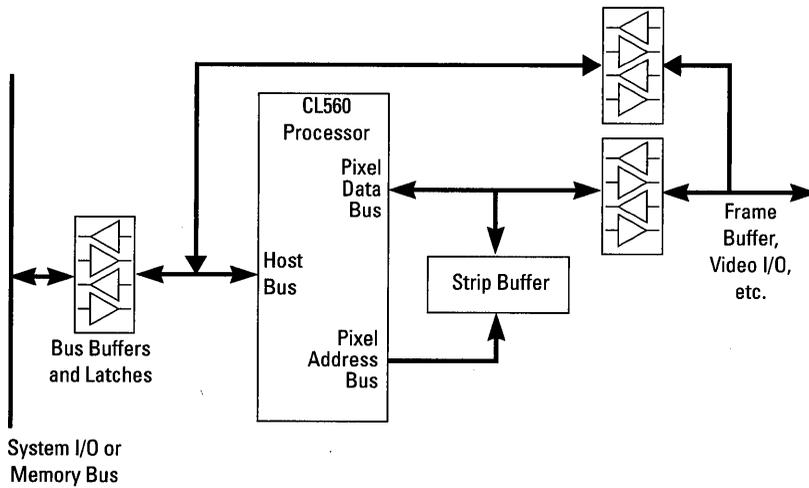
**Figure 8-1 Coprocessor Configuration**

The second basic system configuration is the I/O peripheral configuration (Figure 8-2). This configuration is used in all JPEG video system designs as well as a few high-speed still-image devices such as printers, scanners and digital copiers. In this model, only the CL5xx's Host data bus is connected to the system bus. The Pixel data bus is connected directly to a video capture device or graphics frame buffer. Uncompressed pixel data never crosses the system bus.

Systems can also be designed that combine both coprocessor and I/O peripheral functions by simply making the pixel I/O port accessible from the system bus in addition to the video data path (Figure 8-3).



**Figure 8-2 I/O Peripheral Configuration**



**Figure 8-3 Combined I/O Peripheral and Coprocessor Configuration**

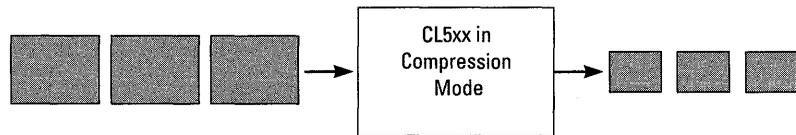
---

## 8.2 JPEG Video Concepts

---

JPEG video is simply a sequence of JPEG-compressed still-images. When these images are compressed or decompressed at video frame rates, it becomes JPEG video.

The CL5xx devices support two distinctly different methods of JPEG video data organization. One obvious way to organize JPEG video data is in a frame-by-frame structure as shown in Figure 8-4. Each compressed field or frame in this structure exists as an individual data item. The frame-by-frame approach is used in both Apple's QuickTime™ system software and Microsoft's Video-for-Windows™ system software.

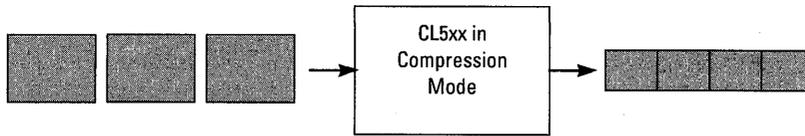


**Figure 8-4 JPEG Video Data in Frame-by-Frame Organization**

In the frame-by-frame video model, each image is compressed or decompressed in real-time, but the CL5xx device has all of the remaining data flushed out and the device is reset at the end of each frame of compression or decompression. The drawback in this approach is that there is a small performance penalty in resetting the CL5xx device between frames because some of the CL5xx device registers must be re-programmed following a device reset. Balancing this is the ability to randomly access individual frames for playback in any order desired. It is also easier to synchronize audio and video using this structure.

The frame-by-frame approach lends itself well to operation in half-duplex mode. In half-duplex mode, an incoming frame is decompressed, the device reset, then an outgoing frame is compressed, and so on. This mode is used in real-time video transmission systems such as tele-conferencing systems.

The CL5xx devices also support the continuous stream method of handling the compressed data. In this model, the CL5xx is not reset between frames and its processing pipeline is never flushed. The device produces a continuous stream of compressed data, with all compressed video frames concatenated together into a single large data stream as opposed to multiple items (Figure 8-5).



**Figure 8-5 JPEG Video Data Organized as a Continuous-Stream**

The advantage of this approach is higher system performance because little or no software intervention is required between frames. This method is preferred by designers of high-end video systems, where video quality is the primary goal. The disadvantages to this approach are that it is more difficult to address random image frames and to maintain audio-video synchronization.

This section gives a detailed description of the procedures used to compress images and video data using the CL550 processor. A general overview of CL550 compression operation is presented, followed by a discussion of the basic compression system concepts for each of the two system configurations (Figure 8-1 and Figure 8-2). For specific details on programming the registers or tables in the CL5xx part, refer to Chapter 7 and Sections 8.7 through 8.11 of this chapter.

---

## 8.3 CL550 Compression Operation

---

### 8.3.1 Overview

During the compression of a frame of image data, the CL550 goes through three basic processing stages. These basic stages are the same whether the system is configured as a still-image coprocessor or as a video compression pipeline. They are:

- SRAM/Pipeline Prefill
- Compression
- SRAM/Pipeline/FIFO Drain

Each of the three processing stages is performed by logic within the Video Bus Interface (see Chapter 5). The paragraphs that follow describe the basic process of compression of a single frame of video using the CL550.

Following chip initialization and start-up (refer to Section 8.7), the CL550 prefills its compression pipeline with data. The CL550 indicates the start of a compression cycle by pulling the  $\overline{\text{VSYNC}}$  output pin LOW. The state of  $\overline{\text{VSYNC}}$  can also be determined by polling the  $\overline{\text{VSYNC}}$  flag in the Flags register. When the active region of the image is reached, the CL550 starts to write the incoming pixel data into the SRAM Strip buffer. Once eight complete lines of pixel data have been accumulated in the SRAM, the CL550 starts to read the first pixels into the compression pipeline.

The compression pipeline has over 320 processing stages, and it takes at least 320 PXCLK cycles for the first data to appear at the FIFO. The Huffman coder starts to take data from the FIFO when the FIFO level reaches 1/4 full. At that point, the CodecNB flag goes to zero. When CodecNB goes back to 1, the first word of compressed data is available in the CODEC register.

As soon as data is available in the FIFO, the process of retrieving compressed data can begin. The host processor can read the compressed data using either programmed I/O transfers or DMA transfers.

The FIFO level must be kept between 1/4 full and 3/4 full for proper operation. Although it is easy to keep the FIFO level above 1/4 full, it is impossible to keep the FIFO level below 3/4 full without hardware support. Often during the compression process, the Huffman coder unit cannot draw data from the FIFO faster than the compression pipeline is filling the FIFO. At other times, other system peripherals may have control of the system bus, preventing the host from servicing the CL550. In either case, an overrun will result unless the pixel flow is regulated using the  $\overline{\text{STALL}}$  mechanism. The CL550 provides two signals to determine the level of the FIFO:  $\overline{\text{NMRQ}}$  and HALF\_FULL.  $\overline{\text{NMRQ}}$  is programmed using the  $\overline{\text{NMRQ}}$  Interrupt Mask register (Chapter 7). The HALF\_FULL signal is fixed to the half-full status flag from the FIFO.

**Example:** If the  $\overline{\text{NMRQ}}$  signal is used to generate  $\overline{\text{STALL}}$ , then the  $\overline{\text{NMRQ}}$  Interrupt Mask register should be programmed (at start-up time) to interrupt when the FIFO reaches 3/4 full. This way, whenever the FIFO level reaches 3/4 full,  $\overline{\text{STALL}}$  is asserted, the video bus will halt, and the FIFO can be drained without the risk of overrun.

Once all the pixels have been written into the SRAM (the end of the im-

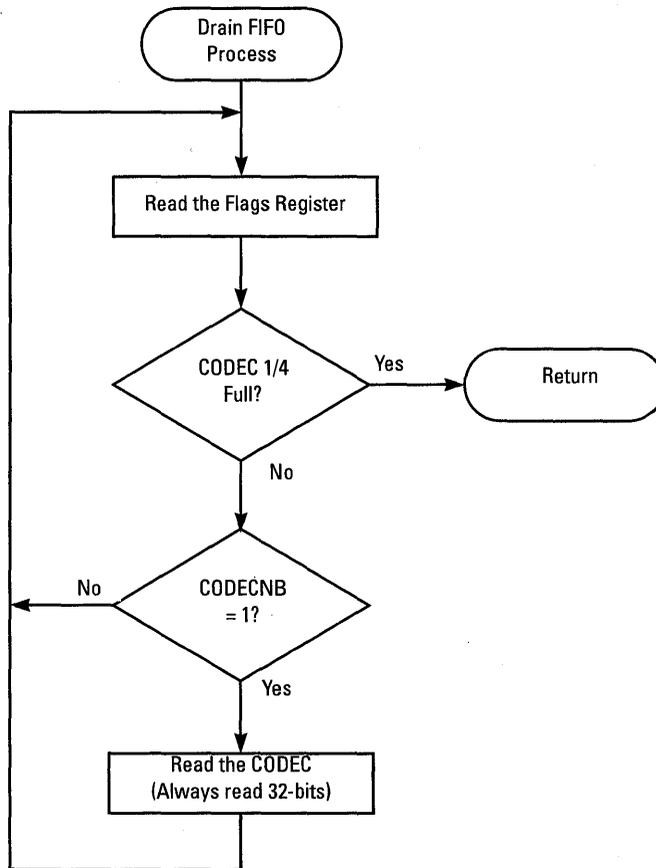
age frame), the process of draining the pipeline can take place. During this period, the CL550 reads any pixels remaining in the SRAM strip buffer (8 lines worth), followed by all of the data remaining in the compression pipeline. When the last word of data has been written to the FIFO, the CL550 sets the VNAC flag in the Flags register to 1. The user may then drain the FIFO until it is empty. When all 1's data appears at the CODEC register (0xFFFFFFFF), the compression process is complete.

A simple flowchart example of a program loop for draining the CL550 FIFO is shown in Figure 8-6. The *CL550/CL560 Program Examples Disk* also contains a code example in the file *550comp.c*. In this example, the FIFO level is taken down to 1/4 full. A check of the CodecNB bit prevents long wait states in case the coder is still busy when the CODEC is read. Figure 8-7 shows a flow chart for clearing the CL550 FIFO at the end of an image frame. The FIFO should never be cleared until VNAC bit in the flags register has been set.

### 8.3.2 CL550 Operation as a Still-image Coprocessor

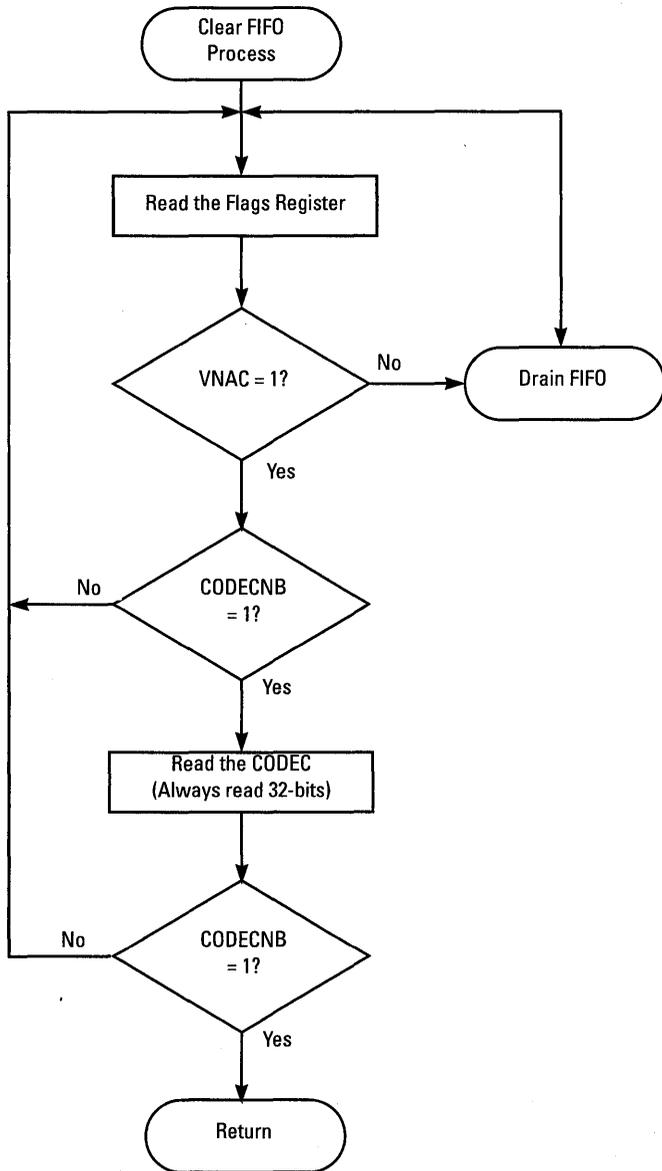
When the CL550 is used as a still-image coprocessor (Figure 8-1) the pixel interface logic must control **STALL** on a pixel-by-pixel basis. Since the CL550's video bus is connected back to the system bus, **STALL** is used to halt the CL550 until the program is ready to provide the pixel data. Each time the program writes a pixel, **STALL** is released for exactly one pixel read cycle. In this way, the FIFO level is regulated by simply not writing pixel data to the CL550. Once all of the pixels have been written, however, **NMRQ** must be used to prevent overflow, because there will be no more activity on the video bus that can be used to generate **STALL**.

The CL550s FIFO is 128 entries deep, and between the 1/4 and 3/4 marks, there are exactly 64 entries. At the maximum rate, the CL550 will need 64 pixel **PXCLK** cycles (with no **STALL** present) to move the FIFO pointer from 1/4 full to 3/4 full. From the video bus point of view if the FIFO level is presently at 1/4 full then up to 64 bytes of pixel data can be written without draining the FIFO. The highest level the FIFO will reach in this case is 3/4 full. This means that for gray-scale images, 64 pixels can be written at one time, for YUV422, RGB-to-YUV422, YUV444-to-YUV422 modes up to 32 pixel can be written at one time,



**Figure 8-6** Flow Chart for the CL550 FIFO Drain Loop

and for 444 and 4444 modes 16 pixels at a time can be written. After each group of pixels is written the host must drain the FIFO back to 1/4 full. This sequence of writing uncompressed pixels to the video bus and then reading compressed data from the FIFO continues until all the pixels have been written. At that time, the host can completely flush the compression pipeline.



**Figure 8-7** Flow Chart for Flushing the CL550 FIFO

### 8.3.3 Operation as a Video Compression Processor

Figure 8-8 illustrates a typical video compression/decompression system using the CL550 processor. During compression the signal of most significance to the system designer is the  $\overline{\text{PXIN}}$  signal. Each time that the CL550 asserts  $\overline{\text{PXI}}$ , it expects to see valid pixel data on the  $\overline{\text{PXDAT}}$  bus. If pixel data is not available, then  $\overline{\text{STALL}}$  must be asserted on that cycle or the CL550 will latch invalid data into the SRAM.

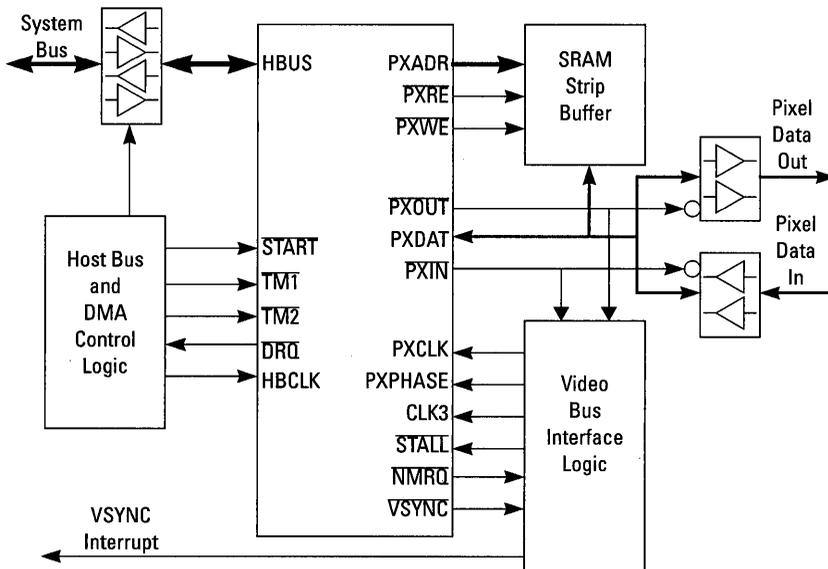
The use of  $\overline{\text{STALL}}$  is required in CL550 based systems, and has three basic uses:

1. To prevent FIFO overrun. The  $\overline{\text{NMRQ}}$  FIFO status signal can be programmed to activate when the FIFO reaches 3/4 Full. This output can be used to generate the  $\overline{\text{STALL}}$  signal, preventing FIFO overflow.
2. To hold the CL550's pixel bus until a pixel becomes available on the bus. A slow capture device can generate an external stall request to halt the CL550 in the event that no pixels are available.
3. To hold off the CL550 between video frames, in order to maintain synchronization with the incoming video stream. By comparing the states of the CL550's  $\overline{\text{VSYNC}}$  output and the  $\overline{\text{VSYNC}}$  signal provided by the capture device,  $\overline{\text{STALL}}$  can be regulated on frame-by-frame basis to control frame rate relative to the incoming  $\overline{\text{VSYNC}}$ . Refer to Chapter 5 for specifics on the CL550's video bus operation.

The signals  $\overline{\text{START}}$ ,  $\overline{\text{TM1}}$ , and  $\overline{\text{TM2}}$  are used to control transfers across the host bus. The use of  $\overline{\text{DRQ}}$  is optional, but it can be used in conjunction with  $\overline{\text{HB15}}$  to control DMA transfers over the host bus (see Chapter 4 for more information).

The software driver operation for the video system is somewhat simpler than the still-image driver. The software does not need to manage pixel transfers. Instead, the driver devotes its attention to handling only compressed data. The CL550's DMA support logic can be used to automate the transfer process.

During frame-by-frame compression operation the CL550 must be flushed, reset, initialized, and re-started every frame in the sequence



**Figure 8-8 Typical CL550 Video Compression System**

(see Section 8.7). These tasks can be performed within an interrupt service routine (described in the next section). Driver operation is simpler during continuous video operation, because the CL550 only needs to be drained after the last frame of the sequence. For all other frames, the host must simply ensure that the FIFO level is maintained between 1/4 and 3/4 full.

During continuous streaming the Coder Coding Interval register can be programmed so that a RST code (0xFFD0-0xFFD7) is produced at the end of each compressed image. The RST code can then be used as a delimiter between the CL550's compressed frames.

**Example:** If the compressed image frame is 320 x 240 pixels and the data is YUV422 (alternatively this can be RGB-to-YUV422), then the size of the Minimum Coded Unit (MCU) is 16x8 pixels, and there are 600 MCU's in the image frame. If the Coder Coding Interval is set to 600, then a RST marker will be inserted at the end of each compressed frame.

Other system-level issues that need to be dealt with are maintaining disk throughput and audio/video synchronization.

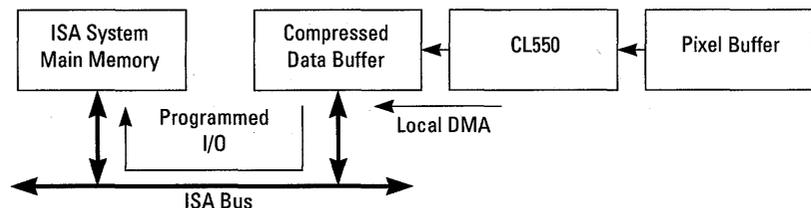
### 8.3.4 Use of DMA and Interrupts with the CL550

In a typical video/multimedia hardware system, the system must be designed to manage data flow as efficiently as possible. For an ISA-based video compression board, compressed data must first be unloaded from the CL550 device, transferred via the ISA bus to main memory, then sent to the disk. Sustained compressed data rates on an ISA-based system may go as high as 500 Kbytes/second. This leaves little time for the CPU to service the compression hardware and address system software needs at the same time. The CL550's  $\overline{\text{DRQ}}$  output provides a way to use hardware to unload the CODEC register, leaving time for the host processor to handle other system-level tasks. See Chapter 4 for hardware considerations related to using the CL550's  $\overline{\text{DRQ}}$  signal.

There are two basic ways in which the CL550's  $\overline{\text{DRQ}}$  output can be used:

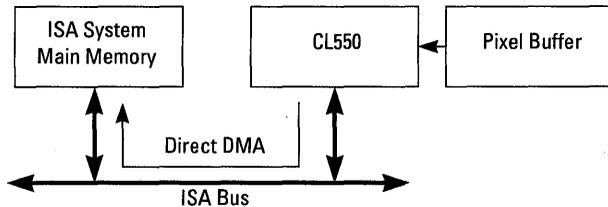
- To transfer compressed data from CL550 to on-board memory. The CPU unloads the on-board memory using a string-move instruction (programmed I/O).
- To transfer compressed data from CL550 directly to main memory using the system's DMA resources.

Programmed I/O (Figure 8-9) provides the fastest possible data transfer across the ISA bus. During programmed I/O the compressed video data is written to a Compressed Data buffer using local DMA. The host then moves the data into system memory using string-move instructions. Programmed I/O is more expensive than DMA because (roughly) 10 to 20 Kbytes of buffering must be provided to allow for the interrupt latency of the system.



**Figure 8-9 Programmed I/O System Architecture**

DMA transfers (Figure 8-10) can be used to move data directly to main memory at a much lower cost. The trade-off is a lack of compatibility with the full range of ISA-based systems. Some older ISA motherboards do not support 16-bit DMA transfers. 8-bit DMA transfers can be used, but the maximum sustainable rate for compressed data will be less than half of what it would be for a 16-bit system.



**Figure 8-10 Direct DMA Architecture**

During compression, the CL550's  $\overline{DRQ}$  output is used as follows:

1. At start-up, the DRQ Mask register is programmed by setting bit 13 (CodecNB) and bit 6 (Fi1q) HIGH.
2. The CL550 will assert  $\overline{DRQ}$  when the FIFO fills to 1/4 full and the coder finishes the first code word. The user can avoid using wait states by using the CodecNB bit in the mask register. Whenever  $\overline{DRQ}$  is asserted, there is a word available in the CODEC register.
3.  $\overline{DRQ}$  activity continues until the end of the image frame or until the last frame of a sequence of frames in the continuous video mode. When the compression pipeline is completely emptied and the FIFO is flushed down to 1/4 full, the CL550 stops activity on  $\overline{DRQ}$ . Note that when this happens, there is still data on-chip that must be drained.
4. The host should use a program loop to remove the remaining data from the CODEC register (Figure 8-7). Data should be read until the word 0xFFFFFFFF is encountered. The program loop can be entered as part of an end-of-image interrupt service routine, as described below.

### 8.3.5 DMA System Programming Example

In a DMA-driven system interrupts are used to keep track of the frames coming in and provide any hardware service necessary between frames. Figure 8-11 shows an example interrupt timing for a 30 field per second compression operation. The circled numbers in the diagram refer to the steps below the diagram.

The VSIN input signal is generated by the NTSC/PAL decoder (or the device that digitizes the incoming video). FIELD is a toggle used to indicate whether the field is even or odd and is used to qualify the start-of-frame interrupts on odd fields only.  $\overline{550VS}$  corresponds to the CL550's VSYNC output (it is re-named for clarity).

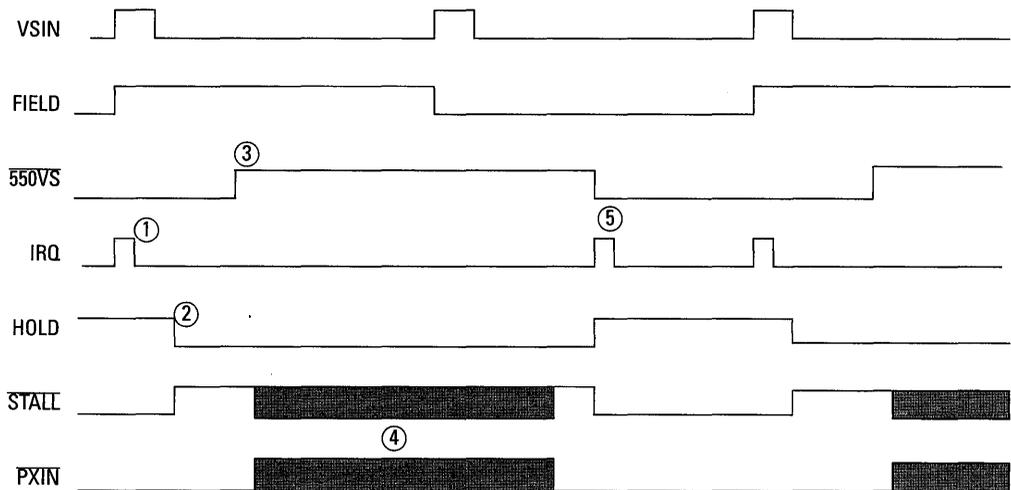
IRQ is generated by either the VSIN signal (start-of-frame) or by the  $\overline{550VS}$  signal (end-of-frame). Software must monitor the state of the two sync signals to determine which interrupt is being given.

HOLD is an external logic term used to stall the CL550 until the field is available. HOLD is cleared by software in response to the start-of frame interrupt. If NMRQ is used to generate STALL, it can be programmed to activate on the VSYNC flag. This will stall the CL550 until the NMRQ mask is reprogrammed to release it.

In this example, the CL550 is initialized and started. When  $\overline{550VS}$  asserts, HOLD is generated. This stalls the CL550 until pixels become available in the pixel buffer. When the start-of-frame interrupt indicates that pixels are available, the interrupt service routine clears HOLD and compression of the frame begins. At the end of the frame the  $\overline{550VS}$  signal activates again, causing another stall. At this time the interrupt server can either clear the FIFO and reset the CL550 (frame-by-frame operation) or allow it to continue without flushing, while performing any other system functions needed at that time (continuous video operation).

Interrupts should alternate between start-of-frame and end-of-frame. If two start-of-frame interrupts occur back-to-back, frame synchronization may be lost. The current compressed frame may have to be discarded in an effort to re-synchronize to the incoming video. Frames can be missed for a number of reasons, but it generally occurs because the compressed data rate is too high for the host to accept in real time (e.g. disk I/O rate, software performance, bus masters taking the bus, etc.). In

response to a frame loss, the program might try to increase the quantization factors to achieve higher compression and a lower compressed data rate.



**Figure 8-11 Interrupt Timing for 30 Frame/Second Compression**

1. VSIN from the Video Interface logic causes the first interrupt. This interrupt indicates the start of an odd field.
2. The interrupt service routine clears the HOLD signal. HOLD is activated whenever the CL550 asserts its  $\overline{\text{VSYNC}}$  output (550VS). Deactivation of HOLD releases the CL550's STALL input. If desired, HOLD can also be the NMRQ signal, programmed to activate on the VSYNC flag.
3. The CL550 begins compressing data from the pixel buffer. 550VS will deassert after its programmed delay.
4. The CL550 starts activity on  $\overline{\text{PXIN}}$ . STALL is asserted as necessary to prevent the FIFO from overflowing.
5. The CL550 asserts 550VS after the frame has been captured. This is used to generate an end-of-frame or "frame done" interrupt. At this time, the host processor can clear the CL550's FIFO and reset the device.

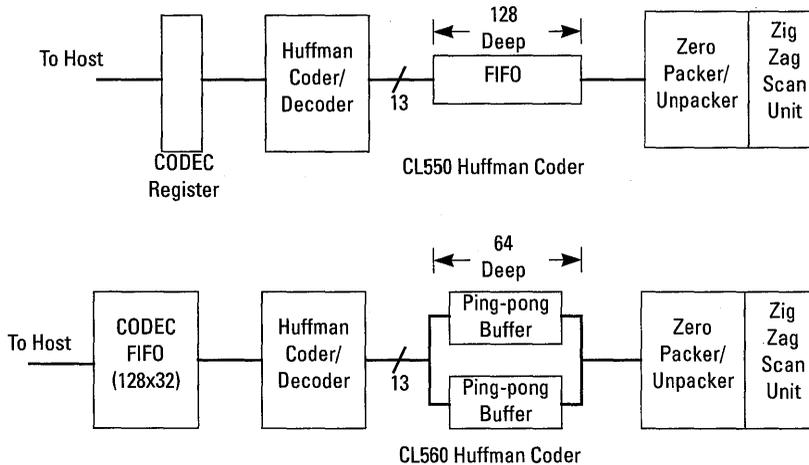
**8.4  
CL560 Compression  
Operation**

The CL560 compresses images and video in much the same way as the CL550. Read section 8.3 before reading this section. In this section, we will describe the enhancements and improvements that the CL560 incorporates to support more demanding applications. A detailed description of the compression operation is given, followed by system descriptions of typical systems that use the CL560 for both still-image and video.

**8.4.1 CL560 Enhancements**

The major differences between the CL550 and CL560 are illustrated in Figure 8-12. The CL560 is different from the CL550 primarily in the areas of the Huffman Coder/Decoder Unit and the location of the host FIFO with respect to the host bus. From the Zero Packer/Unpacker to the Video Bus Interface the CL550 and CL560 are identical.

The CL560 has sufficient performance to pass CCIR601 video at compression ratios as low as 1:1 in real time, either synchronously or asynchronously. The new FIFO arrangement makes programming the device much simpler, as will be seen.



**Figure 8-12 CL550 and CL560 Architecture Differences**

The main enhancement in the CL560 is its single-cycle-per-word

Huffman Coder/Decoder. The CL560 Huffman Coder generates a code in a single PXCLK cycle, where the CL550 takes a number of cycles to code each FIFO entry. The compression pipeline operates at the same rate which allows the Ping-Pong Buffer and Huffman Coder/Decoder run in lock-step with the video unit. All of the CL560's compression elements are locked together as a synchronous pipeline, giving it the ability to operate without stalling. This is a fundamental improvement over the CL550, which can only operate in an asynchronous mode using  $\overline{\text{STALL}}$ . The CL560 can be stalled if desired, but a  $\overline{\text{STALL}}$  halts the Huffman Coder/Decoder in addition to the rest of the pipeline.

The CL560's improved Huffman coder automatically pads the tail of every compressed video frame to a 32-bit boundary as opposed to the 8-bit boundary required by JPEG. This allows each compressed video frame to be aligned to a 32-bit boundary, which greatly simplifies DMA system operation without violating the JPEG standard.

DMA support in the CL560 has also been significantly improved. The CL560 supports a two-clock-per-word transfer mode that is capable of passing data to the host at very high data rates (see Chapter 4), with 96 or more words per burst.

**Example:** Assume that the  $\overline{\text{DRQ}}$  pin is programmed to activate at the 1/2 full mark. In response to the active  $\overline{\text{DRQ}}$  indicator, 64 words can be taken from the FIFO in a minimum of 128 HBCLK cycles. There is no need to monitor  $\overline{\text{DRQ}}$  each time that a word transferred as is the case with the CL550. Further, the CL560's FIFO can be completely flushed by programming the  $\overline{\text{DRQ}}$  output to the "not empty" condition, allowing for full hardware control of the compressed data flow.

The function of the  $\overline{\text{FRMEND}}$  signal has also been modified significantly. In the CL550,  $\overline{\text{FRMEND}}$  is typically not used, because the FIFO must be empty before it is asserted. In DMA-driven systems, this will never happen because the FIFO must be kept over 1/4 full. In the CL560, however, the  $\overline{\text{FRMEND}}$  signal

can be used as an end-of-image interrupt output. The CL560's **FRMEND** output is programmable (See Chapter 7), and can be activated at the end of the compression of a frame of data either at the instant the Huffman coder places the last word of compressed data into the FIFO, or as the last word of the frame is passed out of the FIFO onto the host bus. For DMA-assisted systems, the CL560 compression operation is as simple as enabling DMA, starting the CL560, and getting an interrupt when the operation is complete.

### **8.4.2 CL560 Operational Differences from the CL550**

The CL560 operates in roughly the same fashion as the CL550. Compression takes place in three stages as before: pre-fill, compression, and flush. The significant differences are in the handling of the compressed data. These differences are described below.

During compression in the CL560, when the pipeline begins to prefill, the Zero Packer places the first block's worth of coded symbols into one of the Ping-Pong buffers. The Ping-Pong buffers have the same data format as the CL550's FIFO (128 words by 13 bits). The CL560 uses two FIFOs instead, arranged as 64 words by 13 bits. These buffers swap every 64 clocks.

The Huffman Coder removes data for coding from the Ping-Pong Buffers, decodes it, and places the decoded data into the 128 word by 32 bit CODEC FIFO. This new, larger, FIFO stores data after it has passed through the Huffman compression stage, making it (effectively) five to ten times larger than the CL550's FIFO.

Another advantage of the CL560's FIFO is that more of it can be used. During CL550 compression, the FIFO has to be kept between 1/4 and 3/4 full, giving 64 words of usable FIFO depth. In contrast, during CL560 compression, the FIFO can operate between 3/4 full and empty. The CL560's FIFO level is deterministic: if the 3/4 full flag is active then exactly 96 words can be read without danger of corrupting the data stream. The CL550, on the other hand, will drain an unknown number of FIFO entries each

time that the CODEC register is read. This require status to be checked each CODEC register read.

Figure 8-13 shows a simple loop for draining the CL560's FIFO. In this example, the program tests for the Half-full condition in the FIFO. (The threshold level may also be set to 1/4 or 3/4 if desired.) If the FIFO is not half full, then the FIFO does not require service. When the Half-full flag appears up to 64 words can be taken at one time. At the end of the image (as indicated by either VNAC or FRMEND flags) the FIFO can be cleared by reading the FIFO level register and pulling out exactly that number of entries.

The Compressed Word Count register indicates the exact number of 32-bit words that were compressed during the frame. The Compressed Word Count is valid until the end of the next frame, when it is updated with that frame's word count.

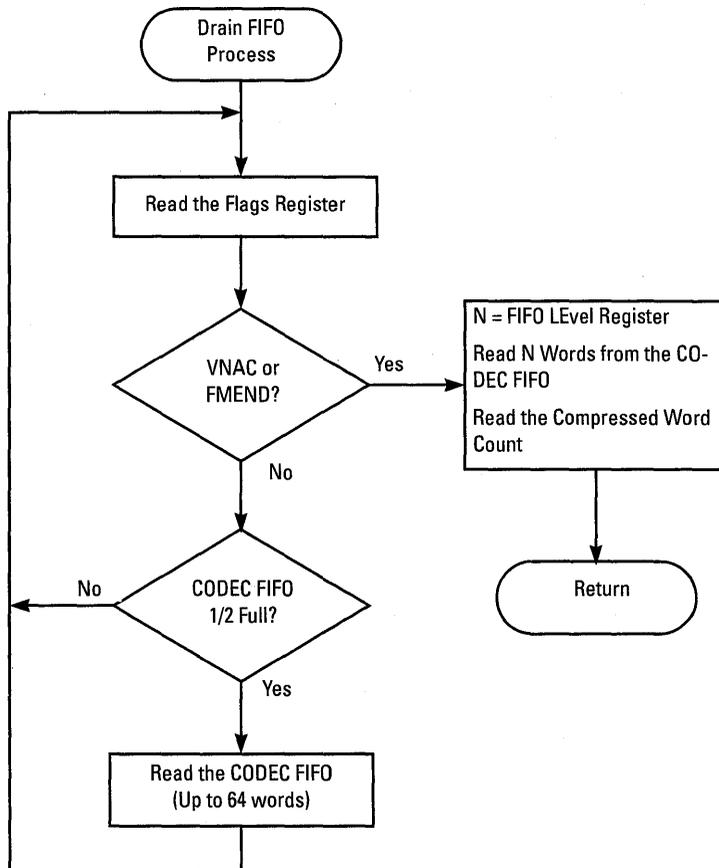
### 8.4.3 Compressed Data Rate Control Mechanisms

The CL560 provides a variety of new mechanisms devoted to data rate control of the CODEC FIFO. There are three ways that the CL560 implements rate control of the compressed data.

The first method of rate control uses the **STALL** input. This method is exactly the same as the one used for the CL550. If the FIFO level is 3/4 full, that condition can be sent out on either the **IRQ1** or **IRQ2** signals and be used to generate **STALL**. **STALL** in turn, halts the compression pipeline, preventing in-flow to the FIFO.

In a system that does not use **STALL** (such as a system with a synchronous, unbuffered video input) an alternative rate control mechanism is provided; the Coder Rate Control Enable register. When the FIFO level programmed into the Coder Rate Control Enable register is reached, the CL560's Huffman Coder automatically trims off all of the remaining AC terms in the image frame. This is highly destructive to the image data, but it slows the rate of FIFO input dramatically. By using the Coder Rate Control Enable register, a slow host might be able to service the device in time to prevent overrun and total corruption of the JPEG stream.

Bit 0 of the Coder Rate Control Active register will read back as a one



**Figure 8-13 Flow Chart for the CL560 FIFO Drain Loop**

when the rate control mechanism is triggered. A write to this register (any data) clears the CRCA flag. Even after rate control is activated, the FIFO will still overflow if it is not serviced. This rate control mechanism is intended to be used as a "graceful failure" option for the system designer.

A third area where the CL560 may trim data is in the area of the Ping-Pong buffers. During a compression operation, the Huffman Coder has exactly 64 clock cycles to completely code the data in one of the Ping

Pong buffers before they switch over. It is possible that the Huffman coder could not code all 64 entries in the 64 clocks allotted. For example, in a worst-case scenario, if the Zero Packer produces a block with 63 significant AC terms, and each of those terms require a 16-bit Huffman Code with zero stuffing and RST code added, the CL560's coder would not be able to generate all the extra codes in the 64 clocks allotted. In this case the Huffman coder automatically stuffs an EOB code into the stream on the 64th clock. Any remaining terms (perhaps the last one or two AC terms of a block) in the Ping Pong Buffer will be lost. When this happens the CL560 sets bit 0 of the Coder Robustness Active register to a one. This tells the programmer that one or more blocks in the image frame had AC terms trimmed off to meet the single-cycle-per-word rate requirement. This condition is not speed-dependent, and may occur in still image systems as well as video systems. This case is extremely rare. When this condition occurs, the image data is being expanded up to 4X instead of being compressed. Needless to say, most applications use enough quantization to never see this condition.

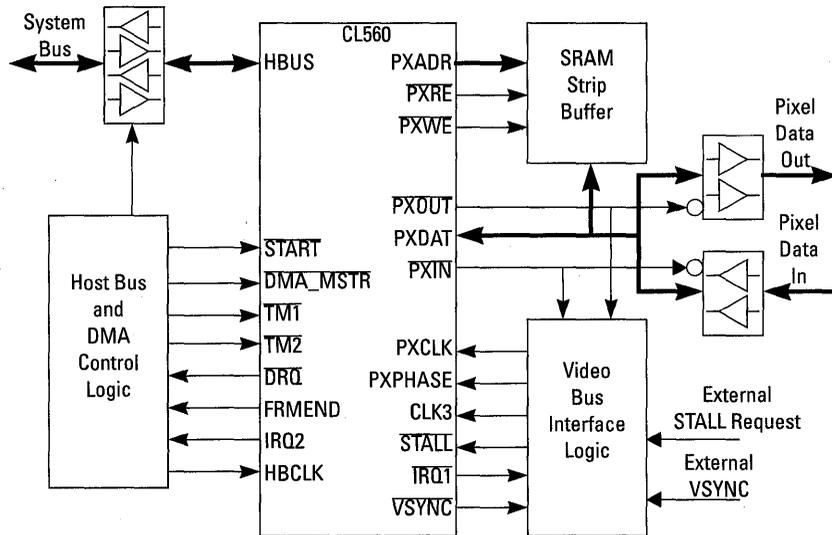
#### 8.4.4 Typical Compression System Using the CL560

The CL560 is useful in a wide range of high-performance still-image and video system designs. Whether the application is for still images or video, the basic system model is the same. A high-speed image scanner passes pixel data at real-time rates much the same way as a video capture device passes its data. Figure 8-14 shows a model of a typical CL560 compression system. Although it is similar to a CL550-based system, the CL560 compression system is faster, more flexible, and easier to program than a CL550 based system. The CL560 can be completely controlled in hardware from the start of the compression to the finish.

In the example shown in Figure 8-14, pixel data to be compressed comes from a capture device such as a scanner or a video digitizer. The logic on the video bus side can be very minimal. The use of  $\overline{STALL}$  is optional, and the only signal of significance to the system designer is the  $\overline{PXIN}$  signal. When the CL560 asserts  $\overline{PXIN}$ , it expects a valid pixel on the bus. If a pixel is not available,  $\overline{STALL}$  must be asserted during that exact clock cycle or the CL560 will latch invalid data into the SRAM.

$\overline{STALL}$  can either be generated by the CL560 or by the video capture device.  $\overline{IRQI}$  can be programmed to be asserted at 3/4 full, and the

$\overline{\text{IRQ1}}$  output can be used to request a stall in order to prevent FIFO over-run. If there are no pixels available, then  $\overline{\text{STALL}}$  can be driven by the video capture device to indicate this condition. The logic should be designed so that  $\overline{\text{STALL}}$  is driven immediately if  $\overline{\text{PXIN}}$  asserts and a pixel is not available. Other signals that are used in the video interface logic but not shown here are  $\overline{\text{VSYNC}}$ ,  $\overline{\text{HSYNC}}$ , and  $\overline{\text{BLANK}}$ .



**Figure 8-14 Typical CL560-based Compression System**

On the host side, the signals  $\overline{\text{START}}$ ,  $\overline{\text{DMA\_MSTR}}$ ,  $\overline{\text{TM1}}$ , and  $\overline{\text{TM2}}$  are used to control host bus transactions.  $\overline{\text{DRQ}}$  is used to indicate FIFO level and  $\overline{\text{FRMEND}}$  is used to indicate the end-of-frame condition. The  $\overline{\text{FRMEND}}$  signal can also be routed internally to the  $\overline{\text{IRQ2}}$  output.

In this system, compression would follow the following steps:

1. At start-up, the user programs the DMA Request Interrupt Mask register to generate a  $\overline{\text{DRQ}}$  on any one of the following flags: Not\_Empty, 1/4 Full, 1/2 Full, or 3/4 Full. CodecNB is not a requirement for FIFO service and the CodecNB bit should never be set.
2. When the FIFO fills to its programmed threshold, then the

CL560 will drive  $\overline{\text{DRQ}}$  LOW. In response to this indication, the host can remove 1, 32, 64, or 96 words, depending on what the threshold is set to.

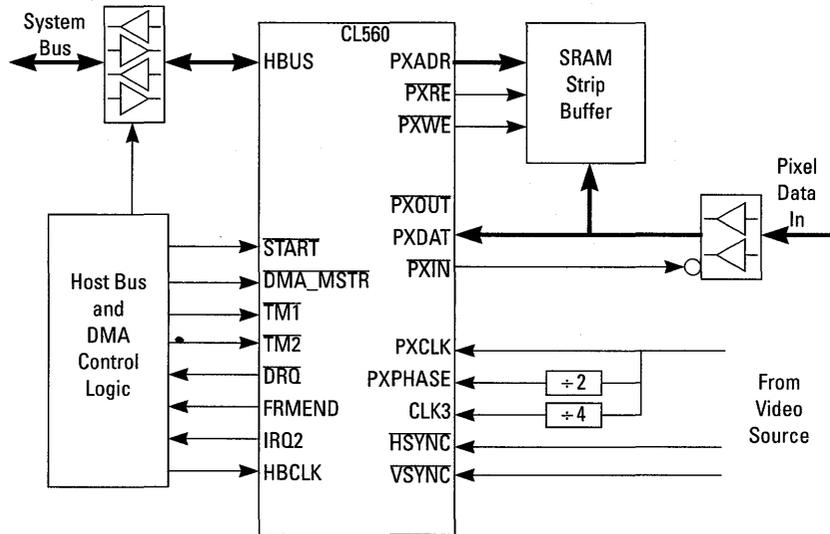
3. At the end of the compression cycle the CL560 flushes the pipeline and Huffman coder, and FIFO input stops. If you are using burst-DMA, you should set the FRMEND Enable register to generate an interrupt on an EOI@Coder condition. When you see a FRMEND interrupt, the FIFO Level register will hold the exact number of words left in the FIFO that are below the burst threshold. Exactly that many words should be read from the FIFO to drain it. If you are using single-word transfers (DRQ on Not\_Empty) you should set FRMEND Enable to either EOI@Host Bus or RST@Host Bus. The FRMEND interrupt will then occur after the last coded word is unloaded.

#### 8.4.5 Slave-mode Compression Operation

The CL560 can run synchronously with input video timing, and can be used in a slave-mode configuration to compress video data directly from a capture device. In slave mode the VSYNC and HSYNC signals function as inputs, and are driven by the video capture device.

The CL550 also has slave-mode capability, but it is typically not useful to the system designer because the CL550 operates asynchronously and must have STALL activated from time to time. The requirement for STALL on the CL550 effectively prevents operation in slave mode, because the VSYNC and HSYNC signals would have to be delayed by exactly the amount of time that STALL is asserted in order for the system to function properly. For the systems designer using the CL550, it is simpler to use the master mode.

Figure 8-14 shows a slave-mode compression system design example using the CL560. In this example, the CL560 is synchronized to the timing of the incoming video. The video capture provides digitized video in YUV4:2:2 format and the timing signals PXCLK, VSYNC, and HSYNC. PXCLK is divided by two to make PXPULSE. CLK3 is not used in YUV4:2:2 mode and is can be tied low.



**Figure 8-15 Typical CL560-based Slave-mode Compression System**

The sequence of events for a slave-mode compression cycle are described below. The host port operates identically to the previous example. Therefore, only the video bus operation is described here.

1. The host processor initializes the CL560 for compression.
2. The host processor writes a 1 to the HVEnable register, followed by a write of 1 to the Start register.
3. The video capture device asserts  $\overline{\text{VSYNC}}$  to indicate the start of an incoming field or frame.
4. The  $\overline{\text{VSYNC}}$  input is negative-edge triggered in slave mode. If the HVEnable and Start registers have been set to one, compression begins with the active  $\overline{\text{VSYNC}}$  edge.
5. To stop the compression at the end of the current frame, write a zero to the Start register after the active  $\overline{\text{VSYNC}}$  edge. In slave mode, the Start register is sampled on every  $\overline{\text{VSYNC}}$  edge. If Start is one, a frame compression starts. If it is zero, the frame compression does not start, and the  $\overline{\text{VSYNC}}$  input is ignored.

6. Once started, the CL560's internal video bus counters begin to operate. Each incoming HSYNC pulse triggers the next line. The CL560's VDelay register is used to count HSYNC pulses until the first active video line is reached.
7. Once the active line is reached, the value in the HDelay register specifies the number of clocks from the active HSYNC edge to the first pixel.
8. At the end of the HDelay interval, the CL560 negates  $\overline{\text{BLANK}}$  (sets it HIGH). It then asserts PXIN to read the first incoming video pixel. The CL560 assumes that valid pixels will be on the bus at that time. It is the responsibility of the system designer to set VDelay and HDelay values that correspond to the input sync timing.

In this section, the CL550 decompression process is described in detail. As there are many similarities to compression operation, you are encouraged to read Section 8.3 on CL550 compression before reading this section.

### 8.5.1 Overview

During the decompression of a frame of image data, the CL550 goes through three basic operations. These steps are the same as in the compression direction, only in reverse. They are:

- Pipeline/SRAM pre-fill
- Active decompression
- SRAM drain

The host must prime the CL550's FIFO with data and clear the late flag after the CL550's registers and tables have been initialized, and before the HVEnable and Start registers are written. The late flag is set whenever the direction bit (bit 9 of the Configuration register) is set to decompression and the FIFO is empty. Because of this, the late flag will come up as soon as the Configuration register is written for decompression. After the FIFO is primed, it must be cleared by software as part of the start-up procedure (See Section 8.7). The FIFO can be primed using either DMA or programmed I/O techniques.

---

## 8.5 CL550 Decompression Operation

---

The host processor in DMA-driven system might use the following sequence at start-up:

1. Load the registers for decompression (the Late bit will activate).
2. Load the tables.
3. Enable DRQ on "Not 3/4 Full", DMA begins as soon as it is enabled.
4. Wait for the 3/4 Full flag.
5. Clear the Late bit by writing 0xffff to Flags register.
6. Set the NMRQ mask to "Not 1/4 Full" (assuming NMRQ is used for STALL).
7. Write the HVEnable and Start registers.

In master mode operation, the CL550 will assert its VSYNC output on start-up to indicate the start of a decompression cycle. At some point prior to the first active line of the video frame (typically 9 lines), the CL550's Zero Unpacker Unit will begin to fetch data from the FIFO for decompression, and the decompression pipeline will begin to fill with data.

STALL should be used to prevent underflow as the FIFO level drops. STALL must be driven externally, using either the NMRQ or HALF\_FULL outputs as FIFO status indicators. The advantage in using NMRQ is that it is programmable and easy to disable in hardware. The HALF\_FULL signal requires an external control bit to disable it.

As the decompression process continues, the host should maintain the level of the FIFO at between 1/4 and 3/4 until all compressed data is loaded. This can be done using either programmed I/O or DMA. Figure 8-16 shows the flow-chart for a simple program loop to load the CL550's FIFO with data. The *CL550/CL560 Program Examples Disk* also contains a code example in the file *550dec.c*.

Once all of the compressed data is loaded, you should disable STALL by clearing the NMRQ mask. This will allow the FIFO to drain completely without causing a STALL.

As the last pixel is removed from the SRAM, the CL560 will activate the VNAC flag in the flags register. If the Start register is left at logic 1,

then  $\overline{\text{VSYNC}}$  will go LOW to begin the next frame.  $\overline{\text{VSYNC}}$  can also be used as an end-of-frame interrupt. In response to this interrupt, frame-by-frame decompression drivers should reset and re-initialize the CL550 for the next frame. Continuous video systems should not reset the CL550 between frames, and should never allow the FIFO to go to empty until the last frame in the sequence.

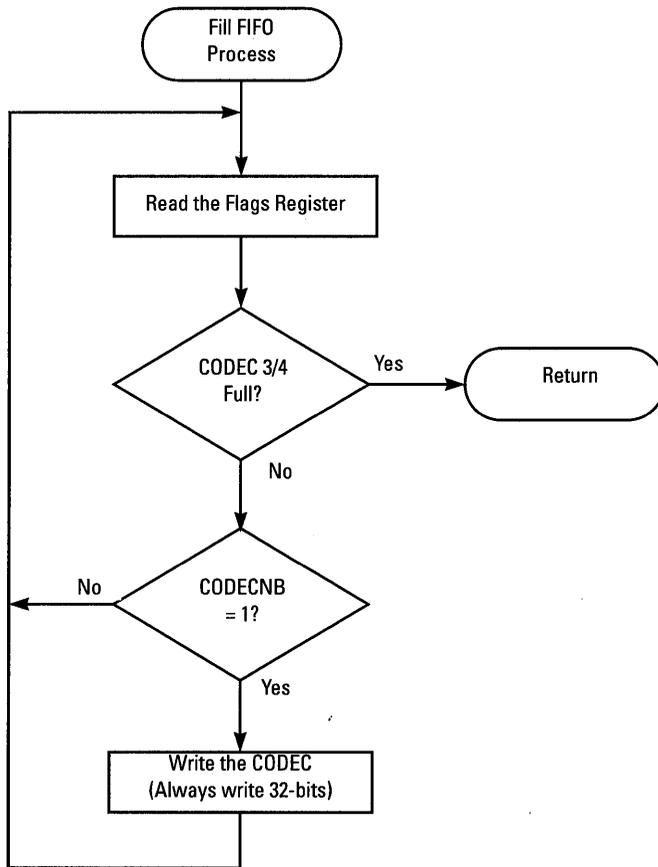


Figure 8-16 Flow Chart for Filling the CL550 FIFO

---

**8.6  
CL560  
Decompression  
Operations**

---

The CL560 operates almost identically to the CL550 in decompression mode. The only significant difference is in how the CODEC FIFO is loaded. Read Sections 8.3, 8.4, and 8.5 prior to reading this section.

The CODEC FIFO in the CL560 provides additional flexibility in two ways. First, it allows a pre-determined amount of data to be loaded (in the CL550, FIFO status must be checked each time that 32-bits of data is written). Second, it allows high-speed, two-clock-per-word loading of the FIFO.

To operate the CL560 in decompression mode, the driver program first must load the registers and tables (the Late bit will activate as with the CL550). The program can now load the FIFO using either DMA or programmed I/O. In a DMA-driven design, up to 96 words of data can be loaded to prime the FIFO. If programmed I/O is used, then a maximum of 128 words can be written. Figure 8-17 shows an example program flow for loading the FIFO in 64 word groups.

The CL560's Huffman decoder will automatically begin pre-fetching data from the FIFO as soon as the level reaches the 1/4 full mark. The decoder will place the first block of symbol data into the first of the two ping-pong buffers to await start-up.

The Decoder Start register is used to manually start the decoder if there are less than 32 words of data in the compressed image (a thumbnail, for example). Writing the Decoder Start register is a standard part of the CL560 start-up sequence in order to handle these special cases.

A decompression driver should follow the following steps to set up the CL560 for decompression.

1. Load the registers and tables (the Late bit activates).
2. Prime the CODEC FIFO to 3/4 full. The decoder will start automatically when the CODEC reaches 1/4 full.
3. Write 0xFFFF to the Flags register, this clears the Late bit.
4. Write to the Decoder Start register, in case the FIFO level is less than 1/4 full.
5. Write the IRQ1 mask to set up **STALL** (assuming IRQ1 is used for **STALL**). Note that with the CL560, **STALL** is not required in decompression. If **STALL** is not used, then it is the responsi-

bility of the host to prevent FIFO underflow. If the FIFO underflows, the Late bit activates. This flag can be routed to IRQ2.

6. Write HVENable and Start to begin decompression.

The CL560 behaves the same as the CL550 from start-up to the end of decompression. The end-of-frame condition can be checked using either the VNAC flag, or the VSYN flag.

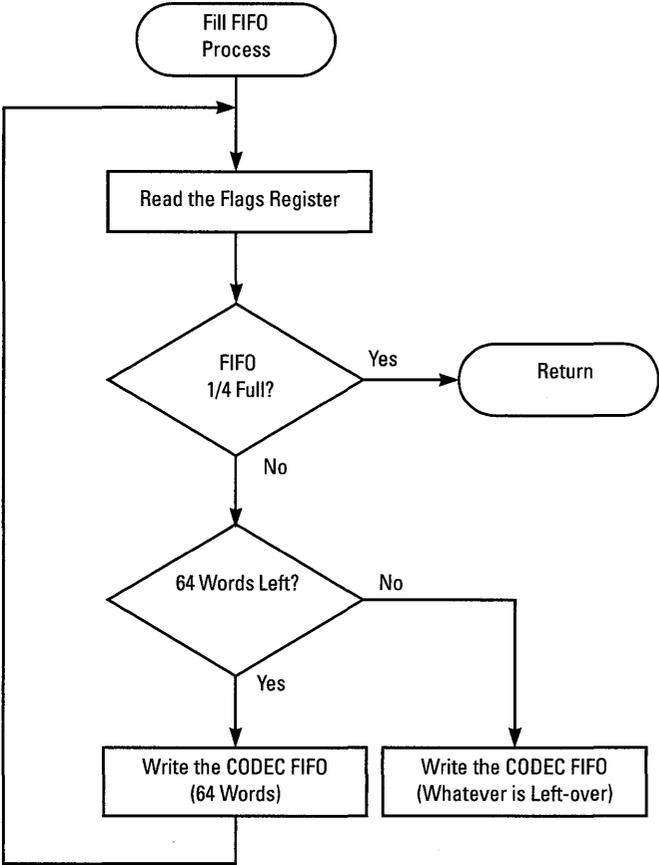


Figure 8-17 Program Flow for Filling the CL560 FIFO

---

## 8.7 CL5xx Initialization Procedures

---

This section describes basic procedures for initializing the CL5xx devices for compression or decompression operation. For details on programming a specific register, refer to Chapter 7. A source code example for initializing the CL5xx registers is given on the *CL550/CL560 Program Examples Disk*.

### 8.7.1 Overview

The initialization process consists of three main parts:

- Loading the Registers
- Loading the Tables
- Going through the Start-up Sequence

The register values to be loaded depend on the direction (compress or decompress), the image dimensions, device type, the JPEG tables specifications, and the overall system architecture.

The CL5xx register set is divided into two groups: registers common to all CL5xx devices, and device-specific registers. Both groups of registers are shown on the following pages.

#### *Common CL5xx Registers*

All of these registers must be loaded prior to device operation. The VControl and HControl registers are used during decompression only, and need not be loaded for compression. The Color Transform Matrix applies only to the RGB-to-YUV4:2:2 pixel mode, and does not need to be loaded for other modes.

- HPeriod
- HSync
- HDelay
- HActive
- VPeriod
- VSync
- VDelay
- VActive
- VideoLatency

- VControl (decompression only)
- HControl (decompression only)
- Init Registers 1-7
- DCT Coefficients (8 entries total)
- Huffman Table Sequence
- DPCM Register Sequence high
- DPCM Register Sequence low
- Coding Interval Register high (compression only)
- Coding Interval Register low (compression only)
- Decoder Table Sequence Length (decompression only)
- Decoder Code Order (decompression only)
- Quantizer A/B Select
- Quantizer Sync
- Quantizer Y/C Sequence
- Quantizer A/B Sequence
- Color Transform Matrix (9 entries total)

### *Device-Specific Registers*

How these registers are programmed depends on whether the device type is a CL550 or a CL560. Some of these registers are unique to either the CL550 or the CL560, and others differ depending on the device type. These registers are listed below

- Config
- Coder Attributes (compression only)
- Coder Sync (CL560 only, compression only)
- Coder RC Enable (CL560 only, compression only)
- Coder Padding (CL560 only, compression only)
- FRMEND Enable
- NMRQ Interrupt Mask (CL550 Only)
- IRQ1 Interrupt Mask (CL560 Only)
- IRQ2 Interrupt Mask (CL560 Only)
- DRQ Mask

### 8.7.2 Programming the Video Interface Control Registers

The Video Frame Control registers determine the signal timing of the Pixel Bus Interface. These registers include:

- HPeriod, HSync, HDelay, HActive
- VPeriod, VSync, VDelay, VActive

The values for these registers are calculated using the formulas in Chapter 7 and are based on the desired reference frame timing. Each of these registers is actually composed of two registers; a holding register and an active count register. When the host writes data, it is written to the holding register, and when the host reads data, it is reread from the active count register. Active count registers are loaded from the holding register at each active  $\overline{\text{VSYNC}}$  transition. Once the registers have been loaded, a device reset will not affect the contents of the holding registers, but it will cause the active count registers to read all 1's.

**Example:** Program the Video Frame Control registers for the frame shown in Figure 8-18. In this example, assume a YUV4:2:2 pixel type and MASTER mode for  $\overline{\text{VSYNC}}$  and  $\overline{\text{HSYNC}}$ . Also assume that the  $\overline{\text{VSYNC}}$  pulse width is 4 lines, and the  $\overline{\text{HSYNC}}$  pulse is 16 pixels.

In this video reference frame, the time (in pixels) between active  $\overline{\text{HSYNC}}$  transitions is specified as 704 pixels. The total number of lines in the video frame (total number of  $\overline{\text{HSYNC}}$  pulses between active  $\overline{\text{VSYNC}}$  transitions) is 272. The number of blank video lines from the active edge of  $\overline{\text{VSYNC}}$  to the first active video line is 16. The number of blank pixels from the active edge of  $\overline{\text{HSYNC}}$  to the first active pixel of a line is 32 pixels. The active video frame width is 640 pixels, and its height is 240 lines.

From these parameters, the H/V control register values are calculated as follows (all values are expressed as decimal):

$$\text{HPeriod} = 704 - 1 = 703$$

$$\text{HSync} = 16 - 1 = 15$$

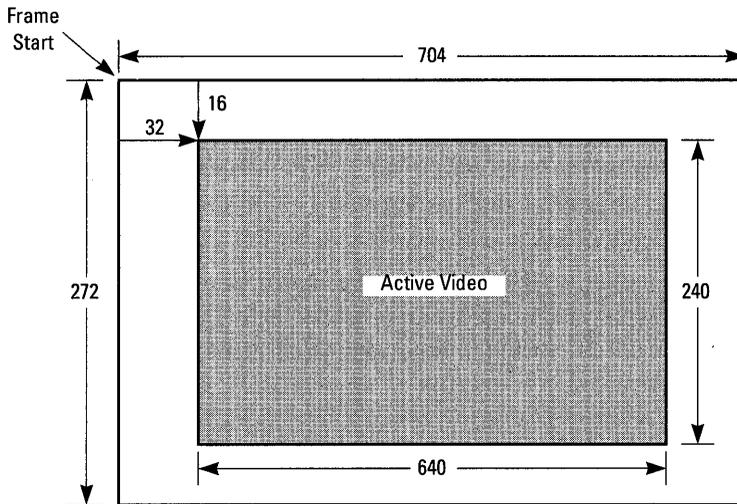
$$\text{HDelay} = 32$$

$$\text{HActive} = 640 / 8 = 80$$

$$\text{VPeriod} = 272$$

$$\text{VSync} = 4$$

$$\text{VDelay (Compression)} = 16$$



Note: All Dimensions are in pixels

**Figure 8-18 Programming Parameters for Video Window Example**

VDelay (Decompression) =

$$16 - 9 + \frac{(\text{Video Latency} + \text{HActiveClocks})}{\text{HPeriodClocks}}$$

where:

Video\_Latency = 383 (constant for YUV4:2:2 mode)

HActiveClocks =  $640 * 2 = 1280$

HPeriod Clocks =  $704 * 2 = 1408$

=  $16 - 9 + (383 + 1280) / 1408$

=  $16 - 9 + 1$

= 6 (Note that this value cannot be less than zero)

VActive =  $240 / 8 = 30$

#### Other Notes:

1. In Master mode, the HPeriod value must be greater than or equal to the active width plus the HDelay time. In the above ex-

ample, the minimum HPeriod value would be  $640 + 32 = 672$  (actually 671, because we subtract one).

2. In Master mode, the VPeriod value must be greater than or equal to the total active lines plus the vertical delay. In the previous example, the minimum VPeriod would be  $240+16 = 256$ .
3. The minimum value for VActive is 1. If VActive is set to 0, the device will not start.
4. (CL550 only) The minimum value for HActive must be such that when the image is compressed, the internal FIFO will reach the 1/4 full mark. Otherwise the Huffman Coder will not start, and the image cannot be compressed. For example, VActive = 1 and HActive = 2 is likely not to compress, as there is not enough data to fill the FIFO to 1/4 full.

### 8.7.3 HControl, VControl Registers Programming

The registers HControl and VControl are used only in the decompression direction, and are typically only used in the continuous-streaming method described in Section 8.2. These registers should be loaded to 0x3FF during single-image decompression or frame-by-frame video operations.

During continuous-stream decompression the HControl and VControl register values must be calculated relative to pixel mode and video window dimensions. The HControl and VControl registers determine the exact point at which the Zero Unpacker Unit stops reading coded entries from the FIFO in order to complete the current frame. In a typical decompression scenario using the CL550, the Zero Unpacker must stop reading code words from the FIFO some time before the last pixel of the current frame emerges on the pixel bus. The remainder of the current frame data which is still in the pipeline is then allowed to flush through. When a pulse on VSYNC starts the next frame, the Zero Unpacker Unit resumes reading from the FIFO.

The values programmed into the HControl and VControl registers must be calculated exactly. If the Zero Unpacker Unit does not stop at the correct time, data from current frame may be left over in the FIFO. In this case, the leftover data is decompressed on the next video frame. If the Zero Unpacker pulls too many words from the FIFO, part of the next frame's data will be flushed out at the end of the current frame and will be

lost. In either case, the next image out will be corrupted. The procedures for computing these register values are fairly complex, and will not be described here. Programmers should refer to the *CL550/CL560 Program Examples Disk* for source code examples.

#### 8.7.4 DCT Lookup Table

The DCT Lookup table is an 8-word table of machine constants that is used by the DCT processing unit. The same values are loaded for all modes of operation. Once loaded, the table will not be affected by reset and will remain valid as long as power is applied to the device. See Section 7.2 for the actual values.

#### 8.7.5 Color Conversion Matrix

The Color Conversion Matrix is only used in the RGB-to-YUV4:2:2 mode of operation. The matrix values are programmable so that different color spaces can be supported. Note, however, that the color-space converter is directly tied to the 4:4:4-to-4:2:2 subsampler. Matrix values for RGB-to-YUV4:2:2 operation are provided in Chapter 7, as well as on the *CL550/CL560 Program Examples Disk*.

If you need to develop custom matrices the hardware-specific values for the matrix are computed using the formula:

$$\text{matrix\_value} * 1024$$

rounded up to the next integer. The data is presented as a 12-bit signed integer. For example, from Table 2-1, we see that  $Y = 0.299R$ . Therefore the matrix value would be computed as:

$$.299 * 1024 = 306.176 \implies 307 \text{ (hex 133)}.$$

#### 8.7.6 Sequence Control Registers

Sequence Control registers are used to control selection of the Huffman and Quantizer tables according to the specified block sequence. For all pixel modes except the MONO (Bypass) mode, these registers are loaded with the constant values supplied in Chapter 7. This is due to the hardwired block sequence ordering produced by the MCU Block Storage Unit.

The MONO pixel mode bypasses the MCU block store and goes directly to the DCT input. In this mode, special block sequences can be used

in some cases. Refer to Section 8.10 for more information on custom sequence programming.

The Sequence Control registers are:

- Quantizer Y/C Sequence
- Quantizer A/B Sequence
- Huffman Table Sequence
- DPCM Register Sequence high
- DPCM Register Sequence low
- Decoder Table Sequence Length (decompression only)

### 8.7.7 Selection of the RST Interval (Compression)

The Coder Coding Interval registers are only used during compression and specify the number of Minimum Coded Units (MCUs) to encode before insertion of the next RST marker code. Bit 4 of the Coder Attributes register is set to a logic 1 to enable generation of RST marker codes. The Coder Coding Interval registers are then loaded with values that correspond to the number of MCU's in the restart interval.

**Example:** Program the coding interval so that a RST code is generated at the end of each 640 x 8 image strip (Figure 8-18). In YUV4:2:2 mode, the number of 8x8 pixel blocks in each MCU is 2 (16 x 8 pixels). Therefore, the coding interval is  $640 / 16 = 40$ . The Coding interval registers are loaded with:

Coder Coding Interval RH = 0x00  
 Coder Coding Interval LH = 0x28 (40 decimal)

### 8.7.8 Pipeline Configuration Registers

The Pipeline Configuration registers are used to configure the various computing elements of the CL5xx compression pipeline. These registers are loaded with constant values that depend on direction and pixel type only. They are not programmable in any other manner, and are the same for all applications. Initialization constants for each register are given in Chapter 7. The Pipeline Configuration Registers include:

- Init Registers 1-7
- Video Latency
- Decoder Code Order (decompression only)

### 8.7.9 Quantizer Tables Loading

The CL5xx quantization tables are on-chip RAM areas used to store the quantization parameters needed by the quantizer unit. The values in these tables are machine-specific, and must be converted from the ISO JPEG Interchange Format. These machine-specific quantization tables must then be downloaded into the appropriate table area in the CL5xx device. Procedures for generating CL5xx quantizer tables from the ISO JPEG Interchange Format are given in Section 8.9.

We recommend that you use the four-table mode for all applications. The two-table mode is useful only for updating Q tables on a frame-by-frame basis either for data rate control purposes or for half-duplex operation. The Quantizer Sequence registers determine the actual order of selection of the tables, regardless of whether the part is in two-table or four-table mode. In some cases, you can use the four-table mode to load four tables at once (for instance, 2 compress tables and 2 decompress tables), then switch into two table mode.

If you are updating the Q-tables every frame, you should keep in mind that after the Quantizer A/B Select register is programmed the tables will not actually switch until an active transition on  $\overline{\text{VSYNC}}$  occurs.

### 8.7.10 Huffman Tables Loading

Like the Quantizer tables, the CL5xx Huffman tables are also machine specific. Before images can be compressed or decompressed using the CL5xx devices, the Huffman table information must be converted from the JPEG interchange format into the CL5xx machine format. Procedures for generating CL5xx machine-specific Huffman tables are given in Section 8.7.

The Huffman Load Enable register must be set to 1 to load the machine-specific tables into the CL5xx device. Once the tables are loaded the Huffman Load Enable register must be cleared before the compress or decompress operation can take place. When a Huffman table has been loaded, it will remain valid as long as power is maintained to the device; Huffman table contents are unaffected by reset.

### 8.7.11 CL5xx Start-up Sequence

The start-up sequence is a series of register writes used to start a compression or decompression operation. This is done only after all of the

registers and tables have been loaded.

For a compression operation, the start-up sequence is:

Write NMRQ Mask = FIFO\_3Q\_Full (used for STALL control)

Write DRQ Mask = CodecNB • FIFO\_1Q\_Full

Write HVEnable = 1

Write Start = 1

For a decompression operation, the start-up sequence is:

Write NMRQ Mask = FIFO\_Not1Q\_Full (used for STALL control)

Write DRQ Mask = CodecNB • FIFO\_Not3Q\_Full

(at this point, use DMA transfers to pre-fill the FIFO to 3/4 full or use software to pre-fill the FIFO)

Write Flags = 0xffff (this clears the LATE flag)

Write HVEnable= 1 (only after FIFO has data in it)

Write Start = 1

*Note: The flag CodecNB in the steps listed above is a CL550 flag only. CL560 based systems only need to initialize the level flags.*

### **8.7.12 Device Reset Considerations**

A reset is performed by either pulling the RESET pin low or by writing to the SReset register. Some of the CL5xx's registers are placed into an initial state following a reset, and these registers need to be re-programmed before operation can resume. The registers that are affected by reset are listed in Table 8-1.

**Table 8-1 CL5xx Register Reset Values**

<b>Register Name</b>	<b>CL550 Address</b>	<b>CL560 Address</b>	<b>Value after Reset</b>
HPeriod	0x8000	0x8000	All 1s
HSync	0x8004	0x8004	All 1s
HDelay	0x8008	0x8008	All 1s
HActive	0x800C	0x800C	All 1s
VPeriod	0x8010	0x8010	All 1s
VSync	0x8014	0x8014	All 1s
VDelay	0x8018	0x8018	All 1s
VActive	0x801C	0x801C	All 1s
Video Latency	0xC030	0xC030	All 1s
HControl	0xC034	0xC034	All 1s
VControl	0xC038	0xC038	All 1s
Configuration	0x9000	0x9000	0
Huffman Table Load Enable	0x9004	0x9004	0
S-Reset	0x9008	0x9008	0
Start	0x900C	0x900C	0
HV Enable	0x9010	0x9010	0
Interrupt Mask	0x9018	-	0
DMA Request Interrupt Mask	0x901C	0x901C	0
Start of Frame	0x9020	0x9020	0
Decoder Code Order	0xA81C	0xA81C	0
Vertical Line Count	0xC03C	0xC03C	0
Init Register 4	0xCF00	0xCF00	0
FRMEND Enable	-	0x092C	0
Compressed Word Count High	-	0xA024	0
Compressed Word Count Low	-	0xA028	0
Coder Rate Control Active	-	0xA02C	0
Coder Rate Control Enable	-	0xA030	0
Coder Robustness Active	-	0xA034	0
Decoding Mismatch	-	0xA824	0
Flags	0x9014	0x9014	1000 0x10 1000 1110
Init Register 3	0xB600	0xB600	001 0000 0000

The Video Window Control registers are actually two registers, a write-register and a read-register, in a master-slave arrangement. The write register functions as a master, and holds the programmed count value that was loaded by the host. The write-register data is not affected by reset. The read register is the active count, and is loaded from the master on the active transition of **VSYNC**. A device reset sets all of the read-registers to 1's, which puts the video interface into an inactive state. Although it appears that the register values have been altered and need to be re-programmed, this is really not necessary because the active count registers will update themselves from the master count registers when the device starts up.

In many frame-by-frame video applications, a minimized restart procedure can be used. Assuming that no tables need to be updated, only those control registers that are affected by reset require programming. These registers are listed in Table 8-2. Programmers should keep a copy of these register values handy to avoid having to compute them over and over again. Once the affected registers have been re-initialized, the CL5xx is ready to be started for the next frame, using the procedure described in Section 8.7.11.

**Table 8-2 Registers Required for Minimized Restart Procedure**

<b>Register Name</b>	<b>Address</b>
Configuration	0x9000
Init_3	0xB600
Init_4	0xCF00
Decoder Code Order	0xA81C
Video Latency	0xC030
HControl	0xC034
VControl	0xC038
DMA Request Interrupt Mask	0x901C
IRQ1/NMRQ Mask	0x9018
FRMEND Enable (CL560)	0x092C

This section describes the tasks associated with creating JPEG Huffman tables. Creating JPEG Huffman tables involves extracting the Huffman tables from the ISO JPEG Interchange format and converting those tables into machine specific formats for the CL550 or CL560 device. Source code examples in C for handling Huffman tables and converting them into CL550/CL560 machine-specific format are provided with the *CL550/CL560 Program Examples Disk*.

---

## 8.8 Programming the Huffman Tables

---

### 8.8.1 Extracting Huffman Tables from ISO JPEG Interchange Format

Before you can decompress images that are stored in the ISO JPEG Interchange format, you must extract the compression parameters and tables from the stream header. Huffman tables are among the information that must be extracted. The syntax for specifying Huffman tables is given in Annex B of the JPEG International Standard specification (ISO/IEC IS 10918-1).

Huffman tables are specified in terms of a 16-byte list (BITS) giving the number of codes for each code length from 1 to 16. This is followed by a list of the 8-bit symbol values (VALUES), each of which is assigned a Huffman code. Annex C of the ISO JPEG specification describes three procedures by which the actual Huffman tables are extracted from the BITS/VALUES syntax. These three procedures are:

- `Generate_Size_Table()` - A procedure to find the sizes of each Huffman code.
- `Generate_Code_Table()` - A procedure to generate the table of Huffman codes based on the size information.
- `Order_Huffman_Codes()` - This procedure orders the list of Huffman codes according to the VALUES list.

Flowcharts for these procedures are listed in Annex C of the International Standard. The *CL550/CL560 Program Examples Disk* contains working C source code implementations of these functions in the file *makehuff.c*.

### 8.8.2 CL550 Huffman Table Formats

Once the JPEG Huffman tables have been extracted from the JPEG file or stream header, the tables must be converted into a CL550/CL560 machine-specific form before they can be stored in the device. This section describes the machine-specific format for CL550 Huffman Tables and

procedures for generating these tables based on any set of JPEG-compatible Huffman Code Tables. The CL550 uses different Huffman table formats depending on whether the CL550 is compressing or decompressing.

The CL560 has four Huffman table RAM spaces; two for storing AC table data and two for storing DC table data. The Huffman Table RAMs are completely static. Once loaded, the contents of the RAMs will remain unchanged as long as power is maintained to the device; even after the device is reset.

### *CL550 Encoder Tables*

The machine-specific table layouts for the CL550 in compression mode are shown in Table 8-3. For brevity, only the luminance encoder tables are shown. Chrominance tables have exactly the same layout, but start at addresses 0xF000 and 0xFC00, respectively.

Each cell in the Huffman table RAM in Table 8-3 is 18 bits wide. RAM cells are accessed from the host bus in two 9-bit halves, with the low-order nine bits at the lower address (E000 for example) and the high-order nine bits at the higher address (E004 for example). In order to access the Huffman table RAM, you must program the Huffman Load Enable register to 1.

The Code word formats for the RAM cells vary depending on the length of the Huffman Code. In a typical application you must derive the Code and Length values from the JPEG interchange format. The procedures for extracting the Code and Length values from JPEG interchange format are described in Annex C of the ISO JPEG specification (ISO/IEC IS 10918-1). Source code examples are also found in the *CL550/CL560 Program Examples Disk* in the file *makehuff.c*. Figure 8-19 shows these code word formats for various code lengths.

The value field of the Huffman Code format in Figure 8-19 is used to contain the Huffman code being loaded. The code is bit-reversed from the JPEG bit order: the MSB appears at bit zero.

**Table 8-3 CL550 Huffman Table Layouts in Compression Mode**

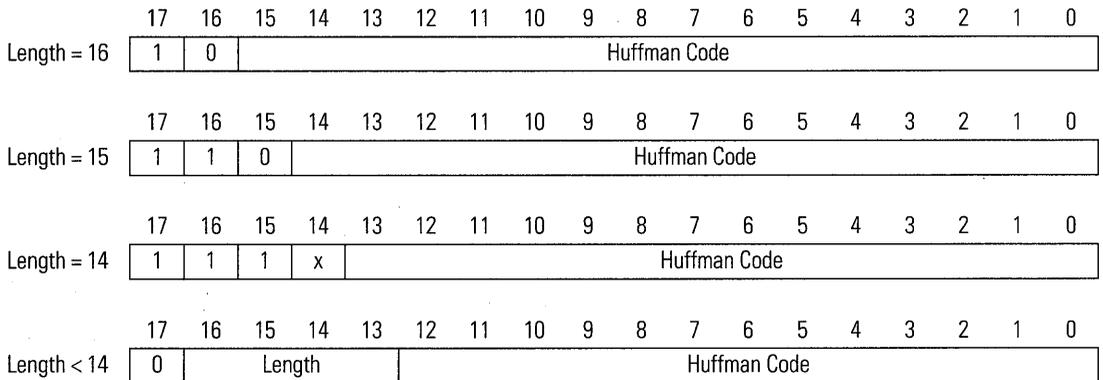
Y-AC	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
0xE000	EOB	0/1	0/2	0/3	0/4	0/5	0/6	0/7	0/8	0/9	0/A	00	00	00	00	00
0xE080	00	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/A	00	00	00	00	00
0xE100	00	2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	2/9	2/A	00	00	00	00	00
0xE180	00	3/1	3/2	3/3	3/4	3/5	3/6	3/7	3/8	3/9	3/A	00	00	00	00	00
0xE200	00	4/1	4/2	4/3	4/4	4/5	4/6	4/7	4/8	4/9	4/A	00	00	00	00	00
0xE280	00	5/1	5/2	5/3	5/4	5/5	5/6	5/7	5/8	5/9	5/A	00	00	00	00	00
0xE300	00	6/1	6/2	6/3	6/4	6/5	6/6	6/7	6/8	6/9	6/A	00	00	00	00	00
0xE380	00	7/1	7/2	7/3	7/4	7/5	7/6	7/7	7/8	7/9	7/A	00	00	00	00	00
0xE400	00	8/1	8/2	8/3	8/4	8/5	8/6	8/7	8/8	8/9	8/A	00	00	00	00	00
0xE480	00	9/1	9/2	9/3	9/4	9/5	9/6	9/7	9/8	9/9	9/A	00	00	00	00	00
0xE500	00	A/1	A/2	A/3	A/4	A/5	A/6	A/7	A/8	A/9	A/A	00	00	00	00	00
0xE580	00	B/1	B/2	B/3	B/4	B/5	B/6	B/7	B/8	B/9	B/A	00	00	00	00	00
0xE600	00	C/1	C/2	C/3	C/4	C/5	C/6	C/7	C/8	C/9	C/A	00	00	00	00	00
0xE680	00	D/1	D/2	D/3	D/4	D/5	D/6	D/7	D/8	D/9	D/A	00	00	00	00	00
0xE700	00	E/1	E/2	E/3	E/4	E/5	E/6	E/7	E/8	E/9	E/A	00	00	00	00	00
0xE780	ZRL	F/1	F/2	F/3	F/4	F/5	F/6	F/7	F/8	F/9	F/A	00	00	00	00	00

0XE780 - 0XEAFc: Load with 00. RRRR/SSSS

**Y-DC**

0xEC00	0	1	2	3	4	5	6	7	8	9	A	B	00	00	00	00
--------	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----	----

Category



**Figure 8-19 CL550 Huffman Table Formats for Compression**

Example: Program the first YAC table entry from the example Y-AC tables listed in Appendix K of the ISO JPEG Specification into the CL550's Y-AC Huffman table RAM.

From the listing in Appendix K, the first value is the EOB code (Run/Size = 0/0). The Huffman code is 0x0a and the Code Length is 4. Using Table 8-3, we see that the EOB code corresponds to address 0xE000. This is where the code will be loaded. Also, since this code is less than 14 bits, we will need to specify a length field as shown in Figure 8-19.

The Huffman Code bits are loaded in reverse bit order, so the code "1010" is first bit-reversed to be "0101". Now, all of the bit fields are assembled such that the following binary code word is produced:

Code Word = 0 0100 000000000101

In order to load this value into the CL550, the code word is split into two 9-bit values, as shown below.

Code Word (low) = 000000101 (loaded into address E000)

Code Word (high) = 001000000 (loaded into address E004)

### *CL550 Decoder Tables*

The CL550 Huffman decoder uses a two-bit-per-grab tree-walk architecture. The decoder operates by grabbing two bits of Huffman code at a time, and performing a tree-walk through the on-chip tables to arrive at the symbol which is represented by the Huffman code. Up to eight grabs are necessary to completely decode the longest Huffman code. The longest Huffman code is 16 bits long.

The format of the CL550 Huffman tables for decompression is shown in Figure 8-20. Each 18-bit RAM cell is divided into two 9-bit nodes. Each group of four consecutive nodes form a state (two 18-bit cells). Each 9-bit node is accessible from the host bus on consecutive addresses (e.g. 0xE000, 0xE004). The low-order nine bits is at the low address. In each node, bit 8 is the "leaf" flag, or "code done" bit. If the leaf flag is zero, then bits 0-7 contain the address of the next state. If the leaf flag is set, then bits 0-7 is the actual 8-bit value represented by the code.

	17	16		9	8	7		0	
0xE004	L				L				0xE000
0xE00C	L				L				0xE008
0xE014	L				L				0xE010
0xE01C	L				L				0xE018

**Figure 8-20 CL550 Huffman Table Format for Decompression**

Example: During a decoding operation, the first two bits of Huffman code are used as an index to the first group of four nodes, or state, in the table space, starting at address zero. Assuming that the code bits taken are '11' then the node at address 0xE00C would be taken. Assume that the leaf flag in this case is zero. In that case, bits 0-7 of that node contain the offset to the next state to be used with the next two bits of Huffman code. This process continues until the leaf flag is found. At that point, the 8-bits of symbol data from that node are taken.

The procedure for generating these tables is complex and will not be described here. The *CL550/CL560 Program Examples Disk* contains C source code for generating these tables in the file *makehuff.c*.

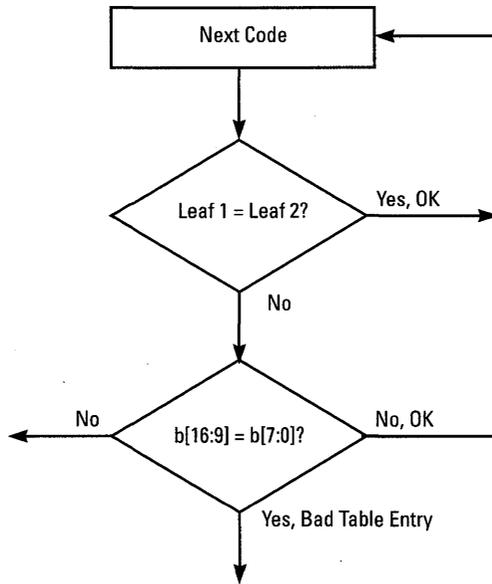
*IMPORTANT NOTE: The CL550's Huffman decoder has a logical bug which renders it unable to decode images with certain custom Huffman tables. Because of this, the CL550 is not 100 percent compliant with the ISO JPEG compliance specification (ISO/IEC IS 10918-2). This bug only affects users who are using custom Huffman tables. The Example tables given in Annex K of the ISO JPEG specification will decode with no problems. Further, users of the Microsoft JPEG/AVI format are not affected, as all AVI-compliant video streams are required to use the tables given in Annex K.*

### *Detecting a Bad Huffman Table*

For each two-node pair in the decoding tree, an error will result if the

leaf bits of node 1 and node 2 are different but bits 0-7 of each node are equal. The procedure for detecting this condition is illustrated in flow-chart form in Figure 8-21. This procedure can be used by programmers to determine if an imported image can be decoded successfully using the CL550. If the table is found to be bad, then an alternate decoding method must be used. Alternative decoding methods include the CL560 processor or a software-only decoder.

If the condition shown in Figure 8-21 results from converting the Huffman table into the CL550 machine-specific format, then the table cannot be used with the CL550. The condition that you are looking for is when the "Run / Level" field matches the "Next Address" field of two consecutive nodes where the leaf flags are not equal.



17	16		9	8	7		0
0	Next Address			1	Run / Level		
1	Run / Level			0	Next Address		

**Figure 8-21 Detecting a Bad Huffman Table**

### 8.8.3 CL560 Huffman Table Format

This section describes the machine-specific format for CL560 Huffman Tables and a procedure for generating these tables based on any set of JPEG-compatible Huffman Code Tables. Unlike the CL550, the CL560's Huffman Tables are homogeneous in the sense that the same machine-specific tables are used for both compression and decompression (the CL550 requires separate machine-specific tables for compression and decompression, respectively). The *CL550/CL560 Program Examples Disk* contains an example source code for generating CL560 machine-specific tables in the file *makehuff.c*.

The CL560 has four Huffman Table RAM spaces, two for storing AC table data and two for storing DC table data. The Huffman Table RAMs are completely static. Once loaded, the contents of the RAMs will remain unchanged even after device reset, as long as power is maintained to the device.

The layout of the Huffman tables is shown in Figure 8-22. This diagram shows the CL560 Huffman table addressing relative to the corresponding Run/Size values in the JPEG Huffman table.

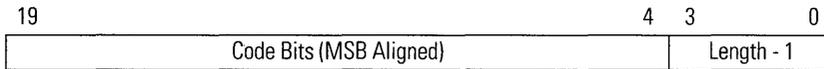
Each of the RAM cells in Figure 8-22 is 20 bits wide. The 20-bit table entry contains two fields, a code field and a code length field, as shown in Figure 2. Bits 3-0 are loaded with the number of significant Huffman code bits, minus 1. Bits 19-4 contain the actual code value left shifted until the most significant code bit is at bit 19. Unused bits are zero-filled.

In a typical JPEG interchange application you must derive the Code and Length values from the JPEG interchange format. Procedures for extracting the Code and Length values from JPEG interchange format are described in Annex C of the ISO JPEG specification (ISO/IEC IS 10918-1). Source code examples are also found on the *CL550/CL560 Program Examples Disk* in the file *makehuff.c*

## Programming the Huffman Tables

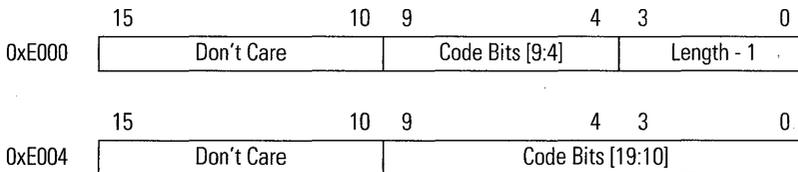
<b>Y-AC</b>	<b>00</b>	<b>08</b>	<b>10</b>	<b>18</b>	<b>20</b>	<b>28</b>	<b>30</b>	<b>38</b>	<b>40</b>	<b>48</b>	<b>50</b>	<b>58</b>	<b>60</b>	<b>68</b>	<b>70</b>	<b>78</b>
0xE000	EOB	11	11	11	11	11	11	11	11	11	11	11	11	11	11	ZRL
0xE080	0/1	1/1	2/1	3/1	4/1	5/1	6/1	7/1	8/1	9/1	A/1	B/1	C/1	D/1	E/1	F/1
0xE100	0/2	1/2	2/2	3/2	4/2	5/2	6/2	7/2	8/2	9/2	A/2	B/2	C/2	D/2	E/2	F/2
0xE180	0/3	1/3	2/3	3/3	4/3	5/3	6/3	7/3	8/3	9/3	A/3	B/3	C/3	D/3	E/3	F/3
0xE200	0/4	1/4	2/4	3/4	4/4	5/4	6/4	7/4	8/4	9/4	A/4	B/4	C/4	D/4	E/4	F/4
0xE280	0/5	1/5	2/5	3/5	4/5	5/5	6/5	7/5	8/5	9/5	A/5	B/5	C/5	D/5	E/5	F/5
0xE300	0/6	1/6	2/6	3/6	4/6	5/6	6/6	7/6	8/6	9/6	A/6	B/6	C/6	D/6	E/6	F/6
0xE380	0/7	1/7	2/7	3/7	4/7	5/7	6/7	7/7	8/7	9/7	A/7	B/7	C/7	D/7	E/7	F/7
0xE400	0/8	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8	9/8	A/8	B/8	C/8	D/8	E/8	F/8
0xE480	0/9	1/9	2/9	3/9	4/9	5/9	6/9	7/9	8/9	9/9	A/9	B/9	C/9	D/9	E/9	F/9
0xE500	0/A	1/A	2/A	3/A	4/A	5/A	6/A	7/A	8/A	9/A	A/A	B/A	C/A	D/A	E/A	F/A
0xE580	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
RRRR / SSSS																
<b>Y-DC</b>																
0xE600	0	1	2	3	4	5	6	7	8	9	A	B	11	11	11	11
Category																
-----																
<b>C-AC</b>	<b>00</b>	<b>08</b>	<b>10</b>	<b>18</b>	<b>20</b>	<b>28</b>	<b>30</b>	<b>38</b>	<b>40</b>	<b>48</b>	<b>50</b>	<b>58</b>	<b>60</b>	<b>68</b>	<b>70</b>	<b>78</b>
0xE800	EOB	11	11	11	11	11	11	11	11	11	11	11	11	11	11	ZRL
0xE880	0/1	1/1	2/1	3/1	4/1	5/1	6/1	7/1	8/1	9/1	A/1	B/1	C/1	D/1	E/1	F/1
0xE900	0/2	1/2	2/2	3/2	4/2	5/2	6/2	7/2	8/2	9/2	A/2	B/2	C/2	D/2	E/2	F/2
0xE980	0/3	1/3	2/3	3/3	4/3	5/3	6/3	7/3	8/3	9/3	A/3	B/3	C/3	D/3	E/3	F/3
0xEA00	0/4	1/4	2/4	3/4	4/4	5/4	6/4	7/4	8/4	9/4	A/4	B/4	C/4	D/4	E/4	F/4
0xEA80	0/5	1/5	2/5	3/5	4/5	5/5	6/5	7/5	8/5	9/5	A/5	B/5	C/5	D/5	E/5	F/5
0xEB00	0/6	1/6	2/6	3/6	4/6	5/6	6/6	7/6	8/6	9/6	A/6	B/6	C/6	D/6	E/6	F/6
0xEB80	0/7	1/7	2/7	3/7	4/7	5/7	6/7	7/7	8/7	9/7	A/7	B/7	C/7	D/7	E/7	F/7
0xEC00	0/8	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8	9/8	A/8	B/8	C/8	D/8	E/8	F/8
0xEC80	0/9	1/9	2/9	3/9	4/9	5/9	6/9	7/9	8/9	9/9	A/9	B/9	C/9	D/9	E/9	F/9
0xED00	0/A	1/A	2/A	3/A	4/A	5/A	6/A	7/A	8/A	9/A	A/A	B/A	C/A	D/A	E/A	F/A
0xED80	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
RRRR / SSSS																
<b>C-DC</b>																
0xEE00	0	1	2	3	4	5	6	7	8	9	A	B	11	11	11	11
Category																

**Figure 8-22 CL550 Huffman Table Layouts in Compression Mode**



**Figure 8-23 CL560 Machine-Specific Huffman Table Entry Format**

Each RAM cell is accessible from the CL560 Host Bus in 10-bit halves, with the least significant half at the lower address. An example is given in Figure 8-24. The CODE bits refer to Figure 8-23 above. Note that in order to read or write Huffman table entries, the Huffman Load Enable Register must be programmed to 1.



**Figure 8-24 Example for Accessing a Table Entry at Address 0xE000**

**Example:** Program the first Y-AC table location (Run/Size = 0/0 or EOB) with the example table value found in Appendix K of the ISO JPEG specification. From these tables, the CODE value is 0x000A, and the Code Length value is 4 bits. The 20-bit CL560 machine-specific code word is calculated as:

$$CL560CodeWord = (( Code \ll (20 - Length) ) + (Length - 1))$$

This results in the hex value 0xA0003. This value is then split into two halves as follows:

$$CL560CodeWord\_Low = CL560CodeWord \& 0x03FF$$

$$= 0x0003 \text{ (loads into address E000)}$$

$$CL560CodeWord\_High = (CL560CodeWord \gg 10) \& 0x03FF$$

$$= 0x0280 \text{ (loads into address E004)}$$

---

## 8.9 CL5xx Quantizer Table Programming

---

This section discusses generation of the CL5xx Quantizer tables. An overview is presented, followed by the detailed procedures for creating Quantization tables for use with the CL5xx devices. Loading of the CL5xx Quantizer tables is discussed in Section 8.7. Functions for handling Quantizer tables are provided on the *CL550/CL560 Program Examples Disk* in the file *makeQ.c*.

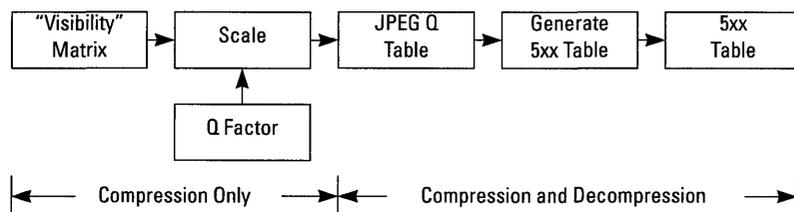
### 8.9.1 Overview

In pure JPEG terms, the actual compression ratio depends on the quantization tables and the Huffman tables, but most of all on the image data itself. For a constant set of Huffman and quantization parameters, the compression ratio will vary depending on the amount of AC terms that are present in the image data. Quantizer tables are used to filter out AC terms that are normally unnoticeable to the human eye. A higher rate of quantization means that more AC terms are reduced to zero, giving a higher compression ratio. The trade-off is that the higher the rate of quantization is, the greater the data loss will be. At some point, the video quality will degrade to a level that is unacceptable to the user.

Although the Quantizer tables themselves do not control the exact compression ratio, they are the only real control that the user has over the compression ratio in a video compression situation. For simple compression applications, the application uses one set of quantization parameters for the entire video sequence. For more advanced applications, the quantizer tables can be updated on a frame-by-frame basis to more tightly regulate the compressed output rate. C-Cube's CL5xx processors can accept the full range of quantization sets allowed under the ISO JPEG Baseline System. They also support switching of quantizer tables on a frame-by-frame basis.

Figure 8-25 illustrates the basic procedures used in creating quantizer tables for the CL5xx devices. The first part of the flow relates only to creating the Q-tables used during compression. It is used to set up the JPEG quantization parameters for a desired compression level based on a standard set of weighting functions, or "visibility matrices" and a scalar, or Q Factor. The second part involves taking the actual quantization parameters and generating machine-loadable forms of the tables for either compression or decompression. This procedure must be followed for each table that is to be used (up to four per image in the baseline

case).



**Figure 8-25 Typical Flow for Generating a CL5xx Quantizer Table**

Color images typically use at least two quantization tables. For example, a YUV4:2:2 image uses two tables: one for the luminance component, and one for the chrominance components. This means that a total of 128 quantization terms need to be updated each time the compression ratio is changed. To simplify this process, the concept of the Q Factor is introduced. The Q factor is not part of the JPEG standard. Rather, it is a simple means of scaling a constant set of values to produce a range of compression ratios. By using the Q Factor, the user needs to specify only one term, as opposed to 128, in order to change the compression ratio.

Consider the psychovisual weighting functions shown in Figure 2-3. These tables (call them "visibility tables") are used to establish the basic shape of the filter curve in two dimensions. They are based on a suite of video test clips submitted to the JPEG working group and are fairly well tuned for video applications. By using these tables as a working base a range of compression ratios can be obtained simply by scaling the base table by the Q Factor. The output of the scaling process is a table that contains actual JPEG quantization parameters that can either be put into the ISO stream header or used to perform compression or decompression.

### 8.9.2 Q Table Scaling Function

The *CL550/CL560 Program Examples Disk* provides a function called `Make JPEG_Q_Table()`. This function takes as its input a visibility table like the component matrix in Figure 2-3 and a Q Factor (an integer between 1 and 255). Each item in the visibility matrix is multiplied by  $Q/$

50, limited between 1 and 255, then stored in zig-zag order as shown in Table 2-2. This scaling function is listed in Figure 8-26.

```

void MakeJPEG_Q_Table
(
    short          QFactor,          /* scalar */
    short          *Input_Visi_Table, /* visibility matrix, non-zig-zag */
    unsigned char  *JPEG_Q_Table     /* output list. zig-zagged */
)
{
    short i;
    unsigned short temp;

    for( i = 0; i < 64; i++)
    {
        /* read input array in zig-zag order, scaling the results
           and placing them into JPEG_Q_Table */

        temp = (short)((Input_Visi_Table[ ZigZag[i] ]
            * QFactor / 50.0) + 0.5 );

        if( temp < 1)
            temp = 1; /* minimum value is 1 */
        if( temp > 255)
            temp = 255; /* maximum value is 255 */

        JPEG_Q_Table[i] = (unsigned char) temp;
    }
}

```

**Figure 8-26 Quantizer Table Scaling Function**

The output of this function is an integer matrix in ISO JPEG order. This table can be attached to the JPEG stream header for export or used to create machine-loadable tables for the CL5xx device.

### 8.9.3 Generating Machine-Loadable Q Tables for CL5xx Devices

Once the JPEG quantizer tables have been created or read in from the incoming file header, they must be converted for use with the CL5xx devices. The CL5xx Quantizer unit is actually a 16 x 16 multiplier. The multiplier unit is shared with the DCT/IDCT unit, which is also a multiplier. To save silicon, these two multipliers are integrated in a single multiplier stage. When generating Q tables for the CL5xx devices, the program must also combine factors used in the DCT computation.

The *CL550/CL560 Program Examples* disk provides a function called `MakeCL5xx_Q_Table()`. This function accepts an input table in ISO JPEG order and produces a machine-loadable table as its output. The function is direction-sensitive, and the direction is specified in the global variable `Direction`. DCT factors are stored in a standard header file called "makeQ.h". There, they are referred to as "K\_Factors". Note that floating-point math is used in this routine. Some C compilers may vary in this area. This function is listed below. Refer to the examples disk for header file constants.

## CL5xx Quantizer Table Programming

```
void MakeCL5xx_Q_Table
(
    unsigned char    *JPEG_Q_Table,
    unsigned short   *CL550_Q_Table
)
{

    short i, j;
    short qshift;
    unsigned short Temp;
    unsigned short UnZigged[64];
    unsigned short Temp_Q_Table[64];

/* 1. first, the JPEG_Q_Table is unzigged from its zig-zag order */

    for( i = 0 ; i < 64 ; i++ )
        UnZigged[ ZigZag[i] ] = (unsigned short) JPEG_Q_Table[i];

/* 2. From the UnZigged table, generate tables for the CL550 */

    if( Direction == COMPRESS )          /* build a compression table */
    {
        for( i = 0; i < 64 ; i++ )
        {
            Temp_Q_Table[i] =
                (unsigned short) (CompKFactor[i] / (UnZigged[i] << 1) + .5);

            if ( Temp_Q_Table[i] > (unsigned short) 32767. )
                Temp_Q_Table[i] = (unsigned short)32767.0;
        }
    }
    else    /* Direction == DECOMPRESS, build a decompression table */
```

**Figure 8-27 Make CL5xx Quantizer Table Function Listing**

```

/* 3. Convert the Temp_Q_Table to CL550 ordering using a corner-to-corner
transposition */

for ( i=0; i<8; i++ )
    for ( j=0; j<8; j++ )
        CL550_Q_Table[i*8+j] = Temp_Q_Table[i+j*8];
}

/*****

/* CountBits() -
   This function returns the number of significant bits for a given input
   value.
*/

static short CountBits( unsigned short quant )
{
int i;
unsigned char mask;
for (i=0, mask=0x80; i<8; i++,mask>>=1)
    if (mask & quant) return(i);
}

{
for( i = 0; i < 64; i++ )
{
    Temp_Q_Table[i] = (unsigned short)(DecompKFactor[i]
        * UnZigged[i]
        * pow( (double)2, (double)CountBits(UnZigged[i]) ) + .5);
    /* truncate to 13 significant bits */
    Temp_Q_Table[i] &= 0xfffc;
    qshift = 7 - CountBits( UnZigged[i] );
    Temp_Q_Table[i] = (qshift<<13) + (Temp_Q_Table[i]>>2);
        /* exponent + mantissa */
}
}
}

```

**Figure 8-27 Make CL5xx Quantizer Table Function Listing**

---

## **8.10 Custom Block Sequencing**

---

This section describes the CL5xx's component block sequence registers. These registers control the sequencing of component blocks within the various processing stages of the CL550. For certain applications, these registers can be programmed to accommodate user-specified block sequences.

### **8.10.1 Restrictions**

To use the CL5xx devices for custom block sequences, the CL5xx device must be configured in the MONO, or "Bypass" mode. In this mode the PXDAT bus is routed directly to the DCT input. All other modes use the MCU Block Store, which generates a pre-defined, hardwired block sequence. For these other modes of operation, C-Cube has specified standard values for these registers (see Chapter 7), and users should not make any changes to them.

A further restriction in this mode is that the strip buffer addressing scheme expects a single-component stream. Multicomponent streams having custom sequences should be passed in block format, and the SRAM Strip buffer should not be used.

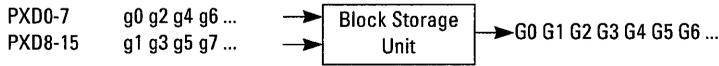
### **8.10.2 CL5xx Internal Component Sequencing**

To understand custom sequencing with the CL5xx it is important to first understand how the CL5xx formats its data internally and prepares it for processing. The first step in this process is the actual data acquisition itself. The Video interface accepts pixel data in one of six different formats: Gray-Scale, YUV4:2:2, YUV4:4:4 (or RGB4:4:4), 4:4:4:4, RGB-to-YUV4:2:2, and YUV4:4:4-to-YUV4:2:2. The format required for the input pixels is 8x8 block ordering. Block ordering can be performed using the strip buffer SRAM which is driven by the CL5xx device

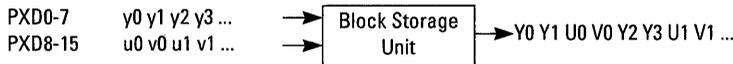
Before the pixels can be processed using JPEG, the pixel blocks themselves must be separated into individual component blocks, each 64 bytes in size. This function is performed by the CL5xx's MCU Block Storage Unit (BSU). The BSU provides four 64-byte RAM queues to separate and sequence the individual component blocks to and from the DCT/IDCT unit. Each 64 ticks of the PXCLK, a new component block is sequenced into the DCT/IDCT unit for processing. The sequencing controls in this unit are hardwired and cannot be altered. The only pixel mode that lends itself to custom sequencing is the Gray-Scale mode. In

this mode, the BSU is effectively bypassed, allowing direct DCT input. Figure 8-28 below describes the internal component sequence orderings for each pixel mode. The left-most symbols are first in time order.

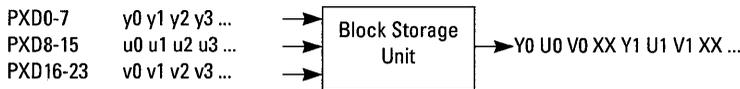
Gray Scale (Bypass) Mode



YUV4:2:2 Mode



YUV4:4:4 Mode (RGB4:4:4)

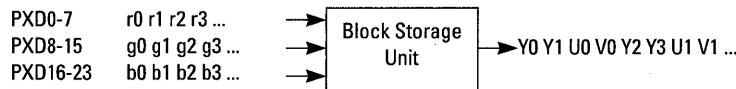


Note: XX designates "Don't Care"; This is an Idle Block Cycle.

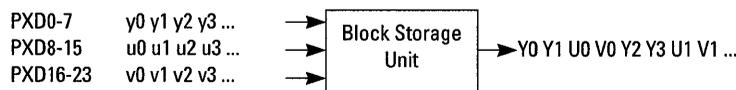
4:4:4:4 Mode



RGB-to-YUV4:2:2 Mode



YUV4:4:4-to-YUV4:2:2 Mode

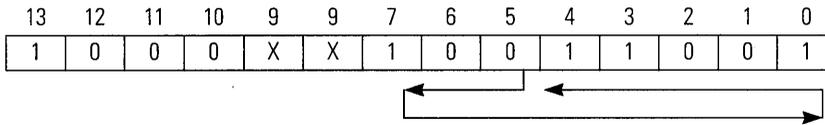


**Figure 8-28 CL5xx Block Storage Unit Component Sequencing**

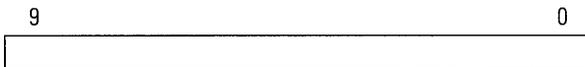
**8.10.3 CL550 Component Sequence Programming**

Several of the CL550's internal processing units have been made programmable to accommodate the various block sequences produced by





**Figure 8-30** Quantizer Y/C Sequence Register Example



**Figure 8-31** Quantizer A/B Sequence Register

This register contains the sequence selection order for the A/B table banks. A zero bit selects the A table bank, and a one selects the B table bank. The method for programming this register is the same as for the Quantizer Y/C Sequence Register. In the YUV4:2:2 example above, only two tables are used, and this register is loaded with zero.

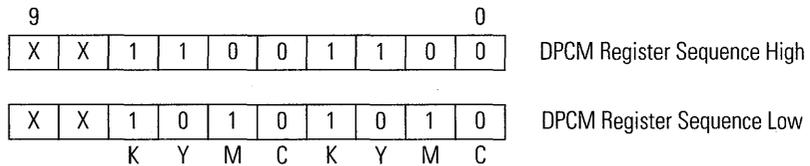
Note: During 4:4:4 mode operations the 'XX' blocks must also be specified in the Y/C and A/B sequence registers, even though the bit has no meaning. This is required as padding in the sequence to keep it synchronized with the BSU.

### *Huffman Table Sequence Registers*

Huffman table selection must also be programmed for each CL550 mode using the Huffman Table Sequence register (Address A000). This register controls the selection of Huffman tables on a block-by-block basis. The register is 10-bits wide, and is write-only. Sequences are specified starting from bit zero, and move to the higher bits. A zero bit selects the Y table set, and a one selects the C table set. The number of bits in the sequence is determined by the sequence length that is programmed into the Decoder Sequence Length register (address A80C). For example, for a sequence YYUVYYUV, the Decoder Sequence Length register is loaded with 1000, and the Huffman Table Sequence Register is loaded with xx11001100. The minimum sequence length allowed is 5, therefore a YYUV block order is specified as YYUVYYUV.

*DPCM Register Sequence Programming*

There are four internal registers that hold the predictor values used when computing the DPCM values for DC terms. Each register holds a predictor for its particular component, and the sequence for selecting these registers must be specified on a block-by-block basis. Two registers are used to specify the table selection sequence. These are the DPCM Register Sequence High (Address A004), and DPCM Register Sequence Low (Address A008) registers. These registers are 10-bits wide, and are write-only registers. They are used in combination to select one of four predictor value registers. For example, bit zero from each register is used to form a two-bit value that selects one of the four predictor value registers. The actual register chosen for a particular component is not important, as long it is used for only one component. For example, consider the sequence CMYKCMYK. The Decoder Table Sequence Length would be loaded with 8, and the DPCM Register Sequence registers would be programmed starting from bit zero:



**Figure 8-32 DCPM Register Sequence**

Note: When programming the Huffman Table Sequence and DPCM Sequence registers for 4:4:4 modes, the 'XX' value of the BSU sequence is not important, and a padding bit is not required. For example, in YUV4:4:4 mode, the Length field of the Quantizer Y/C Sequence Register is programmed for 8, and a sequence field of 8 bits is used, using two padding bits (YUVXYUVX). However, the Huffman Table Sequence register is loaded with 6, and the Huffman table sequence specified follows the sequence YCCYCC, with no padding bits. This is due to the fact that the Huffman Coder/Decoder is separated from the Quantizer by the internal FIFO, and it is not necessary to synchronize to BSU timing.

# Index

## A

absolute maximum ratings, 6-2

AC characteristics, 6-3

AC timing

DRQ, 6-5

HBCLK and RESET parameters,  
6-4

host interface, CPGA package, 6-  
14

host interface, MQUAD package,  
6-15

video interface, 6-16

video interface, CPGA package, 6-  
21

video interface, full-rate compres-  
sion, 6-17

video interface, full-rate decom-  
pression, 6-18

video interface, half-rate compres-  
sion, 6-19

video interface, half-rate decom-  
pression, 6-20

video interface, MQUAD pack-  
age, 6-22

## B

BLANK, 3-10, 5-4

## C

CCIR 601 resolution, 1-2

CIF resolution, 1-2

CL550

absolute maximum ratings, 6-2

AC characteristics, 6-3

applications, 1-3

as still-image coprocessor, 8-7

as video compression processor, 8-  
10

compression concepts, 8-5

custom block sequencing, 8-56

- data rate, 1-2
- DC characteristics, 6-2
- decoder tables, 8-44
- decompression concepts, 8-25
- device reset, 8-38
- DMA system programming example, 8-14
- encoder tables, 8-42
- features, 1-2
- filling the FIFO, 8-27
- functional description, 1-4
- initialization procedures, 8-30
- logic diagram, 3-2
- operating conditions, 6-2
- physical dimensions, CPGA package, 6-24
- physical dimensions, MQUAD package, 6-29
- pinout, CPGA package, 6-26
- pinout, MQUAD package, 6-30
- quantizer table programming, 8-50
- register reset values, 8-39
- slave-mode compression operation, 8-23
- startup sequence, 8-37
- typical system configurations, 8-1
- use of DMA and interrupts with, 8-12
- CL550 and CL560, differences, 4-2
- CL550 and CL560, differences between, 3-3, 8-16
- CL560
  - absolute maximum ratings, 6-2
  - AC characteristics, 6-3
  - applications, 1-3
  - compression concepts, 8-16
  - custom block sequencing, 8-56
  - data rate, 1-2
  - data rate control mechanisms, 8-19
  - DC characteristics, 6-2
  - decompression concepts, 8-28
  - device reset, 8-38
  - DMA transfers, 4-14
  - features, 1-2
  - filling the FIFO, 8-29
  - functional description, 1-6
  - initialization procedures, 8-30
  - logic diagram, 3-2
  - operating conditions, 6-2
  - physical dimensions, CPGA package, 6-24
  - physical dimensions, MQUAD package, 6-29
  - pinout, CPGA package, 6-26
  - pinout, MQUAD package, 6-30
  - quantizer table programming, 8-50
  - register reset values, 8-39
  - slave-mode compression operation, 8-23
  - startup sequence, 8-37
  - synchronous interface, 5-18
  - typical compression system, 8-21
  - typical system configurations, 8-1
  - typical video application, 5-14
- CLK3, 3-11, 5-4
- CMYK, 1-2
- CODEC FIFO, 7-40
- Codec register, 7-39
- Coder Attributes register, 7-21
- Coder Coding Interval registers, 7-22
- Coder Rate Control Active register, 7-24
- Coder Rate Control Enable register, 7-24
- Coder Robustness Active register, 7-25
- Coder RST Padding Control register, 7-25
- Coder Sync register, 7-22
- Coder/Decoder DPCM register, 7-19
- color conversion, 5-12
- color conversion matrix, 8-35
- color transformation matrix, 7-13
- Compressed Word Count registers, 7-

compression ratio  
 range, 1-1  
 compression timing, 5-19  
 Configuration register, 7-28  
 CPGA package  
 physical dimensions, 6-24  
 pinout, 6-26  
 custom block sequencing, 8-56

**D**

DC characteristics, 6-2  
 DCT lookup table, 8-35  
 DCT tables, 7-15  
 DCT, *see* discrete cosine transform, 2-2  
 DCT/IDCT processor, 1-2  
 Decoder Code Order register, 7-27  
 Decoder DPCM Reset register, 7-20  
 Decoder Marker register, 7-26  
 Decoder Resume register, 7-26  
 Decoder Start register, 7-27  
 Decoder Table Sequence Length register, 7-26  
 Decoding Mismatch Error Code register, 7-28  
 Decoding Mismatch register, 7-27  
 decompression timing, 5-29  
 device reset, 8-38  
 discrete cosine transform, 2-2  
 theory, 2-4  
 DMA access timing, 4-12  
 DMA read timing, 4-17  
 DMA Request Interrupt Mask register, 7-35  
 DMA write timing, 4-15  
 DMA\_MSTR, 3-4, 4-12  
 DPCM register sequence programming, 8-60  
 DRQ, 3-4, 4-12  
 AC timing, 6-5

**E**

8-bit grayscale, 1-2

**F**

FIFO Level register, 7-40  
 Flags register, 7-31  
 FRMEND, 3-7  
 FRMEND Enable register, 7-38

**H**

HActive register, 7-9  
 HALF\_FULL, 4-26  
 HALF\_FULL, 3-5  
 HBCLK, 3-7, 4-4  
 AC timing, 6-4  
 HBOUT, 3-4, 4-7  
 HBUS[31:0], 3-3  
 HBUS[31:0], 4-4  
 HBUS\_32, 3-3, 3-4, 4-4  
 HControl register, 7-12  
 HControl registers, programming, 8-34  
 HDelay register, 7-8  
 host bus interface, 1-1  
 host interface  
 AC timing, CPGA package, 6-14  
 AC timing, MQUAD package, 6-15  
 block diagram, 4-2  
 burst-mode read, 4-22  
 burst-mode write, 4-20  
 design considerations, 4-23  
 error conditions, 4-9  
 functional description, 4-1  
 register access timing, 4-3  
 register read, 4-9  
 register write, 4-11  
 signal descriptions, 3-3  
 HPeriod register, 7-7  
 HSYNC, 3-10, 5-4  
 HSync register, 7-8  
 Huffman code tables, 7-20

Huffman Table Load Enable register, 7-20  
Huffman table sequence programming, 8-59  
Huffman Table Sequence register, 7-21  
Huffman tables, 1-2  
    detecting bad, 8-45  
    programming, 8-41  
HV Enable register, 7-13

## I

ID[3:0], 3-4, 4-4  
Init registers, 7-30  
initialization procedures, 8-30  
interface, host bus, 1-1  
interface, video, 1-1, 1-4  
IRQ1, 3-5, 4-26  
IRQ1 Mask register, 7-34  
IRQ2, 4-26  
IRQ2, 3-5  
IRQ2 Interrupt Mask register, 7-37

## J

JPEG, 1-1  
    algorithm, 2-3  
    baseline sequential process, 2-2  
    entropy encoding, 2-8  
    overview, 2-1  
    quantization, 2-5  
    zero run-length coding, 2-7  
JPEG video  
    continuous stream model, 8-4  
    frame-by-frame model, 8-4  
JPEG video concepts, 8-4

## M

MQUAD package, 1-2  
    physical dimensions, 6-29  
    pinout, 6-30

## N

NMRQ, 3-5, 4-26  
NMRQ Interrupt Mask register, 7-33

## O

operating conditions, 6-2

## P

package, MQUAD, 1-2  
package, PGA, 1-2  
PGA package, 1-2  
pixel order conversion, 5-10  
PXADR[15:0], 3-8, 5-3  
PXCLK, 3-11, 5-4  
PXDAT[23:0], 3-8, 5-3  
PXIN, 3-9, 5-3  
PXOUT, 3-9, 5-3  
PXPHASE, 3-11, 5-4  
PXRE, 3-8, 5-3  
PXWE, 3-9, 5-3

## Q

quantizer, 1-2  
Quantizer A/B Table Select register, 7-16  
Quantizer A/B Table Sequence register, 7-18  
Quantizer Sync register, 7-18  
quantizer table programming, 8-50  
quantizer table scaling function, 8-51  
quantizer table sequence programming, 8-58  
quantizer tables, 7-15  
    generating machine-loadable, 8-53  
Quantizer Y/C Table Sequence register, 7-17

## R

register  
    CODEC, 7-39  
    CODEC FIFO, 7-40  
    Coder Attributes, 7-21

- Coder Coding Interval, 7-22
- Coder Rate Control Active, 7-24
- Coder Rate Control Enable, 7-24
- Coder Robustness Active, 7-25
- Coder RST Padding Control
  - Active, 7-25
- Coder Sync, 7-22
- Coder/Decoder DPCM, 7-19
- color transformation matrix, 7-13
- Compressed Word Count, 7-23
- Configuration, 7-28
- DCT tables, 7-15
- Decoder Code Order, 7-27
- Decoder DPCM Reset, 7-20
- Decoder Marker, 7-26
- Decoder Resume, 7-26
- Decoder Start, 7-27
- Decoder Table Sequence Length,
  - 7-26
- Decoding Mismatch, 7-27
- Decoding Mismatch Error Code,
  - 7-28
- DMA Request Interrupt Mask, 7-35
- DPCM, sequence programming,
  - 8-60
- FIFO Level, 7-40
- Flags, 7-31
- FRMEND Enable, 7-38
- HActive, 7-9
- HControl, 7-12
- HControl, programming, 8-34
- HDelay, 7-8
- HPeriod, 7-7
- HSync, 7-8
- Huffman code tables, 7-20
- Huffman Table Load Enable, 7-20
- Huffman Table Sequence, 7-21
- HV Enable, 7-13
- Init, 7-30
- IRQ1 Mask, 7-34
- IRQ2 Interrupt Mask, 7-37
- NMRQ Interrupt Mask, 7-33
- Quantizer A/B Table Select, 7-16
- Quantizer A/B Table Sequence, 7-18
- Quantizer Sync, 7-18
- quantizer tables, 7-15
- Quantizer Y/C Table Sequence, 7-17
- S-Reset, 7-29
- Start, 7-29
- Start of Frame, 7-26
- VActive, 7-9, 7-11
- VControl, 7-12
- VControl, programming, 8-34
- VDelay, 7-10
- Version, 7-30
- Vertical Line Count, 7-12
- Video Latency, 7-11
- VSync, 7-10
- register reset values, 8-39
- registers
  - compression and decompression,
    - 7-15
  - pipeline configuration, 8-36
  - programming video control, 8-32
  - sequence control, 8-35
  - video field, 7-6
  - video interface, 7-6
- RESET, 3-7, 4-25
  - AC timing, 6-4
- RGB, 1-2
- RST interval, selection, 8-36

**S**

- signals, external
  - BLANK, 3-10, 5-4
  - CLK3, 3-11, 5-4
  - DMA\_MSTR, 3-4, 4-12
  - DRQ, 3-4, 4-12
  - FRMEND, 3-7
  - HALF\_FULL, 4-26
  - HALF\_FULL, 3-5

HBCLK, 3-7, 4-4  
 HBOUT, 3-4, 4-7  
 HBUS[31:0], 3-3  
 HBUS[31:0], 4-4  
 HBUS\_32, 3-3, 3-4, 4-4  
 HSYNC, 3-10, 5-4  
 ID[3:0], 4-4  
 ID[3:0], 3-4  
 IRQ1, 3-5, 4-26  
 IRQ2, 4-26  
 IRQ2, 3-5  
 NMRQ, 3-5, 4-26  
 PXADR[15:0], 3-8, 5-3  
 PXCLK, 3-11, 5-4  
 PXDAT[23:0], 3-8, 5-3  
 PXIN, 3-9, 5-3  
 PXOUT, 3-9, 5-3  
 PXPHASE, 3-11, 5-4  
 PXRE, 3-8, 5-3  
 PXWE, 3-9, 5-3  
 RESET, 3-7, 4-25  
 STALL, 3-10  
 STALL, 5-3, 5-7  
 START, 3-6, 4-4  
 TEST, 3-8  
 TM0, 3-6, 4-5  
 TMI, 3-6, 4-5  
 TM2, 3-7, 4-5  
 TMOUT, 3-7, 4-5  
 VSYNC, 3-10, 5-4  
 S-Reset register, 7-29  
 STALL, 3-10  
 STALL, 5-3, 5-7  
 START, 3-6, 4-4  
 Start of Frame register, 7-26  
 Start register, 7-29  
 startup sequence, 8-37  
 synchronous interface, 5-18  
**T**  
 table  
     DCT lookup, 8-35

Huffman, sequence programming, 8-59  
 quantizer, programming, 8-50  
 quantizer, scaling function, 8-51  
 quantizer, sequence programming, 8-58  
 tables  
     DCT, 7-15  
     Huffman code, 7-20  
     Huffman, loading, 8-37  
     Huffman, programming, 8-41  
     quantizer, 7-15  
     quantizer, generating machine-loadable, 8-53  
     quantizer, loading, 8-37  
 TEST, 3-8  
 timing  
     DMA access, 4-12  
     DMA read, 4-17  
     DMA write, 4-15  
 TM0, 3-6, 4-5  
 TMI, 3-6, 4-5  
 TM2, 3-7, 4-5  
 TMOUT, 3-7, 4-5  
**V**  
 VActive register, 7-9, 7-11  
 VControl register, 7-12  
 VControl registers, programming, 8-34  
 VDelay register, 7-10  
 Version register, 7-30  
 Vertical Line Count register, 7-12  
 video field registers, 7-6  
 video interface, 1-1, 1-4  
     AC timing, 6-16  
     AC timing, CPGA package, 6-21  
     AC timing, full-rate compression, 6-17  
     AC timing, full-rate decompression, 6-18  
     AC timing, half-rate compression,

- 6-19
- AC timing, half-rate decompression, 6-20
- AC timing, MQUAD package, 6-22
- block diagram, 5-2
- color conversion, 5-12
- color modes, 5-6
- compression timing, 5-19
- decompression timing, 5-29
- functional description, 5-1
- modes, 5-7
- pixel order conversion, 5-10
- programming control registers, 8-32
- registers, 7-6
- signal descriptions, 3-8
- window management, 5-12
- Video Latency register, 7-11
- video resolution
  - CCIR 601, 1-2
  - CIF, 1-2
- VSYNC, 3-10, 5-4
- VSynch register, 7-10

## **Y**

- YUV, 1-2



# Customer Feedback

C-Cube Microsystems is always working to improve the quality of our documentation. If you have comments or suggestions on this document, please mark up a copy of the page or pages, or send us e-mail. We will acknowledge all comments received. Our address is:

Technical Publications Department  
C-Cube Microsystems  
1778 McCarthy Boulevard  
Milpitas, CA 95035

phone: (408) 944-6300

fax: (408) 944-6314

e-mail: [techpubs@c-cube.com](mailto:techpubs@c-cube.com)

# North American Representatives

---

## Alabama

M<sup>2</sup>I  
1910 Sparkman Avenue  
Huntsville, AL 35816  
phone: 205-830-0498  
fax: 205-837-7049

## Arkansas

TL Marketing  
14850 Quorum Dr., Suite 100  
Dallas, TX 75240  
phone: 214-490-9300  
fax: 214-960-6075

## California

Bager Electronics  
519 Encinitas Blvd.  
Encinitas, CA 92024  
phone: 619-632-8816  
fax: 619-632-8810

Bager Electronics  
17220 Newhope Street, Suite 209  
Fountain Valley, CA 92708  
phone: 714-957-3367  
fax: 714-546-2654

Bager Electronics  
6324 Variel, Suite 314  
Woodland Hills, CA 91367  
phone: 818-712-0011  
fax: 818-712-0160

Norcomp  
2140 Professional Drive, #200  
Roseville, CA 95661  
phone: 916-782-8070  
fax: 916-782-8073

Norcomp  
3350 Scott Blvd.  
Santa Clara, CA 95054  
phone: 408-727-7707  
fax: 408-986-1947

## Colorado

Promotech Sales, Inc.  
2901 S. Colorado Blvd.  
Denver, CO 80222  
phone: 303-692-8484  
fax: 303-692-8416

## Florida

Semtronic Associates  
657 Maitland Avenue  
Altamonte Springs, FL 32701  
phone: 407-831-8233  
fax: 407-831-2844

Semtronic Associates  
1467 S. Missouri Avenue  
Clearwater, FL 34616  
phone: 813-461-4675  
fax: 813-442-2234

Semtronic Associates  
3471 NW 55th Street  
Ft. Lauderdale, FL 33309  
phone: 305-731-2484  
fax: 305-731-1019

## Georgia

M<sup>2</sup>I  
3000 Northwoods Parkway #110  
Norcross, GA 30071  
phone: 404-447-6124  
fax: 404-447-0422

## Hawaii

Bager Electronics  
17220 Newhope Street, Suite 209  
Fountain Valley, CA 92708  
phone: 714-957-3367  
fax: 714-546-2654

## Illinois

Beta Technology Sales, Inc.  
1009 Hawthorn Drive  
Itasca, IL 60143  
phone: 708-250-9586  
fax: 708-250-9592

## Iowa

Cahill, Schmitz & Howe  
226 Sussex Drive. N.E.  
Cedar Rapids, IA 52402  
phone: 319-377-8219  
fax 319-377-0958

## Louisiana

TL Marketing  
14850 Quorum Dr., Suite 100  
Dallas, TX 75240  
phone: 214-490-9300  
fax: 214-960-6075

## Massachusetts

Advanced Technical Services  
348 Park Street, Suite 102  
North Reading, MA 01864  
phone: 508-664-0888  
fax: 508-664-5503

## Minnesota

Cahill, Schmitz & Cahill, Inc.  
315 N. Pierce Street  
St. Paul, MN 55104  
phone: 612-646-7217  
fax: 612-646-4484

## Mississippi

M<sup>2</sup>I  
1910 Sparkman Avenue  
Huntsville, AL 35816  
phone: 205-830-0498  
fax: 205-837-7049

## Montana

Promotech Sales, Inc.  
2901 S. Colorado Blvd.  
Denver, CO 80222  
phone: 303-692-8484  
fax: 303-692-8416

## Nevada

*Clark County Only*  
Bager Electronics  
6324 Variel, Suite 314  
Woodland Hills, CA 91367  
phone: 818-712-0011  
fax: 818-712-0160

## *Other*

Norcomp  
2140 Professional Drive, #200  
Roseville, CA 95661  
phone: 916-782-8070  
fax: 916-782-8073

## **New Jersey**

Parallax  
734 Walt Whitman Road  
Melville, NY 11747  
phone: 516-351-1000  
fax: 516-351-1606

## **New York**

Empire Technical Associates  
29 Fennell Street, Suite A  
Skaneateles, NY 13152  
phone: 315-685-5703  
fax: 315-685-5979

Empire Technical Associates  
349 W. Commercial Street  
Suite 2920  
East Rochester, NY 14445  
phone: 716-381-8500  
fax: 716-381-0911

Parallax  
734 Walt Whitman Road  
Melville, NY 11747  
phone: 516-351-1000  
fax: 516-351-1606

## **North Carolina**

M<sup>2</sup>I  
1200 Trinity Road  
Raleigh, NC 27607  
phone: 919-851-0010  
fax: 919-851-6620

## **North Dakota**

Cahill, Schmitz & Cahill, Inc.  
315 N. Pierce Street  
St. Paul, MN 55104  
phone: 612-646-7217  
fax: 612-646-4484

## **Oklahoma**

TL Marketing  
14850 Quorum Dr., Suite 100  
Dallas, TX 75240  
phone: 214-490-9300  
fax: 214-960-6075

## **South Carolina**

M<sup>2</sup>I  
1200 Trinity Road  
Raleigh, NC 27607  
phone: 919-851-0010  
fax: 919-851-6620

## **South Dakota**

Cahill, Schmitz & Cahill, Inc.  
315 N. Pierce Street  
St. Paul, MN 55104  
phone: 612-646-7217  
fax: 612-646-4484

## **Tennessee**

M<sup>2</sup>I  
3000 Northwoods Parkway #110  
Norcross, GA 30071  
phone: 404-447-6124  
fax: 404-447-0422

## **Texas**

TL Marketing  
14850 Quorum Dr., Suite 100  
Dallas, TX 75240  
phone: 214-490-9300  
fax: 214-960-6075

TL Marketing  
8100 Shoal Creek, Suite 250  
Austin, TX 78758  
phone: 512-371-7272  
fax: 512-371-0727

TL Marketing  
14343 Tory Chase Blvd. Suite I  
Houston, TX 77014  
phone: 713-587-8100  
fax: 713-580-7517

## **Wisconsin**

*Western*  
Cahill, Schmitz & Cahill, Inc.  
315 N. Pierce Street  
St. Paul, MN 55104  
phone: 612-646-7217  
fax: 612-646-4484

## *Eastern*

Beta Technology Sales, Inc.  
9401 W. Beloit Road, Suite 409  
Milwaukee, WI 53227  
phone: 414-543-6609  
fax: 414-543-9288

## **Canada**

Electrosources  
230 Galaxy Blvd.  
Rexdale, ONT  
Canada M9W 5R8  
phone: 416-675-4490  
fax: 416-675-6871

Electrosources  
230 Galaxy Blvd.  
Rexdale, ONT  
Canada M9W 5R8  
phone: 416-675-4490  
fax: 416-675-6871

Electrosources  
340 March Road, Suite 503  
Kanata, ONT  
Canada  
K2K 2E4  
phone: 613-592-3214  
fax: 613-592-4256

Electrosources  
6600 Trans Canada Highway,  
Suite 420  
Pointe Claire, Quebec  
H9R 4S2  
phone: 514-630-7486  
fax: 514-630-7421

## **Others (Not Listed)**

Contact nearest office of  
C-Cube Microsystems

# International Representatives and Distributors

---

## France

NEWTEK (Rep/Dist.)  
8, rue de l'Esterel  
SILIC 583  
94663 Rungis Cedex  
France  
phone: (33) 1-46.87.22.00  
fax: (33) 1-46.87.80.49

## United Kingdom

Kudos Thame Ltd. (Rep/Dist.)  
55 Suttons Park, London Rd.  
Reading, BERKS RG6 1AZ  
United Kingdom  
phone: (44) 734-351010  
fax: (44) 734-351030

## Germany

Metronik GmbH (Rep/Dist.)  
Leonhardsweg 2  
8025 Unterhaching  
Germany  
phone: (49) 89-61108-0  
fax: (49) 89-6116858

## Australia

ZATEK Components Pty Ltd.  
(Rep/Dist.)  
Suite 8, 1059 Victoria Rd.  
West Ryde 2114  
Sydney, Australia  
phone: (61) 2-874-0122  
fax: (61) 2-874-6171

## Hong Kong

MEMEC Asia Pacific  
(Rep/Dist.)  
Unit No 2520-2525  
Tower 1  
Metroplaza  
Hing Fong Road  
Kwai Fong, N.T.,  
Hong Kong  
phone: (852) 410-2780  
fax: (852) 418-1600

## Japan

Kubota C-Cube Inc.  
Fuso Building, 7F, 2-12-8  
Shin-Yokohama  
Kohoku-Ku  
Yokohama, Kanagawa 222  
Japan  
phone: (81) 45-474-7571  
fax: (81) 45-474-7570

## Korea

MEMEC Asia Pacific  
(Rep/Dist.)  
4th Floor, Jae Woong Bldg  
176-11 Nonhyun-Dong  
Kangnam-ku  
Seoul  
Korea  
phone: (82) 2-518-8181  
fax: (82) 2-518-9419

## Singapore

Serial System Marketing  
(Rep/Dist.)  
11 Jalan Mesin  
Standard Industrial Bldg, #06-00  
Singapore 1336  
phone: (65) 280-0200  
fax: (65) 286-6723

## Republic of China

MEMEC Asia Pacific(Rep)  
14F-1, 171 Section 58  
Min Sheng East Road  
Hai Hwa Building  
Taipei  
Taiwan, R.O.C.  
phone: (886) 2 760-2028  
fax: (886) 2 765-1488  
  
ALLY, Inc. (Dist.)  
7F, 18, Alley 1 Lane 768, Sec. 4  
Pa Teh Rd.,  
Taipei  
Taiwan, R.O.C.  
phone: (886) 2 788-6270  
fax: (886) 2 786-3550

# C-Cube Microsystems Sales Offices

---

## Home Office

C-Cube Microsystems  
1778 McCarthy Boulevard  
Milpitas, CA 95035  
phone: 408-944-6300  
fax: 408-944-6314

## Eastern Area Office

C-Cube Microsystems  
One Kendall Square, Suite 220  
Cambridge, MA 02139  
phone: 617-621-7180  
fax: 617-621-7179

## European Office

C-Cube Microsystems  
44 Dartford Road  
Sevenoaks, Kent  
UK TN 133 TQ  
phone: (44) 732 743 256  
fax: (44) 732 450 151



**C-Cube  
Microsystems**

1778 McCarthy Blvd.  
Milpitas, CA 95035  
Tel: (408) 944-6300  
Fax: (408) 944-6314