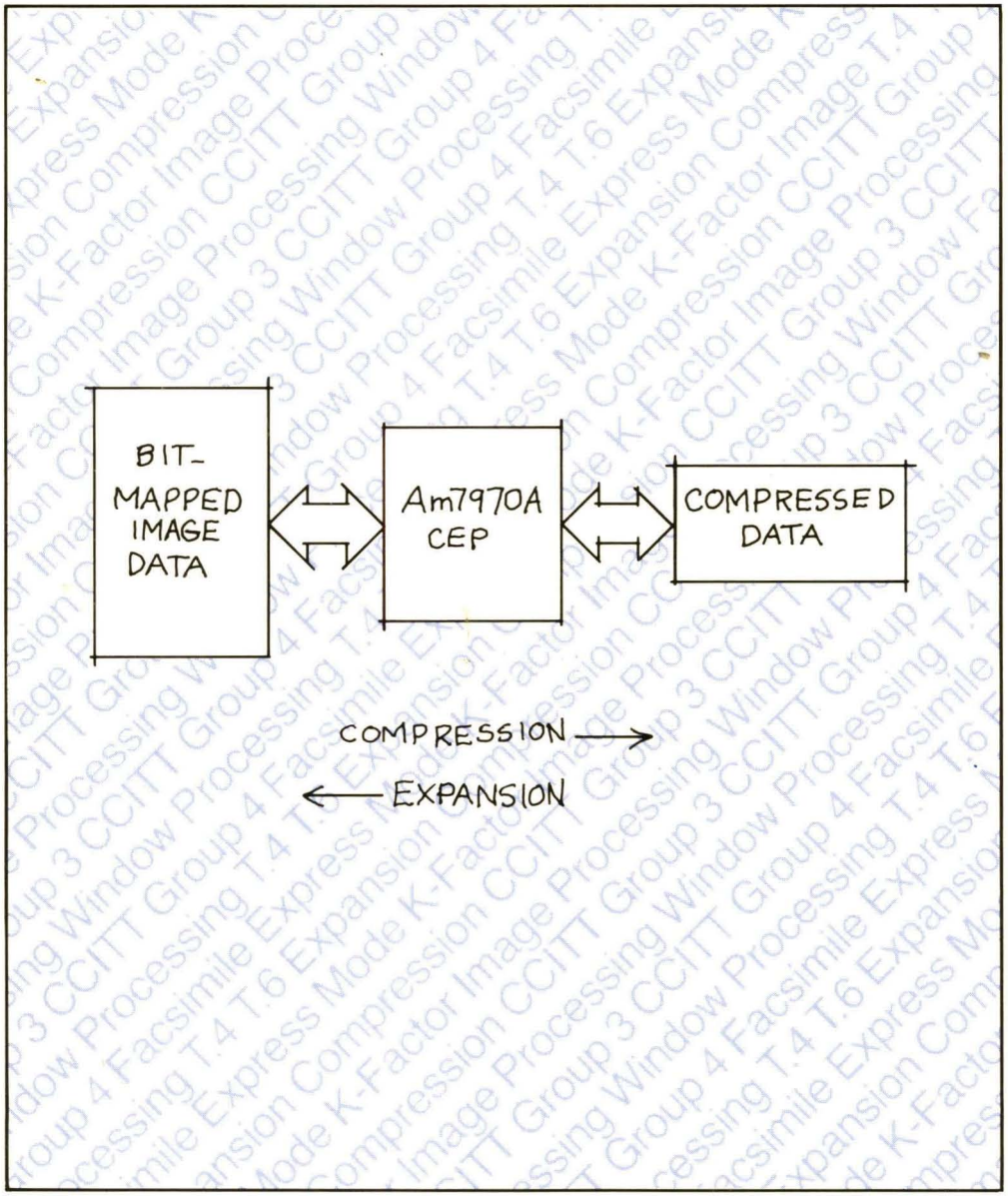
Compression Expansion Processor
Am7970A

Technical Manual



ADVANCED MICRO DEVICES

Advanced Micro Devices

Am7970 Compression Expansion Processor

The International Standard of Quality guarantees a 0.05% AQL on all electrical parameters, AC and DC, over the entire operating range.

INT·STD·500

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

ACKNOWLEDGEMENTS:

This technical manual was written by James Williamson, Field Applications Engineer and Wolfgang Kemmler, Headquarters Applications Engineer. The Senior Technical Writer for this manual is Erland Kyllonen.

Section 4.2 was contributed by Deyoung Hong, Software Engineer.

Peter Alfke, Director of Applications and Joseph Brcich, Manager of Headquarters Applications contributed to the final revisions of this manual.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

1.1 GENERAL DESCRIPTION

The Am7970A Compression/Expansion Processor (CEP) is a high-performance peripheral which compresses and expands two-tone bit image data in accordance with the International Telegraph and Telephone Consultative Committee (CCITT) recommendations. These image-preserving compression protocols allow highly efficient storage and transmission of two-tone pictures and documents without loss of information.

Using advanced one- and two-dimensional compression algorithms, the Am7970A is able to represent a one megabyte document in an average of 64K bytes of storage, a reduction ratio of 15:1. In many cases, the compression ratio is 30:1 and higher. In addition to the memory space saved, this compression applies the same saving to the transmission time. Thus, a document that requires 15 minutes to transmit at 9.6 kb/s requires less than one minute with compression. Typical compression of the eight CCITT test documents is 5x to 50x. The compression ratio varies with the compression mode and the amount of image detail on the document. Tables are presented in the discussion of the coding concepts to show the compression ratios that can be expected for various modes of operation.

Paralleling the use of compression/decompression in the facsimile environment, there are image processing requirements in the commercial office. Generally speaking, these requirements have many similarities to those of facsimile. It is necessary to have the ability to create, capture, view, edit, print and communicate images. The communication of these images may take the form of "hard copy" (facsimile) or "display" via a CRT terminal. Further requirements of such systems are the ability to modify images in content, shape, and size, as well as, to incorporate image data with other forms of information (for instance text). Figure 1-1 shows a document processing workstation. It shows the environment in which the CEP may be used.

The CEP has a standard Am8088/8086-like microprocessor bus interface which is easily adapted to a regular microprocessor interface. CEP operation is set by programming internal control registers. CEP status is available through polled registers; exception conditions may be signaled using an external interrupt. The 42 on-chip registers allow a simple and highly flexible system implementation. After initialization, the CEP processes data with minimal intervention by the host processor.

The Am7970A CEP includes a secondary, local Document Store bus for optional use in conjunction with the CPU bus. The local storage buffer is highly desirable within many system architectures to optimize CPU bus performance. The CEP can linearly address up to 16Mbytes of memory on each bus, for a total of 32 Mbytes. Starting address, buffer length, and current address for raw and processed data are stored within internal registers independently for both the Compressor and the Expander.

The Am7970A performs modified Huffman one-dimensional coding or modified Relative Element Address Designate (READ) two-dimensional coding. This is compatible with CCITT recommendations T.4 and T.6 for Group 3 and Group 4 digital facsimile apparatus.

The Compressor and Expander, which operate in full-duplex, can be independently programmed for one-dimensional encoding/decoding, two-dimensional encoding/decoding, or transparent data transfer. In two-dimensional operation, the programmable K-Parameter defines the number of lines to be encoded in each two-dimensional coding sequence. For error-less systems (Group 4), "K = infinity" allows maximum compression by coding all lines two-dimensionally. Transparent Mode is provided to move data from one memory area to another using the DMA on the CEP.

Accelerated image processing is supported with a Compressor Express Mode which skips one line for every n lines compressed. The Expander counterpart is a Granularity Mode which duplicates the last line expanded after each n lines that are expanded. To expand the document to the same size as the original, the n value must be the same for both compression and expansion.

Document format controls include line length and margins. Line lengths or document widths of up to 16K picture elements may be selected. Programmable top, left, and right margins specify "white space" around image data, supporting both normal margin requirements and also "windowing", defined as overlaying of multiple image blocks or image blocks and character blocks.

1.2 FEATURES

- Compression/Expansion of digital two-tone image data using run-length and relative coding.
- Compatible with CCITT recommendations T.4 and T.6 for Group 3 and Group 4 facsimile apparatus.
- One-Dimensional, Modified Huffman Coding with optional Wraparound Mode.
- Two-Dimensional, Modified READ Coding with programmable K-Parameter.
- General-purpose microprocessor interface.
- Optional local Document Store bus.
- On-chip, dual-bus DMA controller.
- Transparent transfer of unmodified data.
- 16-Mbyte physical addressing range on each bus.
- Programmable paper width up to 16K picture elements.
- Programmable top, left, and right margins.
- Window capability
- Optional Express Mode during compression and Granularity Mode during expansion.
- Full-duplex capability for simultaneous independent compression and expansion.
- High-performance 2 to 8 Mbps throughput with a 5-MHz clock.

The Am7970A is packaged in a 68-pin LCC or Pin Grid Array and uses a single +5 V power supply.

1.3 CCITT STANDARDS

Standards for graphical data compression have been developed and agreed upon by the CCITT. These standards define the document representation, the coding alternatives, the encoding algorithms, and transmission requirements for Facsimile operation. Document definition is discussed in the next Section of this chapter. Chapter 3 is devoted to discussing coding concepts and encoding. These standards are also included as Appendix F.

The standards are divided into four groups to address various categories of equipment. Groups 1 and 2 are for old relatively slow analog equipment and are not discussed in this manual. Groups 3 and 4 provide for both one-dimensional and two-dimensional digital coding, and the inclusion of uncompressed text.

The CEP (Am7970) adheres to the T.4 and T.6 standards recommendations set forth by the CCITT Group 3 and Group 4 committees, respectively. The recommendations establish compatibility among manufacturers of facsimile equipment.

The CCITT compression and expansion techniques are based on the modified Huffman and modified READ codes. In compressing the data, only the image redundancies are removed so that the image is preserved without degradation.

Group 3 (T.4) facsimile standards for document transmission specify the apparatus requirements to enable an ISO A4 document (similar to an 8 1/2 by 11 inch page) to be transmitted over a telephone-type circuit in less than one minute.

In Group 3, the total coded scan line is defined as the sum of the data bits plus any required fill bits (zeros) plus the EOL code (in one-dimensional coding). The EOL code is 000000000001. For two-dimensional coding, the scan line includes all of the above plus a tag bit to specify whether the next line is coded one-dimensionally or two-dimensionally. A scan line is 1728 pixels long. Alternative optional lengths are provided as specified in detail under Document Definition.

In Group 4, the total coded scan line is the sum of the data bits. Fill bits and EOLs are not used except for the last line which is terminated by a sequence of two EOLs and a pad of zeros as needed to end the document on a byte boundary. Coding formats are defined and explained in detail in Chapter 3.

The various transmission rates and communication handshakes for this equipment are specified in the CCITT recommendation T.30. This recommendation specifies modem bit rates according to V.27 for (4800/2400 bps) and/or V.29 (9600/7200 bps). T.30 also specifies V.21 (300 bps) initial interrogation between facsimile equipment to assure compatibility via a preliminary interchange of information.

The minimum transmission time of each total coded scan line is also specified in the CCITT recommendations (refer to Appendix F).

1.4 DOCUMENT DEFINITION

Image data is rapidly becoming an important part of computer data storage and communication. The most common unit of image data is the document, an eight and a half by eleven inch area. This area is divided into many small areas called picture elements (pixels).

Picture elements are of uniform size and are scanned from left to right and from top to bottom as seen when viewing the document held in a vertical plane. The resolution possible is determined by the size of the picture elements. Each pixel is

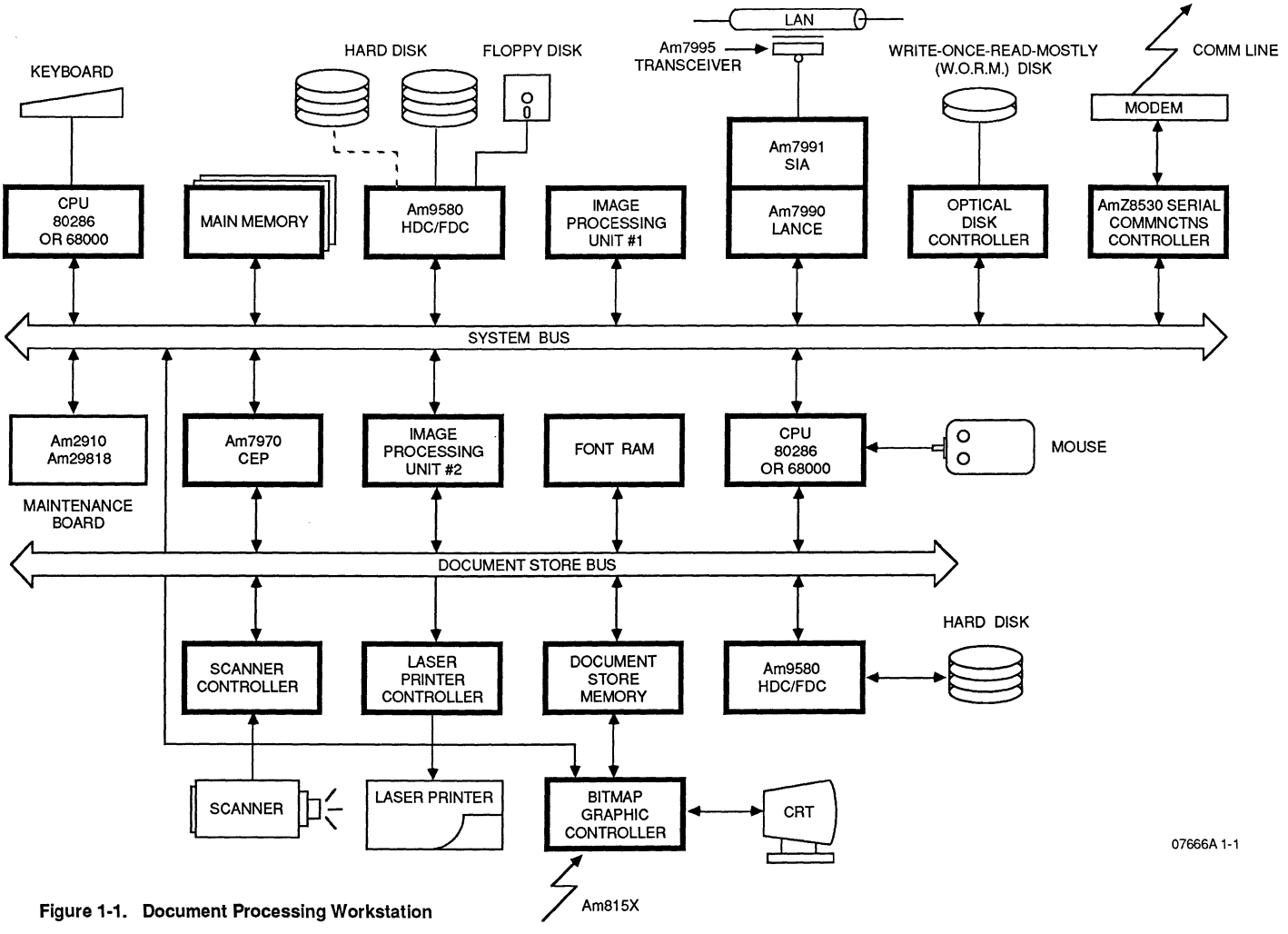


Figure 1-1. Document Processing Workstation

represented as a single binary bit of color information or data in a document image. Therefore, colors are limited to two tones (black and white or other color pairs).

In memory, the bits representing the pixels are combined into bytes. The first pixel at the top left edge of the image must be stored as the least significant bit of the first byte in the memory buffer. This is also the first bit to be sent on a transmission line. The bits of each byte are transmitted serially. The compressed (coded) image follows the same rule. If this rule is violated, additional color changes may be created completely upsetting the compression statistics and reducing the compression ratio. There is no standardization on how a scanner has to present the data.

A standard scan line in a Group 4 document is 215 mm (8.46") long and contains 1728 pixels (same as Group 3). All scanning is from left to right and from top to bottom when viewing the document in a vertical plane. Optionally, the line may be 255 mm long and contain 2048 pixels or it may be 303 mm long and contain 2432 pixels.

CCITT Group 4 standard sizes have been established for the pixels. There may be 200, 300, or 400 pixels per inch horizontally and from 100 to 400 pixels per inch vertically. The number of pixels per inch determines the resolution obtainable.

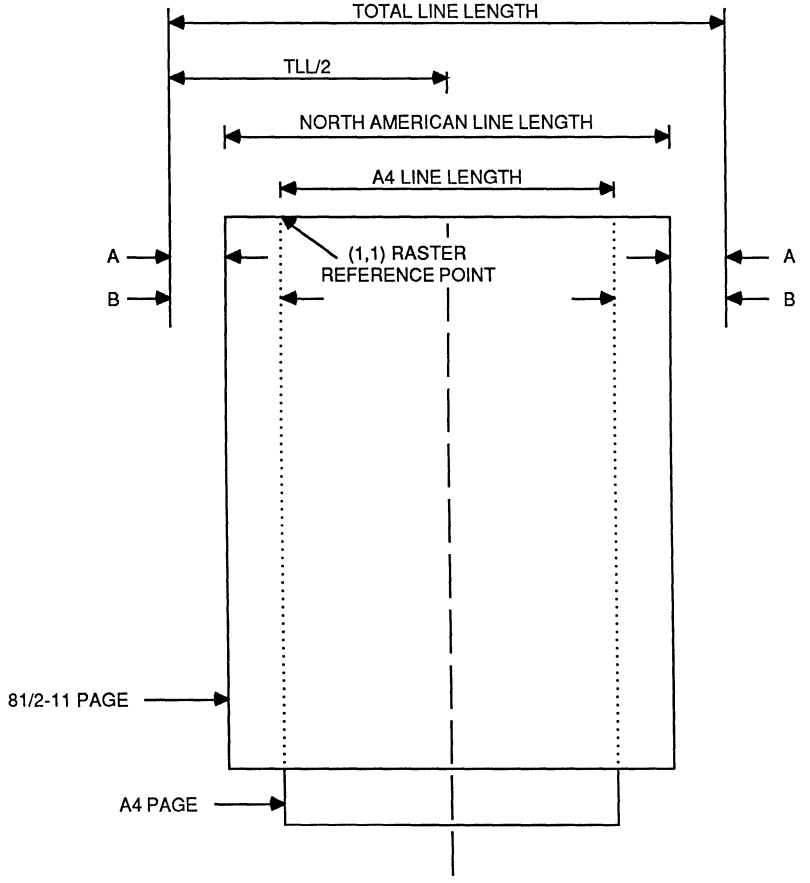
The Group 4 standard of 200 lines per inch means that a line containing 1728 pixels is 8.64 inches long whereas North American letter size paper is 8.5 inches wide.

In the vertical direction, the standard resolution is 100 lines/in (3.85 lines/mm) with an optional resolution of 200 lines/in (7.7 lines/mm). Resolutions of 300 and 400 lines/in. are also allowed in the vertical direction. For comparison, a television picture that is 8 1/2 inches wide and has a 4 MHz video bandwidth has a resolution of 60 to 90 pixels/in. Refer to Figure 1-2.

In addition to specifying the pixel size, one can also specify the left, right, and top margins. Thus, an area of information in an all white (or all black) field can be sent or stored by specifying the margins to include all of the white field above, to the left, and to the right of the image. Figure 1-3 shows the white margins that may be specified within a document.

The memory used to store the document image is called a frame. It may be the same width as the document or it may be wider. Figure 1-4 shows the frame and the document or page within the frame. It shows some of the registers used to define the image area location within the frame. These registers are defined in Chapter 2.

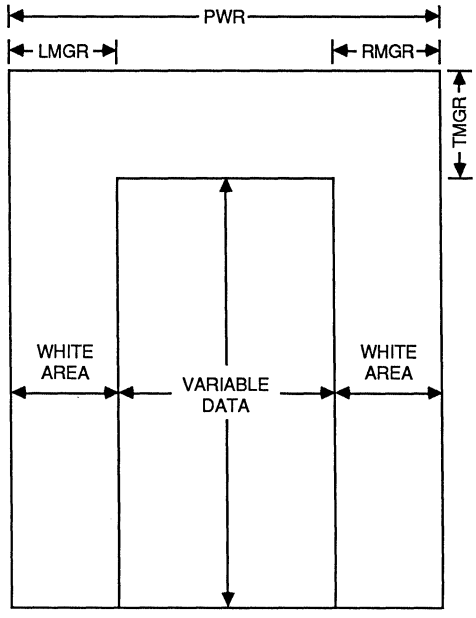
The document is actually a window within the frame. This window may be moved within the frame by manipulating the registers defining its location. Thus, cutting and pasting of information on the display screen can be implemented efficiently.



RESOLUTION (PPI)	PELS PER LINE	81/2x11 LINE (PELS)	A4 LINE (PELS)	BLANKED A (PELS)	MARGINS B (PELS)	ADDRESS (1,1) REF POINT
200 x 200	1728	1700	1654	14	37	(38.1)
300 x 300	2592	2550	2480	21	56	(57.1)
400 x 400	3456	3400	3308	28	74	(75.1)

07666A 1-2

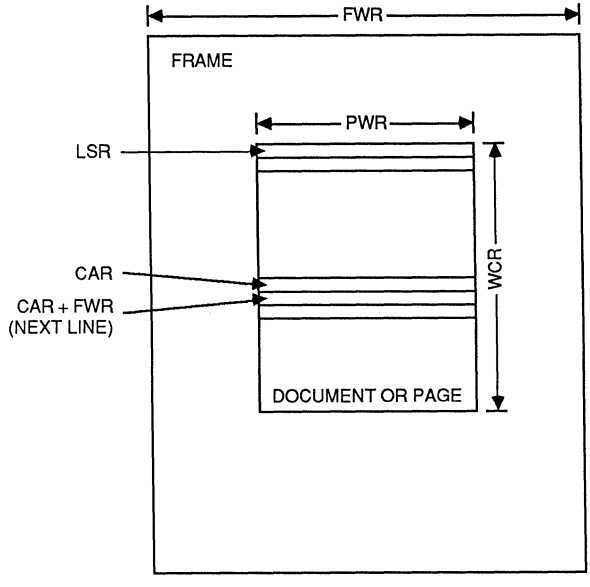
Figure 1-2 Document Format



PWR = PAGE WIDTH REGISTER
LMGR = LEFT MARGIN REGISTER
RMGR = RIGHT MARGIN REGISTER
TMGR = TOP MARGIN REGISTER

Figure 1-3 Document Margins

07666A 1-3



FWR = FRAME WIDTH REGISTER
CAR = CURRENT ADDRESS REGISTER
LSR = LINE START REGISTER
PWR = PAGE WIDTH REGISTER
WCR = WORKING COUNT REGISTER

Figure 1-4 Document Storage

07666A 1-4

Chapter 2
FUNCTIONAL DESCRIPTION

This functional description includes the operational description, register description, and the interface description.

2.1 OPERATIONAL DESCRIPTION

CEP operations consist of three phases: initialization, operation, and termination. In the first phase, the registers (compressor or expander processor) are initialized to specify and control the desired operation. In the second phase, the processing operation itself is started and performed. The final phase involves terminating the selected processor and performing any actions that are appropriate to that termination. These phases are described in detail in the following sections.

The Am7970A contains two separate buses—the System bus and Document Store bus. One DMA Controller on the CEP chip serves both buses. Therefore, DMA data transfers cannot take place on both buses at the same time. However, slave transfers can occur on the system bus while a DMA transfer is taking place on the Document Store Bus. Data transfers between the Am7970A and Main Memory take place on the System bus. Data transfers between the Am7970A and the Document Store Memory take place on the Document Store Bus.

The Am7970A processes two types of data; uncompressed or image data and coded or compressed data. Image data is stored in that portion of memory called the Image Buffer. Compressed data is stored in a portion of memory called a Code Buffer. In an Am7970A system, the Code and image Buffers are external to the CEP and each can be located in either the Main Memory or the Document Store in any combination.

Consideration should be given to the assignment of the buffers to memory. All control information exchanges between the Am7970A and the host processor take place on the System bus. Because of the high data rate of image data, it is recommended that the Image Buffer be placed in the Document Store so that it can be accessed without slowing down the CPU by contention for the DMA. For maximum performance, the Image buffer should be large enough to store one uncompressed document. The Code Buffer can be placed in the Main Memory so that the CPU can access it rapidly during transmission or reception of data. Since the compressed code is considerably smaller than the image data, it does not seriously slow down the system bus and thus impact the CPU.

The Am7970A contains registers to specify the starting address and assigned length of both the Image Buffer and the Code Buffer. The Am7970A Compressor is completely independent of the Expander. The Compressor takes image data from its Image Buffer and loads the resulting compressed data into its Code Buffer. The Am7970A Expander takes compressed data from its Code Buffer for processing and loads the resulting image data into its Image Buffer. In an Am7970A system, the Compressor can be operating from its Image and Code Buffers while the Expander is simultaneously using its own buffers.

For certain images (such as half tone), the compressed data representing a line may be longer than the original line of the image. This is called negative compression. The Am7970A checks for this condition after compressing a line and alerts the host processor via an interrupt and a status bit.

Each compressed line may be delimited by an End of Line (EOL) code according to the CCITT recommendation for Group 3 facsimile apparatus. However, this automatic EOL insertion can be suppressed by appropriate bit settings of the Am7970A (EOL=1).

The CCITT recommendation T.4 for Group 3 equipment requires that each coded line be a certain minimum length. Fill bits are added by the CEP to a shorter line when necessary to meet this requirement. The Am7970A contains a Time Fill Register to specify the minimum line lengths (including zero).

Data is vulnerable to modification by transmission errors. When erroneous data is expanded, the resulting image is very different from the original. The Am7970A checks the expanded line for the number of picture elements required by the specified paper width. If there is a discrepancy, the CPU is alerted via an interrupt. In Group 4 mode, error-free transmission is assumed.

2.1.1 Initialization

The Am7970A has the following initialization requirements:

- Source Buffer definition
- Destination Buffer definition
- Attributes
- Control Parameters

These requirements are met by writing appropriate information into the 42 registers in the CEP. These registers are discussed in detail in this chapter. The following discussion is a summary of the information in these registers as it pertains to initialization. The system program should specify certain initial conditions before starting the operation of the Am7970.

Source Buffer Definition

A Source Buffer is defined by specifying which memory it is in (Main Memory or Document Store), the starting address, the width, and the capacity of the Source Buffer. To specify a source buffer in the Main Memory, the system program must load "0" into the CSC/ESC bit in the Compressor Master Control Register (CMCR)/Expander Master Control Register (EMCR). If a Source Buffer is located in the Document Store, the system program must load a "1" into the CSC/ESC bit. The width of the source buffer memory is stored in the Compressor Frame Width Register CFWR. The EFWR stores the width of the Expander Destination Buffer. Frame width is not applicable to data in the compressed form.

The system program must load a starting address into the Source Address Holding Registers (CSAHR, ESAHR) and the Source Current Address Registers (CSCAR, ESCAR). Also, the system program must load the negative two's complement of the length (in bytes) of the Source Buffer into the Source Count Holding Registers (CSCHR, ESCHR) and the Source Working Count Registers (CSWCR, ESWCR). Additional requirements of the Source Buffer are discussed under the specific source register sections in this chapter.

Destination Buffer Definition

The residency, the starting address, the width, and the capacity of the Destination Buffer must be specified. An "0" in the CDC/EDC bit in the CMCR/EMCR register specifies that the Destination Buffer is located in the Main Memory; a "1" specifies it is in the Document Store. The starting address is specified in the Destination Address Holding Registers (CDAHR, EDAHR) and Destination Current Address Registers (CDCAR, EDCAR). The width is stored in the Expander Frame Width Register EFWR. Destination buffer width is only meaningful for the Expander.

The negative two's complement of the length (in bytes) of the Destination Buffer must be loaded into the Destination Count Holding Registers (CDCHR, EDCHR) and the Destination Working Count Register (CDWCR, EDWCR). The length of the Destination Buffer has some conditions that are are discussed in detail later in this chapter. The system program should adhere to those recommendations.

Attributes

The system program must set the Source Attribute bit, SA, in the Parameter Registers (CPR/EPR) when the CEP is to process a new page. The Data Format Control (DFC) bits in the Parameter Register specifies the compressed data format (byte boundaries and the RTC and EOL suffix codes).

If the automatic insertion of an EOL code is required, the system program must load "0" into the EOL bit in the Compressor Parameter Register. If this bit is "1," automatic insertion of EOL is suppressed.

The system program must load "0" into the EOL bit in the Expander Parameter Register (EPR) when data with attached EOL is going to be expanded. If the data that is to be expanded contains no EOL codes (except at the end of a page), the system program must load "1" into the EOL bit in the EPR register.

The system program specifies the Wraparound, Express, and the Top, Left, and Right Margin options by loading the corresponding registers. The horizontal pixel count is specified in the Page Width Registers (CPWR, EPWR). The width of the memory buffer used to store the picture image is loaded into the Frame Width Registers (CFWR, EFWR). If window processing is used, the width of the window is stored in the Page Width register. Otherwise, the Frame Width Register and the Page Width Register have the same values.

Control Parameters

The operating mode, operation control, interrupt enable, and start/stop are loaded into the Master Control Registers (CMCR, EMCR). The operating modes are: One-dimensional, Two-dimensional, and Transparent. The K-Parameter is specified in the K-Parameter Register when Two-Dimensional processing is required. The granularity option, the G-Parameter, is specified in the Expander Parameter Register.

The GO bit combined with the OC field (bits 0, 1, and 2) in the CMCR or EMCR specify whether

compression or expansion processing or the reset operation for the compressor or expander is to be performed. If 001 is specified in bit positions 2, 1, and 0, the reset operation is executed. 101 specifies multi-line processing and 011 specifies that single-line processing is to begin. The minimum transmission time requirement is loaded into the Time Fill Register (TFLR).

A "1" in the Compressor Interrupt Enable bit (CIE) in the CMCR or in the Expander Interrupt Enable bit (EIE) in the EMCR specifies that an interrupt request is required upon CEP termination. A "0" in this location specifies that an interrupt request is not required.

Compression processing starts as soon as a "1" is loaded into the GO bit in the CMCR. Expansion processing starts as soon as a "1" is loaded into the GO bit in the EMCR.

The Restart Control Registers (CRCR, ERCR) specify whether to continue with the current values or to restore the starting values for the Source and Destination Current Address Registers, Working Count Registers, and Line Start Registers when a new processor operation is initiated.

Line Termination (LT) bits in the CPR are used to specify how many bits of terminating image to add to the end of each line after the last full byte of data. The termination bits for each line have the same value as the last bit of the last byte on that line.

All of the registers are described in detail in this chapter. The recommendations made in this chapter must be followed for initialization.

2.1.2 Start Processing Procedures

The Am7970A has two different operating configurations. In the full-duplex mode, the Expander and the Compressor are operated simultaneously. In the half-duplex mode, either the Expander or the Compressor may be operated separately. A "1" in the GO bit of the CMCR initiates compression. A "1" in the GO bit of the EMCR initiates expansion. For full-duplex operation, load a "1" into the GO bit of each register.

Entire images may be compressed or expanded in one operation if the code buffer and the image buffer are both large enough to contain the entire image. In this case, each start is a start to process a new page and the system program must specify a RESET operation before each start. The reset operation flushes the internal pipeline, sets "busy" to zero, sets up the check for configuration errors, clears status and interrupt registers, and sets the GO bit to zero.

If the code buffer is not large enough during compression to contain the code for an entire image or page, the CEP will stop when the buffer is full. Then, after the coded data is saved, compression can continue without issuing a reset.

If the image buffer is not large enough during expansion to contain an entire image or page, the CEP will stop when the buffer is full. Then, after the image buffer data is saved, CEP processing is resumed without issuing a reset. Thus, compression and expansion are possible using buffers too small to store an entire page.

2.1.3 How To Use The Status Registers

The CEP has three status registers: the Master Status Register (MSR), the Compressor Status Register (CSR), and the Expander Status Register (ESR). Bits 6 and 7 (EBY, CBY) in the MSR provide general status information to the CPU about both the Compressor and Expander. These bits are known as the Expander Busy (EBY) bit and the Compressor Busy (CBY) bits.

If interrupts have not been enabled, the system program should periodically poll EBY and CBY in the MSR register. If the system program is enabled to respond to an interrupt, it should test the EBY bit and the CBY bit in the MSR after a CEP interrupt occurs.

Bits 0 to 3 (EXT, ECD) in the MSR indicate whether a non-CCITT uncompressed mode entry code was detected during expansion. Bit 4 in the MSR indicates that the Expander detected an End of Page (EOP) code. If the system program requires detailed status information, it should test the CSR or the ESR directly.

2.1.4 Interrupt Handling

The Am7970A will drive its interrupt line (INTR) High when the CBY bit in the CSR or the EBY bit in the ESR changes from "1" to "0" while the Compressor Interrupt Enable (CIE) bit in the CMCR or the Expander Interrupt Enable (EIE) bit in the EMCR has been set to "1." The INTR line will remain High until the MSR has been accessed by the system program. The system program may test the MSR register to distinguish Compressor interrupts from Expander interrupts. The system program should isolate the cause of the interrupt by reading the appropriate status register (CSR or ESR). Reading the status register clears the interrupt. The system program may then execute its interrupt service routine to respond to the interrupt.

Compressor Error Recovery Procedures

The Compressor detects several error conditions: a premature source overflow, a premature destination overflow or an illegal command. An error condition is also detected if a new command is attempted while the Compressor is busy or if negative compression occurs. A *premature* source or destination overflow is indicated by the Wraparound Incomplete (WPI) bit or the Line Processing Incomplete (LPI) bit of the Compressor Status Register (CSR).

The error recovery procedure for an LPI error must include the redefinition of the Source Buffer or the Destination Buffer as follows:

1. Premature Source Overflow

```
New CSCHR = N * Hr * Apw/8
New CSWCR = 2's complement of new CSCHR
          - old CSCHR
New CSCAR = CSLSR, new CDCAR = CDLSR
CRCR = All one (X'FF')
New CDWCR = old CDWCR = (CDCAR - CDLSR)
```

2. Premature Destination Overflow

```
New CDCHR = N * Hr * Apw/8
New CDWCR = 2's complement of new CDCHR
         - old CDCHR
New CDCAR = CDLSR, new CSCAR = CSLSR
CRCR = All one (X'FF')
New CSWCR = old CSWCR - (CSCAR - CSLSR)

where:

N   = line count
Hr  = Horizontal resolution
Apw = Actual page width
```

The error recovery procedure for a WPI error without an LPI error may include restarting the Source Buffer or the Destination Buffer without specifying Wraparound Restart (WRC) and/or Two-Dimensional Restart (TDC). If an illegal command is detected, the system program should load a Continue Operation into the CEP or issue a new command to the CEP when the Compressor or Expander Busy and New Operation Attempted error (COA or EOA) is detected. If negative compression is detected, the system program may load a Continue Operation command into the CEP to accept the data as is or the system program may replace the line with uncompressed data using the transparent mode in the CEP.

Expander Error Recovery Procedures

The Expander will detect several error conditions: a premature source overflow, a premature destination overflow, an illegal command, an Expander Busy and New Operation Attempted error, a data error, or an undefined extension code. A premature source or destination overflow is indicated by the WPI bit and the LPI bit of the Expander Status Register (ESR). The error recovery procedure for a premature overflow requires that the Source Buffer and the Destination Buffer be redefined as follows:

1. Premature Source Overflow and LPI without WPI

```
New ESCHR = N * Hr * Apw/8
New ESWCR = 2's complement of new ESCHR
          - old ESCHR
ERCR = All one (X'FF')
New EDWCR = old EDWCR - (EDCAR - EDLSR)
New ESCAR = ESLSR, new EDCAR = EDLSR
```

2. Premature Destination Overflow and LPI without WPI

```
New EDCHR = N * Hr * Apw/8
New EDWCR = 2's complement of new EDCHR
          - old EDCHR
New EDCAR = EDLSR, new ESCAR = ESLSR
ERCR = All one (X'FF')
New ESWCR = old ESWCR - (ESCAR - ESLSR)
```

If the system program detects a premature overflow and the WPI error bit is set without the LPI error bit being set, the system program should restart the Source or Destination Buffer without loading the Wraparound Restart (WRC) and Two-Dimensional Restart (TDC) bit. If an illegal command is detected, the system program should load Continue Operation into the CEP or reissue a new command to the CEP when the Expander Busy and New Operation Attempted error EOA is detected.

If a data error is detected, the system program should replace the error line with a copy of the previous line as follows:

```
New ESCAR = EDLSR N * Hr * Apw/8
N = EWR + 1
New ESWCR = 2's complement N * Hr * Apw/8
New EDCAR = EDLSR
New EDWCR = EDWCR - (EDCAR - EDLSR)
ERCR = All one (X'FF')
EM0 and EM1 = 0 (Transparent Mode)
```

Appendix H gives additional information about expander error recovery.

2.1.5 Stopping The Cep

The CEP compressor may be terminated by writing

to the Compressor Master Control Register while the CEP is busy (called a soft abort). The expander may be terminated by writing to the EMCR while the CEP is busy. The Compressor or Expander will terminate its operation as soon as the internal operation allows it. If the system program is required to stop immediately, the system program should assert the RESET input of the Am7970A. This is called a hardware stop. If the system program executes a hardware stop, the CEP will not save the current status. If the system program executes a software stop, the CEP will terminate its operation (as soon as its internal operation allows it) and keep the Compressor Busy and the New Operation Attempted (COA) or Expander Busy and New Operation Attempted (EOA) status bits; however, this is *not* a resumable operation.

2.2 Register Description

The CPU cannot instantaneously or directly access the CEP internal registers because that would interfere with the CEP's internal operations Instead, a slave access is used to interrupt the internal microprogram. After that, all data transfers to and from the registers are performed by a microprogram. By holding READY Low, the CEP keeps the CPU waiting during this time. (The only exception is a read on the Master Status Register which is directly accessible by the CPU.)

The access time of the registers varies widely for two reasons:

1. The access time depends on the status of the operation that the CEP is currently performing.

2. Access times are optimized with respect to the probability of their usage.

The first statement means that register access time is unpredictable when the CEP is busy. This is important since the access time may be as long as 50 clock cycles. This may have an impact on system design considerations. Typical access times are:

Write Operation with CEP in Idle State:

Case 1, A single write once in a while:

4 clock cycles for all registers.

This write access is internally latched. The addressed register is loaded with the data long after the CPU is released.

Case 2, a sequence of consecutive slave write accesses:

16 clock cycles for paper width, parameter, and command registers.
14 clock cycles for all other registers.

Read Operation with CEP in Idle State:

All cases:

4 clock cycles = MSR only
10 clock cycles = status, parameter, command, and paper width registers.
12 clock cycles = all other registers.

All Operations with CEP Busy:

4 clock cycles for MSR read
All other accesses take an unpredictable number of clock cycles up to 50 depending on the current operation being performed by the internal microprogram.

The block diagram for the Am7970A (CEP) is shown in Figure 2-1. Tables 2-1 and 2-2 list the Compressor and Expander registers respectively. The size and port access address of each register is listed. All CEP registers are located on even boundary addresses. A_0 is completely disregarded for slave accesses.

Registers that are unique to the Compressor are discussed first. These are the Time Fill, Left Margin, Right Margin, Top Margin, and the Express Mode Registers. Then, the Master Status Register (MSR) which is common to both the Compressor and Expander, is discussed. This is followed by a discussion of pairs of registers one of which is in the Compressor and the other in the Expander starting with the Compressor Status Register and the Expander Status Register. The Compressor register is described first, and if the Expander register of the pair is different, the differences are then described. In most of these pairs of registers, the registers are identical. The registers are discussed in the order presented in the Compressor register table.

After initialization by the RESET input, the state of the Status Register, Master Status Register, and GO bits are "0", the status of other bits is not specified.

2.2.1 Time Fill Register (TFLR)

The Time Fill Register, an 8-bit Compressor register, specifies the minimum length of a coded line expressed in bytes. If the compressed line has fewer bytes than this number, time fill bits must be added to meet this requirement. Time fill bits are simply all "0"s. Refer to Figure 2-2.

Table 2-1. Compressor Registers

Abbr.	Name	Size (bits)	No. of Bytes	Port Address(es)
TFLR	Time Fill Register	8	1	44
LMGR	Left Margin Register	16	2	40 (LSB)/42 (MSB)
RMGR	Right Margin Register	16	2	60 (LSB)/62 (MSB)
TMGR	Top Margin Register	16	2	30 (LSB)/32 (MSB)
CER	Compressor Express Register	8	1	68
CSR	Compressor Status Register	8	1	78
CMCR	Compressor Master Control Register	8	1	76
CRCR	Compressor Restart Control Register	8	1	48
CPR	Compressor Parameter Register	8	1	74
CKPR	Compressor K Parameter Register	8	1	66
CWR	Compressor Wraparound Register	16	2	50 (LSB)/52 (MSB)
CPWR	Compressor Page Width Register	16	2	70 (LSB)/72 (MSB)
CFWR	Compressor Frame Width Register	16	2	54 (LSB)/56 (MSB)
CSAHR	Compressor Source Address Holding Register	24	3	3A (LSB)/3C/3E (MSB)
CSCAR	Compressor Source Current Address Register	24	3	0A (LSB)/0C/0E (MSB)
CSCHR	Compressor Source Count Holding Register	24	3	14 (LSB)/16/18 (MSB)
CSWCR	Compressor Source Working Count Register	24	3	04 (LSB)/06/08 (MSB)
CSLSR	Compressor Source Line Start Address Register	24	3	5A (LSB)/5C/5E (MSB)
CDAHR	Compressor Destination Address Holding Register	24	3	4A (LSB)/4C/4E (MSB)
CDCAR	Compressor Destination Current Address Register	24	3	2A (LSB)/2C/2E (MSB)
CDCHR	Compressor Destination Count Holding Register	24	3	34 (LSB)/36/38 (MSB)
CDWCR	Compressor Destination Working Count Register	24	3	24 (LSB)/26/28 (MSB)
CDLSR	Compressor Destination Line Start Address Register	24	3	6A (LSB)/6C/6E (MSB)

Table 2-2. Expander Registers

Abbr.	Name	Size (bits)	No. of Bytes	Port Address(es)
MSR*	Master Status Register	8	1	FE
ESR	Expander Status Register	8	1	F8
EMCR	Expander Master Control Register	8	1	F6
ERCR	Expander Restart Control Register	8	1	C8
EPR	Expander Parameter Register	8	1	F4
EKP	Expander K Parameter Register	8	1	E6
EWR	Expander Wraparound Register	16	2	D0 (LSB)/D2 (MSB)
EPWR	Expander Page Width Register	16	2	F0 (LSB)/F2 (MSB)
EFWR	Expander Frame Width Register	16	2	D4 (LSB)/D6 (MSB)
ESAHR	Expander Source Address Holding Register	24	3	BA (LSB)/BC/BE(MSB)
ESCAR	Expander Source Current Address Register	24	3	8A (LSB)/8C/8E (MSB)
ESCHR	Expander Source Count Holding Register	24	3	94 (LSB)/96/98 (MSB)
ESWCR	Expander Source Working Count Register	24	3	84 (LSB)/86/88 (MSB)
ESLSR	Expander Source Line Start Address Register	24	3	DA (LSB)/DC/DE(MSB)
EDAHR	Expander Destination Address Holding Register	24	3	CA (LSB)/CC/CE(MSB)
EDCAR	Expander Destination Current Address Register	24	3	AA (LSB)/AC/AE(MSB)
EDCHR	Expander Destination Count Holding Register	24	3	B4 (LSB)/B6/B8 (MSB)
EDWCR	Expander Destination Working Count Register	24	3	A4 (LSB)/A6/A8 (MSB)
EDLSR	Expander Destination Line Address Register	24	3	EA (LSB)/EC/EE(MSB)

NOTE: All register addresses are even, the bytes in a register are, therefore, not addressed with contiguous addresses.

* Used by both the compressor and the expander.

Specifying "0" in the Time Fill Register means that no time fill is desired. The minimum length requirement is either not applicable or is handled in some other way.

When the Auto-EOL feature is suppressed, the Am7970A ignores the time fill requirement; no time fill is inserted. When both the Auto-EOL and the byte boundary control are enabled the Am7970A will add fill bits as necessary between the compressed data and the EOL code to end the line on a byte boundary. When the no byte boundaries control is specified, the Am7970A Compressor does not add time fill bits to end lines on a byte boundary.

2.2.2 Left Margin Register (LMGR)

The Left Margin Register, a 16-bit register in the Compressor, specifies the width, in bytes, of the left-hand margin. If the value is is "0", then the original scan line is used without modification. Refer to Figure 2-3.

When a compression operation is initiated with the left margin specified, the Compressor obtains data from the Source Buffer via DMA as usual. However, the margin specification overrides the actual image data and forces the pixels to be "white". Such overriding continues until the programmed margin requirements are satisfied.

If, for example, a "0001" is specified in the Left Margin Register, it means that the first 8 pixels of the line data are overridden with white margin. Compression of the remainder of the scan line proceeds as usual (see also Right Margin Register). The left margin is effective in Wraparound and Express Mode and is included in One-Dimensional, Two-Dimensional, and Transparent Modes of operation.

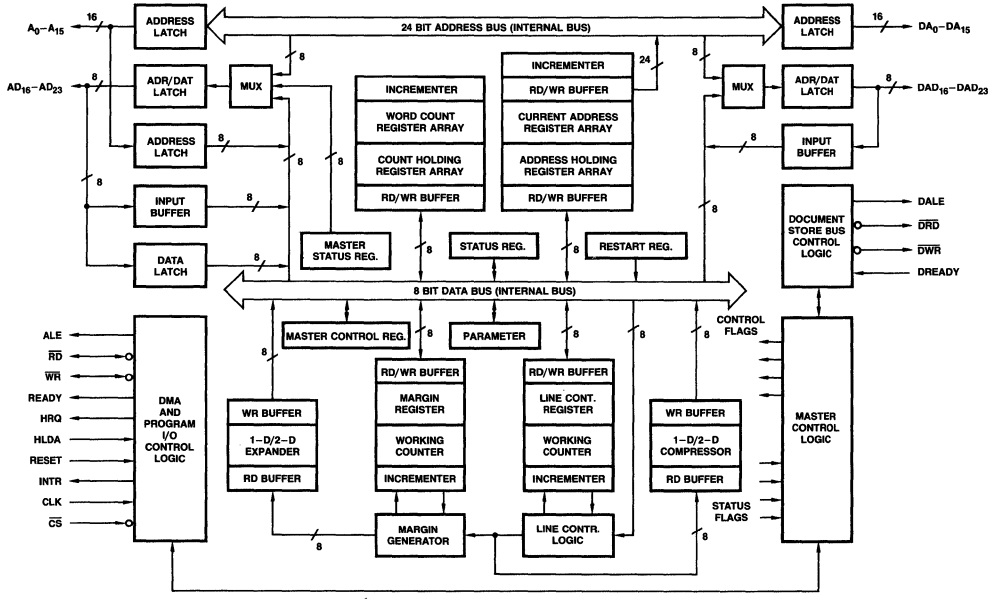


Figure 2-1 Am7970A (CEP) Block Diagram 05557B 2-1

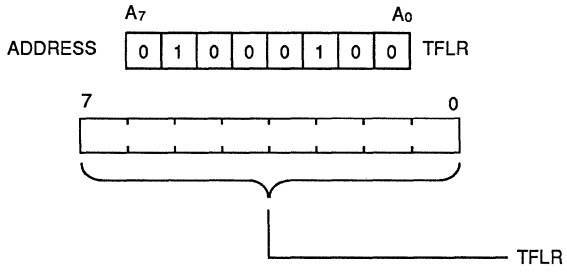


Figure 2-2 Time Fill Register (TFLR) 07666A 2-2

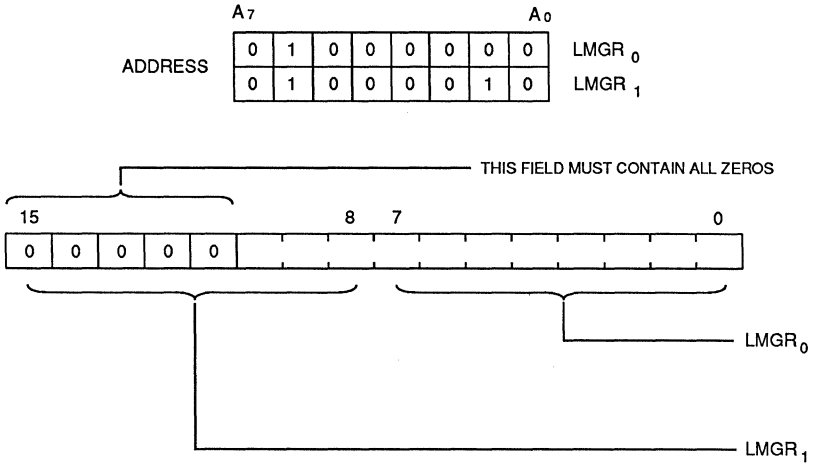


Figure 2-3 Left Margin Register (RMGR) 07666A 2-3

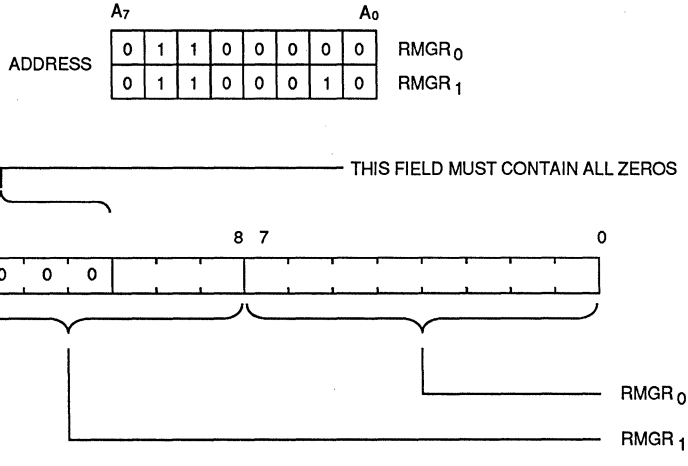


Figure 2-4 Right Margin Register (TMGR) 07666A 2-4

Hence, if the left margin is specified while using Transparent Mode in the transfer of data from the Source Buffer to the Destination Buffer, the data in the Destination Buffer will differ from the data in the source because of the margin. The Compressor does not modify the Left Margin Register during its operation.

The sum of the left and right margin specifications must not be greater than the paper width specified. This would result in an error condition flagged by (CIC) bit in CSR..

Bits 11 through 15 of the Left Margin Register must be set to "0".

2.2.3 Right Margin Register (RMGR)

The Right Margin Register, a 16-bit Compressor register, specifies the width (in bytes) of the right hand margin. A value of "0" means that the original scan line is used without modification. Refer to Figure 2-4.

When a compression operation is initiated with the right margin specified, the Compressor obtains data from the Source Buffer via DMA as usual. However, the margin specification overrides the actual image data and forces the pixels in the margin area to be "white". If, for example, the value in this register is "0002", the last 16 pixels on the line will be overridden by the white margin.

Compression of the scan line up to the start of the right margin proceeds as usual (see also Left Margin Register). The right margin is effective in Wraparound and Express Mode and is included in One-Dimensional, Two-Dimensional, and Transparent Modes of operation.

Hence, if the right margin is specified while using Transparent Mode to accomplish a transfer of data from the Source Buffer to the Destination Buffer, the data in the destination will differ from the data in the source because of the margin. The Compressor does not modify the Right Margin Register during its operation.

The sum of the left and right margin specifications must not be greater than the paper width specified. This would result in an error condition.

Bits 11 through 15 of the Right Margin Register must be set to "0".

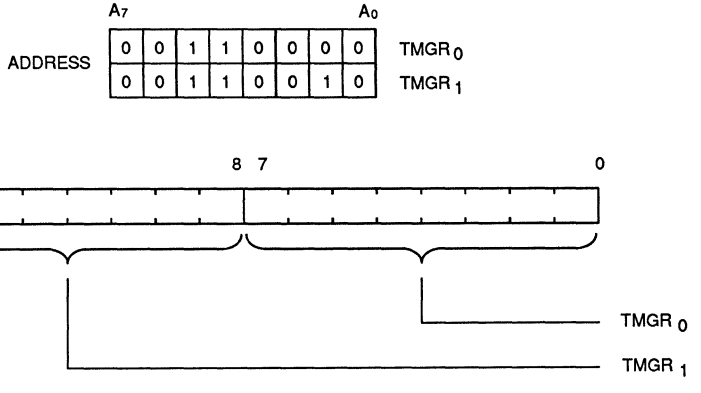
2.2.4 Top Margin Register (TMGR)

The Top Margin Register specifies the top margin of a document. If the Top Margin Register is loaded with a "0", no top margin is specified. Refer to Figure 2-5.

If the Top Margin Register is non-zero, it specifies the desired top margin height in increments of one scan line. When a compression operation is initiated with the top margin specified, the Compressor reads data from the Source Buffer via DMA as usual. However, the top margin specification overrides the data and forces "white" into the Compressor until the top margin requirements are satisfied. From then on, the usual compression operation takes place (also see Left and Right Margin Registers).

Since, by definition, the top margin white space occurs only once per document, the Compressor logic decrements the Top Margin Register by one after processing each scan line until it reaches "0", at which time normal compression proceeds.

The top margin is effective in both Wraparound and Express Modes. However, caution must be



07666A 2-5

Figure 2-5 Top Margin Register (TMGR)

exercised when specifying Express Mode with a top margin since the Compressor logic of the Am7970A skips every "nth" line ("n" being a function of the Express Register) in Express Mode. For example, assume that Top Margin Register specifies "8" and the Express Register specifies "1". The Compressor then processes every other scan line (scan line 1, 3, etc.) in accordance with the Express Register specification. The TMGR is decremented by only those scan lines that are processed. Therefore, since the Top Margin is assigned to be "8", and every other scan line has been skipped, scan line "17" of the original picture will be the first coded line with real picture data on it in this example.

The top margin controls are effective in One-Dimensional, Two-Dimensional, and Transparent Modes. Hence, consideration must be given to the effects of Top Margin Register when using Transparent Mode to transfer data from the Source Buffer to the Destination Buffer.

2.2.5 Compressor Express Register (CER)

The Compressor Express Register, an 8-bit register, specifies (in binary) how many scan lines to compress before skipping one line. For example, if the CER value is 4, every fifth line will be skipped resulting in a vertical image reduction of 20%. If this register is loaded with a "0", every scan line is compressed; this is the normal operating mode. The Compressor logic will not modify this register during its operation. Refer to Figure 2-6.

If the Express Mode is defined with Two-Dimensional Compression, each line that is compressed is also the reference line for the next line that is compressed. Skipped lines are not used as reference lines.

The Am7970A does not allow Wraparound and Express Modes to be specified simultaneously. If such a condition is specified, an error status will be indicated by the CIC bit in CSR. The scan line length is obtained from the Compressor Page Width Register.

2.2.6 Master Status Register (MSR)

Figure 2-7 shows the Master Status Register layout. This 8-bit register provides both Expander and Compressor global status information to the CPU. The various bits in this register are assigned the following meaning:

EXT (Extension). Bits 0, 1, and 2 are the Extension Code bits. When extension code bits have been detected by the Expander, it sets ECD (Bit 3) to "1", and the EXT bits contain (in reverse order) the three least significant extension bits. For example, an extension code of "011" appears in the MSR as "110". When the ECD bit is set to "0", the extension bits are cleared to "0s". The EXT bits are also cleared when a new operation is initiated.

ECD (Extension Code Detected). Bit 3 is the Extension Code Detected bit. This bit is set by the Expander to indicate that an extension code, for which the least significant three bits are not all "1"s, has been detected. The ECD bit is cleared to "0" after a new operation is initiated.

EOP (End Of Page). Bit 4 is the End Of Page detected bit. This bit is set by the Expander to indicate that either an EOP code (in Group 4 expansion) made up of two contiguous EOL codes, has been detected or an RTC code (in Group 3 processing) made up of six EOL codes has been detected. The Transparent Mode cannot detect an EOP code.

The EOP code formats are:

000000000001000000000001 2 EOL codes for Group 4 (EOP)

000000000001 ... 000000000001 6 EOL codes for 1D (RTC)

0000000000011 ... 0000000000011 6 EOL codes for 2D in Group 3 (RTC)

The state of the EOP bit is "0" after a new operation is initiated.

ID (Version I.D.). Bit 5 (the Version I.D. bit) of the Master Status Register is used to identify the CEP as either a 7970 Rev A/A' or as a 7970A as follows:

ID = 0 identifies 7970 Rev A/A'
ID = 1 identifies 7970A

The differences are described in Appendix D.

EBY (Expander Busy). Bit 6 is the Expander Busy bit. This bit is equivalent to the EBY bit in the Expander Status Register.

CBY (Compressor Busy). Bit 7 is the Compressor Busy Bit. This bit is equivalent to the CBY bit in the Compressor Status Register.

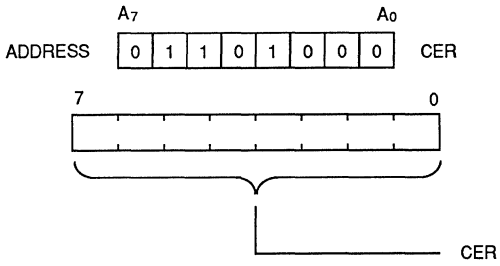
2.2.7 Compressor Status Register (CSR)

The Compressor Status Register, an 8-bit register, indicates the outcome of the last operation. The following paragraphs contain a detailed description of the CSR bits. Refer to Figure 2-8 for the register layout.

NGC – Negative Compression

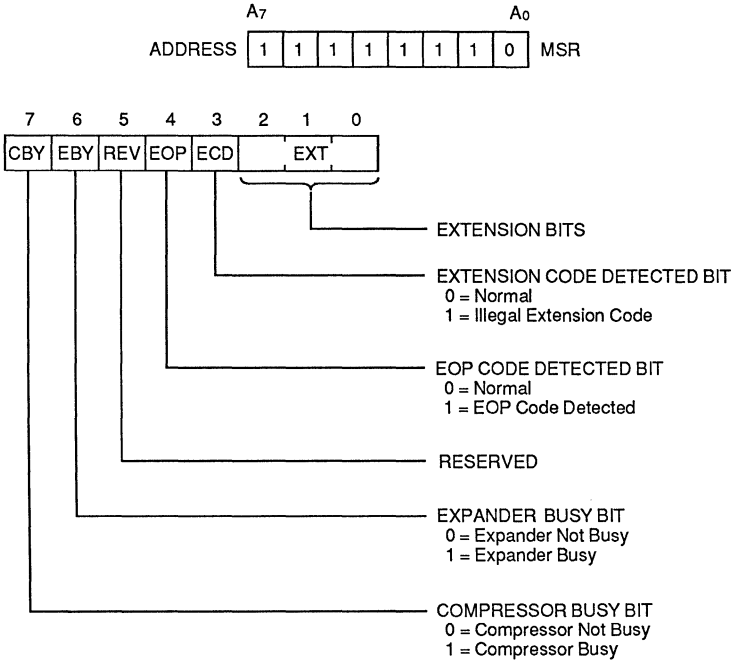
Bit 0 is set to "1" to indicate that compressing the current line resulted in negative compression. This means that the compressed line has more bytes than the original effective line. The Am7970A checks for negative compression only after completely processing an effective line.

Prepositional EOL codes and fill bits, in addition to the actual data, are included in determining the total number of bytes in a compressed line. The Am7970A computes the number of bytes contained in the original effective line based on



07666A 2-6

Figure 2-6 Compressor Express Register (CER)



07666A 2-7

Figure 2-7 Master Status Register (MSR)

the contents of the Page Width Register and the Wraparound Register (when appropriate). Negative compression conditions are not checked during Transparent Mode of operation.

The Am7970A clears the NGC bit to "0" when a new operation is initiated from the CMCR.

COA – Compressor Busy and New Operation Attempted

Bit 1 in the CSR is set to "1" to indicate that an attempt was made to write a new operation into the CMCR while the Compressor was still busy.

All registers which require user-specification in the Am7970A can be read as well as written by the host CPU. However, modifying the registers while the Compressor is operating is not allowed. The Am7970A ignores any attempt to write into a compressor register (other than the CMCR) while the Compressor is busy. Any attempt to write into the CMCR at such times sets the COA bit to "1" and marks the beginning of a Compressor termination. The actual termination is indicated by the Compressor Busy (CBY) bit.

The COA bit is cleared to "0" when a new operation is initiated.

CIC – Compressor Illegal Command

Bit 2 in the CSR is set to "1" when the Compressor is directed to start operating with any of the following illegal conditions present:

When the GO bit is set to "1" (that is, "start Compressor operation"), the Am7970A clears the CIC bit to "0" and sets the CBY bit to "1" to indicate busy status. The conditions for CIC status (b, c, d, or e) are checked only when the current GO command was preceded by a compressor soft reset.

a) The mode bits (MC0 and MC1) or the control bits (CC0 and CC1) in the Master Control Register specify the reserved code.

b) The mode bits (MC0 and MC1) in the CMCR specify 2D compression and the Compressor Wraparound Register specifies wraparound (is non-zero). Wraparound in 2D is illegal.

c) The Compressor Express Register and Compressor Wraparound Register are both non-zero. Wraparound in Express Mode is illegal.

d) The sum of the left and right margins represented by the Left Margin Register and Right Margin Register is greater than the page width specified in Compressor Page Width Register. In other words, specifying overlapping margins is illegal.

e) The Compressor Page Width Register has been specified as "0".

If any illegal condition exists as stated above, the Am7970A will terminate the operation with the CIC bit set to "1" and CBY bit set to "0".

WPI – Wraparound Incomplete

Bit 3 is set to "1" when the Compressor terminates prior to successfully compressing an effective line. This status signifies that the Compressor has not satisfied the Compressor Wraparound Register requirements. The WPI bit being set in conjunction with the LPI bit gives a detailed indication of the status that exists when the Compressor is terminated. The setting of this bit starts the Compressor termination. Actual termination is indicated in the CBY bit of the Status Register. The WPI bit is cleared to "0" when a new operation is initiated.

LPI – Line Processing Incomplete

Bit 4 is set to "1" to indicate that the Compressor terminated without successfully processing a complete line; either a source or destination overflow occurred prematurely. There are three situations in which this occurs:

1. When the last byte obtained from the Source Buffer did not correspond to the last byte of the page, and the Compressor Source Working Count Register overflowed.

2. When a Compressor Destination Working Count Register overflowed before the Compression operation reached the end of a scan line.

3. When a Compressor Destination Working Count overflowed before a fully compressed line could be stored in the destination. The term "fully compressed line" includes EOL (if any), data, time fill (if any), and fill (if any).

The start of Compressor operation termination is marked by the LPI bit being set. Actual termination is indicated by the CBY bit.

The LPI bit is cleared to "0" when a new operation is initiated.

In Multi-Line operation on a Source Buffer overflow, or at the end of the terminal line in Single-Line operation, the Compressor processes the last full byte of source data but truncates any excess bits (one to seven) that do not comprise a byte. The truncated bits are still retained within the compressor and concatenated with subsequent code data unless a compressor soft reset is issued. In this case, the Compressor terminates and the LPI bit is set in the CSR.

CDO – Compressor Destination Overflow

Bit 5 is set to indicate that the Compressor Destination Working Count Register has reached "0". This register is initially loaded with a negative two's complement value and is incremented after each transaction with the Destination Buffer. When the Compressor Working Count Register reaches "0" due to such incrementing, the CDO bit is set to "1", starting the Compressor termination process. The actual termination is indicated by clearing the CBY bit to "0".

The CDO bit is cleared to "0" when a command is initiated.

CSO – Compressor Source Overflow

Bit 6 is set to "1" to indicate that the Compressor Source Working Count Register has reached "0". This register is initially loaded with a negative two's compliment value and is incremented by one after each Source Buffer transaction. When the CSWCR reaches "0" due to such incrementing, source overflow has occurred and the CSO bit is set to "1". The Compressor will begin to terminate its operation after setting the CSO bit. Thus, there will be some elapsed time between setting the CSO bit and the actual termination as indicated by the CBY bit.

The CSO bit is cleared to "0" when a command is initiated.

CBY – Compressor Busy

Bit 7 is set to "1" by the CEP to indicate that the Compressor is busy. Whenever a new operation is initiated (by setting the GO bit), the CBY will indicate busy status. The CBY bit automatically becomes "0" when the Compressor terminates its operation. Modification of any of the Am7970A registers is allowed only when the CBY is "0", indicating "not busy". Otherwise the Am7970A might inadvertently modify the registers during normal operation.

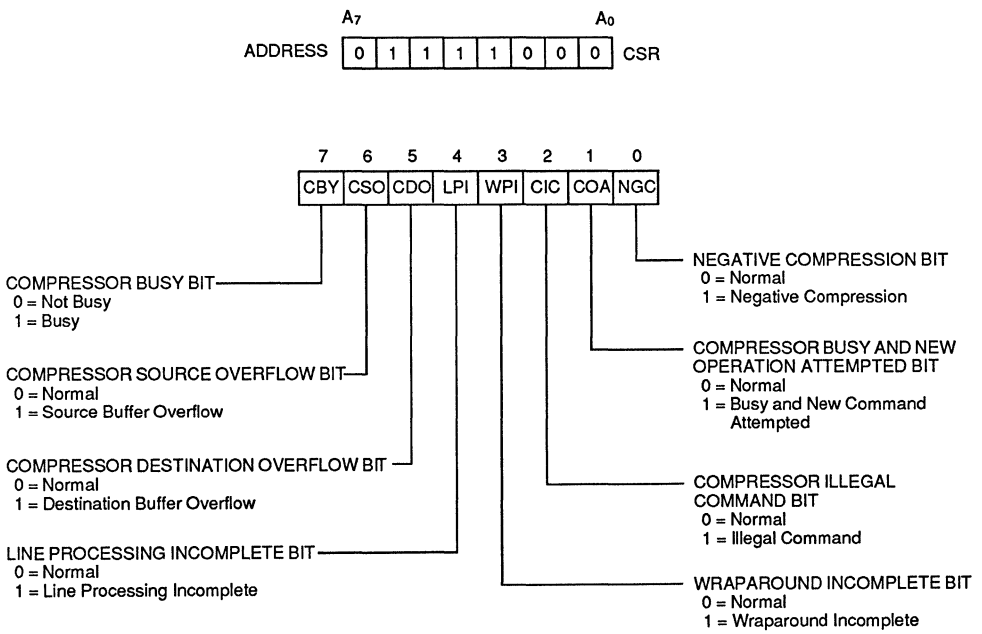


Figure 2-8 Compressor Status Register (CSR)

If the CIE bit in the Master Control Register is "1", an interrupt to the CPU is asserted when the CBY becomes "not busy". The CBY bit can be polled by the host CPU for an indication of the completion of an operation.

2.2.8 Expander Status Register (ESR)

The Expander Status Register bit assignments give the status information about the Expander. Except for bit zero, they are the same as the Compressor Status Register assignments. Refer to Figure 2-9 for a register layout.

DER – Data Error

The CEP stops operation and Bit 0, the Data Error bit (DER) in the ESR, is set to "1" whenever an error is detected in the current effective line during expansion. The address of the erroneous effective line is recorded in the Expander Source Line Start Register. One type of error exists when an EOL code is detected and the bit length of an expanded effective line is not equal to:

$$L = (EPWR \cdot 8) \cdot (EWR + 1)$$

where: EPWR = The value of the Expander Page Width Register

EWR = The value of the Expander Wraparound Register.

Another type of error condition occurs when the CEP detects an illegal code. An error may result from a hardware failure or the inadvertent transformation of a valid Modified Huffman codeword to another bit pattern due to noise or due to a transmission error. An unrecognizable codeword causes the DER bit to be set to "1". If (EOL=1) in the EPR and an EOL code is detected by the expander, the DER bit will be set.

In Group 3 coding in 2D (EOL bit = 0, MC field = 10), the expander stops processing after a data error (DER) but processing is resumable by the CPU. When processing resumes (with EOL bit = 0 and ESA bit = 1), the expander will process the next line as either 1D or 2D according to the tag bit. If the next line is not decodable, the DER bit is set again. If the next line is one-dimensional and has no errors, two-dimensional processing may resume. Since CPU action is required after each DER data error, the CPU can count the lost lines between the error and the recovery.

In Group 4 coding (EOL bit = 1, MC field = 10), the expander processes all lines in 2D. Zero error at this level is assumed. Error detection and correction is done at a higher level.

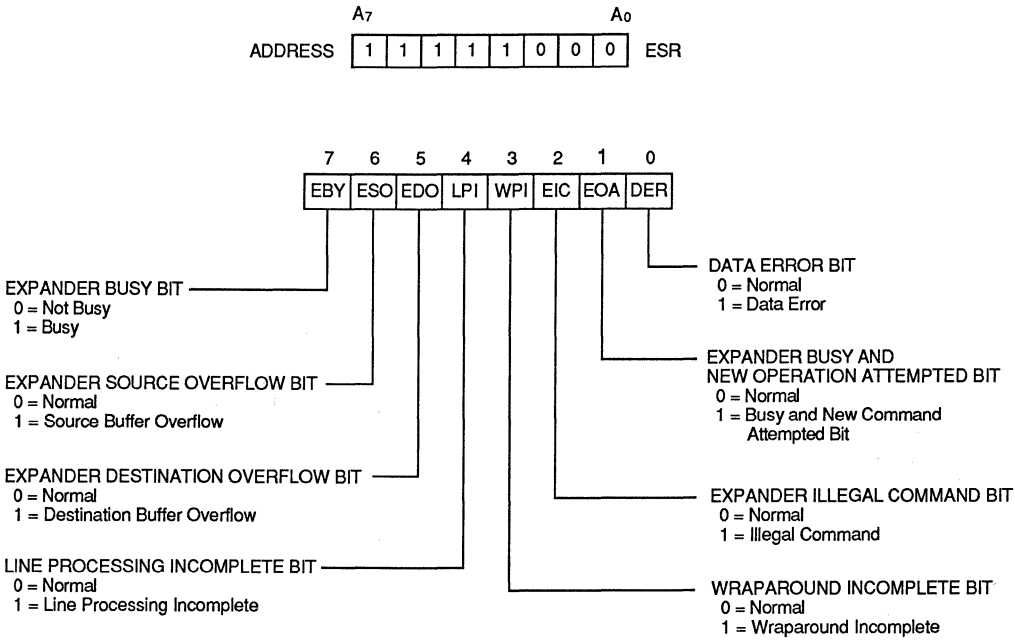


Figure 2-9 Expander Status Register (ESR) 07666A 2-9

The state of the DER bit, after a new command has been initiated by the the CPU, is "0". Refer to Appendix H for more information regarding recovery after a data error during expansion.

EOA – Expander Busy and New Operation Attempted

Bit 1, the EOA bit, is set to "1" when an attempt is made to write a new command into the EMCR while the Expander is still busy. When the EOA bit is set to "1", the Expander prematurely terminates and the Expander Busy (EBY) bit is cleared to "0".

After this command has been initiated by the CPU, the state of the EOA bit is "0".

EIC – Expander Illegal Command

Bit 2, the EIC bit, is set to "1" to indicate that the Expander was directed to start operating with any of the following illegal conditions present:

1. A Two-Dimensional Expansion Mode and a non-zero Wraparound Register have both been specified.

2. A non-zero granularity parameter and a non-zero Wraparound Register have both been specified.

3. Both MC bits in the EMCR are specified as "1".

4. Both OC bits in the EMCR are specified as "1".

5. The page width in the EPWR has been specified as "0".

After an Expander Software Reset, if an Expander command is received when one or more of these conditions are present, the EIC bit is set to "1" and the EBY bit is cleared to "0".

After a new command has been initiated by the CPU, the state of the EIC bit is "0".

WPI – Wraparound Incomplete

Bit 3, the WPI bit, is set to "1" to indicate either A Source Buffer or Destination Buffer overflow after a complete scan line was expanded but before an entire effective line could be expanded.

After a new command has been initiated by the CPU, the state of the WPI bit is "0".

LPI – Line Processing Incomplete

Bit 4, the LPI bit, is is set to "1" to indicate either a Source Buffer or Destination Buffer overflow before an entire scan line could be expanded and written into the Destination Buffer.

The state of the LPI bit is "0" after a new command has been initiated by the CPU.

EDO – Expander Destination Overflow

Bit 5, the EDO bit, set to "1", indicates that the Expander Destination Working Count Register (EDWCR) has reached "0". This register is initially loaded with a negative two's complement value and is incremented after each transaction with the Destination Buffer. When the EDWCR reaches "0", the EDO bit is set to "1" beginning the Expander termination process. The actual termination is indicated by the CEP clearing the EBY bit to "0".

The state of the EDO bit is "0" after a new command has been initiated by the CPU.

ESO – Expander Source Overflow

Bit 6, the ESO bit, set to "1", indicates that the Expander Source Working Count Register (ESWCR) has reached "0". This register is initially loaded with a negative two's complement value and is incremented by one after each Source Buffer transaction. When the ESWCR reaches "0", overflow has occurred and the ESO bit is set to "1" beginning the Expander termination operation. Some time elapses between setting the ESO bit to "1" and the actual Expander termination as indicated by the EBY bit.

The state of the ESO bit is "0" after a new command has been initiated by the CPU.

EBY – Expander Busy

Bit 7 is set to "1" to indicate that the Expander is busy. Whenever a new operation is initiated, the EBY bit is set to "1". When the Expander has completed its operation, the EBY bit is reset to "0".

2.2.9 Master Control Registers (CMCR, EMCR)

The 8-bit CMCR specifies the desired mode of operation (one-dimensional, two-dimensional, or transparent), the location of the source and destination buffers (Main Memory or Document Store), the Interrupt Enable, the operation controls (Reset, Single-Line or Multi-Line), and the initiation of processing (start or stop processing). The function of each of the CMCR bits is described in the following paragraphs. The EMCR register performs the same functions for the

expander and is identical to the CMCR. Refer to Figure 2-10.

GO

Bit 0, the GO bit, when set to "1" by the system program, initiates compressor operation specified in the Operation Control Field. Once set, the Compressor Status Register (CSR) indicates that the compressor is busy. If the Operation Control field in the MCR has been set to (00), the SOftware Reset Operation is performed. If the OC Field is set to (01) or (10) after a software reset and the GO bit is again set to (1), the compressor will check the configuration of the registers before proceeding with the compression. Upon completion of the compressor operation, the GO bit is reset to "0" automatically. Any attempt to load the CMCR when the Compressor is busy, terminates the processor with the appropriate error bits set in the CSR.

OC – Operation Control

Bits 1 and 2, the Operation Control bits, configure the CEP for the following operations which are executed when the GO bit is set:

Bit 2 OC1	Bit 1 OC0	Operation
0	0	RESET
0	1	SINGLE-LINE
1	0	MULTI-LINE
1	1	RESERVED

Reset (00). This operation sets the Compressor to the same state as a hardware reset except that it does not reset the DMA bus. It flushes the input queue and clears the internal working registers, process control flags, sets up the check for configuration errors, clears status and interrupt registers, and sets the GO bit to zero. It does *not* clear the user-programmable control registers, the CMCR, or Compressor Parameter Register (CPR).

The system program *must* issue a Reset operation before starting a new sequence of events such as starting to process a new page. This software Reset between contexts is necessary so that the upcoming sequence of operations is interpreted correctly.

A software reset is performed by writing a reset command (00) into the Operation Control (OC) bits of the Master Control Register. The GO bit must be set to "1" to start the reset operation. The GO bit will be cleared during the reset operation.

The reset operation is a microprogram that takes about 4 microseconds to execute. During this time, the "busy" bit (CBY and EBY) is set active in the Master Status Register. During the reset operation, the "busy" bit can be sampled until it goes inactive to verify that the CEP has completed the reset operation before attempting any other operation. Then the GO bit in the Master Control Register (CMCR or EMCR) is set to 1 to start the new operation.

The software reset is required before processing a new page but it is not needed to resume processing on the same page. If resumable operation of the current processing activity is necessary, the system should not issue a Reset operation since various registers are cleared invalidating any subsequent Restart operation.

Single-Line (01). When Single-Line operation is initiated, one effective line of data from the Source Buffer is processed before the GO bit in the CMCR is cleared to "0". Such an event may be either normal or in error. (See Status Register description for details of error status.) The next line or multiple lines of data can be restarted from where the preceding Single-Line operations ended if no errors have occurred.

The value of "effective line" is determined by the contents of the Compressor Wraparound Register (CWR) (1D or transparent mode only) and the Compressor Page Width Register (CPWR). It is (CWR +1) • (CPWR • 8) bits. If the CWR is "0", the effective line is one scan line (equal to the Page Width Register). The "effective line" value specifies how much buffer space is needed.

Multi-line (10). The Multi-Line operation processes data until either the Compressor Source Working Count Register (CSWCR) or Compressor Destination Working Count Register (CDWCR) reaches "0". At this time the processor terminates and the GO bit in the (CMCR) is cleared to "0". Such an ending may be either normal or in error. (See Status Register description for details of error status.)

Reserved (11). If the Reserved operation is initiated, the CEP terminates with an illegal operation error in the CSR (see Status Register section).

IE – Interrupt Enable

Bit 3 is the Interrupt Enable (IE). When the processor terminates an operation, the CEP will interrupt the CPU if the IE bit is set to "1". If the IE

bit is "0", interrupts are disabled. Any system access of the Master Status Register resets the interrupt.

DC – Destination Control

Bit 4 (Destination Control) assigns the location of the Destination Buffer. A Zero means Main Memory and a One means Document Store.

SC – Source Control

Bit 5 (Source Control) specifies the location of the Source Buffer. A Zero means Main Memory and a One means Document Store.

MC – Mode Control

The Mode Control field (bits 6 and 7) specifies the desired mode according to the following table:

Bit 7 MC1	Bit 6 MC0	Mode
0	0	TRANSPARENT
0	1	ONE-DIMENSIONAL
1	0	TWO-DIMENSIONAL
1	1	RESERVED

Transparent (00). Transparent Mode means that no data modification occurs in the selected processor; data merely passes through the processor via DMA. However, the effects of the Auto-End-of-Line (EOL) insertion feature, Margin Registers, Wraparound, Time Fill, and Express Registers must be fully considered before attempting such information moves. In Transparent Mode, the EOL code is always padded to a byte boundary.

If, for example, the Code Buffer is contained in the Document Store and all data transmissions take place from the main memory, facilities are required to transfer the required information from the Code Buffer in the Document Store to the main memory. By initiating Transparent Mode operation, the processor can transfer the required information in the same way as a conventional DMA controller (with the effects mentioned above). On the Document side, the transparent mode through the compressor section is faster than through the expander because the compressor section provides a 3-byte input FIFO.

One-Dimensional (01). One-Dimensional Mode specifies the standard Modified Huffman Code. During this mode of operation, the processor takes into account the relevant margin registers, Wraparound Register, Express Register, Page Width, and and Time Fill Registers. If the Auto-EOL feature is on, each coded effective line

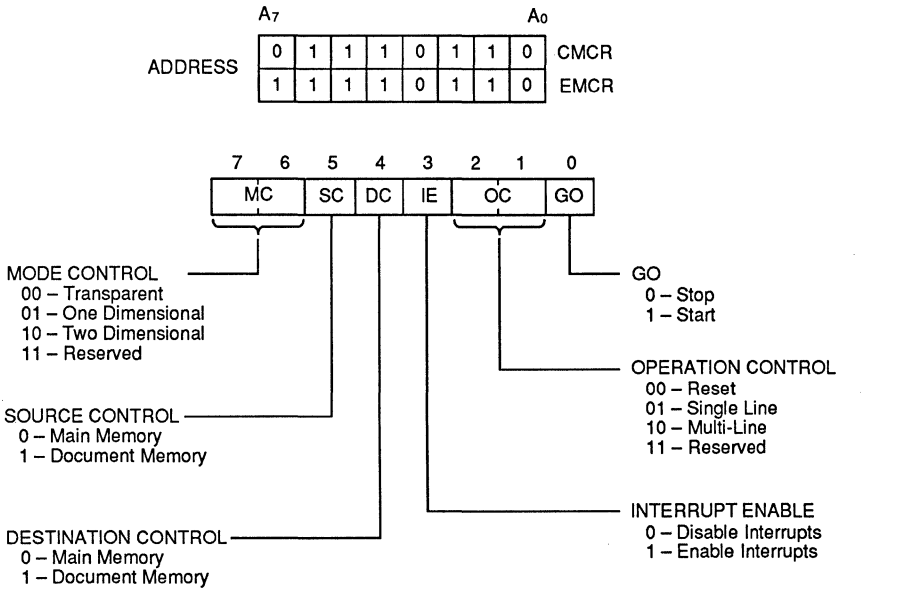


Figure 2-10 Master Control Registers (CMCR, EMCR)

is terminated with an EOL. These registers and features are discussed individually under their respective headings.

Two-Dimensional (10). The two-dimensional mode (2D) in Group 3 (EOL = 0, MC = 10, K > 0) specifies the standard Modified READ Code according to CCITT recommendations T.4. During this mode of operation, the processor takes into account the Page Width, Margins, Express Mode, and Time Fill Registers. The Auto-EOL feature is on. The Wraparound feature is not available with two-dimensional coding. The K Parameter determines how many lines of 2D follow each line of 1D in compression.

In Group 3 expansion (EOL = 0, MC = 10, K > 0), the expander ignores the K-Parameter and uses the tag bit following each EOL to determine how to process the next line. If the tag bit is 1, the next line is decoded as 1D. If the tag bit is 0, the next line is decoded as 2D.

Two-Dimensional Mode (2D) in Group 4 (EOL = 1, MC = 10, K = 0) specifies the standard Modified READ Code according to CCITT recommendation T.6. During this mode of operation, the processor takes into account the Page Width, K-Parameter, Margins, and Time Fill Registers. The Auto-EOL feature is off. The Wraparound feature is not available with two-dimensional coding.

The 2D Mode (MC = 10) in Group 3 utilizes both 1D and 2D processes in conjunction with the K-Register and the K-Working-Register (not visible to the CPU). The first line is encoded in 1D and then K-1 lines are encoded in 2D. The 1D line does not need a reference line. The first 2D line uses the 1D line as the reference line. Each succeeding 2D line uses the preceding line as its reference line.

With 2D compression, the CEP reads the current line and the previous line (reference line) to encode the current line. With 2D expansion, the CEP reads the code from the source buffer and rereads the previous already expanded line to decode the current line.

Reserved (11). If the Reserved operation is specified, the CEP terminates with an illegal operation error in the CSR (see Status Register section).

The fields in the Expander Master Control Register are identical to the CMCR (as described above) and specify the same options (refer to Figure 2-10). The GO bit initiates operation of the Expansion Processor. Issue a software reset by setting the GO bit to (1) and the OC Field to (00) before each new page and each new sequence of events when resumable operation is not required. Then set the OC Fleld to single line or multi-line operation and set the GO bit to start processing.

The OC bits specify whether single line or multi-line operation is required. The IE bit specifies whether CPU interrupts are desired. The source and destination buffer locations are specified in the SC and DC bits. The MC bits specify the Mode Control.

2.2.10 Compressor/Expander Restart Control Registers (CRCR/ERCR)

The CRCR and ERCR are 8-bit registers that specify whether to continue the compression/expansion with the current buffer address and count values or to restore the starting values from the Hold Registers into the Current Buffer Address registers before a new processor operation is initiated by setting the GO bit. The registers affected are: the Source and Destination Current Address Registers, Source and Destination Working Count Registers, and Source and Destination Line Start Registers. The following is a detailed description of the individual bits. These two registers are identical except for bit 5. In the compressor register, bit 5 is reserved whereas the expander register uses bit 5 for byte boundary control. Refer to Figure 2-11.

SCC – Source Count Control

When Bit 0, the SCC bit, is "0", the value in the Source Count Holding Register is loaded into the Source Working Count Register when a new operation is initiated. When this bit is "1", such loading does not take place and the existing value of the Source Working Count Register is used unaltered for the new operation.

SAC – Source Address Control

When Bit 1, the SAC bit, of the CRCR is "0", the value of the Source Address Holding Register is loaded into the Source Current Address Register when a new operation is initiated. When this bit is "1", such loading does not take place and the existing value of the Source Current Address Register is used in the new operation.

DCC – Destination Count Control

When Bit 2 of the CRCR is "0", the contents of the Destination Count Holding Register are loaded into the Destination Working Count Register when a new operation is initiated. When this bit is "1", such loading does not take place and the existing contents of the Destination Working Count Register are used in the new operation.

DAC – Destination Address Control

When Bit 3 of the Restart Control Register is "0", the contents of the Destination Address Holding Register are loaded into the Destination Current Address Register when a new operation is initiated. When this bit is "1", such loading does not take place and the existing contents of the Destination Current Address Register are used in the new operation.

RES – Reserved

Bits 4 and 5 are reserved in the CRCR. Bit 4 is reserved in the ERCR.

BBC – Expander Byte Boundary Control

Bit 5, the BBC bit, must be set in the Expander to specify whether or not byte boundary control was used in the compression. The expander uses this information to distinguish byte padding "0"s at the end of a coded line from the starting code of the next coded line when the compressed data does not have EOL codes. If the compressed data was byte-adjusted, this bit must be set to "0".

In Group 4 expansion, the BBC bit must be set to 1 to specify no byte boundaries.

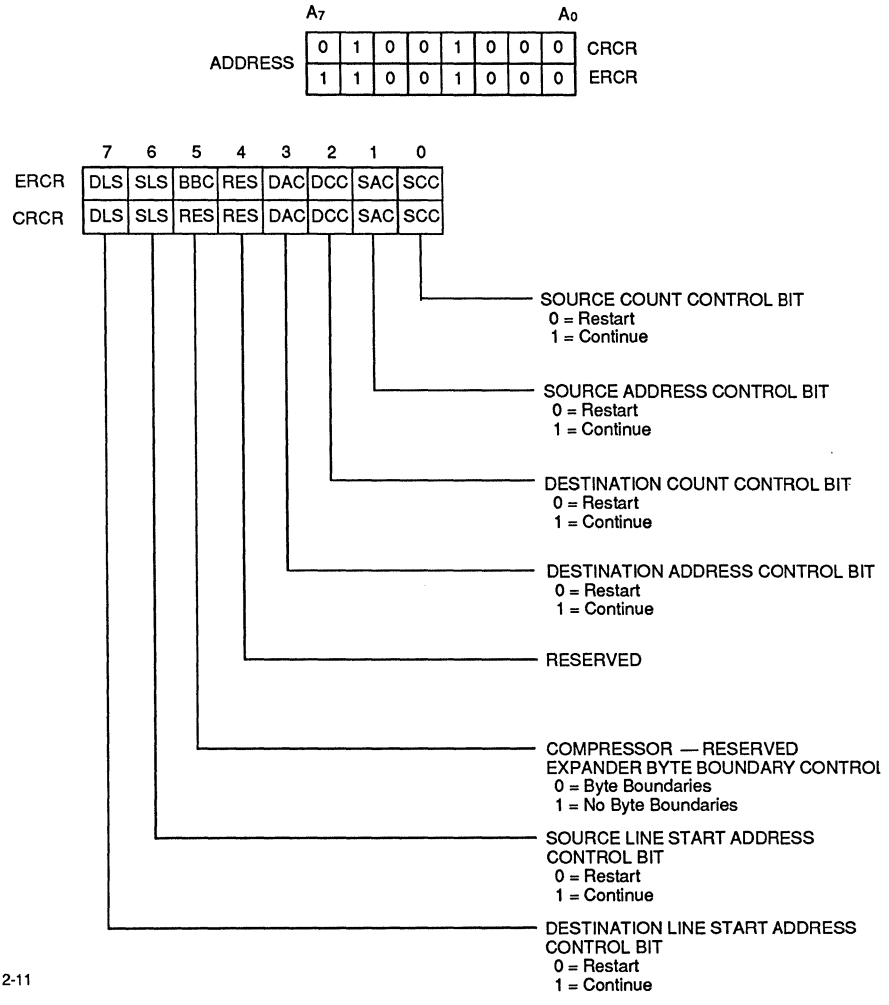


Figure 2-11 Restart Control Registers (CRCR, ERCR)

SLS – Source Line Start Address Control

When Bit 6, the SLS bit, is "0", the Source Line Start Address Register is loaded from either the Source Address Holding Register or the Source Current Address Register depending on whether the SAC bit is "0" or "1" when a new operation is issued. When the Source Line Start Address Control bit is "1", the contents of the Source Line Start Address Register are not modified when a new command is issued.

DLS – Destination Line Start Address Control

When Bit 7, the DLS bit, is "0", the Destination Line Start Address Register is loaded from the Destination Address Holding Register or the Destination Current Address Register, depending on whether the DAC bit is "0" or "1" when a new operation is issued. When the Destination Line Start Address Control bit is "1", the contents of the Destination Line Start Address Register are not modified when a new command is issued.

2.2.11 Compressor Parameter Register (CPR)

The Compressor Parameter Register, an 8-bit register, specifies the Line Termination Parameter, the data format for the Compressor, the beginning of the page attributes, and the EOL control. The function of each of the parameter bits is described in the following paragraphs. Refer to Figure 2-12.

LT – Line Termination Parameter

During compression, the Line Termination field (bits 0, 1, and 2) permits the user to define the width of an image or page on bit boundaries rather than byte boundaries. It specifies the number (from 0 to 7) of bits to be *appended* to the end of each image line after the last full byte of data as follows:

LT2	LT1	LT0	No. of Terminating Bits
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

The termination bits for each line have the same color as the last bit (bit 7) of the last byte of that line. The termination bits as well as the data are encoded. This function is especially useful in conjunction with the frame width specification to align image windows on bit boundaries. In expansion, these added bits are not prefixed to the next line.

During expansion, the termination bits are clipped off as follows: When the expanded data length reaches the line length specified in the Expander Page Width Register (EPWR), the expander resets an internal shift register and thus clips off any image bits comprising an incomplete byte at the end of an image line. The incomplete bytes of from 1 to 7 bits of image data at the end of each line are deleted. Otherwise, the coded color change in the bits of the last incomplete image byte would prevent successful expansion. If there is a color change within these clipped bits, the next line will not be expanded correctly.

DFC – Data Format Control

Bits 4 and 5 (the Data Format Control field) specify the desired compressed data format according to the following table:

Bit 5 CDF1	Bit 4 CDF0	Compressed Data Format
0	0	Process on Byte Boundaries
0	1	No byte boundaries with RTC code at end of document
1	0	No Byte Boundaries without RTC or EOP
1	1	No byte boundaries with EOP code at end of document

The choice of these options depends on the size of the image buffer. If it is large enough to permit processing an entire document or page without stopping, the (01) option is selected for Group 3 and the (11) option is selected for Group 4. The (01) option compresses on no byte boundaries and places an RTC code at the end of the document. The (11) option compresses on no byte boundaries and places an EOP code at the end of the document.

The (10) option is used to process a fraction of a page other than the last fraction. This option compresses on no byte boundaries without placing an RTC or EOP at the end of the fraction being processed. To compress the last fraction with an RTC code or EOP code, the (01) or (11) option is used.

The (00) option (Process on byte boundaries) is only used in processing fractions of a page. When this option is used, the last fraction of the page

must be a single line for which the RTC code option (01) is specified. Group 4 does not allow processing on byte boundaries.

Process On Byte Boundaries (00). When this option is specified, the Am7970A (CEP) is conditioned to end lines on byte boundaries. If a coded line does not end on a byte boundary, the Am7970A adds enough fill bits (one to seven consecutive "0"s) to end a line on a byte boundary.

If Auto-EOL is enabled, and a coded line including Time Fill (if any) plus the EOL code does not end on a byte boundary, fill bits, as needed, are inserted immediately preceding the EOL. Time Fill bits are also zeros and are not distinguishable from other fill bits.

Suffix Return-To-Control (RTC) Code (01). With this option, the compressor processes the document on no byte boundaries and adds an RTC code consisting of six EOLs to the end of the terminal scan line (last line on a page). This identifies the end of a page to the expander. The Compressor is conditioned to end the RTC code on a byte boundary. The RTC code is used only in Group 3 compression. In 2d processing, the RTC consists of six (EOL + 1) codes.

In normal Multi-Line operation, the RTC code is added to the end of the terminal line in which the Compressor Source Buffer overflow occurs, assuming line processing has been successfully completed. If a coded line, including Time Fill (if any) and the RTC code, does not end on a byte boundary, the Compressor adds enough fill bits between the compressed DATA and the RTC code to end the line on a byte boundary.

Error conditions during Multi-Line operation are indicated by the Line Processing Incomplete (LPI) bit in the CSR. These conditions are resumable because the remainder of the coded line (including RTC code, if any) is maintained by the Compressor internally. For example, the Compressor terminates its operation promptly and the RTC code is not added when a Compressor Source Buffer overflow occurs along with either a Compressor Destination Buffer overflow or with the line processing incomplete and wraparound incomplete (if any). If the GO command is issued after the Compressor Source Buffer or the Compressor Destination Buffer has been prepared, the Compressor resumes its previous process from where it left off.

In normal Single-Line operation, an RTC code is added to the end of the line when that line's processing is completed. If a coded line including Time Fill (if any) and the RTC code does not end on a byte boundary, the Compressor adds enough fill bits between the compressed Data and the RTC code to end the line on a byte boundary. If a Compressor Destination Buffer overflow occurs during processing, the Compressor terminates its operation promptly and the RTC code is not suffixed.

No Byte Boundaries (10). When this option is selected, the compressed data (including the Time Fill and EOL codes, if any) is not conditioned to end lines on a byte boundary. No EOL suffix is included at the end of the page (when the working

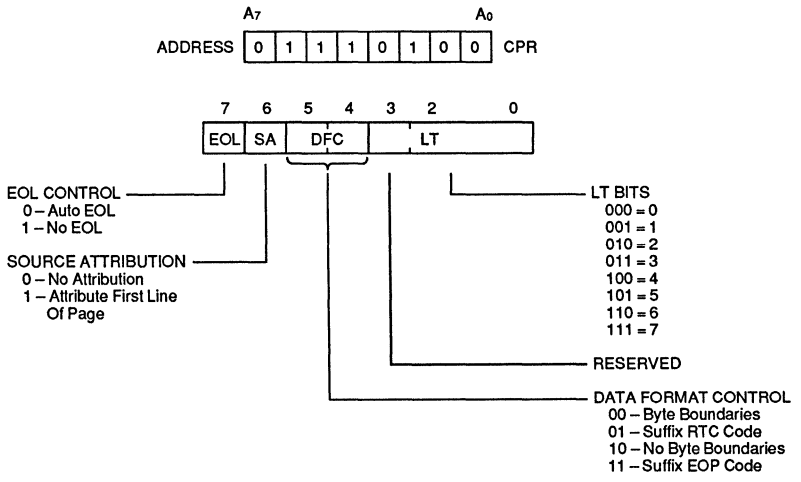


Figure 2-12 Compressor Parameter Register (CPR)

count register is exhausted).

If a Compressor destination overflow occurs, the Compressor terminates its operation promptly and the Compressor retains the remaining bits of the compressed data in an internal register. Under these circumstances, processing may be resumed by issuing the Compressor Go operation after preparing the Compressor Source Buffer or the Compressor Destination Buffer respectively. At this point, the Compressor combines the excess bits from the point of termination with the remainder of the data to be compressed.

Transparent Mode operation cannot be specified together with the no byte boundaries option. The Transparent Mode operation is always conditioned to end a line on a byte boundary, regardless of the byte boundary control bit.

Suffix EOP Code (11). The End Of Page code is used in Group 4 compression to process lines on no byte boundaries and to terminate each coded page with an EOP. The EOP code consists of two consecutive EOL codes as specified in the CCITT recommendation T.4.

The EOP code is followed by fill bits as needed to end the page on a byte boundary.

SA – Source Attribute

Bit 6, the Source Attribute (SA) bit set to "1" specifies processing at the start of a page. In Group 3 processing (EOL field set to "0"), it means that the compressor will insert an EOL at the beginning of a page and the expander will look for an EOL at the start of processing. The SA bit is cleared automatically after the first line of processing is completed. The SA bit is sampled by the CEP when a GO operation is received following a RESET. The initialization of the CEP for a new page includes setting the SA bit by the system (external to the CEP).

The EOL at the beginning of a page is necessary in Group 3 transmissions because the transmitter starts sending "0"s as soon as the connection is established.

When the SA bit is set to "0" at the start of a page, the processors operate without the prefixed EOL code even if EOL = 0.

For Group 4 processing (2D, K = 0, EOP mode, and EOL = 1), when SA = 1, the reference line for the beginning of a page is an imaginary all-white line. If the SA bit is set to 0 after a reset, the CEP will process the first line one-dimensionally. All other lines are processed two-dimensionally according to the Group 4 recommendations.

EOL – End Of Line

For Group 3 processing, bit 7, the EOL bit, is set to "0", and the Compressor automatically adds an EOL code to the end of a compressed line or to the end of a transparent line. With this option (Auto-EOL on), a compressed line or transparent line consists of data followed by Time Fill bits (if any) and an EOL code. This line is conditioned to end on a byte boundary only if byte boundary control is specified.

For Group 4 processing, the EOL bit is set to "1" and a compressed line consists of data only (no time fill bits or EOL codes).

When Auto-EOL is enabled (set to 0) with the Source Attribute (SA) bit in the Compressor Parameter Register set to "1", an EOL code is prefixed to the first compressed line of the page. This prefixed EOL code always ends on a byte boundary when the Transparent Mode has been specified irrespective of the byte boundaries specification. (The EOL code consists of eleven zeros followed by a One.)

1D Mode: 0000 0000 0001
2D Mode: 0000 0000 0001T

T = Tag Bit
= 0 if next line is 2D
= 1 if next line is 1D

2.2.12 Expander Parameter Register (EPR)

(Refer to Figure 2-13 for the register layout).

Reserved

Bits 0, 1, and 2 are reserved.

G – Granularity

Bits 3, 4, and 5) of the EPR contain the Granularity factor (G-Parameter). It specifies how many lines to expand before duplicating the last line expanded. For instance, if G = 4, every fourth scan line is duplicated resulting in a 25% vertical expansion. Every scan line that is duplicated is written into the Destination Buffer a total of two times. Unless the Expander Wraparound Register (EWR) is zero when the G-Parameter is non-zero, an error condition will exist and will be indicated by the EIC bit in the Expander Status Register. These registers are discussed later in this Section. G0, G1, and G2 in the following table refer to Bits 3, 4, and 5 respectively of the EPR. This table lists the G-Parameters and the corresponding code:

Bit 5 G2	Bit 4 G1	Bit 3 G0	G-Parameter
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

SA – Source Attribute

Same as the SA bit in the Compressor Parameter Register.

EOL

Bit 7, the EOL bit, is interpreted as follows in the Expander: When this bit is set to "0", the Source Buffer data is assumed to contain EOL codes. This data will be checked for data errors. If the EOL bit is not set to "0", the data will not be checked for data errors (however, illegal codes and EOL codes generated by the transmission errors will be detected and flagged); in Group 4, error free data is assumed.

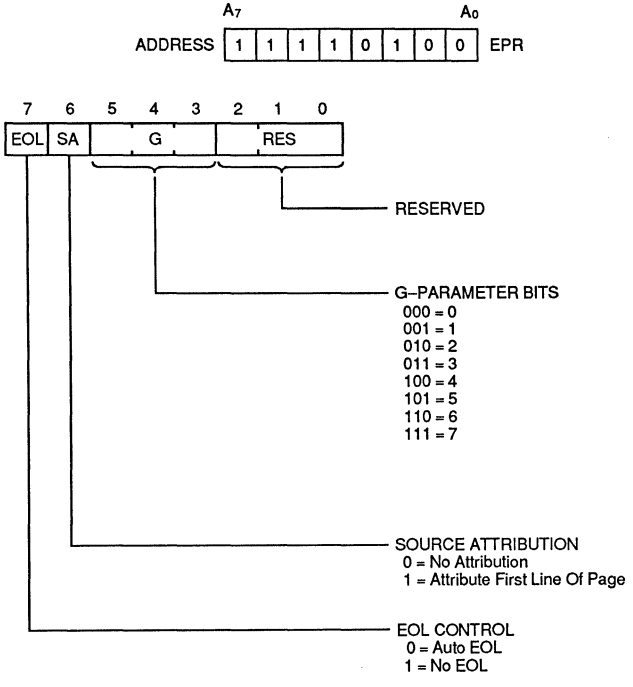
When the EOL bit is a "1" during the Transparent Mode, the 1D, or the 2D Modes, the Source Buffer data is assumed to contain no EOL codes.

If the Expander GO bit is set to "1" with EOL = 0 and SA = 1, the Expansion Processor starts processing data when it detects an EOL code from the previous line or prefixed at the beginning of a page. It continues its operation until it detects an EOL code on the current processing line. When this EOL is detected, the Expansion Processor will check that the bit-length of the expanded current line is equal to:

$$L = (EPWR \cdot 8) \cdot (EWR + 1)$$

Where: EPWR = The value of the Expander Page Width Register.
EWR = The value of the Expander Wraparound Register.

If the current line is not equal to this value, the Expansion Processor terminates in an error-state



07666A 2-13

Figure 2-13 Expander Parameter Register (EPR)

with the DER bit in the Expander Status Register set to "1".

2.2.13 K Parameter Registers (CKPR, EKPR)

The K Parameter Register is used in Group 3 coding (EOL bit = 0, MC field = 10) to specify the number of lines encoded in 2D for each line encoded in 1D. For example, if K = 5, four consecutive lines are compressed using two-dimensional encoding for each line compressed using one-dimensional encoding. The first line is always encoded in 1D in Group 3. If K = 0, all the lines are coded in 2D. This 8-bit register can specify K values from 1 to 255 in binary. Refer to Figure 2-14. In Group 4, all the lines are coded in 2D and K is always zero.

If a destination buffer smaller than the page is used with resumable coding in Group 3, it is necessary to recognize which lines are 1D and which lines are 2D (to determine if the last line encoded before overflow is still needed as a reference line when processing resumes. If the processing stopped so that the next line is a 1D line, a reference line is not needed. The simplest way to do this is to choose a buffer size and K parameter such that the buffer is an integral multiple of K. Since the first line is always encoded in 1D and K-1 lines are coded in 2D, the buffer will overflow stopping the processing after the last line of 2D. Under these conditions, each buffer overflow indicates that the next line to be encoded or decoded is a 1D line. With this arrangement of a buffer, no reference line is needed when coding or decoding resumes since the next line is a 1D line.

If One-Dimensional coding or Transparent Mode is specified in the Master Control Register, the processor logic ignores the K-Parameter. For 2D coding, the 7970A encodes Group 3 using the K Parameter and setting the tag bit following an EOL.

For Group 3 expansion, the 7970A processes each line as either 1D or 2D according to the value of the tag bit immediately following the EOL code terminating the previous line. If the tag bit is 1, the next line is decoded as a 1D line. If the tag bit is 0, the next line is decoded as a 2D line. The K Parameter is not used.

2.2.14 Wraparound Registers (CWR,EWR)

The Compressor and the Expander each have a Wraparound Register. This 16-bit register, specifies the number of additional scan lines that are grouped into one effective line during encoding. If the value is "0", then the effective line is identical to a scan line. This is the normal operating mode. If this register is loaded with a "1", two scan lines make up an effective line and so on. Refer to Figure 2-15.

Wraparound Mode cannot be used simultaneously with either Two-Dimensional Compression or Express Mode. Either of these combinations will result in an illegal command error. The Wraparound Register is not modified by the selected processor during its operation.

2.2.15 Page Width Registers (CPWR, EPWR)

Both the Compressor and the Expander have a Page Width Register. This 16-bit register, specifies the page width or length of a scan line in increments of 8 pixels. The largest line the Am7970A can handle is 16K pixels long because only 11 of these 16 bits are significant. Bits 11 through 15 must be set to "0". Refer to Figure 2-16.

CCITT recommendation T.4 covers compression and expansion of scan lines up to 2560 bits. The Am7970A accommodates much wider pages (up to 16K bits) by the use of multiple make-up codes to efficiently encode long runs.

Before starting a processor operation, the

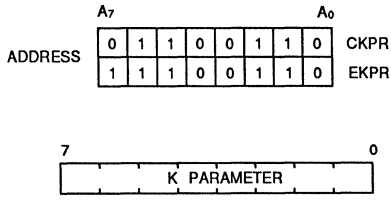


Figure 2-14 K Parameter Registers (CKPR, EKPR) 07666A 2-14

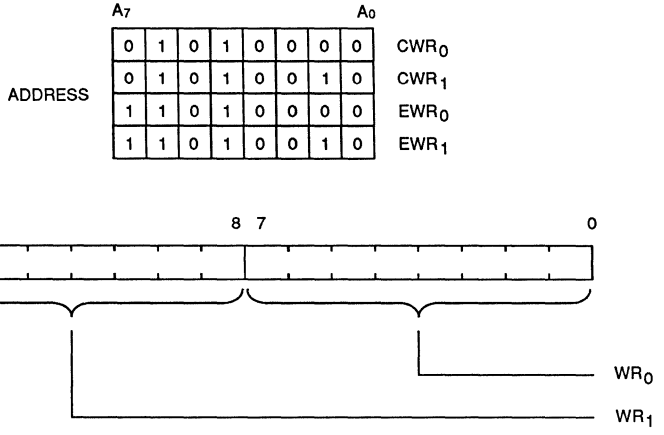
Am7970A checks the Page Width Register value against the Left Margin Register and the Right Margin Register values to ensure that the page width is greater than or equal to the sum of left and right margins. When the margin specifications are not consistent with the page width, the Am7970A will terminate operation after setting the Illegal Command (IC) bit in the appropriate Status Register to "1".

During expansion, the Expander Page Width Register (EPWR) is used in clipping the termination bits as follows: When the expanded data length reaches the line length specified in the EPWR, the expander resets an internal shift register and thus clips off any image bits comprising an incomplete byte at the end of an image line. The incomplete bytes of from 1 to 7 bits of image data at the end of each line are deleted. Otherwise, the coded color change in the bits of the last incomplete image byte would prevent succesful expansion. The processor logic does not modify the Page Width Register during its operation.

2.2.16 Frame Width Registers (CFWR, EFWR)

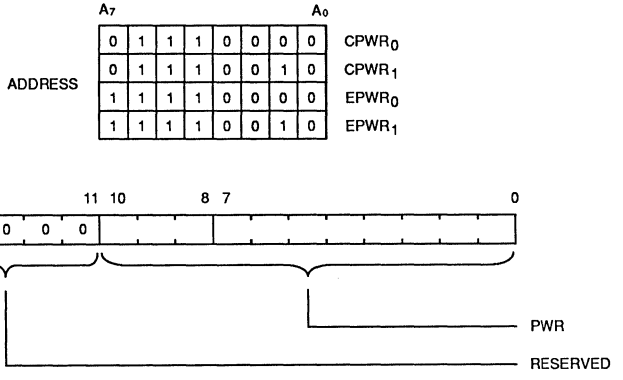
The 16-bit frame width registers define the width of the memory buffer used to store the picture image. These registers are used to calculate the starting point in memory of the next image line. (Am7970A Rev. A does not have Frame Width Registers). The CEP calculates the starting address of line n + 1 as one frame width beyond the starting address of line n. Refer to Figure 2-17.

For "full page" processing, the value in the Frame



07666A 2-15

Figure 2-15 Wraparound Registers (CWR, EWR)



07666A 2-16

Figure 2-16 Page Width Registers (CPWR, EPWR)

Width Register must equal the page width as specified in the Page Width Register. For full page processing, all of the data within the frame is processed. For "window" processing, the CFWR and EFWR define the width of the total image area (called a frame) and the CPWR and EPWR define the width of the window within the image area. Only the data within the window is processed. Each consecutive line of image in the window that is processed is transferred to or from consecutive lines in the frame in memory.

The first line starting address for "full page" and "window" addressing is specified by the CPU in the Source and Destination Address Hold Registers.

Margin control is effective during window processing. Therefore, it is possible to have a window with white margins.

2.2.17 Source Address Holding Registers (CSAHR, ESAHR)

The Compressor and the Expander each have a Source Address Holding Register. This 24-bit register contains the starting address of the Source Buffer for the selected processor. When a restart process is initiated, the Source Address Holding Register provides the initial value to the working register. The contents of Source Address Holding Register are loaded automatically into the Source Current Address Register whenever the GO bit in the Master Control Register is set, in conjunction with the SAC bit in the respective Restart Control Register being "0". The contents of the Source Address Holding Register are not modified by the processor during its operation. Refer to Figure 2-18.

2.2.18 Source Current Address Registers (CSCAR, ESCAR)

The 24-bit CSCAR provides the current address for all Compressor transactions with the Compressor Source Buffer. Likewise, the ESCAR provides the current address for Expander transactions with the Expander Source Buffer. After each transaction this register is incremented by one and will wraparound through "0" after reaching a maximum value of all "1"s. Refer to Figure 2-19.

If the SAC bit in the selected Restart Control Register is "0", the SCAR is loaded from the Source Address Holding Register whenever a new Go operation is initiated. On the other hand, if the SAC bit in the Restart Control Register is "1", the SCAR continues from its current value.

Two-Dimensional processing requires not only data for the current line but also corresponding data from the previous line. The Am7970A calculates the initial address of the reference line using the Source Line Start Address Register. From there on, the address of the reference line is incremented appropriately.

Reference Line Starting Address = SLSR – FWR

where SLSR = Source Line Start Address Register
FWR = Frame Width Register

2.2.19 Source Count Holding Registers (CSCHR, ESCHR)

The Compressor Source Count Holding Register specifies the Compressor Source Buffer length in bytes. The Am7970A requires that this buffer length be specified as a *negative number in two's complement form.* Refer to Figure 2-20.

The contents of the Source Count Holding Register are loaded automatically into the Source

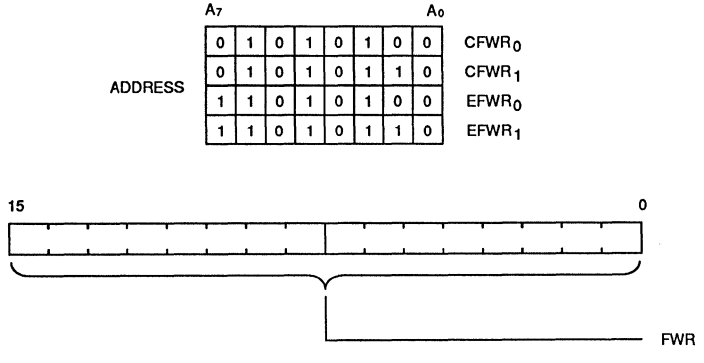


Figure 2-17 Frame Width Registers (CFWR, EFWR) 07666A 2-17

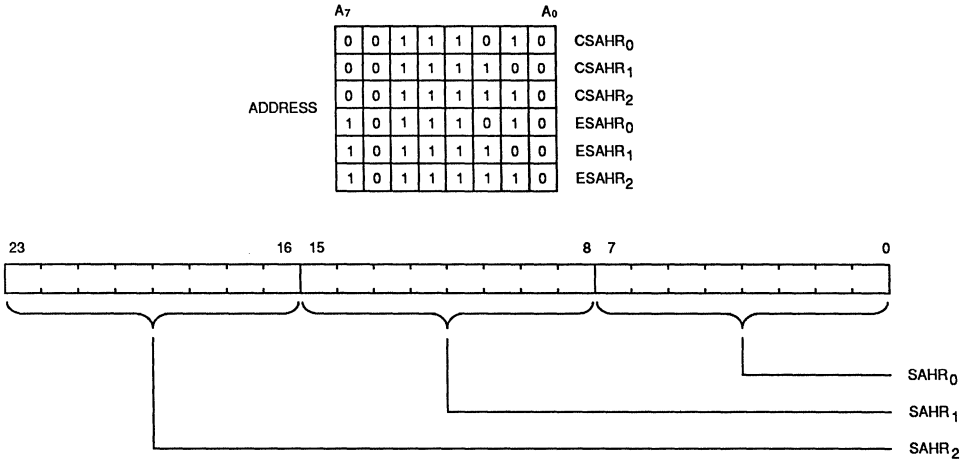
Working Count Register whenever a Go operation is initiated in conjunction with the Source Count Control (SCC) bit in the Restart Control Register being "0".

The Expander Source Count Holding Register provides the identical function in the Expander.

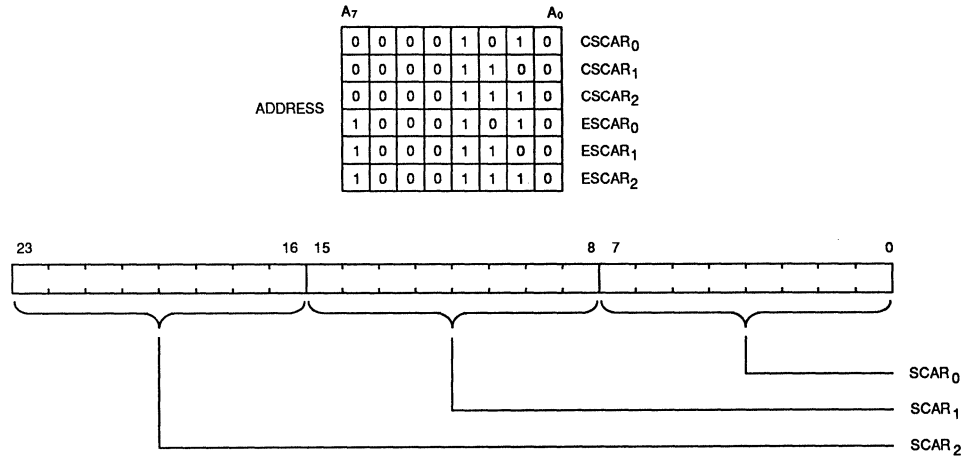
2.2.20 Source Working Count Registers (CSWCR, ESWCR)

The Compressor Source Working Count Register keeps track of the number of Compressor Source Buffer transactions, Initially, the CSWCR contains a 24-bit *negative number in two's complement form* specifying the number of bytes in the Source Buffer. The Am7970A increments the Source Working Count Register by one after completing each source transaction. The Source Overflow bit (SO) in the Status Register is set to "1" immediately after the Source Working Count Register count becomes zero.

The CEP controls the contents of the Working



07666A 2-18 **Figure 2-18 Source Address Holding Registers (CSAHR, ESAHR)**



07666A 2-19 **Figure 2-19 Source Current Address Registers (CSCAR, ESCAR)**

Count Register only when it increments it. The CEP does not observe what value is loaded by the CPU and thus does not interrupt for "0" contents. Refer to Figure 2-21.

It must be emphasized that a transaction does not necessarily mean one access. For example, for Two-Dimensional processing, the Am7970A will access the Source Buffer both for the current line and for the reference line. Thus, a transaction in 2D will include twice as many Source Buffer accesses as an equivalent 1D transaction.

When the last byte of the Source Buffer corresponds to the end of an effective line, the Am7970A will process this byte and will attempt to store it in the Destination Buffer and then terminate the processor. For normal operation, the Am7970A terminates after storing a fully processed line successfully (that is, no premature destination overflow or error status). Abnormal termination due to premature destination overflow or error status is discussed later.

When the last byte obtained from the Source Buffer does not correspond to the end of an effective line, it means that the assigned Source Buffer length is less than what is needed by the Page Width Register and the Wraparound Register (if used). In this condition, the Am7970A will process the byte and then terminate abnormally. The Wraparound Incomplete (WPI) and Line Processing Incomplete (LPI) bits in the Status Register will reflect the appropriate error termination.

When an operation is initiated with the SCC bit in the Restart Control Register set to "0", the Source Working Count Register obtains its initial value from the Source Count Holding Register. When this control bit is a "1" and a Restart operation is initiated, the Source Working Count Register uses the existing value as the initial value. The Expansion Source Working Count Register is an equivalent register in the Expander.

2.2.21 Source Line Start Address Registers (CSLSR, ESLSR)

The 24-bit Compressor Source Line Start Address Register contains the address of the first byte of the current effective line in the Source Buffer. This register is automatically updated when the Am7970A begins processing a new effective line. Thus, after a successful completion of Single-Line operation, the CSLSR contains the starting address of the line that was just completed until processing is started on a new effective line. Refer to Figure 2-22.

During Multi-Line operations, the Am7970A automatically begins processing the next line after successfully processing a line. The Compressor always loads the Source Current Address Register contents into the Source Line Start Register and then begins processing of the new line. This facilitates resetting the source back to the beginning of the line in case of a premature termination. The initial contents (not the updating process) of the Source Line Start Address Register are determined by the setting of certain bits in the Restart Control Register. For details of this operation, see the description under the Restart Control Register section.

The Expander Source Line Start Address Register is identical to the Compressor Source Line Start Address register.

2.2.22 Destination Address Holding Registers (CDAHR, EDAHR)

The 24-bit Compressor Destination Address Holding Register specifies the starting address of the Compressor Destination Buffer during Restart operations. The contents of the CDAHR are automatically transferred into the Compressor Destination Current Address Register whenever the GO bit in the CMCR is set and the DAC bit in the Restart Control Register is "0". The CDAHR contents are not modified by the Compressor during its operation. Refer to Figure 2-23.

The Expander Destination Address Holding Register is identical to the CDAHR and performs the same function in the Expander on comparable Expander registers.

2.2.23 Destination Current Address Registers (CDCAR, EDCAR)

The 24-bit Compressor Destination Current Address Register provides the current address of the Compressor Destination Buffer. The Am7970A increments the CDCAR by one after each transaction and will wraparound through "0" after reaching a maximum value of all "1"s. Refer to Figure 2-24.

If the DAC bit in the Restart Control Register is "0" when the GO bit in the Master Control Register is set to "1", the CDCAR is loaded from the CDAHR. If the DAC bit is "1", the CDCAR continues from the current value upon receipt of the Go operation.

2.2.24 Destination Count Holding Registers (CDCHR, EDCHR)

The 24-bit Compressor Destination Count Holding Register specifies the length (in bytes) of the Compressor Destination Buffer. The buffer length

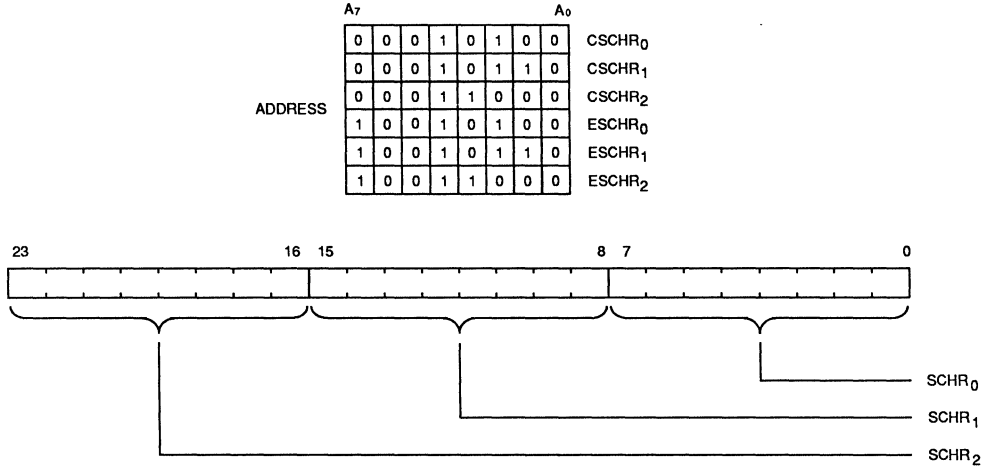
must be specified as a *negative number in two's complement form.* The contents of the CDCHR are loaded automatically into the Compressor Destination Working Count Register whenever a Restart operation is initiated with the Destination Count Control (DCC) bit in the Restart Control Register set to "0".

The Compressor logic does not modify this register during its operation. Refer to Figure 2-25.

The Expander Destination Count Holding Register is an identical register in the Expander.

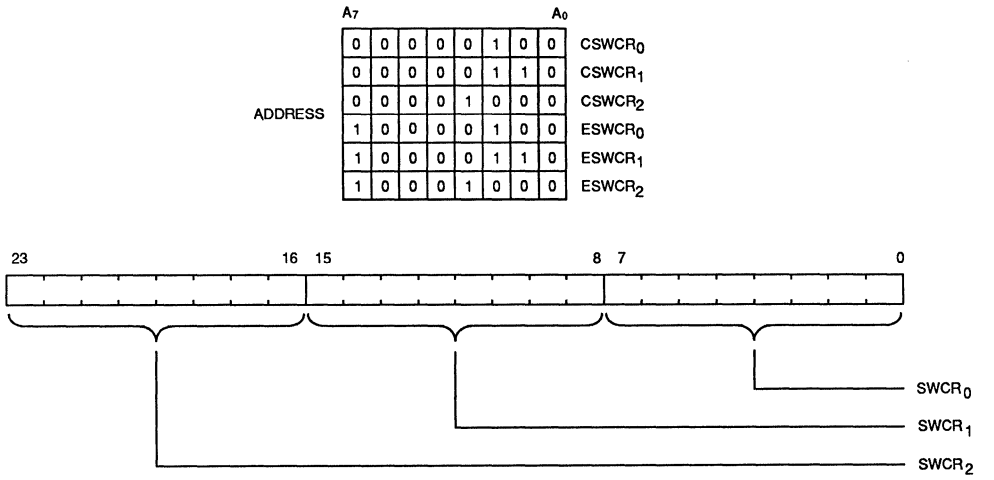
2.2.25 Destination Working Count Registers (CDWCR, EDWCR)

The 24-bit Compressor Destination Working Count Register keeps track of the number of Compressor Destination Buffer accesses. Initially, the Destination Working Count Register contains a 24-bit *negative two's complement number* specifying how many bytes long the Destination Buffer is. Refer to Figure 2-26.



07666A 2-21

Figure 2-20 Source Count Holding Registers (CSCHR, ESCHR)



07666A 2-21

Figure 2-21 Source Working Count Registers (CSWCR, ESWCR)

The Am7970A increments the Destination Working Count Register by one after each destination transaction. As soon as the Destination Working Count Register reaches "0", the processor operation will terminate with processor Destination Overflow (DO) bit set in the appropriate Status Register. It should be noted that such a termination might be normal (error-free) or abnormal.

A normal termination occurs when the Am7970A is able to transfer a fully-processed line into the Destination Buffer without a premature destination overflow and without an error such as negative compression. An abnormal termination due to premature destination overflow will result in an appropriate error status. In this condition, the Am7970A will process the byte and then terminate abnormally. The Wraparound Incomplete (WPI) and Line Processing Incomplete (LPI) bits in the Status Register reflect the appropriate error termination.

When Restart Operation is initiated with the DCC bit in the Restart Control Register set to "0", then the initial value loaded into the Destination Working Count Register is obtained from the Destination Count Holding Register. However, if

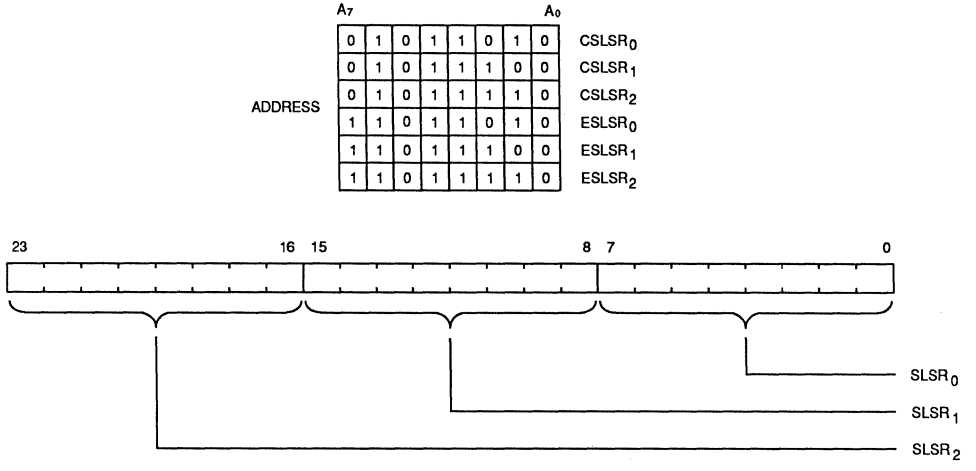


Figure 2-22 Source Line Start Address Registers (CSLSR, ESLSR) 07666A 2-22

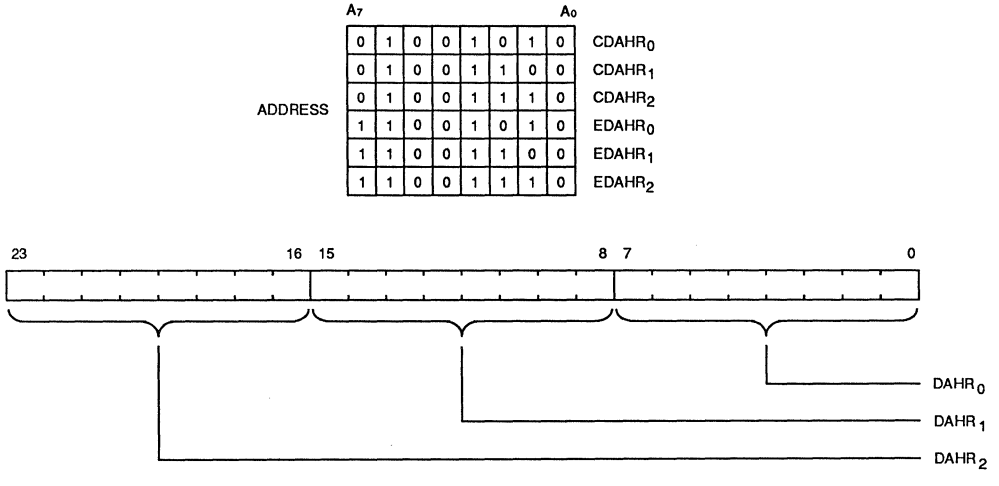


Figure 2-23 Destination Address Holding Registers (CDAHR, EDAHR) 07666A 2-23

the DCC bit is "1", the existing value in the Destination Working Count Register becomes the initial value.

The Expander Destination Working Count Register is an identical register in the Expander.

2.2.26 Destination Line Start Address Registers (CDLSR, EDLSR)

The 24-bit Compressor Destination Line Start Address Register contains the address of the first byte of the current line in the Destination Buffer. This register is updated automatically when the processor starts a new line. Thus, after the successful completion of a Single-Line operation, the Destination Line Start Address Register contains the starting address of the line that was just completed until the processor starts a new line. Refer to Figure 2-27.

During Multi-Line operations, the processor automatically begins processing the next line after successful completion of a line. The Am7970A loads the Destination Line Start Address Register from the Destination Current Address Register before processing of the line begins. The Source Line Start Address Register is also loaded from the

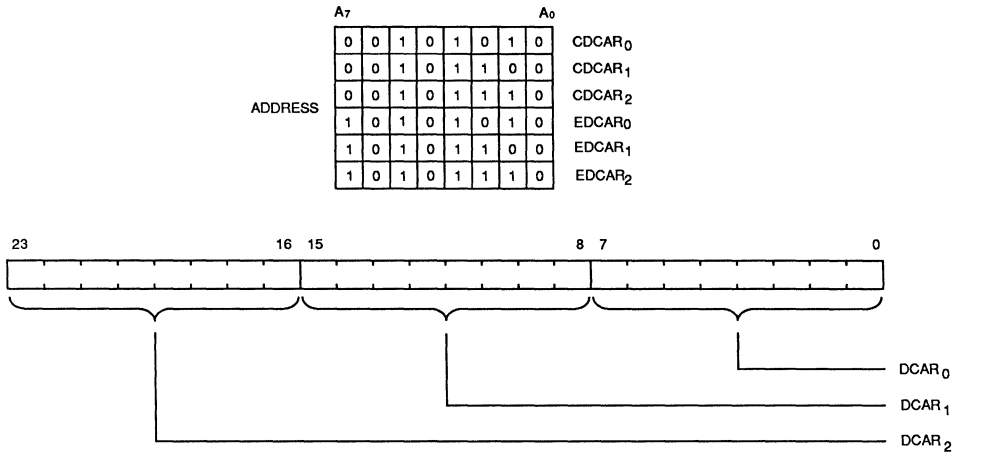


Figure 2-24 Destination Current Address Registers (CDCAR, EDCAR) 07666A 2-24

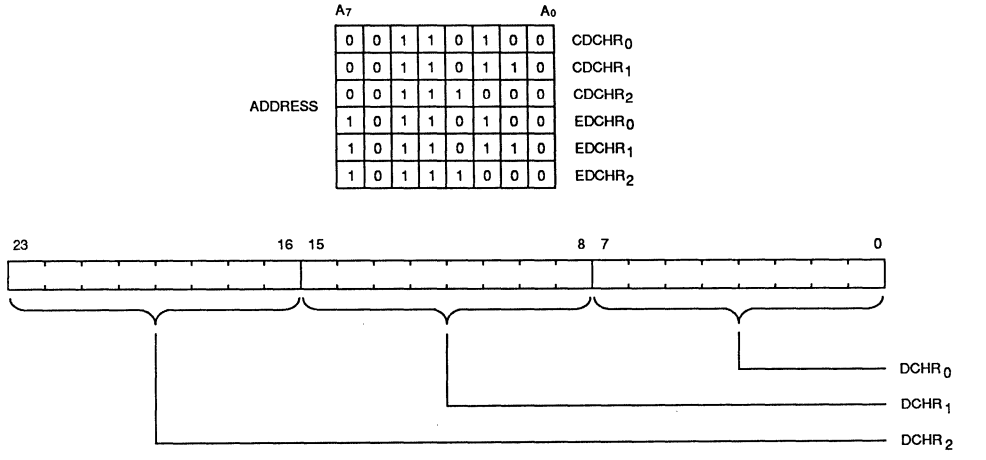


Figure 2-25 Destination Count Holding Registers (CDCHR, EDCHR) 07666A 2-25

Source Current Address Register. Thus before any access to the Destination Buffer for the new line, both the Source Line Start Address Register and the Destination Line Start Address Register are updated. This facilitates resetting the destination back to the beginning of a line in case of a premature termination.

The initial contents (not the updating process) of the Destination Line Start Address Register are determined by values loaded into the Restart Control Register. See the section on this register for further details.

The Expander Destination Line Start Address Register is an identical register in the Expander.

2.3 INTERFACE DESCRIPTION

The interface description includes a description of the signals on the CEP pins, and the sequences involved in CPU access operations, DMA operations, and Document Bus operations.

All inputs to the CEP are directly TTL-compatible. CEP control signals include CLK and RESET. The two bus interfaces are the system bus and the document bus. The system bus control signals include $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, ALE, HRQ, HLDA, READY, and INTR. The document bus control signals include the $\overline{DRD}$, $\overline{DWR}$, $\overline{DALE}$, and DREADY.

One DMA Controller serves both the system bus and the document bus. Therefore, there is never a simultaneous DMA access on both the system bus and the document bus. However, slave access on the system bus is allowed while a document bus DMA is in progress. If the CEP is inactive ("busy" bits inactive), all signals of the document side are tri-stated. This feature can be used for an inexpensive software controlled bus arbitration of the document bus. When a peripheral wants to access the document bus, it can notify the system CPU. The CPU can poll the BUSY bit of the CEP and notify the peripheral when it becomes inactive.

The CEP operation may be stopped by a write to the command register when the CEP is busy (a software abort). However, this abort is not resumable. In other words, setting the "GO" bit back to High is not enough to resume an aborted operation.

The system side interface transfers only one byte per arbitration cycle (no burst DMA). The document side interface transfers single bytes for each bus cycle; contiguous bus cycles can be initiated by the CEP on the document bus.

2.3.1 Signal Description

The interface signals include the CEP control signals, the System Bus signals, and the Document Bus signals.

CLK Clock (Input)

The Clock signal controls most of the CEP's internal operations and determines the rates of its data transfers. It is usually derived from a master

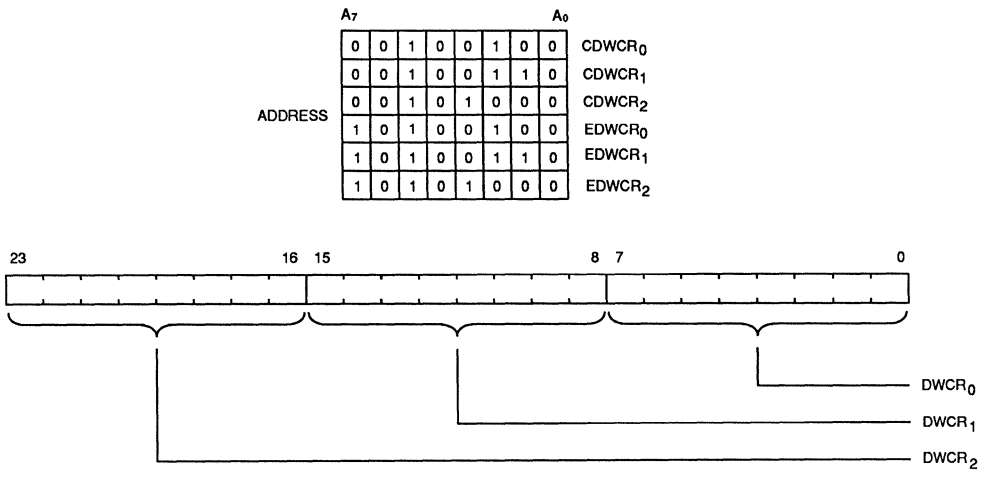


Figure 2-26 Destination Working Count Registers (CDWCR, EDWCR) 07666A 2-26

system clock or the associated CPU clock. The Clock input accepts a TTL voltage level. The input signals $\overline{CS}$, HLDA, $\overline{RD}$, and $\overline{WR}$, can make transitions independent of the CEP clock (asynchronous operation).

Transitions on the inputs, READY and DREADY, must meet set-up and hold requirements relative to the CEP clock, since these inputs do not contain internal synchronizers. Failure to meet these timing requirements may result in incorrect operation from the internal state machine with unpredictable consequences. See the timing diagrams in the data sheet for details.

RESET (Input)

RESET is an asynchronous, active-High input which initializes the Am7970A to an idle state. This input must be driven High for at least four clock cycles.

The hardware reset sets the GO, Status, and Interrupts to zero. It resets the DMA bus. It sets BUSY to zero, flushes out the pipeline, and sets up the checks for configuration errors. The hardware reset also sets the software reset for both the compressor and expander.

$\overline{RD}$ Read (Input/Output, Active low, Three-state)

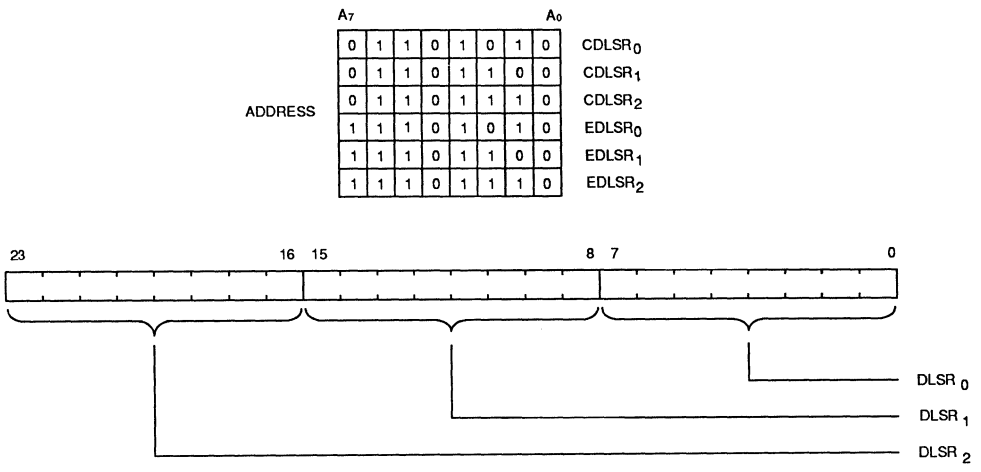
$\overline{RD}$ is a bidirectional, active-Low, three-state signal. A Low indicates that the AD_{16}–AD_{23} bus is being used for a Read Data Transfer. When the CEP is not in control of the system bus and the external system is transferring information from the CEP, $\overline{RD}$ is an asynchronous timing input used by the CEP to move data between registers and the AD_{16}–AD_{23} bus. The $\overline{RD}$ cycle from the system should be completed only after the CEP's READY output has returned High. After $\overline{RD}$ returns to its High state, AD_{16}–AD_{23} will float.

$\overline{RD}$ is an output when the CEP is Bus Master (HRQ and HLDA are both High). The CEP asserts $\overline{RD}$ Low when data from system memory is required. The CEP strobes this data into its internal buffers from the AD_{16}–AD_{23} lines near the Low-to-High transition of $\overline{RD}$.

$\overline{WR}$ Write (Input/Output, Active low, Three-state)

$\overline{WR}$ is a bidirectional, active-Low, three-state signal. A Low indicates that the AD_{16}–AD_{23} bus is being used for a Write Data Transfer. When the CEP is not in control of the system bus and the external system is transferring information to the CEP, $\overline{WR}$ is an asynchronous timing input used by the CEP to move data from the AD_{16}–AD_{23} bus into its internal registers. The data is loaded into the specified register before the CEP's READY output is driven High. This $\overline{WR}$ cycle from the system should be completed only after the CEP's READY output has returned High.

$\overline{WR}$ is an output when the CEP is Bus Master (HRQ and HLDA are both High). The CEP asserts $\overline{WR}$ Low when data is to be written into Main Memory. The CEP drives this data onto its AD_{16}–AD_{23} lines near the High-to-Low transition of $\overline{WR}$. See timing



07666A 2-27 **Figure 2-27 Destination Line Start Address Registers (CDLSR, EDLSR)**

diagrams in the data sheet for more details.

$\overline{CS}$ Chip Select (Active low, Input)

$\overline{CS}$ is an asynchronous, active-Low input. A CPU or other external device uses $\overline{CS}$ to activate the CEP for reading from or writing to its internal registers. Once asserted Low, this input can remain Low until all register accesses have been completed. Once $\overline{CS}$ is negated High, it may not be asserted Low again for at least 100ns. There are no timing requirements between the $\overline{CS}$ input and the CEP clock; the $\overline{CS}$ input timing requirements are only defined relative to the $\overline{RD}$ and $\overline{WR}$ signals. $\overline{CS}$ is ignored when the CEP is in control of the system bus.

Do not use $\overline{CS}$ directly to enable the buffer for $\overline{RD}$ or $\overline{WR}$ into the CEP in slave mode. When $\overline{CS}$ goes Low, the outputs of the buffer will still be unstable for a couple of nanoseconds while the CEP is already expecting true signals. The best solution is to use HLDA for enabling the driver if the system requires buffering the $\overline{RD}$ and $\overline{WR}$ lines.

Make sure $\overline{CS}$ is High within 1 clock cycle after $\overline{RD}$ or $\overline{WR}$ return High if block transfer mode is not desired. Refer to the CPU Block I/O Transaction Timing Diagram in the data sheet for block transfers.

ALE Address Latch Enable (Output)

This active-High signal is provided by the CEP to latch the address signals AD_{16}–AD_{23} into an address latch. This pin is never floated. ALE is asserted High during address time when the CEP is Bus Master; otherwise it is Low. Address is defined as valid prior to the High-to-Low (trailing) transition of ALE.

HRQ Hold Request (Output)

Hold Request is an active-High signal used by the CEP to obtain control of the bus from the system CPU or arbiter. Hold Request lines from multiple devices may be connected to a priority encoder. If the HLDA input is High after the HRQ output has been asserted High, HRQ will remain High until the CEP has completed one memory transaction. The HLDA input may be negated Low prior to HRQ going Low. The HRQ signal remains Low for a minimum of 2 clocks to allow the bus master to arbitrate for the bus. This sequence of events is called a preemption. If HLDA is not asserted, HRQ can be forcibly lowered only by a hardware reset.

HLDA Hold Acknowledge (Input)

HLDA is an asynchronous, active-High input indicating that the CPU has relinquished the bus and that no higher priority device has assumed bus control. Since HLDA is internally synchronized by the CEP before being used, transitions on HLDA do not have to match setup and hold requirements with respect to the CEP clock. The HLDA input can be connected to either the HLDA output from the CPU (8086-type) or to the output of a device such as a priority decoder. The HLDA input normally remains High until the CEP drives the HRQ output Low.

The HLDA input to the CEP can be negated Low prior to HRQ. This forces the CEP to release the bus within a maximum time of 5 clock periods (assuming READY is High and no wait states). The removal of HRQ indicates to the preempting device that the CEP has released the bus. In either case, HRQ remains LOW for a minimum of 2 clocks. Processing is resumed from the point of preemption in the normal course of events. In any case, the system bus transfers only one byte per arbitration cycle.

READY (Input/Output, Three-state)

READY is a synchronous, active-High, three-state, bidirectional signal. READY is used as an input signal when the CEP is Bus Master. In Master Mode, the CEP samples the READY line with the rising edge of T2 before $\overline{RD}$ or $\overline{WR}$ are asserted by the CEP. See timing diagrams in the data sheet for more details.

Slow memories may use READY to extend $\overline{RD}$ or $\overline{WR}$ cycles. This is accomplished by negating READY Low at the appropriate times and thus inserting Wait states until READY is returned High. READY must be High before Main Memory data can be accessed by the CEP. Care must be taken, however, to assure that this signal is synchronized to the CEP clock and meets its set-up and hold requirements as specified in the data sheet. Failure to do so causes unpredictable operation.

READY is used as an output signal when the CEP is Bus Slave. After $\overline{CS}$ has been asserted Low by the CPU, READY is kept Low by the CEP until it is able to provide or accept data for the current transaction. When ready, the CEP asserts READY High at which time the CPU should complete the current read or write cycle by negating $\overline{RD}$ or $\overline{WR}$.

INTR Interrupt Request (Output)

Interrupt Request is an active-High output used to interrupt the CPU. It is driven High whenever an exception or terminating condition exists in either the Compressor (if the Compressor Interrupt Enable bit is set) or Expander (if the Expander

Interrupt Enable bit is set). The INTR line is reset to Low when the CPU reads the CEP Master Status Register or when the CEP is hardware reset.

A_0–A_{15} Lower Address (Input, tri-state outputs)

In the Bus Slave mode, the Lower Address Bus is a non-multiplexed, bidirectional bus of the seven address lines (A_1–A_7). It is used in addressing all system bus I/O and memory transactions.

When the CEP is not in control of the system bus (HRQ and HLDA Low), and the $\overline{CS}$ input is asserted Low, A_1–A_7 are used as input address lines to access the CEP's internal registers. (The CEP's internal registers have been assigned even addresses.) During this time, the address lines A_8–A_{15} are ignored by the CEP. The input addresses on A_0–A_7 do not have to be valid before the $\overline{CS}$ input is driven Low but must remain valid throughout the register transaction. See timing diagrams in the data sheet for more details.

In the Bus Master mode (A_0–A_{15} tri-state outputs), the CEP is in control of the main bus (HRQ and HLDA are High) and the lower address bus is an active-High, three-state bus with A_0 the least significant bit position and A_{15} the most significant bit position.

DMA transactions with the Main Memory will occur. The presence of valid address on A_0–A_{15} is defined by the falling edge of ALE. A_0–A_{15} are used as non-multiplexed output address lines during the memory transactions. These lines are enabled 2 clock cycles after HREQ and HLDA = High. After the High-to-Low transition of HRQ, the A_0–A_{15} lines will float to a three-state condition.

AD_{16}–AD_{23} Address-Data Bus (Input/Output, Three-state)

The Address-Data Bus is a time-multiplexed (in Master Mode only), bidirectional, active-High, three-state bus used for all system bus I/O and memory transactions. When referring to the data cycle on this bus, AD_{16} is the least significant data bit position and AD_{23} is the most significant.

The presence of a valid address during Bus Master operations is defined by the falling edge of ALE and valid data is defined by the $\overline{WR}$ and $\overline{RD}$ signals; otherwise these lines are floating. While the CEP $\overline{RD}$ output is Low, AD_{16}–AD_{23} must contain valid input data from the system while the READY input is High,

When the CEP $\overline{WR}$ output is asserted Low, AD_{16}–AD_{23} has valid CEP output data. The READY input must then return High to acknowledge receipt of the valid data and to allow the completion of the $\overline{WR}$ cycle.

When the CEP is acting as a Bus Slave (HRQ and HLDA Low) and the $\overline{CS}$ input is driven Low, AD_{16}–AD_{23} are used strictly as data lines D_0–D_7. They behave as input data lines when $\overline{WR}$ is asserted Low and as output data lines when $\overline{RD}$ is asserted Low. At all other times they are floated to three-state.

$\overline{DRD}$ Document Store Read (Active low, Output, Three-state)

$\overline{DRD}$ is an active-Low, three-state signal. A Low on this signal indicates that the DAD_{16}–DAD_{23} bus is being used for a Read Data Transfer. When the CEP does not have a source or destination buffer located on the Document Store bus, this pin is floated to three-state. Even if the CEP is programmed to access the Document Bus, these lines go floating whenever the CEP is performing internal operations rather than transfering data through this interface. Therefore, usually a pullup resister must be connected to $\overline{DRD}$.

$\overline{DRD}$ is an output when the CEP is in control of the Document Bus. The CEP asserts $\overline{DRD}$ Low when data from Document Store is required. The CEP strobes this data into its internal buffers near the Low-to-High transition of $\overline{DRD}$.

$\overline{DWR}$ Document Store Write (Active low, Output, Three-state)

$\overline{DWR}$ is an active-Low, three-state signal. A Low on this pin indicates that the DAD_{16}–DAD_{23} bus is being used for a Document Bus write data transfer. When the CEP does not have a source or destination buffer located on the Document Store Bus, this pin is floated to three-state. Even if the CEP is programmed to access the Document Bus, these lines go floating whenever the CEP is performing internal operations rather than transfering data through this interface. Therefore, usually a pullup resister is required on $\overline{DWR}$.

$\overline{DWR}$ is an output when the CEP is Bus Master. The CEP asserts $\overline{DWR}$ Low when data is to be written into Document Store. The CEP drives this data onto its DAD_{16}–DAD_{23} lines near the High-to-Low transition of $\overline{DWR}$. See timing diagrams for details.

DALE Document Store ALE (Output, Three-state)

This active-High signal is provided by the CEP to latch the Document Store address signals

DAD_{16}–DAD_{23} into an address latch to separate addresses from data. When the CEP does not have a source or destination buffer located on the Document Bus, this pin is floated to three-state. Even if the CEP is programmed to access the Document Bus, these lines go floating whenever the CEP is performing internal operations rather than transfering data through this interface. Therefore, a pullup resistor is required on DALE. DALE in conjunction with a pullup resister makes a perfect $\overline{AS}$ signal for 68000-like systems because it changes directly from 3-state to High. This may be a useful low-active memory-enable signal for the document bus.

DALE is asserted High during address time when the CEP is Bus Master; during the remainder of the transaction, it is Low. Address is defined as valid prior to the transition of DALE.

DREADY Ready (Input, Three-state)

DREADY is a synchronous, active-High, three-state signal. DREADY is used as an input signal when the CEP is Bus Master. In Master Mode, the CEP samples the DREADY line with the rising edge of T2 before $\overline{DRD}$ or $\overline{DWR}$ are asserted by the CEP. See timing diagrams in the data sheet for more details.

Slow memories may use DREADY to extend $\overline{DRD}$ or $\overline{DWR}$ cycles. This is accomplished by negating DREADY Low at the appropriate times and thus inserting Wait states until DREADY is returned High. DREADY must be High before Document Store data can be accessed by the CEP. Care must be taken, however, to ensure that this signal is synchronized to the CEP clock and meets its set-up and hold requirements as specified in the data sheet. Failure to do so can result in unpredictable operation.

If the DREADY signal is suppressed on the document bus, the CEP will be frozen the moment that it samples the DREADY line because it is waiting for access to the document bus. No further memory transfers can take place on the system bus either, because each side is waiting for the other to complete the memory cycle. HREQ is inactive in this case. This behavior might be useful for implementing a ring buffer as a destination buffer or in conjunction with transceivers for DA_{016}–DA_{073} and buffers for DA_0–DA_{16},DRD, DWR, and control logic, for a bus arbitration scheme on the document bus.

DA_0–DA_{15} Document Store Lower Address Bus (Output, Three-state)

The Document Store Lower Address Bus is a non-multiplexed, active-High, three-state bus used in addressing all local document memory transactions. DA_0 is the least significant bit position and DA_{15} is the most significant bit position.

When the CEP is in control of the Document Store Bus, the presence of a valid address on DA_0–DA_{15} is defined by the falling edge of DALE. During this Master Mode, DA_0–DA_{15} are used as non-multiplexed output address lines whenever the Compressor or the Expander is using the Document Store as a source or destination buffer for the current transaction, otherwise this bus is floated to three-state.

DA_{16}–DA_{23} Document Store Upper Address-Data Bus (Input/Output, Three-state)

The Document Store Upper Address-Data Bus is a time-multiplexed, bidirectional, active-High, three-state bus used for all local document memory transactions. When referring to the data cycle on this bus, DAD_{16} is the least significant data bit position and DAD_{23} is the most significant. The presence of a valid address during Bus Master operations is defined by the falling edge of DALE and the valid data is defined by the $\overline{DWR}$ and $\overline{DRD}$ signals; otherwise these lines are floating.

The $\overline{DRD}$ and $\overline{DWR}$ outputs return to their inactive-High levels only after the DREADY input has been sampled High. While the CEP $\overline{DRD}$ output is Low, DAD_{16}–DAD_{23} must be provided with valid input data from the system while the DREADY input is High. When the CEP $\overline{DWR}$ output is asserted Low, DAD_{16}–DAD_{23} is driven by the CEP with valid output data. The DREADY input must then return High to acknowledge receipt of the valid data and to allow the completion of the $\overline{DWR}$ cycle.

2.3.2 CPU Access Operations (CEP Slave Mode)

Timing diagrams, Figures 2-28 and 2-29, show idealized read and write timing relationships between the signals to provide a quick overview. Figure 2-30 shows a CPU block I/O transaction timing with the CEP in the Slave Mode. For more detailed timing specifications, refer to the data sheet. The procedure is described as follows:

Read Access Operation

1. The CPU places an address on the CPU address lines A1–A7 to specify the intended register and enable the $\overline{RD}$ and $\overline{WR}$ lines to the CEP.
2. The CPU address decoder (external to the CEP) drives the $\overline{CS}$ input Low.

3. The CEP drives the READY output Low.
4. The CPU drives the $\overline{RD}$ input Low.
5. The CEP READY output is driven High when the CEP register data is available.
6. AD_{16} to AD_{23} is driven by the CEP with valid data.
7. The CPU drives the CEP $\overline{RD}$ input High.
8. CEP drives READY Low a maximum of one clock cycle after $\overline{RD}$ is High.
9. If the $\overline{CS}$ input is driven High, further write accesses can be initiated by executing from Step 1.

   If the $\overline{CS}$ input is kept low, further read or write accesses can be initiated by executing from Step 1 but skipping Step 2. (Block transfer mode, Figure 2-30.)

10. READY returns to High after $\overline{CS}$ High.

Note: If Step 7 precedes Step 5, the read access will be aborted by the CEP. If the read access is aborted, READY is driven High and AD_{16} to AD_{23} will float. No data will be presented on the data lines. Further read accesses can be initiated by executing Step 1.

The register access can take up to 50 clock cycles depending on the internal operation of the CEP. Refer to Section 2.2 for more details.

Write Access Operation

1. The CPU places an address on the CPU address lines A1–A7 to specify the intended register and enable the $\overline{RD}$ and $\overline{WR}$ lines to the CEP.
2. The CPU address decoder (external to the CEP) drives the $\overline{CS}$ input Low.
3. The CEP drives the READY output Low.
4. The CPU drives the $\overline{WR}$ input Low.
5. AD_{16} to AD_{23} is driven by the CPU with valid data.
6. The CEP READY output is driven High after data has been loaded into the appropriate register.
7. The CPU drives the CEP $\overline{WR}$ input High.
8. CEP drives READY Low a maximum of one clock cycle after $\overline{RD}$ is High.
9. If the $\overline{CS}$ input is driven High, further write accesses can be initiated by executing from Step 1.

   If the $\overline{CS}$ input is kept low, further read or write accesses can be initiated by executing from Step 1 but skipping Step 2.

10. READY returns High after $\overline{CS}$ High. (Block transfer mode, Figure 2-30.)

Note: If Step 7 precedes Step 6, the write access will be aborted. If the write access is aborted, READY is driven High. No data will be presented on the data lines. The contents of the specified register are not altered. Further write accesses can be initiated by executing Step 1.

The register access can take up to 50 clock cycles depending on the internal operation of the CEP. Refer to Section 2.2 for more details.

2.3.3 DMA Operation (CEP Master Mode)

Figures 2-31 and 2-32 are idealized timing diagrams of the system side DMA read and write operations. Refer to the data sheet for more detailed timing information. The procedure by which the CEP executes a DMA operation in the CEP Master Mode is described as follows:

Read Access Operation

1. The CEP drives the HRQ output High.
2. The CPU drives the HLDA input High.
3. The CEP enables address/data, $\overline{RD}$, and $\overline{WR}$ lines two clock cycles after step 2 has been executed (Tsync).
4. The CEP drives the ALE output High and places a memory address (24 bits) on the CPU bus during CEP state T1. The address is valid during the High to Low transition of the ALE output (CEP state T2, falling edge).
5. The CEP drives the $\overline{RD}$ output Low (CEP state T2, rising edge).
6. The state (High or Low) of the READY input is sampled by the rising edge of T2.
7. If the READY input is Low, a Wait State is inserted. The READY input should become High when the memory location becomes available (CEP Wait State, rising edge). Any number of wait states will be inserted as long as READY is sampled Low during state Tw.
8. If the READY input is High, then AD_{16} to AD_{23} must be driven with valid data (from the indicated memory location). The rising edge of T3 samples the data on AD_{16} to AD_{23}.
9. The CEP drives the $\overline{RD}$ output High (CEP state T3, rising edge).
10. The CEP disables address/data, $\overline{RD}$, and $\overline{WR}$ lines.
11. The CEP drives the HRQ output Low. It should remain Low for at least two clock cycles (falling edge after T3).
12. The CPU drives the HLDA input Low.

Note: If Step 12 precedes Step 11 (preemption),

the CEP will complete the current bus transaction. The HRQ output will then be driven LOW for at least two clock cycles. If additional bus transactions are required by the CEP, the CEP will drive the HRQ output High after a minimum of two clocks have elapsed (Step 1).

Write Access Operation

1. The CEP drives the HRQ output High.
2. The CPU drives the HLDA input High.
3. The CEP enables address/data, $\overline{RD}$, and $\overline{WR}$ lines two clock cycles after step 2 has been executed (Tsync).
4. The CEP drives the ALE output High and places a memory address (24 bits) on the CPU bus (state T1, falling edge). The address is valid during the High to Low transition of the ALE output.
5. AD_{16} to AD_{23} is driven by the CEP with valid data (CEP state T2, falling edge).
6. The CEP drives the $\overline{WR}$ output Low (CEP state T2, rising edge).
7. The state (High or Low) of the READY input is sampled by the CEP at the state T2 rising edge.
8. If the READY input is Low, a Wait State is inserted. The READY input should be driven High after data has been loaded into the

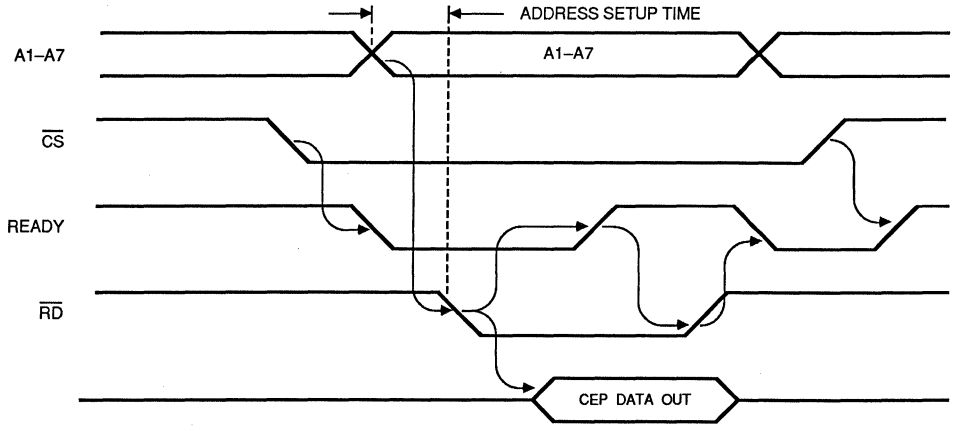


Figure 2-28 CPU Read Timing (CEP Slave Mode) 07666A 2-28

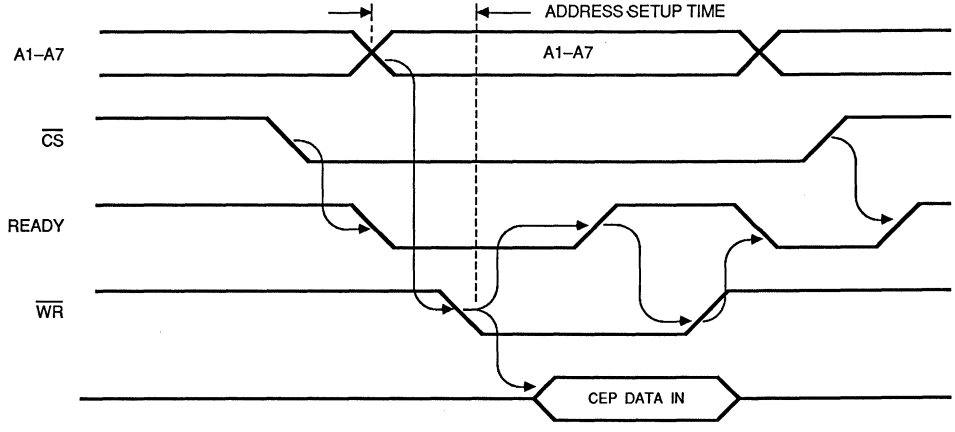


Figure 2-29 CPU Write Timing (CEP Slave Mode) 07666A 2-29

appropriate memory location (CEP Wait State, rising edge). Any number of wait states will be inserted as long as READY is sampled Low during state Tw.

9. The CEP drives the $\overline{WR}$ output High (CEP state T3, rising edge).
10. The CEP disables address/data, $\overline{RD}$, and $\overline{WR}$ lines.
11. The CEP drives the HRQ output Low.
12. The CPU drives the HLDA input Low.

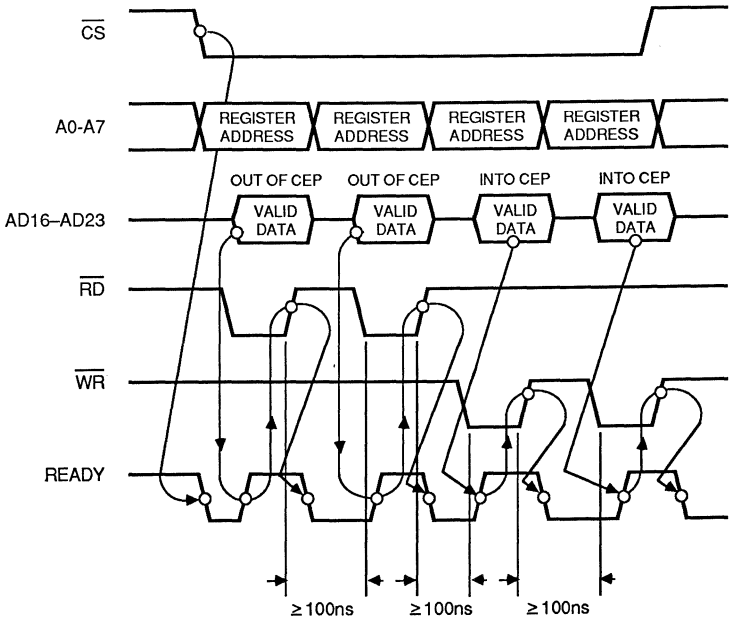
Note: If Step 12 precedes Step 11 (preemption), the CEP will complete the current bus transaction. The HRQ output will then be driven Low for two clock cycles. If additional bus transactions are required by the CEP, the CEP will drive the HRQ output High (Step 1).

2.3.4 Document Bus Operation

Figures 2-33 and 2-34 show an idealized timing diagram of the Document Store bus DMA read and write operations. The procedure by which the CEP executes a Document Bus memory operation is described as follows:

Read Access Operation

1. The CEP enables the address/data, DALE, $\overline{DRD}$, and $\overline{DWR}$ lines and drives DALE High during T_{Float}. TF is indicated by the dotted lines in the diagram.
2. The CEP places a Document Store address (24 bits) on the Document Bus (CEP state T1). The address is valid during the High to Low transition of the DALE output (CEP state T2, falling edge).
3. The CEP drives the $\overline{DRD}$ output Low (CEP state T2, rising edge).
4. The state (High or Low) of the READY input is sampled by T2, rising edge.
5. If the DREADY input is Low, a Wait State is inserted (Tw). The READY input should become High when the Document Store location becomes available (CEP Wait State, rising edge). Any number of wait states will be inserted as long as READY is sampled Low during state Tw.
6. If the DREADY input is High, then DAD_{16} to DAD_{23} must be driven with valid data (from the indicated Document Store location) (CEP state T3, falling edge).
7. The CEP drives the $\overline{DRD}$ output High (CEP state T3, rising edge).
8. The CEP drives DALE High.
9. The CEP disables the address/data, DALE, $\overline{DRD}$, and $\overline{DWR}$ lines (TF dotted lines). The CEP may process any number of consecutive



07666A 2-30

Figure 2-30 CPU Block I/O Transaction Timing (CEP Slave Mode)

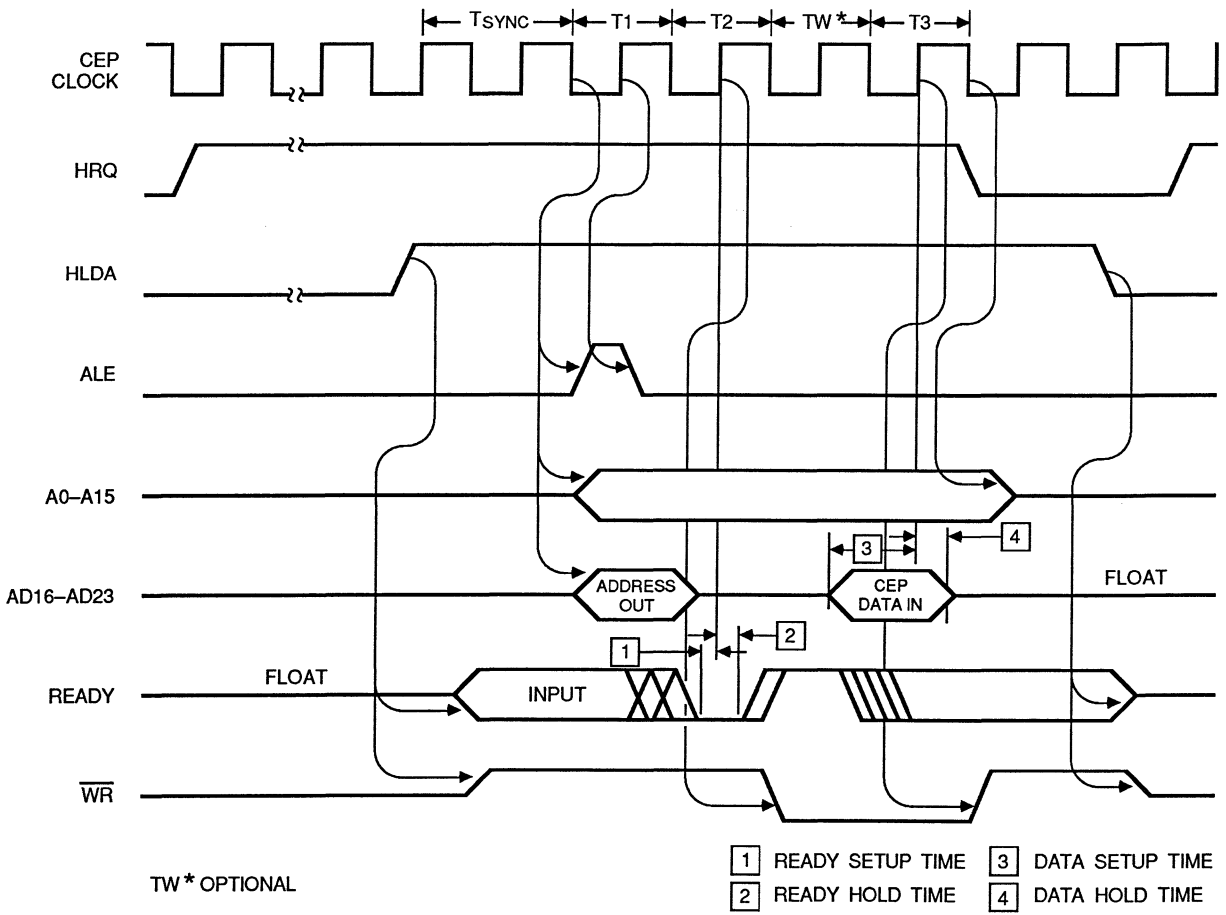


Figure 2-31. System Side DMA Read Operation (CEP data in)

07666A 2-31

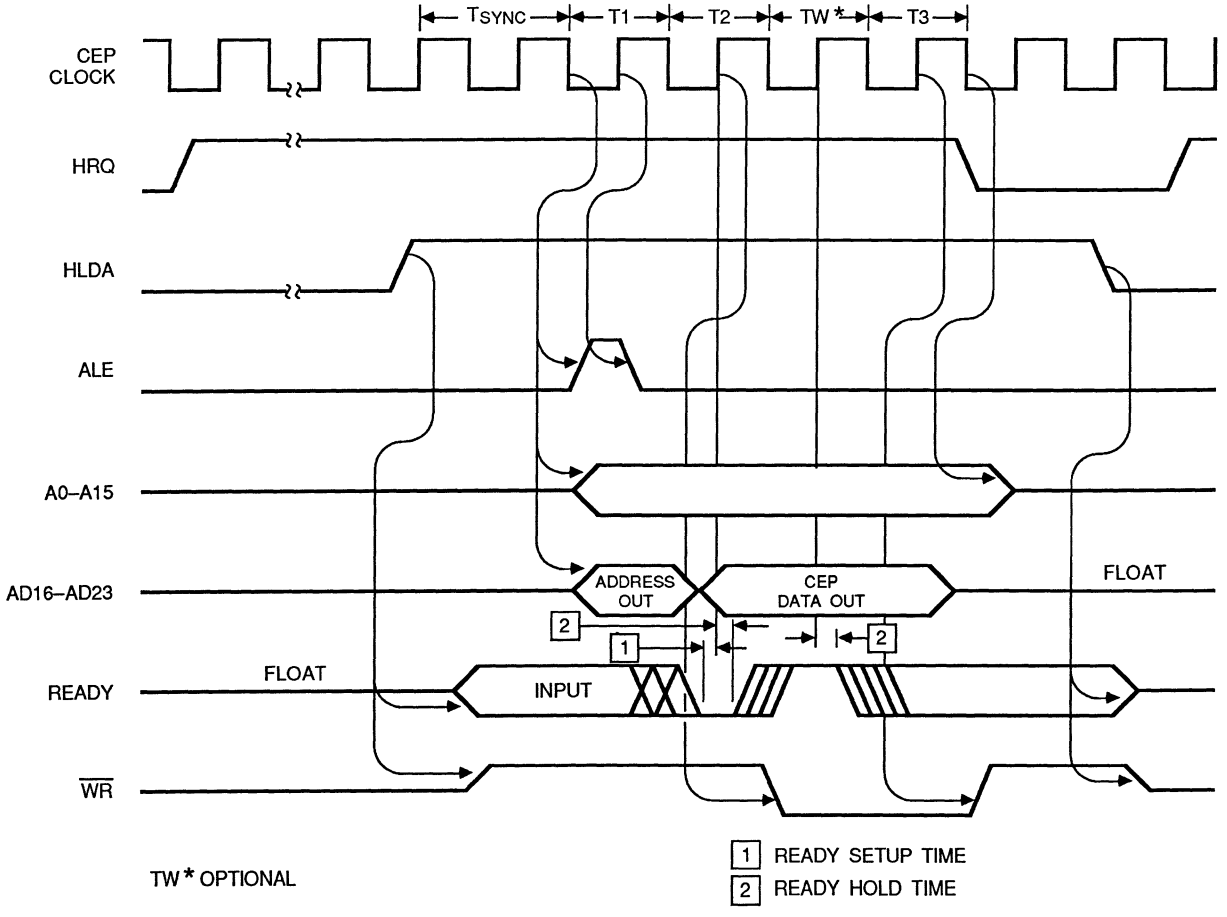


Figure 2-32. System Side DMA Write Operation (CEP data out)

07666A 2-32

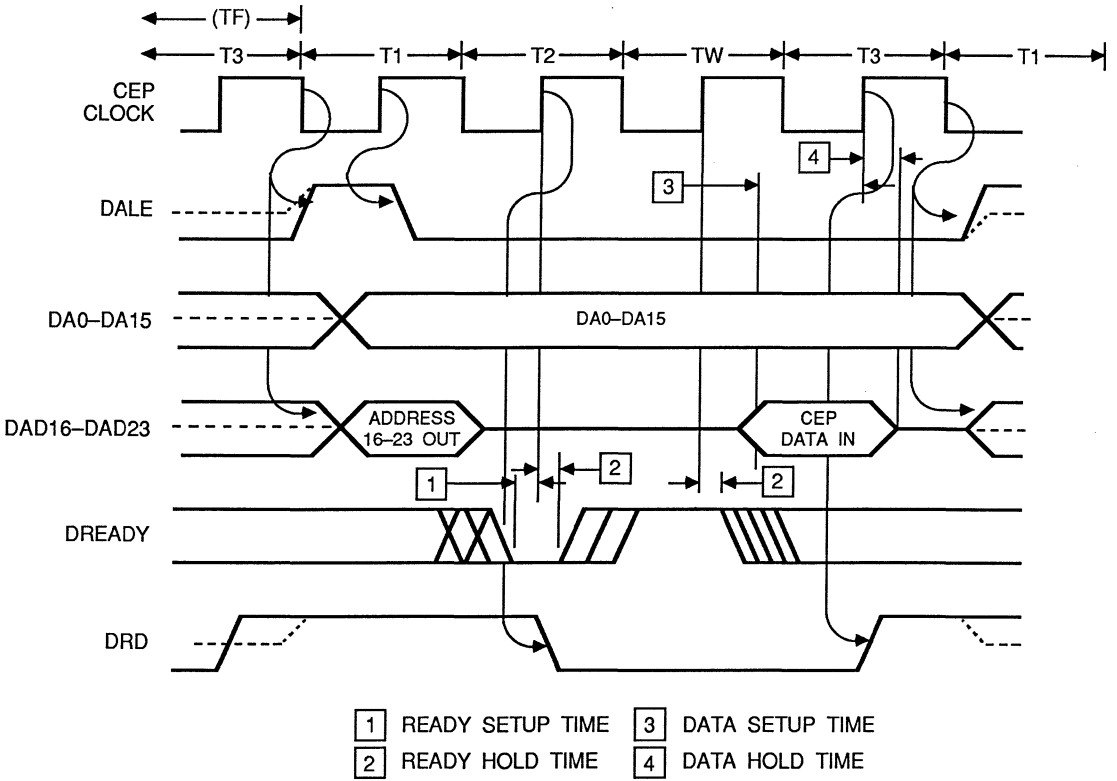
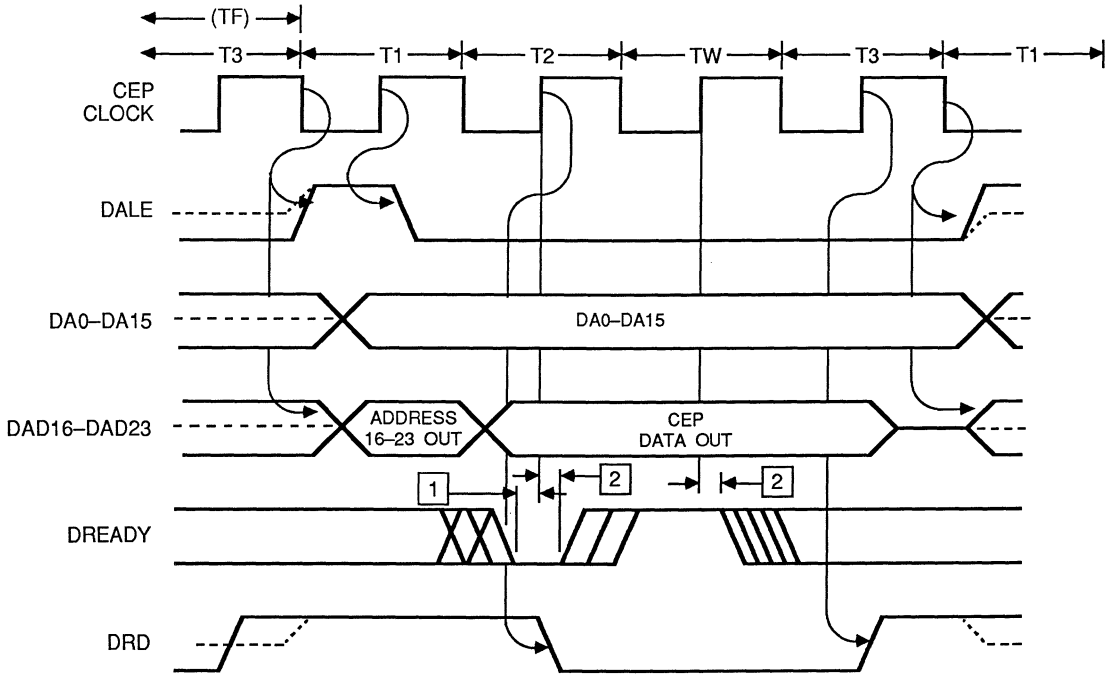


Figure 2-33. Document Store Bus DMA Read Operation

07666A 2-33



TF = FLOAT OR IDLE STATE

--- DASHED LINES SHOW TRI-STATE CONDITION DURING IDLE STATE WHEN Am7970A IS NOT BUS MASTER

Figure 2-34. Document Store Bus DMA Write Operation

07666A 2-34

accesses without disabling the interface by skipping step 9 and going to step 2. This case is indicated by the solid lines in the diagram.

Write Access Operation

1. The CEP enables the address/data, DALE, $\overline{DRD}$, and $\overline{DWR}$ lines and drives DALE High.
2. The CEP places a Document Store address (24 bits) on the Document Bus (CEP state T1, falling edge). The address is valid during the High to Low transition of the DALE output (CEP state T2, falling edge).
3. DAD_{16} to DAD_{23} is driven by the CEP with valid data (T2 falling edge).
4. The CEP drives the $\overline{DWR}$ output Low (CEP state T2, rising edge).
5. The state (High or Low) of the DREADY input is sampled by the T2 rising edge.
6. If the DREADY input is Low, a Wait State is inserted (Tw). The DREADY input should be driven High after data has been loaded into the appropriate Document Store location (CEP Wait State, rising edge). Any number of wait states will be inserted as long as READY is sampled Low during state Tw.
7. The CEP drives the $\overline{DWR}$ output High (CEP state T3, rising edge).
8. The CEP drives DALE High.
9. The CEP disables the address/data, DALE, $\overline{DRD}$, and $\overline{DWR}$ lines (TF dotted lines). The CEP may process any number of consecutive accesses without disabling the interface by skipping step 9 and going to step 2. This case is indicated by the solid lines in the diagram.

Chapter 3
CODING

3.1 CODING CONCEPTS

This chapter presents an overview of the concepts of coding image data in the form of black-and-white documents. Details of the application of these concepts to compressing this data are also given. This is followed by a comprehensive discussion of the modified Huffman one-dimensional and modified READ two-dimensional Coding schemes.

Document image data is defined by the CCITT as picture elements along a scan line. Scan lines move from left to right starting at the top of a document. Successive scan lines are immediately below the preceding line. A picture element (Pixel) is a unit area of white or black color.

Encoding is used to compress image data without the loss of picture detail. The essence of data compression is the elimination of redundancy without the loss of information. Both one-dimensional and two-dimensional coding strategies have been developed. In one-dimensional coding, the idea is to represent run lengths of identical elements by codes that are shorter than the run length. For example, a run length of 55 white pixels is represented by an eight bit code, 01011000. Two-dimensional coding takes advantage of the similarity of the pixels in one line to the pixels in the line above it.

The information content of typical image data has been analyzed by many researchers over many years and an encoding algorithm has been designed to provide the shortest codes for the most frequently occurring sequences of data.

In addition to the CEP's 1-D and 2-D modes, there is a transparent mode, un-compressed data mode, granularity mode and an express mode. These coding methods are also discussed along with a discussion of minimum transmission times which apply to all modes.

The communication environment also needs to be considered. Handshaking is required to let the receiver know the format used in the data compression so that the receiver can expand the data properly. This handshaking consists of a preamble that is sent preceding the document. The preamble specifies the parameters to be used in processing the document. The details of these communications standards can be found in in the CCITT documents T.5, T.30, T.72, and T.73.

3.1.1 Encoding Digital Facsimile

The two primary encoding techniques used to compress and de-compress digital facsimile data are the Modified Huffman (MH) and Modified READ (MR) schemes. There is also a derivative of the MR code standardized for next generation (group 4) devices sometimes called the Modified Modified READ Code (MMR). In the course of discussing these techniques, it is useful to review the basic elements of facsimile technology.

Facsimile systems are based on the concept of scanning (typically on a horizontal line basis). This scanning creates a stream of data representing the lightness or darkness of the information being scanned at any given time. The resulting stream of data is then transmitted to another facsimile system where it is used to drive an image-reproducing device. Generally speaking, the operation of a facsimile device is identical to the raster scan technology used in television and CRT displays.

Scanned images are usually classified as either *photographic* images, in which the original copy is reproduced faithfully with all of its grey scale tonal gradations in tact, or *document* images, in which the original copy is reproduced strictly in black and white (two-tone). The topics covered here refer to document images only, unless otherwise stated as a specific example.

For facsimile systems, the clarity of the final image depends upon the fineness of the original scan. Normally, 100 to 200 scan lines per inch are required to legabily reproduce a page of text and image material. Thus, a typical 8 1/2 x 11 sheet of paper requires somewhere between 1275 to 1700 scan lines. Each scan line in turn consists of at least 1728 picture elements (pixels), resulting in a total of 2 to 3 million bits for an 8 1/2 x 11 sheet of paper.

To transmit this data at 4800 bits per second (bps) without compression requires a minimum of 416 seconds or approximately seven minutes. Compression, however, can reduce this transmission time to well under 1 minute. In fact, the Am7970A CEP can process most typical documents in 1 to 2 seconds using a 5MHz clock (not including the other system processing components).

The Consultative Committee for International Telegraph and Telephone (CCITT) has classified

document facsimile machines into four groups, Group 1, Group 2, Group 3, and Group 4. Group 1 and 2 machines are completely analog and do not use data compression techniques. Therefore, they are not discussed here. Where differences exist between Group 3 and Group 4, they are mentioned.

In Group 3 and Group 4 equipment, data compression techniques are utilized to reduce the amount of redundancy in the image data. As illustrated in Figure 3-1, an original data stream is operated upon according to the selected algorithm to produce a compressed data stream. This compression of the original data stream is referred to as the encoding process, and the resulting compressed data is called the encoded or "coded data". When referring to black and white (two-tone) pixel image data, the original data stream is often called the "raw" or "picture" data.

Reversing the process, the compressed data stream is expanded (decompressed) to reproduce the original data stream. Since this expansion process results in the decoding of the compressed data stream back into its original state, this decoded or expanded data is also referred to as the "raw" or "picture" data.

The amount of data reduction obtained as a result of the compression process can be expressed as the compression ratio. This represents the quantity of compressed or encoded data with respect to the quantity of original data. Clearly, the higher this ratio, the more effective the compression technique.

where:

$$\text{Compression ratio} = \frac{\text{Size of the original data}}{\text{Size of the compressed data}}$$

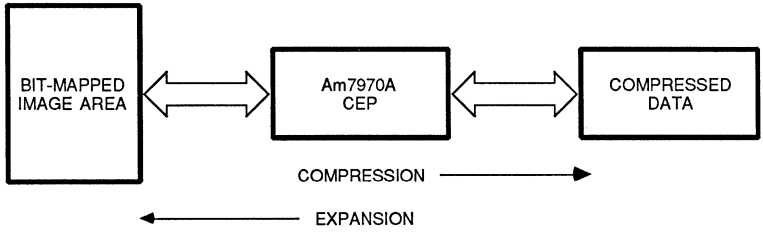
Figure 3-2 shows the basic block diagram of a Group 3 facsimile machine. It utilizes a simple flat-bed scanning system, in which scanning is performed as the original document remains stationary on a flat surface. This scanning is proformed electronically with a charge-coupled device (CCD) image sensor. Shown here, is the widely used thermal recording method for writing the facsimile copy, although more recently, the laser beam print technology is becoming economically feasible for such equipment.

The raw picture data is sent to the data compression section. Here, signals are temporarily stored in a line buffer, which holds from one to five scan lines, after which encoding (compression) takes place.

Since data compression uses a statistical encoding technique resulting in variable length codes, a code buffer for the compressed data is necessary to average or smooth the encoded byte stream into a uniform bit stream for uninterrupted transmission. This buffer is generally the same size as the line buffer in order to accommodate the possibility of negative compression which can occur in the extreme case, when the code assignments are longer than the pixel streams being encoded. To prevent overflow or underflow of this memory, the document feed is controlled in increments of one scan line by a stepper motor.

Facsimile encoding algorithms are statistical in nature. These statistical encoding methods take advantage of the probabilities of occurrence of events so that short codes can be used to represent frequently occurring events while longer codes are used to represent less frequently encountered events. Events can be run lengths, relative distances, or control codes to identify the beginning or end of a document or the end of a line. A summary of parameters for Group 3 and Group 4 equipment is included as Table 3-1.

Statistical encoding can be used to obtain an



07666A 3-1

Figure 3-1 Group 3 and Group 4 Data Compression

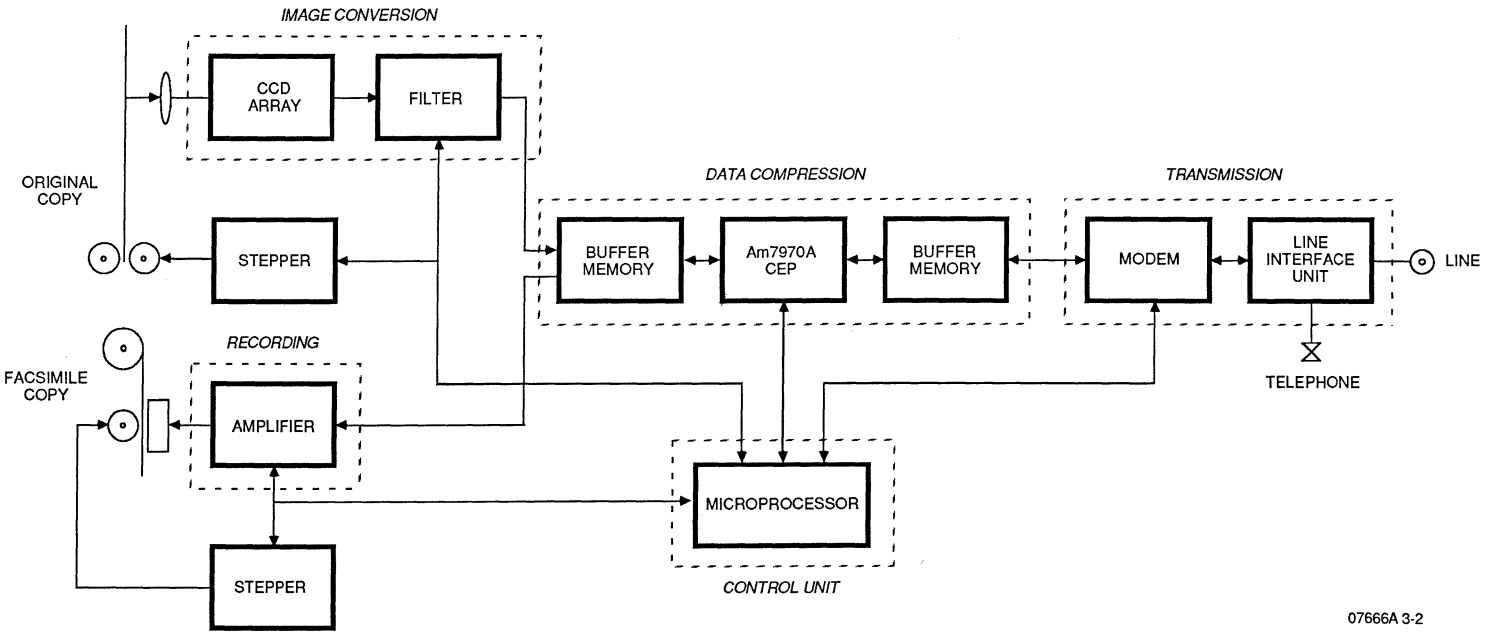


Figure 3-2. Block Diagram of Group 3 Equipment

optimization of the average code length of the encoded data. This is analogous to the manner in which Samuel Morse selected the dot and dash representations of characters most frequently used for telegraphy so that, for example, a single dot was used to represent the letter E, which is the most frequently encountered character in the English language, while longer strings of dots and dashes were used to represent characters that appeared less frequently.

Included in the class of statistical compression techniques is the Huffman coding technique. Although the Huffman technique is not directly used in facsimile equipment, its derivative the Modified Huffman scheme is. However, prior to discussing these statistical encoding techniques in detail, a brief discription of some basic information theory concepts is useful. These concepts should help to provide an understanding of how redundancy can be statistically reduced.

Table 3-1. Summary of Standardized Parameters for Group 3 and Group 4 Equipment

Parameter	Apparatus			
			Group 4***	
	Group 3	Class I	Class 2	Class 3
Apparatus Recommendation	T.4	T.5	T.5	T.5
Network	PTN (PTN.ISDN)****	PDN (PTN.ISDN)****	PDN (PTN.ISDN)****	PDN (PTN.ISDN)
Transmission Time/A4 (min.)	Approx. 1			
Number of pels Along a Scan Line	1728	1728, 2074* 2592*, 3456*	1728, 2074** 2592, 3456*	1728, 2074** 2592, 3456*
Scanning Density	3.85, 7.7* (1p/mm)	200, 240** 300*, 400* (1 p/i)	200, 240** 300, 400* (i p/i)	200, 240** 300, 400* (1 p/i)
Modem	PM (V.27ter). AMPM (V.29)*			
Data Rate (kb/s)	2.4, 4.8 7.2*, 9.6*	2.4, 4.8 9.6, 48	2.4, 4.8 9.6, 48	2.4, 4.8 9.6, 48
Coding Scheme	MH, MR* (T.4)	Modified MR (T.6)	Modified MR (T.6)	Modified MR (T.6)
Control Procedure, Protocol, Recommendation	T.30	T.62, T.70 T.71, T.73	T.62, T.70 T.71, T.72 T.73	T.62, T.70 T.71, T.72 T.73
Remarks			Reception only for teletex and mixed-mode	Transmission and reception for teletex and mixed-mode

*Option
**Required for teletex and mixed-mode reception
***Was recommended end of 1984
****Further study

3.1.2 Information Theory

For equipment capable of transmitting at n distinct levels at u second intervals, the number of different signal combinations in T seconds is $n^{T/u}$. Since the quantity of information is directly related to the length of time of transmission, the logarithm of $n^{T/u}$ expresses the information transmitted in T seconds or (T/u)logn. This is the definition of the information unit, H. For digital systems, i.e. base 2, H becomes:

$$H = \frac{T}{u} \log_2 n \text{ bits.}$$

As an example, for the digital transmission of data over a 20 second period at 1 second intervals, the information content becomes:

$$H = \frac{20}{1} \log_2 2 \text{ bits} = 20 \text{ bits.}$$

From statistics we know that the relative frequency of occurrence of any one combination or event is defined as the probability, P, where:

$$P = \frac{\text{number of times an event occurs}}{\text{total number of possibilities}}.$$

If information with n possible signal levels is to be transmitted, then P = 1/n for signals that are equally likely to occur. Lets consider the case where different events or signal levels do not have equal probabilities of occurrence. Lets assume the digital case where just two levels are to be transmitted, 0 or 1, the first with probability P and the second with probability Q, where P + Q = 1. Then:

$$P = \frac{\text{number of times 0 occurs}}{\text{total number of possibilities}}$$

$$Q = \frac{\text{number of times 1 occurs}}{\text{total number of possibilities}}.$$

For a long message, consisting of many 0s and 1s, the information content is related to $P * \log_2 P + Q * \log_2 Q$, and generally, we can let the probability of each possible signal level or signal be expressed by P_i, where $P_1 + P_2 + \ldots + P_n = 1$. Thus, each interval contains $-\text{Log}_2 P_i$ bits of information. By summing the average information in bits contributed by each symbol appearing $t * P_i$ times over t intervals, we obta

$$H = -t * \sum_{i=1}^{n} P_i \log_2 P_i \text{ bits in t periods.}$$

For the interval T, we then obtain:

$$H = -T/t * \sum_{i=1}^{n} P_i \log_2 P_i \text{ bits in T seconds.}$$

For the most general case, a message with n possible symbols and a probability of occurrence P_i to P_n, the average information per symbol interval of u seconds is:

$$H_{avg} = -\sum_{i=1}^{n} P_i \log_2 P_i \text{ bits/symbol interval.}$$

This is the mathematical definition of entropy used in information theory to calculate the average number of bits required to represent each symbol of a source alphabet.

A simple coin tossing model can be used to illustrate the concept of entropy. The two sides of a coin, heads (H) and tails (T), can be used to define a four symbol alphabet using two coins for each toss. If we assign codes of T = 0 and H = 1, the coin toss results are:

Symbol	Probability	Code
TT	0.25	00
TH	0.25	01
HT	0.25	10
HH	0.25	11

The entropy or average number of bits required to represent each possible outcome or symbol becomes:

$$H_{avg} = -\sum_{i=1}^{4} P_i \log_2 P_i = -4 * 0.25 \log_2 0.25 = 2 \text{ bits.}$$

If, however, the probability of tails occurring is P(T) = 0.75, and the probablilty of heads P(H) = 0.25, then, the outcome of the coin tossing is:

Symbol	Probability	Code
TT	0.5625	00
TH	0.1875	01
HT	0.1875	10
HH	0.0625	11

Although the symbols and codes have remained the same, the outcome probabilites have changed. The entropy of this alphabet is now:

$$H_{avg} = -\sum_{i=1}^{4} P_i \log_2 P_i$$

$$= 0.565 \log_2 0.5625 + 0.1875 \log_2 0.1875 + 0.1875 \log_2 0.1875 + 0.0625 \log_2 0.0625$$
$$= 1.62 \text{ bits/symbol}$$

This says that the average number of bits required to represent a symbol with this probability distribution has been reduced to 1.62 bits from 2. So, by choosing a different coding scheme to represent the four symbols, about 20 percent of redundancy can be removed from the two bits per symbol previously used. This is accomplished by assigning shorter codes to the most frequently occurring symbols and longer codes to the the symbols that do not occur as often. It is the basis for what is called Huffman coding.

3.1.3 Huffman Coding

Huffman coding is a statistical data-compression technique, and is the most familiar variable-length coding scheme. Its purpose is to reduce the average code length required to represent the symbols of an alphabet. This is accomplished by assigning the shortest code word to the most frequently occurring symbol, longer code words to less frequently occurring code words and so on until the longest code word is assigned to the least frequently occurring symbol. This alphabet can be of any type.

The Huffman code results in the shortest average code length of all statistical encoding methods. One of the reasons for this is that, Huffman codes are designed to be self-delimiting so that no shorter code group is duplicated as the beginning of a longer group. Thus, no symbol can be mistaken for another. This removes the need for delimitors such as "spaces" between codes such as found in the Morse Code. It can be considered one of Huffman's greatest contributions to data compression.

The Huffman code is developed by using a tree

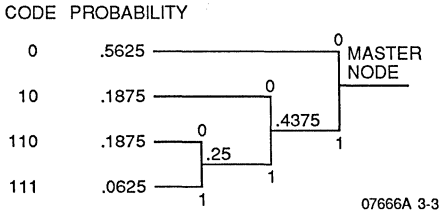


Figure 3-3 Simplified Huffman Coding Tree

structure as shown in Figure 3-3. The symbols are first listed in descending order of probability. Then starting with the two lowest entries (HH and HT), the probabilites are combined into a node with a joint probability of 0.25, in this example. This node is then joined with the next lowest probability from the list, in this case forming a node with a probability of 0.4375 and finally with the topmost entry forming the master node with a probability of unity. By assigning 0s and 1s to every branch, we can derive the Huffman code for each symbol. This is obtained by noting the 1s and 0s encountered in tracing from the master node to each symbol.

To calculate the average length of the coded word, we multiply the Huffman code lengths by their probability of occurrence:

$$L = P_i L_i$$
$$= (1 * 0.5625) + (2 * 0.1875) + (3 * 0.1875) + (3 * 0.0625)$$
$$= 1.63 \text{ bits/symbol}$$

Notice how closely the Huffman coding approaches its theoretical entropy. The coding efficiency, E, can be defined as the ratio of the entropy to the average word length, L, and in this example:

$$E = H/L = 1.62/1.63 = .99 \text{ bits/symbol or } 99\%$$

As with many things, the Huffman Coding process becomes much more complex when applied to real alphabets with many substitutions, but the preceeding example should provide a basic understanding of the concept involved.

The Huffman code, like other statistical coding techniques, relies on an apriori knowledge of the statistical distribution of the message. Therefore, Huffman coding ceases to be optimal when the source statistics are fluctuating as is often the case in real applications.

3.1.4 Modified Huffman Coding

Although the Huffman coding scheme is excellent in theory, it has not found widespread usage because variable-length code words are difficult to implement. The Huffman scheme requires a lot of memory to store the code alphabet and is also very difficult to decode. So, in practice, a number of modifications have been necessary to adapt Huffman coding to document image encoding.

When applying Huffman coding to facsimile transmission or document processing applications, each scan line of an image can be viewed as consisting of a series of black or white "runs", each run being a succession of similar picture elements (pixels). If the color of the first run is known, then the color of all successive runs will be known because black and white runs must alternate. The probability of occurrence of each run length of a given color can be calculated and short code words can be used to represent run lengths that have a high frequency of occurrence while longer code words can be used to represent run lengths that have a lower probability of occurrence.

For instance, the average white run length can be expressed as:

$$L(W) = \sum_{i=1}^{n} l_i(W) P_i(W)$$

where $P_i(W)$ is the probability of a white run of $l_i(W)$ pixels, and n is the total number of white runs on the document. The entropy or average information content for a white run is expressed as:

$$H(W) = -\sum_{i=1}^{n} P_i(W) \log_2 P_i(W)$$

The equations for black run lengths L(B) and entropy H(B) are expressed in a simular fashion. Since by definition white and black runs alternate, the number of black runs equals the number of white runs (N). Therefore, the overall average run length is L(W)/2 + L(B)/2, and the average entropy per run is H(W)/2 + H(B)/2. The entropy per pixel of a run length is expressed as:

$$H_{pixel} = \frac{H(W) + H(B)}{L(W) + L(B)}$$

which imposes a lower bound on the theoretical number of bits required per pixel and when inverted expresses the maximum limit of the compression ratio:

$$CR = 1/H_{pixel}$$

One of the problems with using Huffman coding for facsimile is that the statistics for the run-length probabilities associated with line scans change on a line-to-line and document-to-document basis. Thus, an optimum or near optimum code for a particular line or document may be far from optimum for a different line or document. A second major problem is the fact that the creation of the Huffman code on a real-time basis requires alot of processing power, normally in excess of the capabilities of facsimile machines.

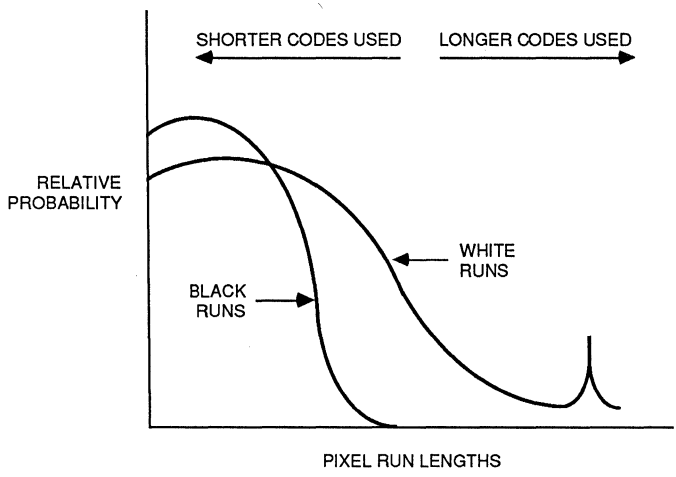
To reduce some of the real-time processing requirements of using the Huffman code, a table look-up approach is needed. But, since CCITT standards require a minimum of 1728 pixels per line, the use of a table look-up technique requires the storage of 1728 variable length locations, each containing a binary code word corresponding to a particular run length, an un-economic approach.

These implementation problems resulted in the development and standardization of the Modified Huffman coding scheme which is more suitable to the hardware cost constraints of the competitive facsimile marketplace. This is one of the coding schemes used in the Am7970A CEP.

In the Modified Huffman coding scheme some changes were made which, while only rarely permitting the average symbol length to approach entropy, do permit significant compression while minimizing hardware and processing requirements. Here, the probability of occurrence of different pixel run lengths were calculated based upon statistics obtained from the analysis of a group of 8 documents recommended by the CCITT as being typical (see Figures G-1 through G-8 in Appendix G). Figure 3-4 shows the relative probablities of occurrence of pixel run lengths based on these documents. Table 3-2 shows typical compression ratios obtained using these documents.

To reduce table look-up storage requirements, the Huffman code set was split up into two much shorter code tables by the creation of a base 64 representation of each run length in one table and the remainder in the other.

Based upon the run-length probabilities of the 8 CCITT documents, code tables were developed for run lengths ranging from 1 to 63 pixels. Since the frequency of occurrence of white runs differs from the frequency of occurrence of black runs, a separate table was developed for each. A part of this dual table set is listed in Table 3-3 for run



07666A 3-1

Figure 3-4 Relative Probabilities of Various Pixel Run Lengths

lengths from 0 to 63 pixels. The codes in this table represent the least significant digit (LSD) of the code word and are referred to as the Terminating Code.

In order to permit the encoding of runs in excess of 63 pixels, another pair of code tables must be employed to handle runs ranging from 64 pixels to the maximum line scan length. A sample of these codes are listed in Table 3-4. These represent the most significant digits of the code word and are known as the Make-up codes. The complete Modified Huffman code tables are specified in CCITT document T.4 in Appendix G.

Table 3-2. Typical Compression Ratios Using Eight Standard CCITT Test Documents

(Compression Ratio = Scanned Data/Compressed Data)

Pels per Inch	200 x 100	200 x 200	200 x 200	400 x 400
Test Doc	1D	2D(K=4)	2D(K=INF)	2D(K=INF)
1	15.2	20.1	27.7	37.7
2	15.1	24.2	40.5	48.2
3	8.7	13.3	18.6	26.9
4	5.3	6.7	7.7	12.7
5	8.5	12.4	16.5	20.5
6	10.2	17.7	29.0	39.9
7	4.8	6.1	7.1	13.5
8	7.9	13.0	19.9	26.8
Ave.	9.5	14.2	22.8	32.1

Table 3-3. Terminating Codes

White Runs		Black Runs	
Length	Code Word	Length	Code Word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
.	.	.	.
.	.	.	.
.	.	.	.
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111

Table 3-4. Make-up Codes

White Runs		Black Runs	
Length	Code Word	Length	Code Word
64	11011	64	0000001111
128	10010	128	000011001000
192	010111	192	000011001001
256	0110111	256	000001011011
.	.	.	.
.	.	.	.
.	.	.	.
1600	010011010	1600	0000001011011
1664	011000	664	0000001100100
1728	010011011	1728	0000001100101
EOL	000000000001	EOL	000000000001

When a run of 63 pels or less is encountered, the appropriate type of LSD code set is accessed to obtain a terminating code word. To encode a run of 64 pels or more, two code words must be used. First, the Make-up code word is obtained from the MSD code table such that N * 64 does not exceed the run length. Next, the difference between the run length and N * 64 is obtained and the Terminating code word is accessed from the appropriate LSD code table.

To employ the Modified Huffman coding scheme successfully, some rules have been developed and must be followed to alleviate a number of deficiencies inherent in statistical encoding techniques. In such techniques, code words do not contain any inherent positional information which is necessary for synchronization. This can be compensated for by making sure that all runs alternate between black and white, and that each line begins with a white run, even if that run length is zero. To denote the beginning and end of each line, a unique end-of-line (EOL) code is used, and a number of 0s may be added to a line to meet minimum timing requirements prior to transmitting the EOL.

The end result of the incorporation of these rules permits a line format to be defined as shown in Figures 3-5, 3-6, and 3-7. Figure 3-5 shows Group 3 format of compressed code with no byte boundary. Group 3 format of compressed code with byte boundary and Auto EOL is shown in Figure 3-6. Group 3 one-dimensional coding with byte boundary, Auto-EOL, and Fill is shown in Figure 3-7.

3.1.5 The CEP's One-Dimensional Mode

The one-dimensional mode of the CEP applies only to Group 3 equipment. It employs the Modified Huffman coding scheme as outlined in the previous section. It is simply the replacement of each run of one color (either white or black) with the code that represents the length of the run. Each line starts with a white run even if it is of zero length. White runs and black runs alternate. The coding used to represent the run lengths is shown in Table 3-3 and Table 3-4. It is discussed later in this chapter and in the CCITT document T.4 in Appendix F.

Group 3 standards of the CCITT specify that the start of a document is to be identified by an EOL code (000000000001). The end of each line must be marked with an EOL and the end of the document must be marked with an Return-to-Control Code, RTC (six EOLs). Figure 3-5 shows the format for Group 3 One-Dimensional coding with no byte boundary specified. Fill is automatically added to each line as needed to meet the minimum transmission time requirements. Pad bits are also added as needed to terminate the coded document on a byte boundary. Each document must begin and end on a byte boundary.

The format for Group 3 One-Dimensional coding with Byte Boundary specified in the DFC field in the Compressor Parameter Register (CPR) is shown in Figure 3-6. Pads are automatically added as needed at the end of each data line to terminate each line at a byte boundary. Auto-EOL is implemented by setting the EOL field in the CPR to 0. EOLs are required at the end of each line in Group 3 coding.

For the last line of a page, the DFC field is set to 01 to specify the RTC code. By specifying the source buffer as one line less than the full page, a source buffer overflow interrupt occurs before the last line so that the RTC can be changed. The Operation Control (OC) field in the CMCR is set to 01 to specify a single line operation for the last line. The Mode Control (MC) field in the CMCR is set to 01 to specify One-Dimensional Mode. Fill bits are automatically added when the data fields are too short.

To maintain color synchronization at the receiver, each data line begins with a white run length code word. If the line actually begins with a run of black, a white run length of zero is specified as the first code.

3.1.6 Modified Read Coding

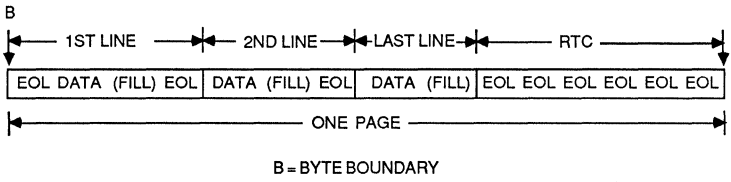
In the previous section we discussed the Modified Huffman coding scheme which reduces redundancies in the scanning direction by encoding pixel runs. Thus, it is referred to as a one-

dimensional encoding scheme.

Many images, particularly text and line drawings, also exhibit a strong vertical correlation from scan line to scan line (due to edge continuity). In fact, it has been found that approximately 50% of all the transitions from black to white or vise versa are directly underneath a transition on the line above it. About 25% differ by only one pixel. Therefore, approximately 75% of all documents can be defined by a relationship which is plus or minus one pixel from the line above it. This is the underlying basis for the two-dimensional Modified READ (MR) code illustrated in the lower portion of Figure 3-8.

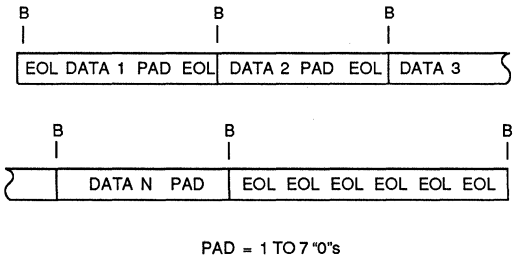
By definition then, the MR scheme must store a history or reference line since it always refers back to the image of the preceding scan line while encoding a next scan line.

MR encoding is separated into three basic modes: Horizontal, Vertical, and Pass. These modes are defined in detail later in this chapter and in CCITT



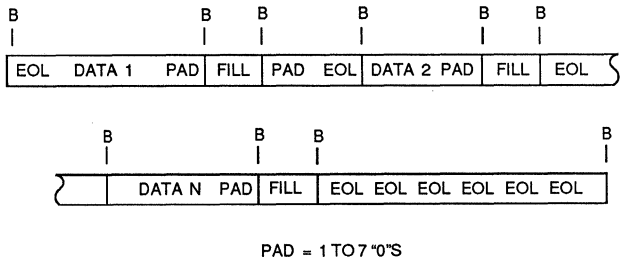
07666A 3-5

Figure 3-5 Group 3 Format of Compressed Code



07666A 3-6

Figure 3-6 Group 3 Format of Compressed Code with Byte Boundary and Auto EOL



07666A 3-7

Figure 3-7 Group 3 Format of Compressed Code with Byte Boundary, Auto EOL and Fill

recommendations T.4 and T.6 included as Appendix F. Basically, MR coding sets up a group of five delimiters along the current and reference scan lines starting at the beginning of the line. The relationship between these delimiters then determines which of the three basic modes will be used to encode the pixel information bounded by these delimiters.

If the trailing delimiter for the reference line is to the left of the leading delimiter of the current line, then the pixel streams are not adjacent and a Pass code is inserted followed by a readjustment of the delimiters to the next pixel stream. If the relative distance between the leading edges of the reference line and the current line are within three pixels in either direction, the code will be selected from the vertical mode tables. And finally, if the conditions for either pass mode or vertical mode are not met, the positions between the current line delimiters are coded one-dimensionally using the Modified Huffman code for that run length.

Figure 3-8 shows a comparison of the encoding operations for a sequence of black and white runs of various pixel sizes using the Modified Huffman code and using the Modified Read code. In the upper portion of this illustration, the relationship between a series of original video data and its representation in the Modified Huffman code is tabulated. In the lower portion, the same video data is represented in the Modified READ code.

3.1.7 The CEP's Two-Dimensional Mode

The CEP performs the modified Relative Element Address Designate (READ) method for two-dimensional coding. Using this method, the

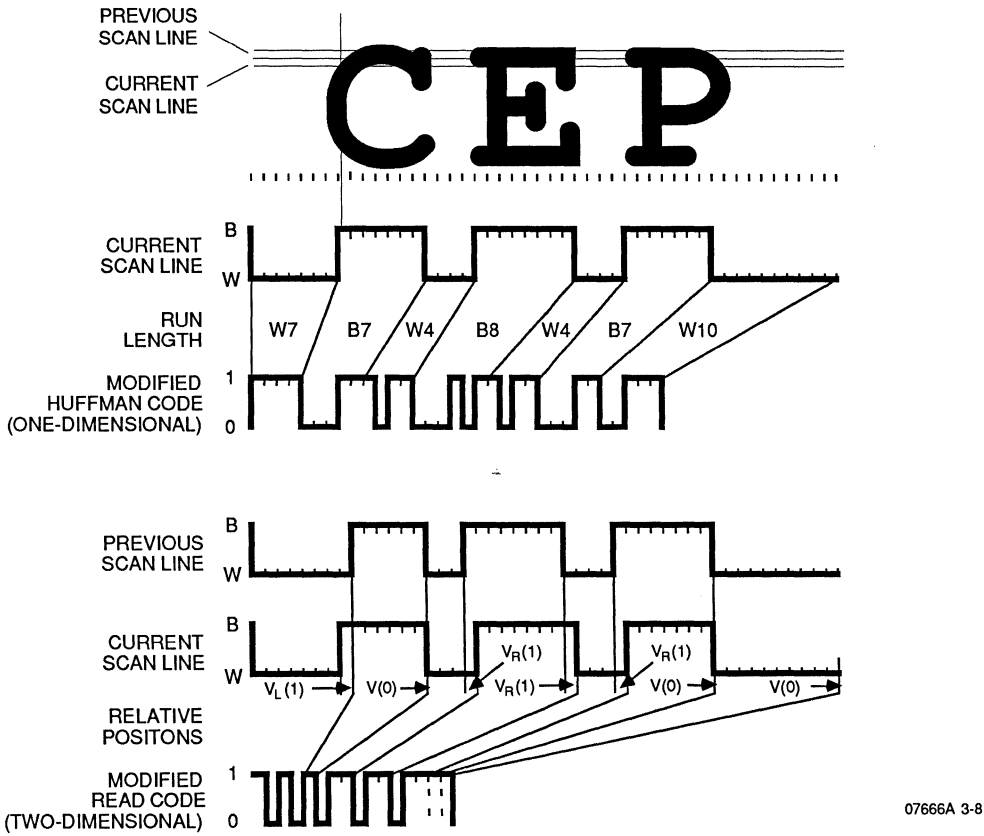


Figure 3-8 Comparison of Run-length and Relative Encoding

position of each changing pixel on the current or coding line is encoded with respect to the position of a corresponding reference pixel situated on the line immediately above it. The line above the current coding line is called the reference line. After a line has been coded, it becomes the reference line for the next coding line.

The first line of the a page is encoded using the Modified Huffman coding and then the following K-1 lines are encoded using the Modified READ techniques, where K is an error immunity factor which requires that 1D lines be inserted at K intervals.

A prefix EOL is inserted preceeding the first element of the first line (source attribute) during compression operations and is assumed to be found there during expansion for Group 3 equipment.

Figures 3-5, 3-6, and 3-7 show various formats for Group 3 two-dimensional compression. To specify two-dimensional Group 3 coding, set the MC field of the CMCR/EMCR to 10. Set the SA field in the CPR to 1 to insert a starting EOL. Also set the K field in the CPR to specify the number of lines of two-dimensional code to be coded for each line of one-dimensional code. Set the DFC field to 00 to process on byte boundaries and reset it to 01 for the last line to specify an RTC code.

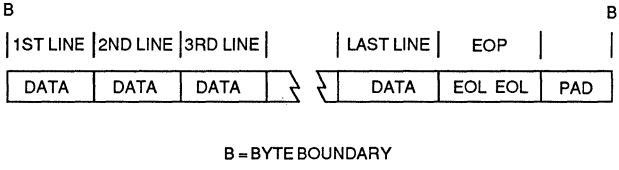
Figure 3-9 shows a format for Group 4 two-dimensional compression. Group 4 encoding differs slightly from Group 3 in that it contains only 2D coded lines (K = infinity). Additionally, the first line (source attribute) is defined as being an imaginary all white line above the top of the page, and there are no EOLs contained on any line. This coding scheme is sometimes referred to as the modified modified READ coding or MMR. Set the MC field in the CMCR/EMCR to 10. In the CPR, set the K field to zero to specify K = infinity and set the DFC field to 10 to specify no byte boundaries. Reset it to 11 for the last line to specify the EOP required in Group 4. Set the SA field to 0 to omit the EOL at the start of the document. Set the EOL field to 1 to omit EOLs at the end of each line.

Three different possibilities are defined in 2D coding: pass mode, vertical mode, and horizontal mode. To explain these modes, it is helpful to define five changing pixel delimiters. A changing pixel is a pixel whose color (black or white) is different from the previous pixel on the same line. Refer to Figure 3-10. The five changing pixels are defined as:

A_0: The reference or starting pixel on the coding line. At the start of the coding of a line, A_0 is set on an imaginary white pixel situated to the left of the first actual pixel on the coding line. During the coding, A_0 is moved to the right as various changing pixels are coded.

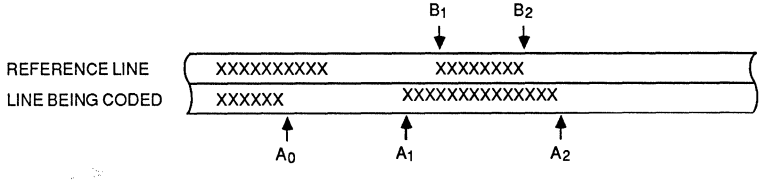
A_1: The next changing pixel to the right of A_0 on the line and the next to be coded. Its color is opposite to that of A_0.

A_2: The next changing pixel to the right of A_1 on



07666A 3-9

Figure 3-9 Group 4 Format of Compressed Code



07666A 3-10

Figure 3-10 Changing Picture Elements

the same coding line.

B_1: The first changing pixel on the reference line to the right of A_0 and having the same color as A_1.

B_2: The next changing pixel to the right of B_1 on the reference line.

Using this coding method, one of the three modes is chosen to encode the position of each changing pixel along the coding line. The coding modes are defined below and illustrated by Figures 3-11 through 3-13. The code assignments are shown in Table 3-5. The flowchart is shown in the CCITT document T.4 in Appendix F.

Pass Mode

The pass mode is used when A_1 is to the right of B_2 as seen in Figure 3-11. This is coded using the word 0001. Then A_0 is moved to the right to the B_2 position. B_1 moves to the right to the first changing pixel of opposite color to A_0 and to the right of A_0. The new B_2 is at the next changing pixel to the right of B_1. A_1 and A_2 are always the next two changing pixel to the right of A_0. The purpose of the pass mode is to identify and encode pixel runs on the coding line which are not adjacent to the corresponding runs on the reference line.

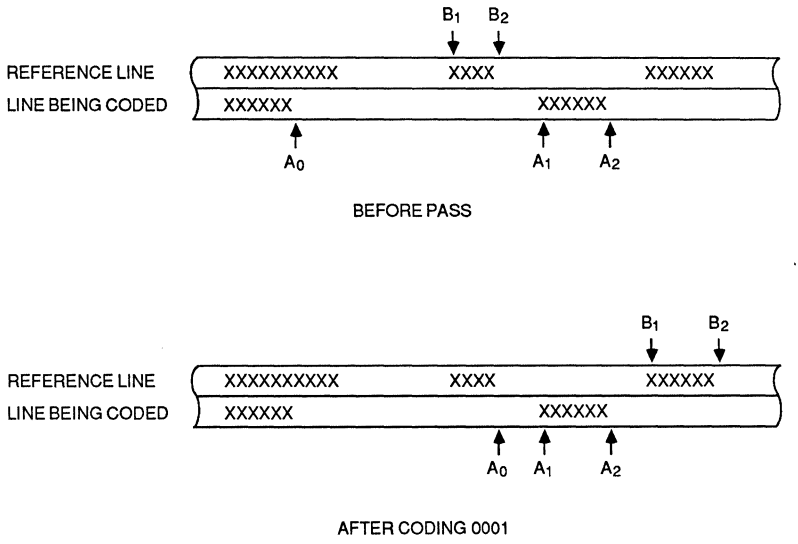
Vertical Mode

If a changing pixel, A_1, is within three pixels horizontally from the corresponding changing pixel, B_1, in the line above, the vertical mode of coding is used. The vertical mode is specified by a short code which indicates the relative distance (0 to plus or minus 3) between A_1 and B_1. After A_1 is coded, A_0 is moved to the A_1 position, A_1 takes the place of A_2, and B_1 takes the place of B_2. A_2 and B_2 each move to the next changing element to the right on the line they are on. Refer to Table 3-5 and Figure 3-12.

If the suggested uncompressed mode is used on a line designated to be one-dimensionally coded, the coder must not switch into the uncompressed mode following any code word ending in the sequence 000. Any code word ending in 000 followed by a switching code 000000001 will be mistaken for an end-of line code.

Horizontal Mode

If the changing pixel, A_1, is more than three units horizontally from the changing pixel, B_1, in the line above, the horizontal mode is used. The code 001 is used to specify the horizontal mode followed by the appropriate Modified Huffman coding to specify the run length (A_0 to A_1). This is followed by another MH code to specify the run length A_1 to A_2. Refer to Figure 3-13. In the



07666A 3-8

Figure 3-11 Pass Mode

horizontal mode, two run lengths are coded and then A_0 is moved to the A_2 location.

3.1.8 Express Mode

The Express Mode is a compression concept in which a specified number of lines are skipped for each line that is coded during compression. It should be noted that although this mode is supported by the CEP, it is not a standard mode as defined by the CCITT.

The resulting expanded picture may or may not have a perceivable loss of quality depending on the level of detail in the picture. However, the scan lines in a typical document are quite close to each other (100 or more lines per inch) and many documents do not require this fine a detail. Line wraparound may not be specified when in the Express Mode. If Express Mode is used with 2-D coding, the current coded line becomes the reference line for the next line to be coded which is n scan lines below it.

Granularity

Granularity is also a non-standard operating mode. It is the expansion counterpart of Express Mode, i.e. the specified number of lines are duplicated during expansion operations. Thus, this mode may be useful in conjuction with Express Mode in certain applications.

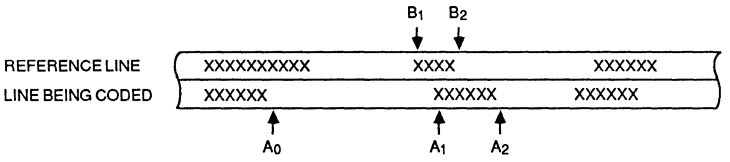
3.1.9 Transparent Mode

The transparent mode is provided to permit moving documents from one memory to another through the CEP via DMA without compression or expansion. However, unlike conventional DMA, the data being transferred is affected by EOL insertion, margin registers, wraparound, time fill, and the express register. Figure 3-14 shows the result of transferring data through the CEP in transparent mode when Auto-EOLs, byte

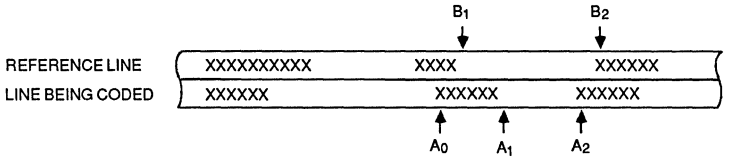
Table 3-5. Two-Dimensional Code Table

Mode	Elements to be Coded	Notation	Code Word
Pass	B_1, B_2	P	0001
Horizontal	A_0A_1, A_1A_2	H	001 + M(A_0A_1) + M(A_1A_2) (Note 1)
Vertical	A_1 under B_1 $A_1B_1 = 0$	V(0)	1
	A_1 to the Right of B_1 $A_1B_1 = 1$ $A_1B_1 = 2$ $A_1B_1 = 3$	$V_R(1)$ $V_R(2)$ $V_R(3)$	011 000011 0000011
	A_1 to the Left of B_1 $A_1B_1 = 1$ $A_1B_1 = 2$ $A_1B_1 = 3$	$V_L(1)$ $V_L(2)$ $V_L(3)$	010 000010 0000010
Extension	Transfer from 1-D line to uncompressed mode		000000001111
	Transfer from 2-D line to uncompressed mode		00000001111
	Other 2-D extensions		00000001XXX
	Other 1-D extensions		000000001XXX (Note 2)

Note 1: Code M() of the horizontal mode represents the code words in Tables 3-2 and 3-3.
Note 2: It is suggested that the uncompressed mode be recognized as an optional extension of the two-dimensional coding scheme for Group 3 apparatus. The bit assignment for the xxx bits is 111 for the uncompressed mode of operation. Further study is needed to define other unspecified xxx bit assignments.



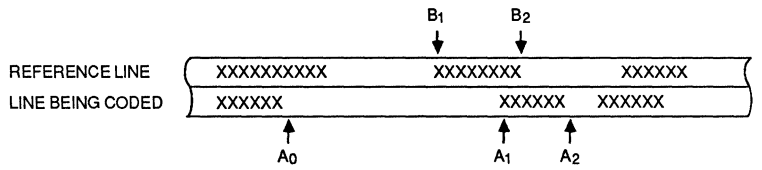
BEFORE CODING A_0–B_1



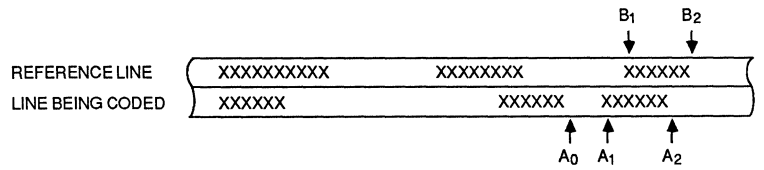
AFTER CODING A_0–B_1

07666A 3-12

Figure 3-12 Vertical Mode



BEFORE CODING A_0–A_1 AND A_1–A_2



AFTER CODING A_0–A_1 AND A_1–A_2

07666A 3-13

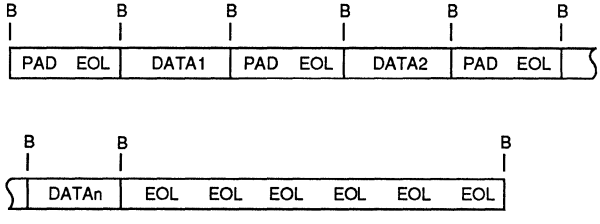
Figure 3-13 Horizontal Mode

boundaries, and RTC are set in the CPR. Transparent mode, although supported by the CEP, is not a standard CCITT mode.

3.1.10 Uncompresed Mode

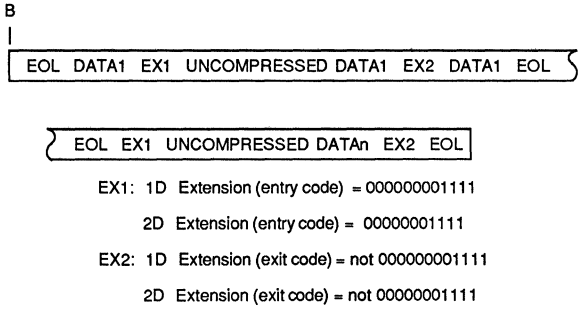
Uncompressed data may be inserted into a compressed data stream at any time by entering an extension code shown in Table 3-6 into the data stream. It should be noted however that this operation is up to the system CPU and is not performed by the CEP (compressor). To return to compressed data format, an exit code is entered as shown in Table 3-6. The exit code includes a tag bit to specify the color of the next run. Figure 3-15 is an example of the coding format.

The purpose of entering the uncompressed mode is that the scan line is not compressing (negative compression). This is a phenominon which can occur when trying to encode highly random data or encoded data such as grey scale images. The CEP will interrupt the host processor in the event that it is unable to compress a line, and it is the responsability of the host to take whatever action is appropriate, including entering the uncompressed mode. However, the expander section of the CEP will detect and correctly expand the uncompressed mode.



07666A 3-10

Figure 3-14 Uncompressed Data Transfer in Transparent Mode



07666A 3-10

Figure 3-15 Uncompressed Data Format

Table 3-6. Uncompressed Mode Code Words

Description	Uncompressed Data	Code Word
Uncompressed Mode	1	1
	01	01
	001	001
	0001	0001
	00001	00001
	00000	000001
Exit from Uncompressed Mode		0000001T
	0	00000001T
	00	000000001T
	000	0000000001T
	0000	00000000001T

T denotes a tag bit which tells the color of the next run (black = 1, white = 0).

The Am7970A Expander detects CCITT recommended extension codes (including exit codes). The CEP's response to detected extension codes is as follows:

1. If the three least significant bits of the detected extension code are *not* all "1s," the Extension Code Detected (ECD) bit in the Master Status Register is set to "1," the Extension (EXT) bits in the Master Status Register are loaded with the three least significant bits of the detected extension code and the Expander immediately terminates.

2. If the three least significant bits of the detected extension code are all "1s," all subsequent data is treated as uncompressed data until a CCITT recommended exit code is detected. After a CCITT recommended exit code has been detected, the CEP resumes its normal 2D Expander Mode of operations.

"Uncompressed data" is passed from the Source Buffer to the Destination Buffer without being expanded. "Uncompressed data" that has been written into the Destination Buffer differs from the Source Buffer data in two ways:

1. Extension code and Exit code will have been removed by the Expander.

2. Each time that the pattern 000001 occurs within the "uncompressed" data, it will be replaced by the pattern 000000.

3.1.11 Transmission Time Constraints

To allow an additional level of error checking, various minimum times are specified for the transmission of each line. If the compressed data line is too short to meet these minimum times, fill bits must be added as needed to increase the transmission time to the minimum. Fill consists of a sequence of zeros. The identification of the minimum time selected is made in the pre-message (phase B) portion of the transmission.

The total coded scan line is defined as the sum of data bits plus any required fill or pad bits plus the EOL bits. Pad bits are zeros added to terminate the line on a byte boundary. For 2-D coding, the total coded scan line is defined as all of the above plus a tag bit following the EOL. The tag bit identifies the coding of the next line as either one-dimensional or two-dimensional coding.

Two alternatives for minimum transmission times are provided. In Type 1 equipment, the minimum transmission time of the total coded scan line is the same for both the standard resolution and for the optional higher resolution.

Additional information regarding minimum transmission times can be found in the CCITT documents in Appendix F.

Chapter 4
PROGRAMMING

4.1 REGISTER SETUP ROUTINES

Compression/Expansion Processor (CEP) Register Assignment Program (Written by Deborah Strickland, November 1984)

4.1.1 Program Listing

A listing of the routines that may be used to setup the various registers is given on the following pages. The routines are:

MAIN PROGRAM	COMP_DEST
INITIALIZE	EXPAND_DEST
MIN_TIME	EXPAND_SOURCE
PAPER_SIZE	WRITE_CEP
MARGIN_SET	BOTH_REG
GPARAM	EXPAND_ONLY
KPARAM	COMP_ONLY
DUMP_REG	READ_CEP
COMP_SOURCE	

```
/************************ PROGRAM DESCRIPTION ******************************/
/*                                                                         */
/* PURPOSE :  Provide functions that will enable the user of the Am7970 to */
/*            load all the registers (compression only, expansion only, or */
/*            both) with the correct set-up configurationn.                */
/*                                                                         */
/*     Note - Not all the functions are available at the user's discretion.*/
/*            Some are called from other functions which utilize the same  */
/*            information and do not need to be called by the user.        */
/*                                                                         */
/*            ** It is suggested that the initialize function be called    */
/*               before any of the others, as it sets up the operating     */
/*               mode and document resolution.                             */
/*                                                                         */
/* Register Address Definitions -                                          */
/*                                                                         */
/*            Each CEP register has a distinct 8-bit port address.  The    */
/*            address for the compression and expansion register pairs (most */
/*            of the registers have corresponding registers in the expansion */
/*            side) differ only by the most significant bit of the address.*/
/*            Since there is a distinct pattern in the register addresses, */
/*            this simplifies the transfer of information to and from the  */
/*            CEP and its driving system.  An extra byte is affixed to the */
/*            port address to allow the program to distinguish between one, */
/*            two, and three byte registers (this is necessary when sending */
/*            out the data).  Depending on the status of the two most signi- */
/*            ficant bits (bit 15 and 16), the program configures the data */
/*            and the port address as necessary.  For example :            */
/*                                                                         */
/*          --Register Name----Compression Address----Expansion Address--  */
/*                                                                         */
/*              C/E DCAR           00101010              10101010          */
/*                                (low byte)            (low byte)         */
/*                                 11000000              11000000          */
/*                                (high byte)           (high byte)        */
/*                                                                         */
/*            If two msb of the high byte = 11 (three byte register)       */
/*            If two msb of the high byte = 10 (two byte register)         */
/*            If two msb of the high byte = 01 (one byte register)         */
/*                                                                         */
/*            This added byte is only for testing and is masked off before */
/*            data is transferred.                                         */
/***************************************************************************/
```

```
/********************** REGISTER ADDRESS ASSIGNMENTS ************************/

#define SWCR 0xC004          /* Source Working Count Register */
#define SCAR 0xC00A          /* Source Current Address Holding Register */
#define SCHR 0xC014          /* Source Count Holding Register */
#define DWCR 0xC024          /* Destination Working Count Register */
#define DCAR 0xC02A          /* Destination Current Address Register */
#define DCHR 0xC034          /* Destination Count Holding Register */
#define SAHR 0xC03A          /* Source Address Holding Register */
#define DAHR 0xC04A          /* Destination Address Holding Register */
#define SLSR 0xC05A          /* Source Line Start Address Register */
#define DLSR 0xC06A          /* Destination Line Start Address Register */

#define TMGR 0x8030           /* Top Margin Register */
#define LMGR 0x8040           /* Left Margin Register */
#define RMGR 0x8060           /* Right Margin Register */
#define WR   0x8050           /* Wraparound Register */
#define PWR  0x8070           /* Page Width Register */

#define TFLR 0x4044           /* Time Fill Register */
#define RCR  0x4048           /* Restart Control Register */
#define CER  0x4068           /* Express Register */
#define PR   0x4074           /* Parameter Register */
#define CR   0x4076           /* Command Register */
#define SR   0x4078           /* Status Register */


/* NOTE:  Not all registers have corresponding expansion addresses */
/*      TMGR, LMGR, RMGR, TFLR, CRCR, and CER are the exceptions    */

/*****************************************************************************/
/*                                                                           */
/*                              MAIN PROGRAM                                 */
/*                                                                           */
/*****************************************************************************/

#include "stdio.h"

#define COMP   0
#define EXPAND 1
#define BOTH   2

float width;          /* document width in millimeters */
float len;            /* document length in millimeters */
unsigned resol;       /* original horiz/vert resolution of document */
unsigned bm;          /* bottom margin, if any */
int hr;               /* horizontal resolution of original document */
int vr;               /* vertical resolution of original document */
int milli;            /* inches to millimeters (1" = 254 mm) */

main()
{
    long read_cep();

    /*      Insert individualized main program here or "include" these func-
            tions with your program to utilize the following functions and
            to implement other control conditions according to the functional
            specifications */
}
```

```
/*****************************************************************************/
/*                                                                           */
/*                              INITIALIZE                                   */
/*                                                                           */
/*****************************************************************************

Purpose - Set up initial values for CER, CWR, EWR, CCR and ECR registers
    needed by the remainder of the program.


Parameters - express ........................... hold CER value
             command ........................... hold command value
             wrap .............................. hold wraparound value
             hr ................................ horizontal resolution
             vr ................................ vertical resolution


Notes - CWR/EWR (wraparound register) is a 16-bit register used to specify the
    additional scan lines to be grouped into one effective line used for
    encoding and decoding.  If the wrapaparound register is loaded with zero,
    the effective line is identical to a scan line (normal operating mode).
    The contents of the wraparound register + 1 = effective line.

    CCR/ECR (command register) is an 8-bit register used to specify the mode
    of operation (one-dimensional, two-dimensional, or transparent), location
    of source and destination buffers, interrupt enable, operation controls,
    and initiation of processing.  This function is only concerned with the
    mode of operation.

    CER (express register) is an 8-bit register used to specify the number of
    lines to skip before compressing the next line after processing the
    current line.  If loaded with zero, every scan line will be compressed. If
    defined in two-dimensional mode, the current compressed line will become
    the reference line for the next compressed line (located at "n" scan lines
    below it).

    Express and wraparound modes can not occur simultaneously.  An error
    message and error code will be generated if needed.
                                                */


initialize(express,wrap,command,hr,vr)
unsigned express,wrap,command;
{
        milli = 254;

        write_cep(CER,express,COMP);

        write_cep(WR,wrap,BOTH);

        if ((!command) && (command != 1) && (command != 2))
                return(-1);
        else {
                command = command << 6;
                write_cep(CR,command,BOTH);
        }
        if (vr * hr % 2)        /* to ensure even number of bytes for buffer */
                resol = vr * hr + 1;
        else resol = vr * hr;
}
```

```
/*****************************************************************************/
/*                                                                           */
/*                              MIN_XTIME                                    */
/*                                                                           */
/*****************************************************************************

Purpose - Load the TFLR (time fill register) with the value of the following
        equation :

TFLR = (modem speed * min transmission time) / 8

If the user does not require a non-zero value for TFLR, then zero may be be
        loaded.


Parameters - modem_spd ..................... modem speed (bps)
                xtime ....................... min transmission time (mS)



Notes - TLFR is an 8-bit register used to specify the minimum length of a coded
        line.  If the number of bytes in a compressed line is greater than this
        number, time fill bits are needed.  Time fill bits will be added to the
        compressed line such that the sum of the code bits and time fill bits
        is equal to or greater than the required line length.  Time fill bits
        are all 0's.
                */



min_xtime(modem_spd,xtime)
int modem_spd,xtime;
{
        unsigned time_fill;

        time_fill = ((long) modem_spd * xtime) / 8000;
        write_cep(TFLR,time_fill,0);
}

/*****************************************************************************/
/*                                                                           */
/*                              PAPER_SIZE                                   */
/*                                                                           */
/*****************************************************************************

 Purpose - Initialize the Paper Width Register (CPWR/EPWR) as follows :

           CPWR/EPWR = (Actual paper width * horz resolution) / 8

      This function calls margin_set and comp_source functions, passing width,
      length, horizontal and vertical resolution to each.


 Parameters - flag ......................... register fill determinator


 Notes - CPWR/EPWR is a 16-bit register which specifies the page width or length
      of a scan line in increments of 8 pels (1 = 8 pels, 2 = 16 pels, etc).
```

```
     Since only the first 11 bits are significant, the largest line that can be
     handled is 16K pels long.

     Bits 11 to 15 must be set to zero.                                        */


paper_size(flag)
int flag;
{
     unsigned long pwidth;

     pwidth = width * hr / 8;
     write_cep(PWR,pwidth,flag);
}


/*****************************************************************************/
/*                                                                           */
/*                              MARGIN_SET                                   */
/*                                                                           */
/*****************************************************************************

Purpose - Load margin values (left, right, top) into respective registers
     according to the following equations :

          LMGR = (horiz resolution * left margin) / 8

          RMGR = (horiz resolution * right margin) / 8

          TMGR = vertical resolution * top margin

     This function calls expand_dest (EDCHR) and comp_source (CSCHR).

Parameters - top ........................ top margin (lines)
             lm,rm ...................... left/right margin (mm)

Notes - The LMGR, RMGR, and TMGR are 16-bit registers used to specify the width
     (or length in lines as needed for TMGR). If any of these registers
     contain zero, then the original document margins will be used. Margin
     register specifications override the margins of the original document thus
     resulting in an altered transmitted copy.

     An error message and code (-2) will be generated if the sum of the left
     and right margins are greater than the page width (as specified in the
     CPWR/EPWR).

     Bits 11 through 15 must be set to 0.
     */

margin_set(top,lm,rm)
unsigned top;
float lm,rm;
{
        unsigned long left,right,tm,pwidth;

        left = hr * lm  / 8;
        write_cep(LMGR,left,0);

        right = hr * rm / 8;
        write_cep(RMGR,right,0);

        tm = top * vr;
        write_cep(TMGR,tm,0);
```

```
        pwidth = width * hr / 8;

        if ((left + right) > pwidth)
                return(-2);
        else {
                expand_dest(width,len,hr,vr,bm);
                comp_source(width,len,hr,vr,bm);
        }
}

/*****************************************************************************/
/*                                                                           */
/*                                  GPARAM                                   */
/*                                                                           */
/*****************************************************************************

Purpose - Set bits 3,4 and 5 (G0, G1, G2) of the expander parameter register
        (EPR) with the granularity control (G-Parameter).


Parameters - param ..... hold G value (0 - 7)


Notes - The G-Parameter is used to specify the number of times that each scan
        line (as specified by the EPWR) should be duplicated in the destination
        buffer.  For instance, when G = 3, each scan line accessed in the
        source buffer and expanded will be written into the destination buffer
        a total of four times.  The EDWCR will be incremented four times, each
        time the ESWCR is incremented once.  The G-Parameters values are :

                ---------------------------------------------
                G2          G1          G0          G-Parameter
                ---------------------------------------------
                0           0           0               0
                0           0           1               1
                0           1           0               2
                0           1           1               3
                1           0           0               4
                1           0           1               5
                1           1           0               6
                1           1           1               7
                ---------------------------------------------

        An error will result if the EWR is not zero when the G-parameter is
        nonzero.  This error will be indicated by the EIC bit in the ESR
        (expander status register).                                          */


gparam(param)
unsigned char param;
{
        unsigned char gvalue;

        gvalue = read_cep(PR,1);
        param = gvalue | (param << 3);
        write_cep(PR,param,1);
}
```

```
/*****************************************************************************/
/*                                                                           */
/*                                  KPARAM                                   */
/*                                                                           */
/*****************************************************************************


Purpose - Load the K-value in the Parameter Register (CPR/EPR).


Parameters - param ........................... k-parameter value
             flag ............................ register test variable


Notes - The CPR and EPR are 8-bit registers whose contents specifies the
     K-parameter, the G-parameter (only for EPR), data format control   bits
     (only for CPR), source buffer attribution bits, and the auto EOL bit.

     The K-parameter is specified in the first three bits of both parameter
     registers.  The allowed values are :

                  ------------------------------------------
                  Bit2      Bit1       Bit0       K-Parameter
                  ------------------------------------------

                    0         0          0          infinity
                    0         0          1             1
                    0         1          0             2
                    0         1          1             3
                    1         0          0             4
                    1         0          1             5
                    1         1          0             6
                    1         1          1             7
                  ------------------------------------------

        The K value is only used during two dimensional compression/expansion.
        This value is ignored during transparent, or one dimensional mode. The
        K parameter value specifies the number of lines to be included in the
        compression/expansion process.                                        */
kparam(param,flag)
unsigned char param;
int flag;
{
        unsigned char cparam,eparam;

        cparam = read_cep(PR,0);      /* fetch current CPR value */
        eparam = read_cep(PR,1);      /* fetch current EPR value */

        if ((param < 0) || (param > 7))     /* invalid K value found */
                return(-10);
        else {
                if (flag == 2) {       /* fill CPR and EPR */
                    param = param | cparam;
                    write_cep(PR,param,2);
                }
                if (flag == 1) {       /* fill EPR only */
                    param = param | eparam;
                    write_cep(PR,param,1);
                }
                if (flag == 0) {       /* fill CPR only */
                    param = param | cparam;
                    write_cep(PR,param,0);
```

```
                }
        }
}

/*****************************************************************************/
/*                                                                           */
/*                              DUMP_REG                                     */
/*                                                                           */
/*****************************************************************************

Purpose - Allow user to dump the register contents and verify the status
       after the initial set-up procedure is completed.

Parameters - none.

Notes - This function will list the contents of all the CEP registers.  It is
       suggested that this function be called after the initialization
       procedure to ensure that the intended values are installed.          */

dump_reg()
{
 printf("\n\t------------------------");
 printf("\n\tAm7970 Register Status\n");
 printf("\t------------------------\n");

 printf("CSWCR = %lx : ESWCR = %lx\n",read_cep(SWCR,0),read_cep(SWCR,1));
 printf("CSCAR = %lx : ESCAR = %lx\n",read_cep(SCAR,0),read_cep(SCAR,1));
 printf("CSCHR = %lx : ESCHR = %lx\n",read_cep(SCHR,0),read_cep(SCHR,1));
 printf("CDWCR = %lx : EDWCR = %lx\n",read_cep(DWCR,0),read_cep(DWCR,1));
 printf("CDCAR = %lx : EDCAR = %lx\n",read_cep(DCAR,0),read_cep(DCAR,1));
 printf("CDCHR = %lx : EDCHR = %lx\n",read_cep(DCHR,0),read_cep(DCHR,1));
 printf("CSAHR = %lx : ESAHR = %lx\n",read_cep(SAHR,0),read_cep(SAHR,1));
 printf("CDAHR = %lx : EDAHR = %lx\n",read_cep(DAHR,0),read_cep(DAHR,1));
 printf("CSLSR = %lx : ESLSR = %lx\n",read_cep(SLSR,0),read_cep(SLSR,1));
 printf("CDLSR = %lx : EDLSR = %lx\n",read_cep(DLSR,0),read_cep(DLSR,1));

 printf("CWR   = %lx : EWR   = %lx\n",read_cep(WR,0),read_cep(WR,1));
 printf("CPWR  = %lx : EPWR  = %lx\n",read_cep(PWR,0),read_cep(PWR,1));
 printf("CRCR  = %lx : ERCR  = %lx\n",read_cep(RCR,0),read_cep(RCR,1));
 printf("CPR   = %lx : EPR   = %lx\n",read_cep(PR,0),read_cep(PR,1));
 printf("CCR   = %lx : ECR   = %lx\n",read_cep(CR,0),read_cep(CR,1));
 printf("CSR   = %lx : ESR   = %lx\n",read_cep(SR,0),read_cep(SR,1));

 printf("TMGR  = %lx\n",read_cep(TMGR,0));
 printf("LMGR  = %lx\n",read_cep(LMGR,0));
 printf("RMGR  = %lx\n",read_cep(RMGR,0));
 printf("TFLR  = %lx\n",read_cep(TFLR,0));
 printf("CER   = %lx\n",read_cep(CER,0));
}

/*****************************************************************************/
/*                                                                           */
/*                              COMP_SOURCE                                  */
/*                                                                           */
/*****************************************************************************
```

```
Purpose - Load the compressor source count holding register (CSCHR) and
     initialize the compressor source working count register (CSWCR) in
     one- and two-dimensional modes as specified by the command register
     (CCR/ECR).  This function calls comp_dest (CDCHR) and expand_source
     (ESCHR) routines.


Parameters - none.


Notes - This function reads the CCR for 1D, 2D or transparent mode
     (bits 6 and 7 of the CCR are called the Mode Control Field and are
     set at 01,10,00 respectively).  Depending upon the operating mode,
     wraparound and/or express mode is also checked (appropriate error
     messages are generated).                                              */



comp_source()
{
        unsigned cwr,ccr,cer;
        unsigned long buffer;

        cwr = read_cep(WR,0);     /* fetch contents of wraparound, */
        ccr = read_cep(CR,0);     /*    command and */
        cer = read_cep(CER,0);    /*    express registers */

/* test for one-dimensional mode */
        if ((ccr & 0x40) == 0x40) {      /* two most sig bits = 01 */
                if ((cwr) && (cer))
                        return(-3);
                else if (!cwr) {            /* in wraparound mode */
                        buffer = resol * (cwr + 1) * width / 8;
                        buffer = ~buffer + 1;

                        write_cep(SCHR,buffer,0);
                }
                else if (!cer) {            /* in express mode */
                        buffer = resol * (cer + 1) * width / 8;
                        buffer = ~buffer + 1;

                        write_cep(SCHR,buffer,0);
                }
                else {                      /* ordinary compression mode */
                        buffer = resol * width / 8;
                        buffer = ~buffer + 1;

                        write_cep(SCHR,buffer,0);
                }
        }

 /* test for two-dimensional mode */
        if ((ccr & 0x80) == 0x80) {       /* 2D most sig bits = 10 */
                if (cwr)
                        return(-4);
                else {
                        buffer = resol * (cer + 1) * width / 8;
                        buffer = ~buffer + 1;

                        write_cep(SCHR,buffer,0);

                        comp_dest(resol,width);
```

```
                        expand_source(resol,width);
                }
        }
        else if ((ccr & 0x00) == 0x00) {    /* transparent mode */
                if ((!cwr) || (!cer))
                        return(-5);
                else {
                        buffer = resol * width / 8;
                        write_cep(SCHR,buffer,0);
                }
        }
}

/*****************************************************************************/
/*                                                                           */
/*                                COMP_DEST                                  */
/*                                                                           */
/*****************************************************************************


Purpose - Loads the Compressor Destination Count Holding Register (CDCHR). The
     CDWCR is initialized with the same value as CDCHR by the Am7970.


Parameters - none.


Notes - The CDCHR is a 24-bit register used to specify the length (in bytes) of
     the destination buffer.  The buffer length is specified as a two's
     complement number.  Contents of the CDCHR is automatically loaded into the
     CDWCR whenever a restart operation is initiated.                      */


comp_dest()
{
        unsigned long buffer;
        unsigned cwr;

        cwr = read_cep(WR,0);
        buffer = resol * (cwr + 1) * width / 8;
        buffer = ~buffer + 1;

        write_cep(DCHR,buffer,0);
}

/*****************************************************************************/
/*                                                                           */
/*                               EXPAND_DEST                                 */
/*                                                                           */
/*****************************************************************************


Purpose - Loads the Expander Destination Count Holding Register (EDCHR). The
     EDWCR is initialized with the same value as EDCHR by the CEP.


Parameters - none.


Notes - The EDCHR is a 24-bit register used to specify the length (in bytes) of
     the destination buffer.  The buffer length is specified in two's
```

```
     complement form.  The initial value is loaded into the destination working
     count register (EDWCR) whenever a restart operation is initiated.

     A worst case compression ratio of 10:1 is assumed when calculating the
     destination buffer (divide by 80 instead of 8).  If negative compression
     is encountered, the CEP will interrupt the CPU signaling the shortage of
     buffer space.                                    */


expand_dest()
{

        unsigned ewr,even;
        unsigned long buffer;

        ewr = read_cep(WR,1);

        buffer = resol * (ewr + 1) * width / 80;

        buffer = ~buffer + 1;
        write_cep(DCHR,buffer,1);
}

/*****************************************************************************/
/*                                                                           */
/*                              EXPAND_SOURCE                                */
/*                                                                           */
/*****************************************************************************


Purpose - Loads the value of the source count holding register (ESCHR). The
     ESWCR is initialized with the same value as ESCHR by the CEP.


Parameters - none.


Notes - The ESCHR is a 24-bit register used to specify the source buffer length
     in bytes.  This value is specified in two's complement form.  The initial
     value is loaded into the destination working count register(EDWCR)
     whenever a restart operation is initiated.    */


expand_source()
{
        unsigned long buffer;

        buffer = resol * width / 8;
        buffer = ~buffer + 1;

        write_cep(SCHR,buffer,1);
}

/*****************************************************************************/
/*                                                                           */
/*                                WRITE_CEP                                  */
/*                                                                           */
/*****************************************************************************


 Purpose -  To evaluate the register address (passed as addr) and determine if
      it is a one, two, or three byte register.  The value to be loaded is
```

```
     divided into one, two, or three bytes before sending it out to the CEP.
     The base address (addr) is incremented as needed to write to the CEP.

Parameters - addr ............................. register address
             value ............................. register contents
             flag .............................. register fill flag

Notes -  A "flag" is passed to determine if the corresponding expansion
     register is also to be filled with "value".  If flag = 1 then the msb of
     the port address will be toggled and the expansion register will be
     loaded; otherwise only one register is filled on each function call.  If
     the register is one of the five exceptions (neither compression or or
     expansion, as mentioned above), then pass flag = 0.

     CAUTION -- This function uses a C86 library routine called "outportb".
     This is a hardware dependent routine and may not be applicable toward your
     particular system.  It may be necessary to furnish an I/O driver that
     serves the same purpose.  This function expects the following :

             unsigned outportb(portno,value)
             unsigned int portno;
             char value;

     Returns the byte output to the port (user supplied).  The port number must
     be valid for the addressed device.  For the CEP, an 8-bit number is
     required, but in other cases a 16-bit number may be needed.  If this is
     the case, it is suggested to place the port number in both upper and lower
     bytes of portno.                                                    */

write_cep(addr,value,flag)
unsigned addr;
long value;
int flag;
{
     char byte1,byte2,byte3;

     byte1 = value;
     byte2 = value >> 8;
     byte3 = value >> 16;
     if ((addr & 0xC000) == 0xC000) {      /* 3-byte register */

             addr &= 0x00FF;
             if (flag == BOTH)
                     both_reg(addr,byte1,byte2,byte3,3);
             else if (flag == EXPAND)
                     expand_only(addr,byte1,byte2,byte3,3);
             else if (flag == COMP)
                     comp_only(addr,byte1,byte2,byte3,3);
             else return(-6);
     }
     else if ((addr & 0x8000) == 0x8000) {  /* 2-byte register */
             addr &= 0x00FF;
             if (flag == BOTH)
                     both_reg(addr,byte1,byte2,byte3,2);
             else if (flag == EXPAND)
                     expand_only(addr,byte1,byte2,byte3,2);
             else if (flag == COMP)
                     comp_only(addr,byte1,byte2,byte3,2);
             else return(-6);
     }
     else if ((addr & 0x4000) == 0x4000) {
             addr &= 0x00FF;
```

```
            if (flag == BOTH)
                    both_reg(addr,byte1,byte2,byte3,1);
            else if (flag == EXPAND)
                    expand_only(addr,byte1,byte2,byte3,1);
            else if (flag == COMP)
                    comp_only(addr,byte1,byte2,byte3,1);
            else return(-6);
    }
    else return(-7);
}

/*****************************************************************************/
/*                                                                           */
/*                              BOTH_REG                                     */
/*                                                                           */
/*****************************************************************************


Purpose - To send out the port number and corresponding value to be loaded into
     the register.  Flag variable from write_cep function is passed as "size"
     which determines the size (1,2 or 3 bytes) of the register to be loaded.
     This variable signifies that the value to be sent is a one,two, or three
     byte register.


Variables - addr ............................. base port address
            byte1 ............................ lsb of loaded value
            byte2 ............................ second lsb of loaded value
            byte3 ............................ msb of loaded value
            size ............................. register size in bytes


Notes - All variables are passed from write_cep function.  A C86 function
     called "outportb" is used in this function (as well as expand_only and
     comp_only) which is hardware dependent (See note in write_cep
     description).                                                        */
both_reg(addr,byte1,byte2,byte3,size)
unsigned addr;
char byte1,byte2,byte3;
int size;
{
        if (size == 3) {
                outportb(addr,byte1);
                outportb(addr + 2,byte2);
                outportb(addr + 4,byte3);

                outportb(addr | 0x80,byte1);
                outportb(addr + 2 | 0x80,byte2);
                outportb(addr + 4 | 0x80,byte3);
        }
        else if (size == 2) {
                outportb(addr,byte1);
                outportb(addr + 2,byte2);

                outportb(addr | 0x80,byte1);
                outportb(addr + 2 | 0x80,byte2);
        }
        else if (size == 1) {
                outportb(addr,byte1);
                outportb(addr | 0x80,byte1);
        }
```

```
        else return(-8);
}

/*****************************************************************************/
/*                                                                           */
/*                              EXPAND_ONLY                                  */
/*                                                                           */
/*****************************************************************************


Purpose - To send out port number and value for expansion register only.
     Use "size" to determine size in bytes of the register to be loaded.


Variables - addr ............................. base port address
            byte1 ............................ lsb of value
            byte2 ............................ second lsb of value
            byte3 ............................ msb of value
            size ............................. register size in bytes


Notes - All variables are passed from write_cep function.  A C86 function
     called "outportb" is used in this function (as well as expand_only and
     comp_only) which is hardware dependent (See note in write_cep
     description).                                                        */



expand_only(addr,byte1,byte2,byte3,size)
unsigned addr;
char byte1,byte2,byte3;
int size;
{
        if (size == 3) {
                outportb(addr | 0x80,byte1);
                outportb(addr + 2 | 0x80,byte2);
                outportb(addr + 4 | 0x80,byte3);
        }
        else if (size == 2) {
                outportb(addr | 0x80,byte1);
                outportb(addr + 2 | 0x80,byte2);
        }
        else if (size == 1)
                outportb(addr | 0x80,byte1);
        else return(-8);
}

/*****************************************************************************/
/*                                                                           */
/*                                COMP_ONLY                                  */
/*                                                                           */
/*****************************************************************************


Purpose - To send out port number and value for compression register only.  Use
     variable "size" to determine the number of bytes of the register to be
     loaded.


Variables - addr ............................. base port address
            byte1 ............................ lsb of value
            byte2 ............................ second lsb of value
```

```
            byte3 ............................. msb of value
            size .............................. register size in bytes


Notes - All variables are passed from write_cep function.  A C86 function
     called "outportb" is used in this function (as well as expand_only and
     comp_only) which is hardware dependent (See note in write_cep
     description).                                                        */



comp_only(addr,byte1,byte2,byte3,size)
unsigned addr;
char byte1,byte2,byte3;
int size;
{
        if (size == 3) {
                outportb(addr,byte1);
                outportb(addr + 2,byte2);
                outportb(addr + 4,byte3);
        }
        else if (size == 2) {
                outportb(addr,byte1);
                outportb(addr + 2,byte2);
        }
        else if (size == 1)
                outportb(addr,byte1);
        else return(-8);
}

/*************************************************************************/
/*                                                                       */
/*                              READ_CEP                                 */
/*                                                                       */
/*************************************************************************


Purpose - To read the contents of a CEP register.  The register "base" address
     is sent out and depending on the size of the register, one, two, or three
     bytes are fetched.  The fetched bytes are then combined into one 32-bit
     value to be returned in each call.


Parameters - addr ............................. "base" address of register
             flag ............................. distinguish between
                                                compression & expansion
                                                register (0 or 1)


Notes - This function returns the register contents in a 32-bit value even
     though the registers are only one, two, or three bytes in length.  The
     user is then responsible for modifying this larger value as needed for
     specific purposes.  The function returns only one register value at a
     time; therefore, it must be called twice in succession (with different
     flag values) to obtain the values for both register pairs.

     CAUTION - This function uses a C86 library routine called "inportb".
     This is a hardware dependent function and may not be applicable toward
     your particular system.  It may be necessary to furnish an I/O driver
     that serves the same purpose.  This function expects the following:

                    unsigned char inportb(portno)
```

```
                    unsigned int portno;
                    unsigned char byte;

    This function returns the byte contained at "portno" but the read_cep
    function returns a 32-bit value.  This is to accommodate the various sizes
    of the registers.                                                          */


long read_cep(addr,flag)
unsigned addr;
int flag;
{
        unsigned char byte1,byte2,byte3;
        long value;

        if ((addr & 0xC000) == 0xC000) {     /* 3-byte register */
                addr &= 0x00FF;
                if (flag == COMP) {

                        byte1 = inportb(addr);
                        byte2 = inportb(addr + 2);
                        byte3 = inportb(addr + 4);

                        value = (byte3 << 16) + (byte2 << 8) + byte1;

                        return(value);
                }
                else if (flag == EXPAND) {
                        byte1 = inportb(addr | 0x80);
                        byte2 = inportb(addr + 2 | 0x80);
                        byte3 = inportb(addr + 4 | 0x80);

                        value = (byte3 << 16) + (byte2 << 8) + byte1;
                        return(value);
                }
        }
        else if ((addr & 0x8000) == 0x8000) {
                addr &= 0x00FF;
                if (flag == COMP) {
                        byte1 = inportb(addr);
                        byte2 = inportb(addr + 2);

                        value = (byte2 << 8) + byte1;
                        return(value);
                }
                else if (flag == EXPAND) {
                        byte1 = inportb(addr | 0x80);
                        byte2 = inportb(addr + 2 | 0x80);

                        value = (byte2 << 8) + byte1;
                        return(value);
                }
         }
         else if ((addr & 0x4000) == 0x4000) {
                 addr &= 0x00FF;
                 if (flag == COMP)
                         return((long) inportb(addr));
                 else if (flag == EXPAND)
                         return((long) inportb(addr | 0x80));
         }
         else return(-9);
 }
 /*****************************************************************************/
 /************************** END OF PROGRAM ***********************************/
```

4.1.2 Error Return Messages

Return Code	Message
-1	Invalid operation mode for CMCR/EMCR. Enter '0' for transparent mode Enter '1' for one-dimensional mode Enter '2' for two-dimensional mode Error returned from Initialize function.
-2	Sum of left and right margins is greater than paper width. Check paper_size and margin_set functions.
-3	Wraparound and express mode can not occur simultaneously in one-dimensional mode. Check CWR, EWR, CER registers. Error returned from source_buffer function.
-4	Wraparound can not occur in one-dimensional mode. Check CWR, EWR, CMCR, EMCR registers. Error returned from source_buffer function.
-5	Wraparound and/or express mode can not occur in transparent mode. Check CMCR/EMCR registers. Error returned from Source_buffer function.
-6	Invalid "flag" variable passed. Enter 0,1, or 2 only to signalboth_reg, expand_only, or comp_only function calls. Error returned from write_cep function.
-7	Invalid register address detected. Check if program register define statements have been altered. Most significant byte of register address should start with C0__, 80__, 40__ only.
-8	Invalid register "size" variable passed. Enter 1,2 or 3 bytes only. Returned from comp_only, expand_only, or both_reg functions.
-9	Invalid register address detected. Can not read data from CEP with this address. Check register address define statements. Send only "base" address as it is defined. Errorreturned from read_cep function.
-10	Invalid K-parameter value entered. Must be 0-7 (000 to 111) where 0 (000) is infinity.

4.2 Image File Analysis Program Description

(written by Deyoung Hong—12/18/84)

This is a functional description of the Image File Analysis Program. The program is written in C language for execution on an IBM PC. Its main task is to analyze image files for the approximation of compression ratios, compressor throughputs, and expander throughputs with either one or two dimensional computations. The program consists of a header declaration section, a main function, and other sub-functions. A listing of this program is given in Appendix B.

4.2.1 Header Declaration Section

This section contains some macro statements for substitutions of constants, and the declaration of external variables used throughout the program. The external variables used in this program are for tables of data, and for files. The tables are declared as arrays, and are useful for quick reference in order to minimize the execution time. The name and characterization of each table is as follows:

ncc =	Table containing the number of color changes in each eight bit binary sequence from 00000000 to 11111111. The value of the sequence (byte) is the index value for access to this table.
wtermc =	Table containing the length of the white terminating codeword for each run length from 0 to 63. It is taken from the Modified Huffman Code Table. The run length of code is the index value for access to this table.
btermc =	Table containing the length of the black terminating codewords for each run length from 0 to 63. It is taken from the Modified Huffman Code Table. The run length of code is the index value for access to this table.
wmakec =	Table containing the length of the white make-up codewords taken from the Modified Huffman Code Table. The run length the make-up code represents divided by 64 is the index value for access to this table.
bmakec =	Table containing the length of the black

make-up codewords taken from the Modified Huffman Code Table. The run length the make-up code represents divided by 64 is the index value for access to this table.

4.2.2 Main Function

This function is the control routine of the program. It prompts user to enter all necessary data before it actually performs the analysis. It then calls other subroutines to do certain tasks of computation. Finally it prints the result of analysis, and also saves the result in an output file.

The prompt questions for filenames and parameters are:

> Name of image file to be analyzed? (Name of the image file to be analyzed—drive name and file extension are considered)

> Number of wait cycles introduced by the document memory? (This data is used for throughput calculations)

> Number of wait cycles introduced by main memory? (This data is also used for throughput calculations)

> Maximum number of pixels per line? (This is for the horizontal resolution of the image file. User should enter the number of bits that are in each scan line)

> Maximum number of lines per page? (This is for the vertical resolution of the image file. User should enter the number of lines to be scanned as a page in the file).

> Dimension of coding (1 or 2)? (Enter 1 for one dimensional coding or enter 2 for two dimensional coding. If Case 2 is entered, the parameter k is then asked:

> Enter parameter k (0 for infinity): k is the number of lines to be coded in two dimensions. One line is coded in one dimension for every k lines are coded in two dimensions. Enter 0 for two dimensional coding on all lines)

> Name of file to store output results? (Enter name of file to store the result of analysis.

After reading all these data, the routine will then call either function scan1d() to compute with one dimensional coding or scan2d() to compute with two dimensional coding the compression ratio. These functions also compute the fractional number of bytes that contain color changes count in the document. The compressor throughput is calculated by the function comp1d(), and the expander throughput is calculated by the function expa1d(). Due to the complexity of analyzing the case of two dimensional calculations for the compressor and expander throughputs, the results are estimated using the formulas for one dimensional calculations multiplied by 4. The results are printed on the screen and saved in the file which user entered earlier.

4.2.3 Sub Functions

The sub-function are huffcode(), code1d(), code2d(), scan1d(), scan2d(), comp1d(), expa1d(), and presult(). The description of each function is as follows:

huffcode() – This function takes as input the white or black runs and returns the corressponding length of the compressed code after it looks up the tables in the header declaration section.

code1d() – This function takes as input one line of the original data and return the length of the compressed line in one dimension. It scans the number of white and/or black runs then each time calls function huffcode() to get and add up the length of compressed code.

code2d() – This function takes as input two lines of code, one is the line to be coded in two dimension, and the other is the previous line as reference.

scan1d() – This function analyzes the whole document by scanning one line at a time for the whole page. It looks up the table ncc to count the number of bytes that contain the same number of color changes. It also calls function code1d() to add up the total length of the compressed data in a page. It then can calculate the fractional number of bytes that contain number of color changes count and the compression ratio for one dimensional coding.

scan2d() – This function analyzes the whole document using the parameter k to perform either one dimensional coding for every first line of k lines or two dimensional coding for k lines. It thus calls both functions code1d() and code2d(), but depends on the value of k. Like scan1d() it also looks up the table ncc to count the number of bytes that contain the same number of color changes. It calculates the fractional number of bytes that contain number of color changes count and the two dimensional compression ratio.

comp1d() – This function calculates and returns the compressor throughput of a document in one

dimension.

expa1d() – This function calculates and returns the expander throughput of a document in one dimension.

presult() – This function saves a summary of the analysis program consisting of the input data and output results in an output file.

Here are the two output results of the CCITT document #5 stored in output files produced by the Image File Analysis Program:

4.2.4 Image Analysis Program Execution Report

```
  Image file analysed:  b:ccitt5.cp
  Number of wait cycles introduced by the document memory:  0
  Number of wait cycles introduced by the main memory:  0
  Horizontal resolution:  1680 pixels/line
  Vertical resolution:  1188 lines/page

% ONE DIMENSIONAL CODING -- GROUP III CODING IS ASSUMED %

  Fractional number of bytes which contain 0 through 7 color changes:
        f0 = 0.889
        f1 = 0.067
        f2 = 0.042
        f3 = 0.002
        f4 = 0.000

  Compression Ratio = 7.303
  Compressor Throughput (in MBPs) = 8.160
  Expander Throughput (in MBPs) = 9.068

*****************************************************************************


*************** IMAGE ANALYSIS PROGRAM EXECUTION REPORT ****************

  Image file analysed:  a:ccitt5.cp
  Number of wait cycles introduced by the document memory:  0
  Number of wait cycles introduced by the main memory:  0
  Horizontal resolution:  1680 pixels/line
  Vertical resolution:  1188 lines/page

% TWO DIMENSIONAL CODING (K = INFINITY) -- GROUP IV CODING IS ASSUMED %

  Fractional number of bytes which contain 0 through 7 color changes:
        f0 = 0.889
        f1 = 0.067
        f2 = 0.042
        f3 = 0.002
        f4 = 0.000

  Compression Ratio = 10.125
  Compressor Throughput (in MBPs) = 2.175
  Expander Throughput (in MBPs) = 2.436

Refer to Appendix B for a copy of the program used to obtain these results.
```

5.1 Am7970A CEP INTERFACE TO THE 68000 CPU

This design presents an example of how to use the CEP in a 68000 system. Though the Am7970A was designed for easy interface to the iAPx family, it can easily be adapted to the 68000.

5.1.1 General Discussion

The example may be a part of a workstation environment or an image storage application such as an optical disc storage device. Also note that all FAX applications (Group 3 and 4) are well served.

Figure 5-1 is the Am7970A CEP interface to the 68000 CPU. This illustration only shows how the system interface of the CEP is embedded in such a system. If very high throughput is desired, the document interface of the CEP should be connected to a large memory bank to buffer the image data. The logic for the document buffer interface is straightforward. Using memory connected to the document side as a source buffer (image data) and the system interface as the destination buffer (coded data), a whole page of image data with a resolution of 300 pixels per inch can be compressed in 1-4 seconds.

The document buffer may be loaded through the CEP system interface in transparent mode. It could also be designed as a dual port memory which is loaded directly by the CPU or by a DMA device. A third approach could load the document buffer directly by a scanner or a image processing peripheral device. The last method reduces the necessary data transfers to an absolute minimum and is therefore the preferred solution for very high performance applications.

This design assumes that the 68000 is connected to a memory bank, either onboard or via a bus interface. By setting the appropriate mode in the CEP's command register, the user determines whether this memory contains either the source or the destination buffers for the CEP, or both.

5.1.2 Hardware Description

A latch and two drivers are used to demultiplex the data from address bits A16-A23 of the CEP and to direct the byte-oriented data stream of the CEP to the upper and lower bytes of the data bus of the 68000. On even addresses, data is transfered through the upper half of the bus; on odd addresses, data passes through the lower half of the 68000 data bus.

All register accesses into the CEP are performed through the upper data bus because all CEP register addresses are even. They are addressed by the signals A0-A7.

Almost all of the conversion logic for the control signals was combined into one PAL. This minimizes the hardware required for customizing the CEP to any kind of processor. The AmPAL22V10 was chosen because it provides more outputs than most other PALs and provides full freedom in choice of output characteristics (polarity, latched/unlatched function). The PAL equations are written for the PLPL PAL assembler. They can easily be changed for any other available PAL Assembler. Refer to Figure 5-2 and Figure 5-3 for the Pal Device equations.

The PAL converts the $\overline{RD}$, $\overline{WR}$ and A0 signals of the CEP to $\overline{UDS}$, $\overline{LDS}$, and R/$\overline{W}$ signals of the 68000. It provides the control signals for the data transceivers and transforms the two-wire bus arbitration signals of the CEP (HRQ, HLDA) to the three-wire arbitration scheme of the 68000 ($\overline{BR}$,$\overline{BG}$ and $\overline{BGACK}$).

The 68000 CPU uses a memory mapped I/O address scheme. The I/O interface logic assigns a memory area to the CEP internal registers using standard address comparators. The $\overline{CS}$ output is validated by $\overline{AS}$ LOW. In sophisticated operating systems the CEP access should be reserved to supervisor level memory accesses. Here this is accomplished by an LS138 decoding this access mode from the signals FC0-2. The output is used to enable the comparators.

When designing the memory interface, care should be taken that the setup time for the READY input is meet. If the environment does not provide this demand, the READY signal coming from the memory must be synchronized with a flip-flop register.

5.1.3 Operation

Interrupt Handling

The CEP notifies the CPU about an exception condition (e.g. end of page) by driving the INTR line HIGH. The CEP does not produce interrupt vectors by itself. If a specific application demands a user vector to be asserted by the peripheral, an

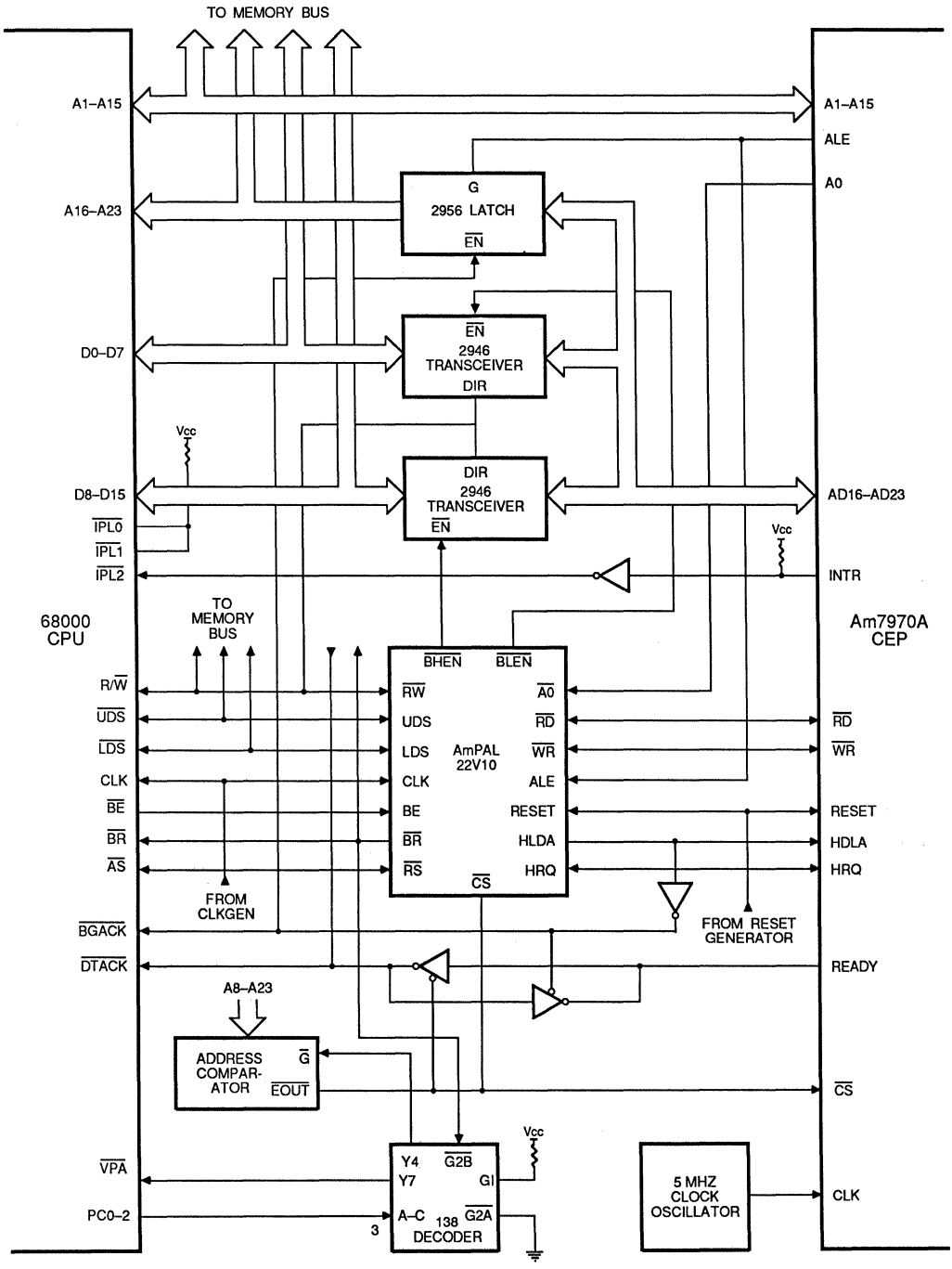


Figure 5-1 Am7970A CEP to 68000 CPU Interface

07666A 5-1

```
DEVICE (AMPAL22V10)
"7970A CEP to 68000 Interface Controller
CEP68KPAL          VERSION 1.0
AMD                Wolfgang Kemmler 9-12-85 "

PIN
CLK     = 1        VCC     = 24
/CS     = 2        /BHEN   = 23
ALE     = 3        /BLEN   = 22
/BG     = 4        A0      = 21
HRQ     = 5        HLDA    = 20
RESET   = 6        /RD     = 21
NC      = 7        /WR     = 19
NC      = 8        /UDS    = 18
NC      = 9        /LDS    = 17
NC      = 10       /RW     = 16
NC      = 11       NC      = 15
GND     = 12       NC      = 14

BEGIN

IF ( RESET ) THEN ARESET() ;

IF ( HLDA ) THEN ENABLE ( RW ) ; HLDA RW = WR ;

IF ( HLDA ) THEN ENABLE( UDS ) ; UDS = RD * /A0 + WR * /A0 ;

IF ( HLDA ) THEN ENABLE( LDS ) ; LDS = RD * A0 + WR * A0 ;

IF ( /HLDA ) THEN ENABLE( RD ) ; RD = /RW * UDS ;

IF ( /HLDA ) THEN ENABLE( WR ) ; WR = RW * UDS ;

IF ( /HLDA ) THEN ENABLE( A0 ); A0 = UDS ;

BHEN = HLDA * /A0 * RD + HLDA * /A0 * WR + CS * UDS ;

BLEN = HLDA * A0 * RD + HLDA * A0 * WR + CS * LDS ;

BR := HRQ * BG * BR * AS + HRQ * /BG * /HLDA ;

/HLDA := /HRQ + /HRQ * /BG + /HRQ * AS + /HRQ * /HLDA
       + BG * /HLDA + AS * /HLDA ;
END
```

Figure 5-2 Am7970A CEP to 68000 Interface Controller PAL Device

```
PAL16R4                          CEP to 68000 Interface Controller Part B
VERSION 1.0
CEP68KB
AMD                              WOLFGANG KEMMLER 9-12-85

CLK /RD /WR HRQ ALE /CS /BG NC NC GND
/OE /DTACK READY /BR HLDA NC NC /AS /BGACK VCC

BR := HRQ * BG * BR * AS + HRQ * /BG * /HLDA

/HLDA := /HRQ + /HRQ * /BG + /HRQ * AS + /HRQ * /HLDA
        + BG * /HLDA + AS * /HLDA

IF ( CS ) DTACK = READY

IF ( HLDA ) READY = DTACK * RD + DTACK * WR

IF ( HLDA ) AS = ALE

BGACK = HLDA
```

Figure 5-3 CEP to 68000 Interface Controller, Part B 07666A 5-3

interrupt controller such as the Am9519A must be used.

To avoid an additional interrupt controller, this design follows an easier approach to service the interrupt request for the CPU, using the 68000 auto vector mode. The status decoder generates the interrupt acknowledge signal from the status lines F0-F2. This signal is used to drive the VPA input of the CPU. If this line instead of DTACK is asserted during an interrupt acknowledge cycle, the 68000 will use the internal auto vectors instead of an externally supplied vector.

The interrupt inputs of the 68000 are directly connected to the inverted INTR signal of the CEP without using the usual priority encoder. Assuming that the auto vector mode of the CPU is used as described above, 2 more peripherals could notify an interrupt request to the CPU by this method. With respect to all possible combinations of pending interrupt requests, the auto vector table would have to look like this:

Exception Vector Table

Vector No.	Assignment
25	Auto Vector 1
26	Auto Vector 2
27	Auto Vector 2
28	Auto Vector 4
29	Auto Vector 4
30	Auto Vector 4
31	Auto Vector 4

The vectors are selected by the 68000 according to the the priority of the interrupt inputs IPL0-IPL2. This schematic shows the CEP connected to IPL2 giving it the highest priority. The CEP removes INTR with the next access to a command register.

68000 Accesses To The Am7970A CEP Registers (Slave Mode)

By driving CS LOW, the address decoder notifies the CEP that the CPU wants to access the CEP internal registers. The CEP reacts by driving READY LOW and interrupting its internal microprogram. The READY signal is an output of the CEP as long it is in slave mode. Depending on the internal status of the CEP, READY is released after 4 - 50 CEP clock cycles.

The CEP provides a totally asynchronous slave interface. This keeps the logic very simple. The data hold time for a "slave write access" is 20 ns minimum which perfectly matches the 68000 up to a CPU clock frequency of 10MHz.

Data transfers in slave mode are generally passed through the upper bus driver (D8-D15) because all registers are located at even addresses.

Am7970A CEP System Memory Access (Master Mode)

The CEP drives HRQ HIGH to gain bus control. As soon as HLDA goes HIGH it enables its system interface lines and start a memory access.

In this operating mode, READY is an input to the CEP. READY is connected to the inverted DACK of the 68000 system. The CEP samples the READY line before driving the RD or WR signals

LOW. These signals are used to provide the $\overline{UDS}$ and $\overline{LDS}$ signals which normaly are asserted much earlier in typical 68000 systems. Therefore, the $\overline{DTACK}$ line which signals the completion of the memory access, cannot be asserted earlier than $\overline{RD}$ or $\overline{WR}$. This causes an automatic wait state for each CEP memory transfer.

The full performance of the CEP in a 68000 system can only be reached if the memory design is optimized not only for the 68000 but also for the specific CEP timing. If $\overline{UDS}$ and $\overline{LDS}$ are only used to enable the data driver of the memory banks and if the memories are fast enough, and if the READY line is driven HIGH during master access all the time (disregarding $\overline{DTACK}$), then the CEP can be used without a wait state.

NOTE:

The CEP needs only 3 clock cycles for a memory transfer while the 68000 CPU takes 4. An additional wait cycle would equal the access times of both devices, assuming they are running at the same clock frequency. A CEP running at 5 MHz without a wait state, on the other hand, would match the memory access time of a 8 MHz 68000. A 5MHz CEP does not necessarily reduce the performance of faster clocked CPUs.

5.2 Am7970A CEP INTERFACE TO THE 80188 CPU

This applicaton note shows how to use the Am7970A CEP in a low cost environment. The 8 bit data interface of the CEP to the 8 Bit 80188 microprocessor is simple. It reduces the number of additional drivers, latches and control logic to an absolute minimum. The 80188 also provides an interrupt controller and a chip select decoder. No additional parts are necessary to access and control the Am7970A CEP.

5.2.1 General Discussion

This example assumes a single board approach with an onboard memory bank which may vary from 64kBytes to 1MBytes. By adding drivers to the control signals and to the address lines A8-A15 it could easily be expanded into a bus controlled system. One MByte of memory is sufficient for storing the image data of one page with a resolution of 300 pixels per inch. This allows the CEP to compress or expand such a picture without interruption and with a minimum of software overhead. A smaller memory bank might be chosen for cost reasons.

This design does not use the faster document buffer interface of the CEP. Both the image data and the compressed data are passed through the system interface. Therefore, bus arbitration is incurred for every single transferred byte limiting the use of this example to low throughput applications.

It is definitely sufficient for FAX applications and may even serve very well for image storage applications where speed is not the most important factor. In these cases a memory bank of 64 to 128 kBytes is sufficient because the CEP is capable of processing fractions of a whole page without producing inconsistencies in the coded image data.

An additional memory bank connected to the document buffer interface will improve the overall throughput of the compression or expansion by a factor of four. This buffer should be used only to store the image data because it requires a data rate approximately 10 times higher than the compressed data.

5.2.2 Hardware Description

Figure 5-4 is a diagram of the Am7970A CEP to 80188 CPU interface. It shows how the address and data lines of the CPU and the CEP are transformed into a common demultiplexed memory bus. Additional peripheral devices could either be located on this memory bus or be connected directly to the CPU interface.

The Am7970A CEP multiplexes the data on the address lines A16-A23 while the CPU multiplexes data on the address lines A0-A7. These are the only differences between the two interfaces. All control signals can be used without conversion.

The 80188 CPU provides a 50% duty cycle clock output which can be used to drive the CEP. If the CPU runs on a higher clock rate, the CEP has to be driven by either an additional clock generator or a CPU clock that has been divided down. This may be very useful because the CEP needs only three clock cycles for a memory access while the 80188 needs four cycles. Running the CEP with a slower clockrate than the CPU does not necessarily result in slower memory access.

The READY signal coming from the memory interface must meet the set up time of the CEP READY input. For Revision A of the Am7970, a wait state must also be inserted for each memory access from the CEP. Figure 5-5 shows the logic to be added to the design above.

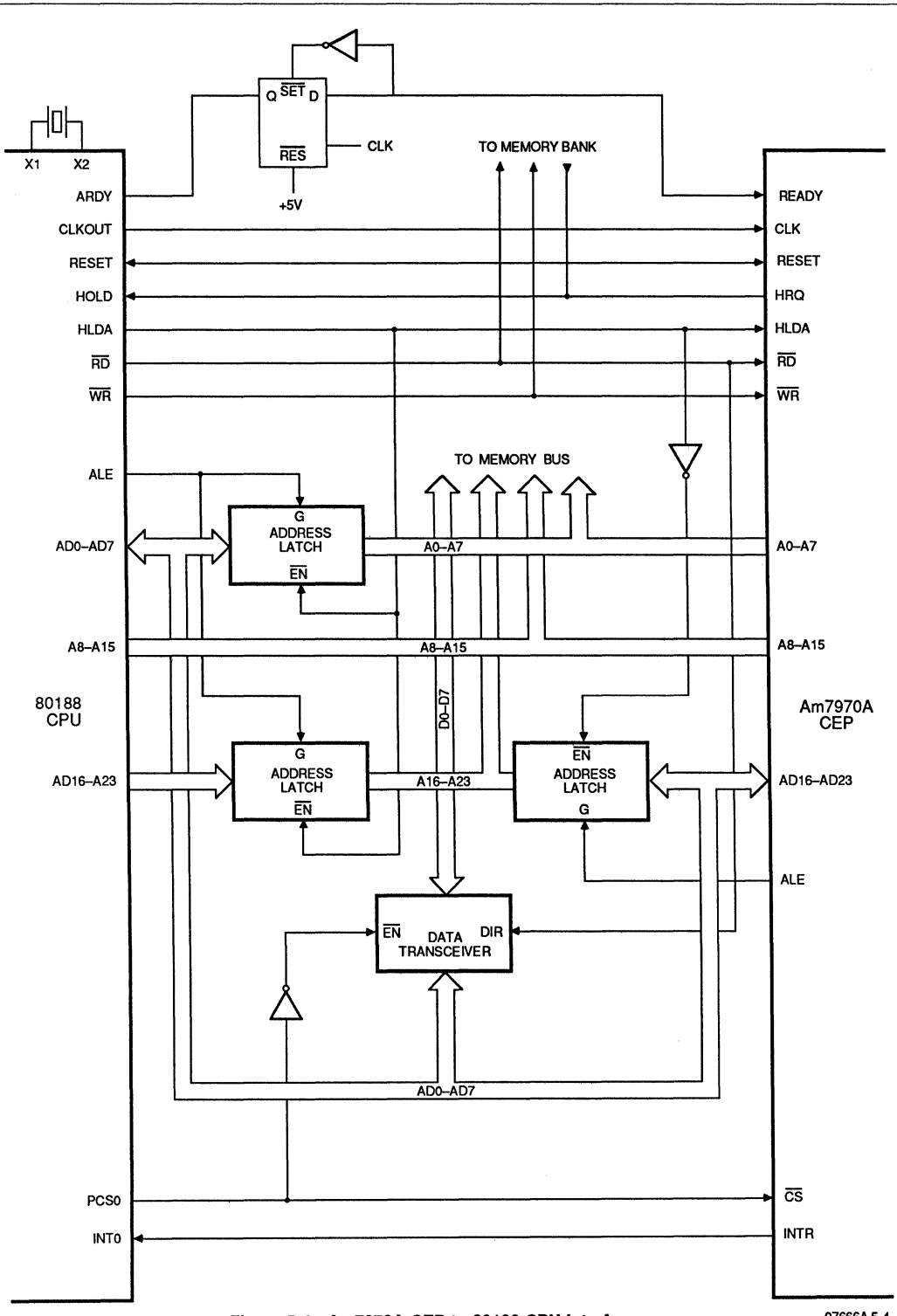
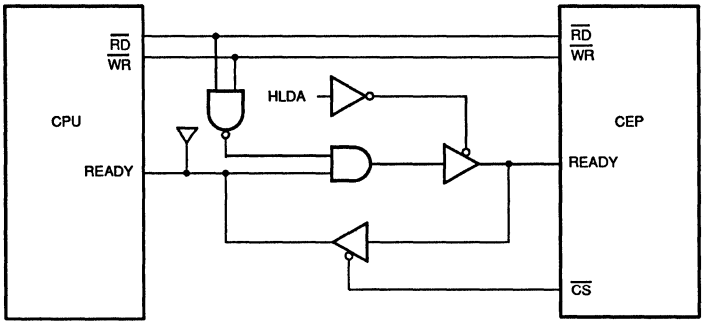


Figure 5-4 Am7970A CEP to 80188 CPU Interface

07666A 5-4



07666A 5-5

Figure 5-5 Wait State Circuit

5.2.3 Operation

80188 CPU Access to the Am7970A CEP

The CEP has many registers which provide programmability of many different options such as paper size, memory address control, and status information. Access to these registers is started by driving the $\overline{CS}$ signal LOW. This input is driven by the Peripheral Chip Select Output of the 80188. This signal also disables the data transceiver to the memory bus. The different registers are addressed by the address lines A0-A7 which are directly connected to the CEP. The CEP drives READY LOW as long as it needs to move the data to or from the appropriate register. This operation is called "slave access".

Am7970A CEP Access to the Memory

As soon as the CEP is started, it activates its internal DMA device to gain control of the memory buffer. It signals this to the CPU by driving HRQ HIGH. It then waits until HLDA is driven HIGH by the CPU to acknowledge that the bus is released. The CEP then starts a memory access. HLDA also enables the address latch for the CEP and disable the ones for the CPU. The CEP releases the bus after each byte transfer. This operation is called "master access".

The data transceiver is activated all the time except during a slave access. It normally drives data onto the memory bus. Only when $\overline{RD}$ goes LOW and $\overline{CS}$ is not active, the direction of the data transceiver is switched to the opposite direction.

5.3 Am7970A CEP EVALUATION BOARD

5.3.1 Features

- Interface for IBM PC/XT or IBM AT on the same board.
- Automatic recognition of PC or AT enviroment.
- Full master mode capability in IBM AT using AT memory for system bus access.
- 1 MByte dynamic memory on board.
- Dualport arbitration allows memory access from CEP system and document side.
- PC/XT has full access into the on board memory adressing it as 16 64KByte blocks in page mode.
- All operating modes of CEP can be evaluated with maximum performance.
- CEP hardware reset initiated by I/O address access.
- Clock rate supplied by plug in exchangable clock generators or from an external input.
- All I/O adresses are memory mapped.
- Jumper selectable DRQ/DACK and IRQ lines.
- Performs master access to AT memory

5.3.2 The CEP Evaluation Board In An IBM PC/XT

The IBM PC/XT does not allow another DMA master beside its own on-board DMA device working on the extension bus. Any attempt do so without changing the logic on the mother board will cause serious bus contention. That requires a separate memory bank dedicated to the CEP and accessible by the IBM PC/XT.

It is also quite useful to have enough memory dedicated to the document bus to hold a whole page of image data with a resolution of 300 Pixel/inch in memory. The evaluation board solves this problem by giving both sides of the CEP full access to a 1 MByte dynamic memory bank. That gives the user the freedom to asign as much memory to any side of the CEP as neccessary.

The on-board "system bus" is shared by the CEP system interface and the CPU. So that makes it a three port memory design.

Normally the "system bus" is dedicated to the CEP system interface. If the CPU wants to access the evaluation board through this bus from the extension bus it drives "SBUSRQ" LOW by accessing an I/O address. The CPU is then kept waiting by the logic with "IOCHRDY" until the CEP releases "HRQ". Then "HLDA" is driven LOW by the interface logic to prevent the CEP from reaccessing the bus.

As long as "SBUSRQ" is LOW, the CPU has free access to the system bus. The document bus side is kept in Wait state while "SBUSRQ" is active.

"SBUSRQ" is latched and must be reset by another I/O access to a different address after completion of the read and write cycles onto the evaluation board. The CPU accesses the CEP registers by driving "CEPRQ", it accesses the memory bank by driving "MEMRQ" and accesses the page latch by driving "PAGE" LOW through different I/O addresses.

Since the IBM PC/XT I/O address layout does not support enough consecutive I/O addresses, all CEP I/O addresses are memory mapped in this design.

While the CEP is compressing or expanding a document, the CPU either polls the status register of the CEP or waits for an interrupt caused by the completion or an exception of the process.

5.3.3 The CEP Evalution Board In An IBM AT

The IBM PC/XT extension bus connector is fully compatible with the new IBM AT connector. The AT introduces an additional connector to provide the extra signals needed for the increased memory size and the 16 Bit data format. The AT also offers a fully compatible PC/XT mode. Thus all functions of the evaluation board designed for the IBM PC/XT will also work on the AT without any change in software and hardware.

In addition the evaluation board wants to make use of the master mode capability offered by the AT extension bus. To do so it uses the signals of the added connector to perform a proper bus arbitration on the extension bus. A ground pin on that connector will tell the board that it is connected to the AT.

For slave mode, everything said in the previous chapter will work the same except that there is now no need for driving the "SBUSRQ" signal before accessing the board because the system bus is automatically released by the CEP due to the bus arbitration. The access of the AT onto the evaluation board is still in 8 bit PC/XT compatible mode.

In master mode the system interface of the CEP will no longer access the on-board memory but will place its address and data signals onto the AT extension bus giving free access to all the memory that is provided by the IBM AT. The bytes coming out of and going into the CEP have to be divided into the upper and lower data bus of the AT according to the address being even or uneven.

Figure 5-6 is a system memory map of the Evaluation Board. Figure 5-7 is a block diagram. The PAL device equations are given in the following pages.

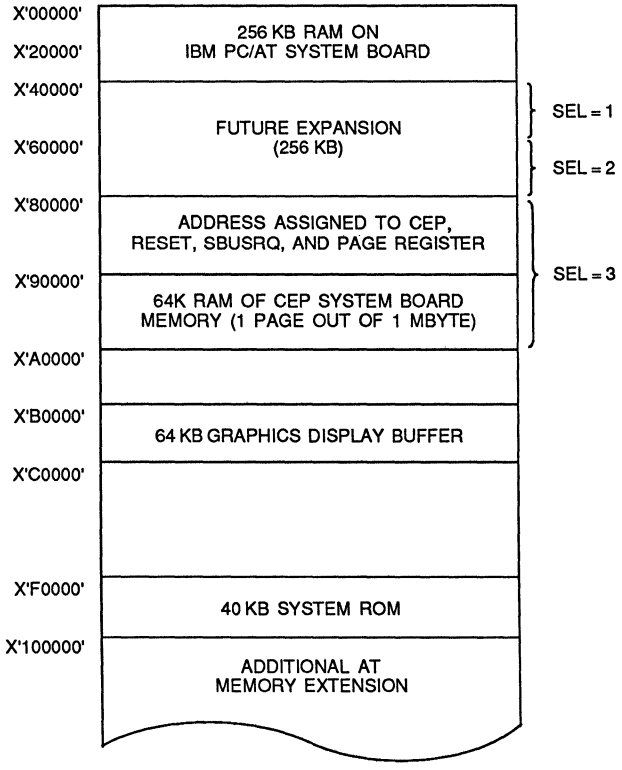


Figure 5-6 Evaluation Board System Memory Map

07666A 5-6

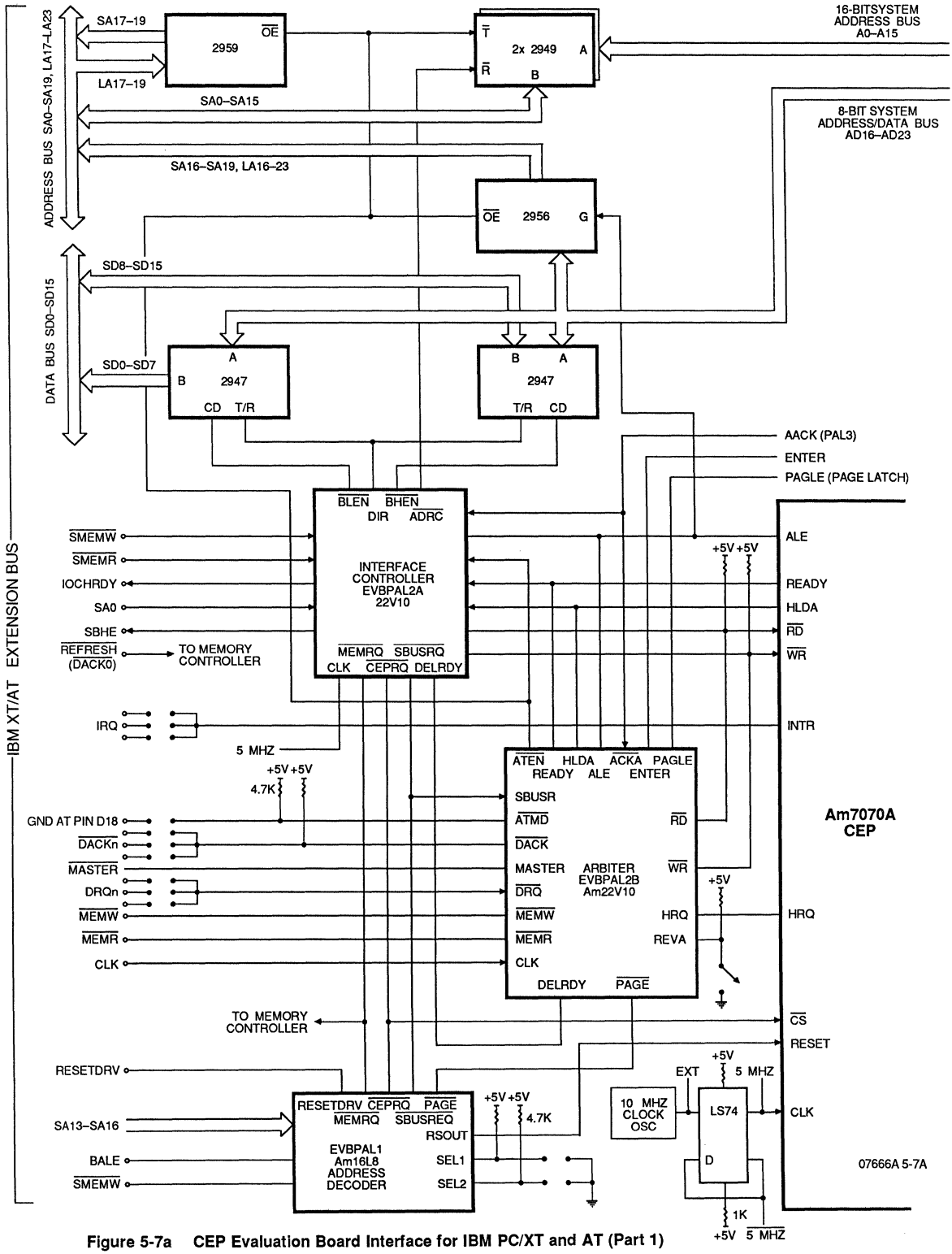


Figure 5-7a CEP Evaluation Board Interface for IBM PC/XT and AT (Part 1)

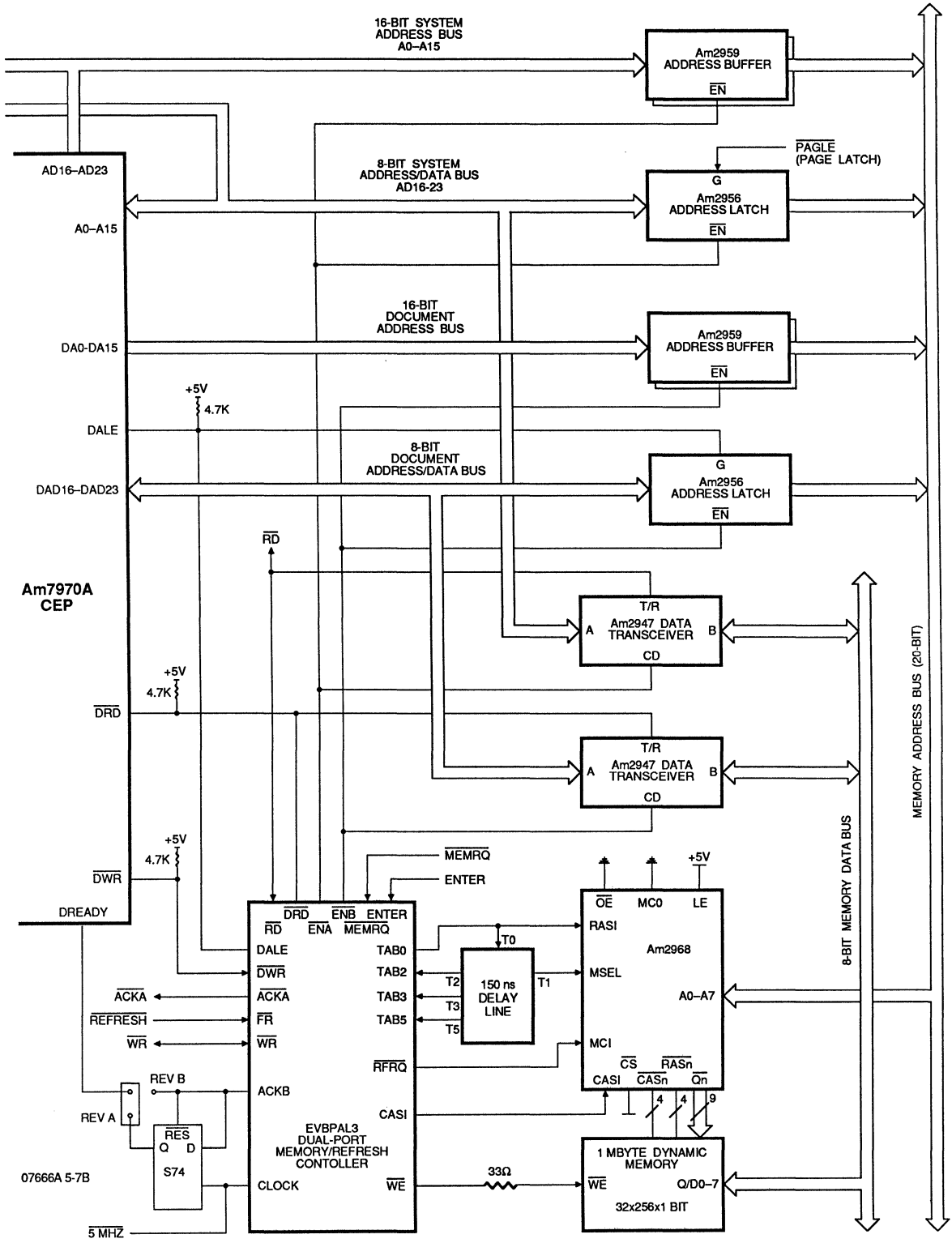


Figure 5-7b CEP Evaluation Board Interface for IBM PC/XT and AT (Part 2)

5.3.4 Evaluation Board PAL Device Equations

```
PARTNO          NONE;
NAME            EVBPAL1;
DATE            11-6-85;
REV                     1.1;
DESIGNER                Wolfgang Kemmler;
COMPANY         Advanced Micro Devices;
ASSEMBLY                CEP Evaluation Board;
LOCATION                U61;

/* Pal Assembler: CUPL (Assisted Technology)*/
/*********************************************************/
/*                                                       */
/*      Address Decoder for                              */
/*      Am7970 CEP Evaluation Board                      */
/*                                                       */
/*********************************************************/

/* Target Device Type : PAL16L8                          */

/* Outputs :            */

PIN 19  = !PAGE;    /* Latch enable foe Page Latch */
PIN 17  = !CEPRQ;   /* CEP Chip Select */
PIN 16  = !SBUSRQ; /* On Board System Bus Request */
PIN 15  = !ADDR;    /* Interm. Signal for Addr. Decoding */
PIN 14  = RSOUT;    /* RESET */
PIN 12  = !MEMRQ;   /* On Board Memory Request */


/* Inputs :              */
PIN [1..7] = [A13..19]; /* Addresses from Extension Bus */
PIN [8..9] = [S0..1];   /* Address range select inputs */
PIN 10     = GND;
PIN 11     = !SMEMW;    /* Write Signal from Ext. Bus */
PIN 13     = RSIN ;     /* RESET Input from Ext. Bus */
PIN 18     = BALE;      /* Address latch enable " "   */
PIN 20     = VCC;

/* Declarations and Intermediate Variable Definitions : */
FIELD   UADDR = [A17..19]; /* Upper Address Range */
FIELD   LADDR = [A13..15]; /* Lower Address Range */
FIELD   SEL   = [S0,S1];   /* Address range select field */

/* Address range definitions: */

/* Upper Address range : */
USEL3 = UADDR:[80000..9ffff];
USEL2 = UADDR:[60000..7ffff];
USEL1 = UADDR:[40000..6ffff];

/* Lower Address range: */
ADR0 = !A16&ADDR&LADDR:[0000..1fff]; /*CEP register access      */
ADR1 = !A16&ADDR&LADDR:[2000..3fff]; /*ask for S-Bus           */
ADR2 = !A16&ADDR&LADDR:[4000..5fff]; /*release S-Bus           */
ADR3 = !A16&ADDR&LADDR:[6000..7fff]; /*Pageaddr. into Pageregister */
ADR4 = !A16&ADDR&LADDR:[8000..9fff]; /*CEP hardware RESET aktiv    */
ADR5 = !A16&ADDR&LADDR:[a000..bfff]; /*CEP hardware RESET inaktiv  */
```

```
/*LOGIC EQUATIONS : */
ADDR =  SEL:3 & USEL3 & !BALE     /*SEL selects 3 allowed */
        # SEL:2 & USEL2 & !BALE   /*   "addressranges (blocks of*/
        # SEL:1 & USEL1 & !BALE ; /* "128kBytes )*/

MEMRQ = A16 & ADDR & !BALE ;            /*Access on board memory access*/

SBUSRQ = ADR1 & SMEMW           /*Request the S-Bus */
        # SBUSRQ & !(ADR2 & SMEMW) ;

PAGE =  ADR3 & SMEMW;           /*Load Page address into */
                                        /*Pageregister*/

RSOUT = ADR4 & SMEMW & RSIN             /*Software controlled */
        # RSOUT & !(ADR5 & SMEMW) ;     /* Hard Reset */

CEPRQ = ADR0 & !RSOUT ;         /*CEP Register Access */
                                /* (Memory mapped )*/

PARTNO          NONE;
NAME            EVBPAL2A;
DATE            11-6-85;
REV             1.2;
DESIGNER        Wolfgang Kemmler;
COMPANY         Advanced Micro Devices;
ASSEMBLY        CEP Evaluation Board;
LOCATION        U31;
/* Pal Assembler: CUPL (Assisted Technology)            */
/*******************************************************/
/*                                                     */
/*   IBM XT/AT Extension Bus Interface Controller      */
/*   for Am7970 CEP Evaluation Board                   */
/*                                                     */
/*******************************************************/

/* Target Device Type : AmPal 22V10                     */

/* Outputs:    */
PIN 23 = IOCHRDY; /* Open Coll. Extension Bus Signal */
PIN 22 = !RD;     /* CEP Signal */
PIN 21 = !WR;     /*    "       */
PIN 20 = !BLEN;   /* Data transceiver low enable */
PIN 19 = !BHEN;   /* Data transceiver high enable */
PIN 18 = !ADDRC;  /* Address receiver enable */
PIN 17 = DIR;     /* Direction controll of data transc. */
PIN 16 = SBHE;    /* Extension Bus Signal */
PIN 15 = DELRDY;  /* Signal helps generating two wait states */
                     /* for Rev.A CEP - IBM AT interface combination */
PIN 14 = IONRDY;  /* Interm. I/O Not Ready Signal */

/* Input: */
PIN 1 = CLK;      /* CEP clock (5 MHz) */
PIN 2 = !SBUSRQ;  /* On board system bus access request */
PIN 3 = !CEPRQ;   /* CEP register access request */
PIN 4 = !MEMRQ;   /* On board memory access request */
PIN 5 = !ATEN;    /* AT Mode enable */
PIN 6 = HLDA;     /* CEP signal */
PIN 7 = READY;    /* CEP signal */
PIN 8 = ACKA;     /* Acknowledge from on board memory */
PIN 9 = !SMEMW;   /* Extension bus signal */
PIN 10 = !SMEMR;  /*         "            */
PIN 11 = SA0;     /*         "            */
PIN 12 = GND;
PIN 13 = !PAGE;   /* Page latch access request */
PIN 24 = VCC;
```

```
/* Logic Equations :    */

WR.OE = !HLDA ; WR  = SMEMW & (CEPRQ # MEMRQ) ;

RD.OE = !HLDA ; RD  = SMEMR & (CEPRQ # MEMRQ) ;

BHEN =  ATEN & SA0 & (RD # WR) ;

BLEN =  ATEN & !SA0 & (RD # WR)
        # CEPRQ & !HLDA & (SMEMR # SMEMW)
        # MEMRQ & !HLDA & (SMEMR # SMEMW)
        # PAGE & !HLDA ;

IOCHRDY.OE = IONRDY; IOCHRDY = !IONRDY ;

IONRDY = HLDA & SBUSRQ
         # !HLDA & (SMEMR # SMEMW)
           & (MEMRQ & !ACKA # CEPRQ & !READY) ;

ADDRC = !HLDA & (CEPRQ # MEMRQ) ;

DIR = HLDA & WR # !HLDA & SMEMR ;

SBHE.OE = ATEN ; SBHE = !SA0 ;

DELRDY.D = RD # WR ;
DELRDY.AR = 'b'0 ; DELRDY.SP = 'b'0 ;



PARTNO            NONE;
NAME              EVBPAL2B;
DATE              1-13-85;
REV               1.3;
DESIGNER          Wolfgang Kemmler;
COMPANY           Advanced Micro Devices;
ASSEMBLY          CEP Evaluation Board;
LOCATION          U30;
/*Pal Assembler: CUPL (Assisted Technology)              */
/*******************************************************/
/*                                                     */
/*      IBM XT/AT Extension Bus Arbiter                */
/*      for the CEP Evaluation Board                   */
/*                                                     */
/*******************************************************/

/* Target Device Type : AmPal22V10                                */

/* Inputs :       */
PIN 1  = CLK;      /* Extension bus clock */
PIN 2  = !RD;      /* CEP signal */
PIN 3  = !WR;      /*    "       */
PIN 4  = HRQ;      /*    "       */
PIN 5  = ACKA;     /* On board memory acknowledge */
PIN 6  = !SBUSRQ;  /* Onboard system bus request */
PIN 7  = !DACK;    /* Extension bus arbitration signal */
PIN 8  = !PAGE;    /* Page Latch access request */
PIN 9  = !ATMD;    /* AT Mode Indicator */
PIN 10 = REVA;     /* CEP Revision A Indicator (adds 1 wait state)*/
PIN 11 = ALE;        /* CEP signal */
PIN 13 = DELRDY;   /* Delay Ready */
/* Outputs :      */
PIN 23 = !MASTER;  /* Extension bus arbitration signal */
```

```
PIN 22 = DREQ;      /*                "                  */
PIN 21 = !MEMW;     /* Extension bus signal */
PIN 20 = !MEMR;     /*          "          */
PIN 19 = READY;     /* CEP signal */
PIN 18 = HLDA;      /*     "      */
PIN 17 = !ATEN;     /* Enable master interface to AT ext. bus */
PIN 16 = ENTER;     /* Onboard memory access request */
PIN 15 = EARLY;     /* Intermediate signal for "ENTER" */
PIN 14 = PAGLE;     /* PAGE LATCH ENABLE signal */
/* Logic Equations :    */

ATEN.D  = MASTER & ATMD;
ATEN.AR = 'b'0 ; ATEN.SP = 'b'0 ;

MASTER.OE = HRQ & HLDA & ATMD;
MASTER.D  = HRQ & HLDA & ATMD;
MASTER.AR = 'b'0 ; MASTER.SP = 'b'0 ;

DREQ.D  = HRQ & ATMD & !DACK
          # DREQ & HRQ & ATMD ;

DREQ.AR = 'b'0 ; DREQ.SP = 'b'0 ;

MEMW.OE = ATEN ; MEMW = WR ;

MEMR.OE = ATEN ; MEMR = RD ;

HLDA = !ATMD & HRQ & !SBUSRQ
        # !ATMD & HLDA & HRQ
        # ATMD & DACK & DREQ
        # ATMD & HLDA & DREQ ;

EARLY = ALE # EARLY & !RD & !WR ;

ENTER = (EARLY & !ALE # RD # WR) & !ATMD & HLDA ;

PAGLE = ALE & HLDA # PAGE & !HLDA ;

READY.OE = HLDA ;
READY = HLDA & ATMD & (RD # WR) & (DELRDY & REVA # !REVA)
        # ACKA & !ATMD & (REVA & (RD # WR) # !REVA) ;



PARTNO            NONE;
NAME              EVBPAL3;
DATE              11-13-85;
REV               1.4;
DESIGNER          Wolfgang Kemmler;
COMPANY           Advanced Micro Devices;
ASSEMBLY          Am7970 CEP Evaluation Board;
LOCATION          U64;
/* Pal Assembler: CUPL (Assisted Technology)                */
/************************************************************/
/*                                                          */
/* Dual Port Dynamic Memory Access/Refresh Controller       */
/*      for Am7970 CEP Evaluation Board                     */
/*                                                          */
/************************************************************/
/* Target Device Type : PAL22V10                            */

/* Inputs :         */
PIN 1   = clock ;   /* 5MHz clock synchr. & inverted */
```

```
                    /* to CEP clock */
PIN 2    = tab3 ;   /* 60% Tab of 150ns delay line          */
PIN 3    = tab5 ;   /* 100%        "                        */
PIN 4    = dale ;   /* CEP signal document side             */
PIN 5    = !drd ;   /*    "                "                 */
PIN 6    = !dwr ;   /*    "            "                     */
PIN 7    = tab2 ;   /* 40% Tab of delay line                */
PIN 8    = !wr ;    /* CEP signal system side               */
PIN 9    = !rd ;    /*    "            "                    */
PIN 10   = !memrq ; /* On board memory request              */
PIN 11   = fr ;     /* REFRESH from IBM AT; DACK0 from IBM XT */
PIN 12   = GND ;
PIN 13   = enter ;  /* CEP system interface access request  */

/* Outputs :        */
PIN 23   = tab0 ;   /* Input of delay line                  */

PIN 22   = !rfrq ;  /* Interm. signal for refresh arb.      */
PIN 21   = !we ;    /* Write enable for on board memory     */
PIN 20   = endcyc;  /* Intermediate signal for document     */
                    /* side arbitration */
PIN 19   = fh;      /* Interm. signal for refresh arb.      */
PIN 18   = !enb ;   /* Enable document side to memory       */
PIN 17   = ackb ;   /* Acknowlege document side access      */
PIN 16   = acka ;   /*     "      system    "     "         */
PIN 15   = !ena ;   /* Enable system side to memory         */
PIN 14   = casi ;   /* CASI for 2968                        */

/* Logic Equations :    */
ena = enter & !rfrq & !enb & !(!fh & fr) & !tab3
      # (rd # wr) & !rfrq & !enb & memrq & !tab3
      # ena & (enter # rd # wr) & !rfrq ;

enb = (!dale & !endcyc # drd # dwr) & !(!fh & fr)
      & !enter & !rfrq & !ena & !tab3
      # enb & (!dale & !endcyc # drd # dwr) & !rfrq & !ena ;

endcyc = drd # dwr # endcyc & !dale ;

tab0 = ena # enb # rfrq & (!tab3 # !tab5) ;

acka.d = ena & !rfrq & !enb ;

ackb.d = enb & !rfrq & !ena ;

fh = fr & rfrq & tab0 # fh & fr ;

rfrq = fr & !fh & !ena & !enb & !tab3
      # rfrq & (tab0 # tab2) ;

we = ena & !enb & !rfrq & wr # enb & !ena & !rfrq & dwr ;

casi = tab2 & (we # ena & rd # enb & drd) ;

acka.ar = 'b'0 ;
acka.sp = 'b'0 ;
ackb.ar = 'b'0 ;
ackb.sp = 'b'0 ;
```

Appendix A
THROUGHPUT PERFORMANCE, 5 MHz CLOCK

The CEP throughput performance for 1-D, Group 3 and Group 4, compression and expansion, under various conditions is given in the following pages. The performance measurements were done with one wait state on both main memory and document store buses. Without the wait state, the performance can be improved. The figures in the "no wait" column in the tables are the calculated performance results without a wait state.

The throughput performance was measured using the following configurations:

1. Throughput performance of expansion for 200 ppi document. Picture data buffer is in the document store and coded data buffer in the main memory.

2. Throughput performance of compression for 200 ppi document. Picture data buffer is in the document store and coded data buffer in the main memory.

3. Throughput performance of compression for 200 ppi document. Both picture data buffer and coded data buffer are in the document store.

4. Throughput performance of expansion for 400 ppi document. Picture data buffer is in the document store and coded data buffer is in the main memory.

5. Throughput performance of compression for 400 ppi document. Picture data buffer is in the document store and coded data buffer is in main memory.

TEST #1

Test Conditions

1. Compressed data buffer in main memory
2. Picture data buffer in document store memory
3. One wait state on both main memory and document store buses.
4. The data in the "no wait" column is the estimated throughput without the wait state.
5. Test documents are CCITT standard test documents
6. Resolution is 200 pixels per inch

A. 1-D Expansion

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	5.89	7.00
2	6.96	8.56
3	5.08	5.93
4	3.24	3.58
5	4.78	5.52
6	6.17	7.50
7	5.12	6.02
8	6.42	7.81

B. Group 3 Expansion

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	3.91	4.88
2	3.44	4.16
3	3.00	3.55
4	1.91	2.14
5	2.85	3.36
6	3.35	4.06
7	2.02	2.27
8	2.75	3.21

C. Group 4 Expansion

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	3.00	3.54
2	2.94	3.45
3	2.09	2.33
4	1.76	1.93
5	2.02	2.25
6	2.62	3.01
7	1.85	2.04
8	2.22	2.49

TEST #2

Test Conditions:

1. Compressed data buffer in main memory
2. Picture data buffer in document store memory
3. One wait state on both main memory and document store buses.
4. The data in the "no wait" column is the estimated throughput without the wait state.
5. Test documents are CCITT standard test documents
6. Resolution is 200 pixels per inch

A. 1-D Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	6.22	7.36
2	6.75	8.11
3	5.37	6.21
4	3.89	4.31
5	5.16	5.92
6	5.98	7.04
7	4.07	4.53
8	5.90	6.92

B. Group 3 Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	4.08	4.96
2	4.09	4.98
3	2.96	3.40
4	1.96	2.14
5	2.78	3.16
6	3.54	4.19
7	2.03	2.23
8	3.34	3.91

C. Group 4 Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	3.74	4.60
2	3.67	4.55
3	2.61	3.00
4	1.69	1.84
5	2.44	2.77
6	3.17	3.76
7	1.75	1.93
8	2.96	3.48

TEST #3

Test Conditions

1. Compressed data buffer in document store memory
2. Picture data buffer in document store memory
3. One wait state on document store bus.
4. The data in the "no wait" column is the estimated throughput without the wait state.
5. Test documents are CCITT standard test documents
6. Resolution is 200 pixels per inch

A. 1-D Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	6.25	7.40
4	3.91	4.33
7	4.09	4.56

B. Group 3 Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	4.08	4.97
4	1.97	2.16
7	2.04	2.23

C. Group 4 Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
1	3.75	4.61
4	1.69	1.84
7	1.76	1.93

TEST #4

Test Conditions:

1. Compressed data buffer in main memory
2. Picture data buffer in document store memory
3. One wait state on both main memory and document store buses.
4. The data in the "no wait" column is the estimated throughput without the wait state.
5. Test documents are CCITT standard test documents
6. Resolution is 400 pixels per inch

Group 4 Expansion

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
4	2.84	3.31
7	2.86	3.34

TEST #5

Test Conditions:

1. Compressed data buffer in main memory
2. Picture data buffer in document store memory
3. One wait state on both main memory and document store buses.
4. The data in the "no wait" column is the estimated throughput without the wait state.
5. Test documents are CCITT standard test documents
6. Resolution is 400 pixels per inch

Group 4 Compression

Doc. No.	Throughput Performance (in mbit/sec.)	
	One Wait	No Wait
4	2.73	3.16
7	2.71	3.12

Appendix B
IMAGE FILE ANALYSIS PROGRAM LISTING
(Deyoung Hong—10/17/84)

```
*****************************************************************************/

#include "stdio.h"
#define CLKRATE 5.0e6          /* 5 MHz clock rate */
#define HTIME   5              /* CPU arbitration time */
#define EOL     12+1           /* End Of Line + 1 bit */
#define RTC     5*EOL          /* Return To Control */
#define EOFB    2*EOL          /* End of Facsimile Block */
#define PASSMODE 4             /* Pass Mode Coding */
#define VRL0    1              /* Vertical 0 Mode Coding */
#define VRL1    3              /* Vertical 1 Mode Coding */
#define VRL2    6              /* Vertical 2 Mode Coding */
#define VRL3    7              /* Vertical 3 Mode Coding */
#define CPMEOF  0x1A           /* end of file */


/* Table of number of color changes contains in each byte (value 0 to 255) */
char ncc[256] = {0,1,2,1,2,3,2,1,2,3,4,3,2,3,2,1,2,3,4,3,4,5,4,3,2,3,4,3,2,3,
                 2,1,2,3,4,3,4,5,4,3,4,5,6,5,4,5,4,3,2,3,4,3,4,5,4,3,2,3,4,3,
                 2,3,2,1,2,3,4,3,4,5,4,3,4,5,6,5,4,5,4,3,4,5,6,5,6,7,6,5,4,5,
                 6,5,4,5,4,3,2,3,4,3,4,5,4,3,4,5,6,5,4,5,4,3,2,3,4,3,4,5,4,3,
                 2,3,4,3,2,3,2,1,1,2,3,2,3,4,3,2,3,4,5,4,3,4,3,2,3,4,5,4,5,6,
                 5,4,3,4,5,4,3,4,3,2,3,4,5,4,5,6,5,4,5,6,7,6,5,6,5,4,3,4,5,4,
                 5,6,5,4,3,4,5,4,3,4,3,2,1,2,3,2,3,4,3,2,3,4,5,4,3,4,3,2,3,4,
                 5,4,5,6,5,4,3,4,5,4,3,4,3,2,1,2,3,2,3,4,3,2,3,4,5,4,3,4,3,2,
                 1,2,3,2,3,4,3,2,1,2,3,2,1,2,1,0};

/* Length of the compressed codewords in the Modified Huffman Code Tables */
char wtermc[64] = {8,6,4,4,4,4,4,4,5,5,5,5,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,
                   8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
                   8,8,8,8,8,8};
char wmakec[41] = {8,5,5,6,7,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,6,9,11,
                   11,11,12,12,12,12,12,12,12,12,12};
char btermc[64] = {8,3,2,2,3,4,4,5,6,6,7,7,7,8,8,9,10,10,10,11,11,11,11,11,11,
                   11,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
                   12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12};
char bmakec[41] = {8,10,12,12,12,12,12,12,13,13,13,13,13,13,13,13,13,13,13,13,13,
                   13,13,13,13,13,13,13,13,11,11,11,12,12,12,12,12,12,12,12,
                   12,12};

/* files are used as global variables */
char fnamin[20];               /* image file name */
char fnamout[20];              /* output file to store results */
int fdin;                      /* input file descriptor */
FILE *fdout;                   /* ouput file descriptor */


main()
{

float comp1d();                /* floating point function comp1d() */
float expa1d();                /* floating point function expa1d() */
int ndwait;                    /* # of wait cycles introduced by doc memory */
int ncwait;                    /* # of wait cycles introduced by main mem */
int linelen;                   /* number of pixels in one scan line */
int pagelen;                   /* number of lines in a page */
float fcc[8];                  /* fraction of color-change count */
float cpr1d;                   /* compression ratio for one dimension */
float cpth1d;                  /* compression throughput for one dimension */
float exth1d;                  /* expander throughput for one dimension */
float cpr2d;                   /* compression ratio for two dimensions */
```

```
float cpth2d;          /* compression throughput for two dimensions */
float exth2d;          /* expander throughput for two dimensions */
int dim;               /* coding dimension */
int kfact;             /* k-factor of dimension */
int k;

printf("\n*************** IMAGE FILE ANALYSIS PROGRAM ***************\n\n");
printf("> Name of image file to be analyzed?  ");
scanf("%s",fnamin);
if ((fdin = open(fnamin,BREAD)) < 0)          /* open file for binary read */
{   printf("\7@ERROR STATUS: Can't open file %s\n",fnamin);
    exit();
}
printf("> Number of wait cycles introduced by the document memory?  ");
scanf("%d",&ndwait);
printf("> Number of wait cycles introduced by main memory?  ");
scanf("%d",&ncwait);
printf("> Maximum number of pels per line?  ");
scanf("%d",&linelen);
printf("> Maximum number of lines per page?  ");
scanf("%d",&pagelen);
printf("> Dimension of coding (1 or 2)?  ");
scanf("%d",&dim);
kfact = -1;
if (dim == 2)
{   printf("> Enter parameter k (0 for infinity):  ");
    scanf("%d",&kfact);
}
printf("> Name of file to store output results?  ");
scanf("%s",fnamout);
if ((fdout = fopen(fnamout,"w")) == 0)        /* open file for ASCII write */
{   printf("\7@ERROR STATUS: Can't open file %s\n",fnamout);
    exit();
}
if (kfact == 0) printf("\n%% GROUP IV CODING IS ASSUMED %%\n\n");
else printf("\n%% GROUP III CODING IS ASSUMED %%\n\n");
if (kfact < 0)                              /* case of one dimensional coding */
{   scan1d(linelen,pagelen,fcc,&cpr1d);
    printf("Fractional number of bytes which contain ");
    printf("0 through 7 color changes:\n");
    for (k = 0; k <= 7; k++)
        if (fcc[k] > 0.000) printf("\tf%0d = %0.3f\n",k,fcc[k]);
    printf("\n1-D Compression Ratio = %0.3f\n",cpr1d);
    cpth1d = comp1d(fcc,ncwait,ndwait,1.0/cpr1d) / 1.0e6;
    printf("1-D Compressor Throughput (in MBPs) = %0.3f\n",cpth1d);
    exth1d = expa1d(fcc,ndwait,1.0/cpr1d) / 1.0e6;
    printf("1-D Expander Throughput (in MBPs) = %0.3f\n",exth1d);
    presult(linelen,pagelen,ndwait,ncwait,kfact,fcc,cpr1d,cpth1d,exth1d);
}
else                                        /* case of two dimensional coding */
{   scan2d(linelen,pagelen,kfact,fcc,&cpr2d);
    printf("Fractional number of bytes which contain ");
    printf("0 through 7 color changes:\n");
    for (k = 0; k <= 7; k++)
        if (fcc[k] > 0.000) printf("\tf%0d = %0.3f\n",k,fcc[k]);
    printf("\n2-D Compression Ratio = %0.3f\n",cpr2d);
    cpth2d = comp1d(fcc,ncwait,ndwait,1.0/cpr2d) / 4.0e6;
    printf("2-D Compressor Throughput (in MBPs) = %0.3f\n",cpth2d);
    exth2d = expa1d(fcc,ndwait,1.0/cpr2d) / 4.0e6;
    printf("2-D Expander Throughput (in MBPs) = %0.3f\n",exth2d);
    presult(linelen,pagelen,ndwait,ncwait,kfact,fcc,cpr2d,cpth2d,exth2d);
}
close(fdin);                                              /* close files */
fputc(CPMEOF,fdout);
fclose(fdout);
```

```
printf("\n*****************************************************************\n");

}

/*-------------------------------------------------------------------------
Function huffcode takes as input the number of white or black pixels and
returns the number of bits correspond to the Huffman code table.
-------------------------------------------------------------------------*/

int huffcode(npixels,bw)
int npixels, bw;
{
    switch (bw)
    {   case 0: if (npixels < 64) return(wtermc[npixels]);
                return(wmakec[npixels/64] + wtermc[npixels%64]);
        case 1: if (npixels < 64) return(btermc[npixels]);
                return(bmakec[npixels/64] + btermc[npixels%64]);
        default: return(0);
    }
}

/*-------------------------------------------------------------------------
Function codeld scans a line of code and return the number of count of the
compression data of one dimensional coding.
-------------------------------------------------------------------------*/

int codeld(line)
char *line;
{
    unsigned char flag;         /* flag signaling change of pels */
    int pixel[3];               /* pixel count - 0 white - 1 black */
    int compcount;              /* compression data count in a line */
    int j;

    flag = 2;
    compcount = 0;
    for (j = 0; *(line+j) != '\0'; j++)       /* scan through string of pels */
    {   if (flag != (*(line+j) - '0'))                    /* if pels change */
        {   compcount = compcount + huffcode(pixel[flag],flag);
            flag = *(line+j) - '0';                          /* reset flag */
            pixel[flag] = 1;                              /* count first pixel */
        }
        else ++(pixel[flag]);                     /* otherwise upcount pels */
    }
    compcount = compcount + huffcode(pixel[flag],flag);
    return(compcount);                          /* return compress data count */
}


/*-------------------------------------------------------------------------
    Function code2d scans a line of code and its reference line, then return the
    number of count of the compression data of two dimensional coding.
-------------------------------------------------------------------------*/

int code2d(codeln,refln)
char *codeln, *refln;
{
     int a0, a1, a2, b1, b2;     /* reference points of lines */
     int compcount;              /* compression data count */
     int j;

compcount = 0;
for (a0 = 1; *(codeln+a0) == *(codeln+a0-1); a0++);     /* detect first a0 */
do                                                          /* repeat loop */
{    if (*(codeln+a0) == '\0')              /* if a0 is at end of line then */
```

```
    {   a1 = a2 = a0;                   /* set all references at end of line */
        b1 = b2 = a0;
    }
    else                                /* otherwise detect a1, b1, and b2 */
    {   for (a1 = a0+1; *(codeln+a1) == *(codeln+a0-1); a1++);
        for (b1 = a0+1; *(refln+b1) == *(refln+b1-1); b1++);
        if (*(refln+b1) == *(codeln+a0))
        for (b1 = b1+1; *(refln+b1) == *(refln+b1-1); b1++);
        if (*(refln+b1) == '\0') b2 = b1;
        else for (b2 = b1+1; *(refln+b2) == *(refln+b2-1); b2++);
    }
    if (b2 < a1)                        /* if b2 is at left of a1 then */
    {   compcount += PASSMODE;                  /* do pass mode coding */
        a0 = b2;
    }
    else            /* else check for vertical or horizontal mode coding */
    {   switch (a1-b1)
        {   case 0:                     /* case V(0) - vertical mode */
                compcount += VRL0;
                a0 = a1;
                break;
            case 1: case -1:                    /* case VR(1) or VL(1) */
                compcount += VRL1;
                a0 = a1;
                break;
            case 2: case -2:                    /* case VR(2) or VL(2) */
                compcount += VRL2;
                a0 = a1;
                break;
            case 3: case -3:                    /* case VR(3) or VL(3) */
                compcount += VRL3;
                a0 = a1;
                break;
            default:                        /* horizontal mode coding */
                if (*(codeln+a1) == '\0') a2 = a1;          /* detect a2 */
                else for (a2 = a1+1; *(codeln+a2) == *(codeln+a2-1); a2++);
                compcount = compcount + 3 + huffcode(a1-a0,*(codeln+a0));
                compcount = compcount + huffcode(a2-a1,*(codeln+a1));
                a0 = a2;
                break;
        }
    } while (*(codeln+a0) != '\0');             /* until end of line */
    return(compcount);                  /* return compresiion data count */
}


/*-------------------------------------------------------------------------
    Function scan1d reads data from the specified document and computes the
    one dimensional compression ratio and the fractional number of bytes which
    contain number of k-color changes.
-------------------------------------------------------------------------*/

scan1d(linelen,pagelen,fcc,cpr1d)
int linelen, pagelen;
float fcc[8];
float *cpr1d;
{
    extern long ltell();
    int nbytes;                 /* number of bytes in one scan line */
    int nbread;                 /* number of bytes read each time */
    unsigned char *buffer;      /* line containing data from file */
    unsigned char *pxline;      /* bit representation of data line */
    unsigned char flag;         /* flag containing current bit */
    long compdata;              /* total compression data count */
```

```
    long cc[8];                 /* color changes count */
    int pixel[3];               /* black and white pixels count */
    int j, k;

    compdata = 0;                                  /* initialize variables */
    for (k = 0; k <= 7; k++) cc[k] = 0;
    nbytes = linelen/8;
    pxline = alloc(linelen + 1);                /* allocate memory space */
    buffer = alloc(nbytes + 1);
    printf("File reading: ");
    for (j = 0; j < pagelen; j++)                 /* scan the whole page */

    { if ((nbread = read(fdin,buffer,nbytes)) <= 0) break; /* read a line */
      for (k = 0; k < nbread; k++)
    { ++(cc[ncc[*(buffer+k)]]);              /* count color-changed bytes */
      sprintf(pxline+k*8,"%08b",*(buffer+k));          /* bytes to pels */
    }
    compdata = compdata + code1d(pxline) + EOL;     /* 1-D coding with EOL */
    if ((j%75) == 0) putchar('\n');
    putchar('.');
    }
    compdata = compdata + RTC;                      /* RTC at end of file */
    putchar('\n');
    putchar('\n');
    if (nbread < 0)                            /* case of error during read */
    { printf("\7@WARNING: Error during read file %s\n");
      printf("         Data Analysis may be incorrect.\n");
    }
    for (k = 0; k <= 7; k++) fcc[k] = cc[k] * 1.0 / ltell(fdin);
    *cpr1d = 8.0 * ltell(fdin) / compdata;      /* 1-D compression ratio */
    free(pxline);                                /* return to free space */
    free(buffer);
  }


/*-------------------------------------------------------------------------------
    Function scan2d reads data from the specified document and computes the
    two dimensional compression ratio and the fractional number of bytes which
    contain number of k-color changes.
-------------------------------------------------------------------------------*/

scan2d(linelen,pagelen,kfact,fcc,cpr2d)
int linelen, pagelen, kfact;
float fcc[8];
float *cpr2d;
{
    extern long ltell();
    int nbytes;                 /* number of bytes in one scan line */
    int nbread;                 /* number of bytes read each time */
    unsigned char *buffer;      /* line containing data from file */
    unsigned char *codeln;      /* the coding line of pixels */
    unsigned char *refln;       /* the reference line of coding line */
    unsigned char flag;         /* flag containing current bit */
    long compdata;              /* total compression data count */
    long cc[8];                 /* color changes count */
    int pixel[3];               /* black and white pixels count */
    int i, j, k;

    k = 1;                                         /* initialize variables */
    compdata = 0;
    for (j = 0; j <= 7; j++) cc[j] = 0;
    nbytes = linelen/8;
    codeln = alloc(linelen + 1);                   /* allocate memory space */
    refln = alloc(linelen + 1);
    buffer = alloc(nbytes + 1);
```

```
    printf("File reading: ");
    for (j = 0; j < linelen; j++) *(refln+j) = '0'   /* imaginary white line */
    *(refln+j) = '\0';
    for (i = 0; i < pagelen; i++)                    /* scan the whole page */

    {   if ((nbread = read(fdin,buffer,nbytes)) <= 0) break;  /* read a line */
        for (j = 0; j < nbread; j++)
            {   ++(cc[ncc[*(buffer+j)]]);      /* count color-changed bytes */
                sprintf(codeln+j*8,"%08b",*(buffer+j));   /* bytes to pels */
            }
        if (kfact == 0)                              /* if k is infinity then */
            compdata = compdata + code2d(codeln,refln);   /* do 2-D coding */
        else if ((k %= kfact) == 1)          /* if first line of k lines then */
            compdata = compdata + code1d(codeln) + EOL;   /* do 1-D coding */
        else                                  /* else do 2-D coding with EOL */
            compdata = compdata + code2d(codeln,refln) + EOL;
        ++k;
        strcpy(refln,codeln);                 /* assign the reference line */
        if ((i%75) == 0) putchar('\n');
        putchar('.');
    }
    if (kfact == 0)                       /* if k is infinity then EOFB code */
        compdata = compdata + EOFB;
    else compdata = compdata + RTC;             /* else RTC at end of file */
    putchar('\n');
    putchar('\n');
    if (nbread < 0)                           /* case of error during read */
    {   printf("\7@WARNING: Error during read file %s\n");
        printf("        Data Analysis may be incorrect.\n");
    }
    for (j = 0; j <= 7; j++) fcc[j] = cc[j] * 1.0 / ltell(fdin);
    *cpr2d = 8.0 * ltell(fdin) / compdata;        /* 2-D compression ratio */
    free(codeln);                                  /* return to free space */
    free(refln);
    free(buffer);
}


/*-------------------------------------------------------------------------
  Function comp1d  computes  the  one dimensional compressor throughput of a
  document.
-------------------------------------------------------------------------*/

float comp1d(fcc,ncwait,ndwait,qeol)
float fcc[8];
int ncwait, ndwait;
float qeol;
{
    float fetchtime;            /* time spent fetching data from DS */
    int dcc[8];                /* # of cycles required of each color change */
    int i;

    dcc[0] = 0;
    dcc[1] = 6;
    for (i = 2; i <= 7; i++) dcc[i] = dcc[i-1] + 3;
    fetchtime = 0.0;
    for (i = 0; i <= 7; i++)
        fetchtime = fetchtime + (fcc[i] * (3.0 + ndwait + dcc[i]));
    return((CLKRATE * 8.0) / (qeol * (3.0 + ncwait + HTIME) + fetchtime));
}


/*-------------------------------------------------------------------------
  Function  exp1d  computes  the  one dimensional expander throughput  of a
  document.
```

```
-------------------------------------------------------------------------------*/

float expa1d(fcc,ndwait,qeol)
float fcc[8];
int ndwait;
float qeol;
{
  float fetchtime;              /* time spent fetching data from DS */
  int dce[8];                   /* # of cycles required of each color change */
  int i;

  for (i = 0; i <= 7; i++) dce[i] = i * 2;
  fetchtime = 0.0;
  for (i = 0; i <= 7; i++)
     fetchtime = fetchtime + (fcc[i] * (3.0 + ndwait + dce[i]));
  return((CLKRATE * 8) / (qeol * (3.0 + ndwait + HTIME) + fetchtime));
}


/*-------------------------------------------------------------------------------
  Function presult writes the output results of the analysed image file to the
  ouput file.
-------------------------------------------------------------------------------*/

presult(linelen,pagelen,ndwait,ncwait,kfact,fcc,cpr,cpth,exth)
int linelen, pagelen, ndwait, ncwait, kfact;
float fcc[8];
float cpr, cpth, exth;
{
     int k;

     fprintf(fdout,"**************** IMAGE ANALYSIS PROGRAM EXECUTION REPORT ");
     fprintf(fdout,"****************\n\n");
     fprintf(fdout,"  Image file analysed:  %s\n",fnamin);
     fprintf(fdout,"  Number of wait cycles introduced by the document memory:");
     fprintf(fdout,"  %d\n",ndwait);
     fprintf(fdout,"  Number of wait cycles introduced by the main memory:");
     fprintf(fdout,"  %d\n",ncwait);
     fprintf(fdout,"  Horizontal resolution:  %d pels/line\n",linelen);
     fprintf(fdout,"  Vertical resolution:  %d lines/page\n\n",pagelen);
     if (kfact < 0)
     {   fprintf(fdout,"%% ONE DIMENSIONAL CODING -- ");
         fprintf(fdout,"GROUP III CODING IS ASSUMED %%\n\n");
     }
     else if (kfact == 0)
     {   fprintf(fdout,"%% TWO DIMENSIONAL CODING (K = INFINITY) -- ");
         fprintf(fdout,"GROUP IV CODING IS ASSUMED %%\n\n");
     }
     else
     {   fprintf(fdout,"%% TWO DIMENSIONAL CODING (K = %d) -- ",kfact);
         fprintf(fdout,"GROUP III CODING IS ASSUMED %%\n\n");
     }
         fprintf(fdout,"  Fractional number of bytes which contain ");
         fprintf(fdout,"0 through 7 color changes:\n");

     for (k = 0; k <= 7; k++)
         if (fcc[k] > 0.000) fprintf(fdout,"\tf%0d = %0.3f\n",k,fcc[k]);
     fprintf(fdout,"\n  Compression Ratio = %0.3f\n",cpr);
     fprintf(fdout,"  Compressor Throughput (in MBPs) = %0.3f\n",cpth);
     fprintf(fdout,"  Expander Throughput (in MBPs) = %0.3f\n\n",exth);
     for (k = 0; k < 73; k++) fputc('*',fdout);
     fputc('\n',fdout);
}

/*******************************************************************************/
```

Appendix C
GLOSSARY

AEOL	Automatic End-of-Line
ALE	Address Latch Enable
BBC	Byte Boundaries Control bit
CBY	Compressor Busy bit
CDAHR	Compressor Destination Address Holding Register
CDCAR	Compressor Destination Current Address Register
CDCHR	Compressor Destination Count Holding Register
CDC	Compressor Destination Control bit
CDF	Compressed Data Format Control Field
CDLSR	Compressor Destination Line Start Address Register
CDO	Compressor Destination Overflow bit
CDWCR	Compressor Destination Working Count Register
CED	Called station identification (2100 hertz)
CER	Compressor Express Register
Changing element	An element whose color (black or white) is different from that of the previous element along the same scan line. It is the first element of a code word.
CIC	Compressor Illegal Command bit
CIE	Compressor Interrupt Enable bit
CKR	Compressor K-Register
CMCR	Compressor Master Control Register
CNG	Calling tone (1100 hertz on for 0.5 sec. off for 3 sec.)
COA	Compressor Busy and New Operation Attempted bit
Code word	A run length of either all white or all black elements
CPR	Compressor Parameter Register
CPWR	Compressor Page Width Register
CRCR	Compressor Restart Control Register
CSA	Compressor Source Attribute bit
CSAHR	Compressor Source Address Holding Register
CS	Chip Select
CSC	Compressor Source Control bit
CSCAR	Compressor Source Current Address Register
CSCHR	Compressor Source Count Holding Register
CSDN	Packet-switched data network
CSLSR	Compressor Source Line Start Address Register
CSO	Compressor Source Overflow bit
CSR	Compressor Status Register
CSWCR	Compressor Source Working Count Register
CWR	Compressor Wraparound Register
DAC	Destination Address Control bit
DALE	Destination Store Address Latch Enable
DC	Destination Control bit
DCC	Destination Count Control bit
DER	Data Error bit
DFC	Data Format Control bits
DLC	Destination Line Start Address Control bit
DO	Destination Overflow bit
DRD	Document Store Read
DREADY	Document Store Ready
Document	One page full of data (8 1/2 x 11) (ISO A4 document)
DWR	Document Store Write
EBY	Expander Busy bit

ECD	Extension Code Detected bit
EDAHR	Expander Destination Address Holding Register
EDC	Expander Destination Control bit
EDCAR	Expander Destination Current Address Register
EDCHR	Expander Destination Count Holding Register
EDLSR	Expander Destination Line Start Address Register
EDO	Expander Destination Overflow bit
EDWCR	Expander Destination Working Count Register
EIC	Expander Illegal Command bit
EIE	Expander Interrupt Enable bit
EKR	Expander K-Register
EMCR	Expander Master Control Register
EOA	Expander Busy and New Operation Attempted bit
EOL	Code word following each line of data in Group 3. In addition it occurs prior to the first data line of a page. The end of a document transmission is marked by six consecutive EOLs in Group 3. Group 4 document ends with two EOLs. Format of EOL: 000000000001
EOL	End-of-Line
EOP	End-of-Page in Group 4 (two EOLs)
EPR	Expander Parameter Register
EPWR	Expander Page Width Register
ERCR	Expander Restart Control Register
ESA	Expander Source Attribute bit
ESAHR	Expander Source Address Holding Register
ESC	Expander Source Control bit
ESCAR	Expander Source Current Address Register
ESCHR	Expander Source Count Holding Register
ESLSR	Expander Source Line Start Address Register
ESO	Expander Source Overflow bit
ESR	Expander Status Register
ESWCR	Expander Source Working Count Register
EWR	Expander Wraparound Register
EXT	Extension bits
Fill	Variable length string of 0's. Fill may be inserted between a line of data and an EOL but not within a line of data. Fill is added to ensure that the transmission time of the total coded scan line is not less than the minimum. The standard minimum is 20 milliseconds with 10, 5, and 40 millisecond options.
G-Parameter	Granularity Parameter
Group 1	Facsimile transmission specification T.2 of CCITT. Six minutes to Transmit one ISO A4 document over a telephone-type circuit (no bandwidth compression).
Group 2	Facsimile transmission specification T.3. Three minutes to transmit one ISO A4 document (uses bandwidth compression).
Group 3	Facsimile transmission specification based on 200 pels per inch resolution. One minute to transmit one document (uses data compression and may use bandwidth compression). Allows Teletex and mixed reception. Recommendation T.4 of CCITT.
Group 4	Facsimile transmission based on 200 (Class 1), 300 (Class 2 and 3), and 400 pels (optional) per inch resolution. Allows Teletex and mixed transmission and reception. Recommendation T.6 of CCITT.
HLDA	Hold Acknowledge
HRQ	Hold Request
Horizontal Mode	Coding mode used in two-dimensional coding when the changing element is more than three elements away from the changing element on the reference line above. The code consists of: $001 + M(a_0a_1) + M(a_1a_2)$.
IC	Illegal Command bit
IE	Interrupt Enable bit
INTR	Interrupt Request
ISDN	Integrated services digital network

ISO	International Organization for Standardization
ISO A4 paper	215 mm wide.
LMGR	Left Margin Register
LPI	Line Processing Incomplete bit
MC	Master Control bits
Mixed	Combination of Image data and Text in one document
MSR	Master Status Register
NGC	Negative Compression bit
OC	Operation Control bits
OSI	Open Systems Interconnection
PDN	Public Data Network
PIS	Procedure interrupt signal
PSDN	Packet-switched data network
PSTN	Public-switched telephone network
Pass mode	coding procedure in two-dimensional coding. Pass mode is identified when B_2 (on reference line) is to the left of A_1 on the coding line.
Pixel, Pel	Picture element
RD	Read
READY	Ready
RESET	Reset
RMGR	Right Margin Register
RTC	Return to Control. Six consecutive EOLs sent at the end of a document transmission.
RTC	Return-to-Control code (six EOLs)
Run length	Number of identical white or black picture elements in sequence
SA	Source Attribute bit
SAC	Source Address Control bit
SC	Source Control bit
SCC	Source Count Control bit
SLC	Source Line Start Address Control Bit
SO	Source Overflow bit
Standard	3.85 lines/mm $\pm$ 1%
Resolution	Optional resolution - 7.7 lines/mm $\pm$ 1%)
TDC	Two-Dimensional Restart
TFLR	Time Fill Register
TMGR	Top Margin Register
Total coded	Sum of data bits plus any required scan line fill bits plus EOL bits. For two-dimensional coding, same as above plus a tag bit.
Two-dimen-sional Coding	Line by line coding method in which sional coding the position of each changing picture element on the current or coding line is coded with respect to the position of a corresponding reference element located on either the coding line or the reference line immediately above it.
Vertical Mode	Coding procedure in two-dimensional coding. Identified when A_1 on the coding line is less than four elements away from the changing element on the reference line.
WPI	Wraparound Incomplete bit
WR	Write
WR	Wraparound Register
WRC	Wraparound Restart

Appendix D

7970A DIFFERENCES RELATIVE TO 7970 REVISION A/A'

Am7970 Rev A/A' was manufactured during 1985. Am7970A is to be introduced in July, 1986. The following recaps the differences.

Recap:

Defintion changed in: CPR, EPR, CER, and MSR

New Registers added: CKPR, EKPR, CFWR, and EFWR

Summary (refer to Chapter 2 for details):

1. The CEP can be identified as either 7970A or 7970 Rev. A/A' by reading the version I.D. bit (ID), bit 5 of the Master Status Register (MSR).

ID = 0 specifies 7970 Rev. A/A'
ID = 1 specifies 7970A (7970 Revision B)

2. Group 3 error recovery during expansion in 2D mode (EOL = 0, MC = 10)

Both the 7970 and 7970A encode using the K Parameter and for Group 3 setting the tag bit.

For Group 3 expansion, the 7970A processes each line as either 1D or 2D according to the value of the tag bit immediately following the EOL code terminating the previous line. If the tag bit is 1, the next line is decoded as a 1D line. If the tag bit is 0, the next line is decoded as a 2D line. The K Parameter is not used. However, 7970 Rev. A' does not use the tag bit for expansion in Group 3. It expands using the K Parameter.

If DER (Data Error) is set during this mode, the expander stops but processing is resumable by the CPU. When processing resumes, the expander will process the next line as 1D or 2D depending on the tag bit. If the next line has unrecognizable code, the DER bit is set again. Multi-line processing will continue only after a one-dimensional line with no errors has been found and decoded. Halting after each DER allows the CPU to count the number of lines lost when an error occurs.

3. CPR change — The K Parameter field in 7970 Rev. A' has been removed from the Compressor Parameter Register in 7970A. Bits 0, 1, and 2 are the Line Termination field (LT). It specifies the number of bits (0 to 7) of terminating image to be appended to each image line following the last full byte of data. The terminating bits all have the same value as the last bit (bit 7) of the last full byte on each line. Each line of encoded data includes the line termination bits.

4. EPR change — The K Parameter field has been removed from the Expander Parameter Register in 7970A. Bits 0, 1, and 2 are now reserved bits.

5. Group 4 expansion (EOL = 1, MC = 10) of coded data from lines not terminating with a full byte in 7970A is transparent to the software driver.

When the expanded data length reaches the line length specified in the Expander Page Width Register (EPWR), the expander resets an internal shift register and thus clips off any image bits which may comprise an incomplete byte at the end of the image line. This allows coded data for page lengths that end with a single color incomplete byte of image to be expanded into pages that terminate with the last byte of image a full byte. The incomplete byte at the end of each line is deleted. Otherwise, the last incomplete byte would prevent successful expansion.

In the 7970 Rev. A', when the specified line length is reached, the expander places any remaining incomplete byte at the beginning of the next line.

6. New compressor and expander K Parameter registers, CKPR and EKPR, have been added in 7970A. In 7970 RevA', these registers are not used. These 8-bit registers define the K parameter in binary representing K values from 1 to 255. The value 0 specifies a K value of infinity. These registers are located at the following addresses:

CKPR: 8 bits, 1 byte @ '66'
EKPR: 8 bits, 1 byte @ 'E6'

7. New compressor and expander Frame Width Registers, CFWR and EFWR, have been added in 7970A. These 16-bit registers define the width of the image area in the memory buffer. These registers are used by the CEP to calculate the starting point of the next image line in the memory buffer. The CEP calculates the starting address of line N + 1 as being one frame width beyond the starting address of line N.

For "full page" processing, the frame width must equal the page width (CFWR = CPWR). For

"window" processing, the Frame Width Register defines the width of the overall image area or frame and the Page Width Register defines the width of the window to be compressed or expanded within the frame.

Each line of image is transferred by DMA to or from the frame address space in memory immediately following the frame address space of the previous line referenced within the frame by the Page Width Register.

For "full page" or "window" processing, the first line starting address is specified by the CPU as defined in the data sheet for the CEP.

The CFWR may not be written into when the compressor is busy; the EFWR may not be written into when the expander is busy.

In the 7970 Rev. A', the Page Width Register value is used to calculate the starting address of each next line. Frame width is implicitly equal to page width.

These registers are located at the following addresses:

CFWR: 16 bits, 2 bytes @ '54' (LSB)/'56' (MSB)
EFWR: 16 bits, 2 bytes @ 'D4' (LSB)/'D6' (MSB)

8. CER change — In 7970A, the eight bits in the Compressor Express Register, CER, defines (in binary) the number of image lines to compress before skipping one line. For example, if CER value is 4, every fifth line is skipped resulting in a 4/5 vertical image reduction. If the value is 0, every line is compressed.

In the 7970 Rev. A', the CER value defines how many scan lines to skip before compressing the next line.

9. EPR change — In 7970A, the expander granularity mode bits (bits 3, 4, and 5) in the Expander Parameter Register define in binary how many lines to expand before duplicating the last line. For example, if the value in this field is 4, every fourth line is duplicated resulting in a 5/4 vertical image expansion.

In 7970 Rev. A', the EPR bits 3, 4, and 5 G-parameter value defines the number of times each expanded line is duplicated.

Appendix E

Am7970A CEP DESIGN HINTS

The following are common mistakes, oversights, or points to pay particular attention to when designing with the Am7970A.

1. In memory, the bits representing the pixels are stored as bytes. The first pixel at the top left edge of the image must be stored as the least significant bit of the first byte in the memory buffer. This is also the first bit to be sent on a transmission line. The compressed coded image follows the same rule. If this rule is violated, additional color changes may be created completely upsetting the compression coding. There is no standardization on how a scanner has to present the data.

2. If the CEP is inactive ("busy" bits inactive), all signals of the document side are tri-stated. Because an abort of the CEP assures this, this feature can be used for a cheap software controlled bus arbitration of the document bus.

3. One DMA Controller serves both the system bus and the document bus. Therefore, both buses cannot be simultaneously accessed.

4. The system side interface transfers only one byte per arbitration cycle.

5. A software reset is performed by writing a reset command (00) into the Operation Control (OC) bits of the Master Control Register. The reset operation is a microprogram that takes about 4 microseconds to execute. During this time, the "busy" bit (CBY and EBY) is set active in the Master Status Register. During the reset operation, the "busy" bit can be sampled until it goes inactive to verify that the CEP has completed the reset operation before attempting any other operation. Then the GO bit in the Master Control Register (CMCR or EMCR) is set to 1 to start the operation.

6. The software reset is required before processing a new page but it is not needed to resume processing on the same page.

7. Do not use $\overline{CS}$ directly to enable the buffer for $\overline{RD}$ or $\overline{WR}$ into the CEP in slave mode. When $\overline{CS}$ goes Low, the outputs of the buffer will still be unstable for a couple of nanoseconds while the CEP is already expecting true signals. The best solution is to use HLDA for enabling the driver.

8. Make sure $\overline{CS}$ is High within 1 clock cycle after $\overline{RD}$ or $\overline{WR}$ return High if block transfer mode is not desired. Refer to the CPU Block I/O Transaction Timing Diagram in the data sheet for block transfers.

9. In Master Mode, the CEP samples the READY and DREADY line with the rising edge of T2 before $\overline{RD}$ or $\overline{WR}$ are asserted by the CEP. See timing diagrams in the data sheet for more details.

10. READY and DREADY must be High before Main Memory data can be accessed by the CEP. Care must be taken, however, to assure that this signal is synchronized to the CEP clock and meets its set-up and hold requirements as specified in the data sheet. Failure to do so causes unpredictable operation.

11. When the CEP does not have a source or destination buffer located on the Document Store bus, the $\overline{DRD}$, $\overline{DWR}$, and DALE pins are floated to three state. Therefore, pullup resisters must be connected to these pins.

12. DALE in conjunction with a pullup resister makes a perfect $\overline{AS}$ signal for 68000 systems because it changes directly to High from tristate. This may be a useful low-active memory-enable signal for the document bus.

13. The CPU cannot instantaneously or directly access the CEP internal registers because that would interfere with the CEP internal operations on its internal bus. Instead, a slave access is used to interrupt the internal microprogram. After that, all data transfers to and from the registers are performed by a microprogram. The CPU is kept waiting during this time by holding READY Low. (The only exception is a read on the Master Status Register which is directly accessible by the CPU.

14. The access times of the registers varies widely for two reasons:

    A. The access time depends on the status of the operation that the CEP is currently performing.

B. Access times are optimized with respect to the probability of their usage.

15. Presently, register access time is unpredictable when the CEP is busy. In Am7970 this is no problem because slave accesses are forbidden. In Am7970A, this is important since the access time may be as long as 50 clock cycles. This may affect system design considerations severely. Typical access times are:

    Write Operation with CEP in Idle State:

    Case 1, A single write once in a while:

    4 clock cycles for all registers.
    This write access is internally latched. The addressed register is loaded with the data much later than the CPU is released.

    Case 2, a sequence of consecutive slave accesses:

    6 clock cycles for papersize, parameter, and command registers.
    14 clock cycles for all other registers.

    Read Operation with CEP in Idle State:

    All cases:

    4 clock cycles = MSR only
    10 clock cycles = status, parameter, command, and papersize registers.
    12 clock cycles = all address and option registers.

    All Operations with CEP Busy:

    4 clock cycles for MSR read All other accesses take an unpredictable number of clock cycles up to 50 depending on the current operation being performed by the internal microprogram.

16. One DMA Controller serves both the system bus and the document bus. Therefore, there is never a simultaneous access on both the system bus and the document bus. If the CEP is inactive ("busy" bits inactive), all signals of the document side are tri-stated. This feature can be used for a cheap software controlled bus arbitration of the document bus. When a peripheral wants to access the document bus, it can notify the system CPU. The CPU can poll the BUSY bit of the CEP and notify the peripheral when it becomes inactive.

17. The CEP operation may be stopped by a write to the command register when the CEP is busy (a software abort). However, this abort is not resumable. In other words, setting the "GO" bit back to high is not enough to resume an aborted operation.

18. If the DREADY signal is suppressed on the document bus, the CEP will be frozen the moment that it samples the DREADY line because it is waiting for access to the document bus. No further memory transfers can take place on the system bus either because each side is waiting for the other to complete the memory cycle. HREQ is inactive in this case. This behavior might be useful for implementing a ring buffer as a destination buffer.

19. Bits 0, 1, and 2 are the Extension Code bits. When ECD (Bit 3) is set to "1", the EXT bits display (in reverse order) the three least significant extension code bits which have been detected by the Expander. For example, an extension code of "011" appears in the MSR as "110". When the ECD bit is set to "0", the extension bits are cleared to "0s". The EXT bits are also cleared when a new operation is initiated.

20. The CEP controls the contents of the Working Count Register only when it increments it. The CEP does not observe what value is loaded by the CPU and thus does not interrupt for "0" contents. Refer to Figure 2-21.

21. A destination buffer that is smaller than page size can be used if the buffer size and the K parameter are chosen such that the buffer is an integral multiple of K and the first line encoded is a 1D line. The buffer will overflow after the last line of 2D. Under these conditions, each time that the buffer overflows, the next line to be encoded or decoded is a 1D line. When coding or decoding resumes, no reference line is needed since the next line is a 1D line.

Recommendation T.4

STANDARDIZATION OF GROUP 3 FACSIMILE APPARATUS FOR DOCUMENT TRANSMISSION

(Geneva, 1980, amended at Malaga-Torremolinos, 1984)

The CCITT,

considering

(a) that Recommendation T.2 refers to Group 1 type apparatus for ISO A4 document transmission over a telephone-type circuit in approximately six minutes;

(b) that Recommendation T.3 refers to Group 2 type apparatus for ISO A4 document transmission over a telephone-type circuit in approximately three minutes;

(c) that there is a demand for Group 3 apparatus which enables an ISO A4 document to be transmitted over a telephone-type circuit in approximately one minute;

(d) that for a large number of applications black and white reproduction is sufficient;

(e) that such a service may be requested either alternatively with telephone conversation, or when either or both stations are not attended; in both cases, the facsimile operation will follow Recommendation T.30;

unanimously declares the view

that Group 3 facsimile apparatus for use on the general switched telephone network and international leased circuits should be designed and operated according to the following standards:

1 Scanning track

The message area should be scanned in the same direction in the transmitter and receiver. Viewing the message area in a vertical plane, the picture elements should be processed as if the scanning direction were from left to right with subsequent scans adjacent and below the previous scan.

2 Dimensions of apparatus

Note – The tolerances on the factors of cooperation are subject to further study.

2.1 The following dimensions should be used:

a) A standard resolution and an optional higher resolution of 3.85 line/mm ± 1% and 7.7 line/mm ± 1% respectively in vertical direction.

b) 1728 black and white picture elements along the standard scan line length of 215 mm ± 1%.

c) Optionally, 2048 black and white picture elements along a scan line length of 255 mm ± 1%.

d) Optionally, 2432 black and white picture elements along a scan line length of 303 mm ± 1%.

2.2 Input documents up to a minimum of ISO A4 size should be accepted.

Note – The size of the guaranteed reproducible area is shown in Appendix I.

3 Transmission time per total coded scan line

The total coded scan line is defined as the sum of DATA bits plus any required FILL bits plus the EOL bits.

For the optional two-dimensional coding scheme as described in § 4.2, the total coded scan line is defined as the sum of DATA bits plus any required FILL bits plus the EOL bits plus a tag bit.

To handle various printing methods, several optional minimum total coded scan line times are possible in addition to the 20 milliseconds standard.

3.1 The minimum transmission times of the total coded scan line should conform to the following:

1) Alternative 1, where the minimum transmission time of the total coded scan line is the same both for the standard resolution and for the optional higher resolution:

   a) 20 milliseconds recommended standard,

   b) 10 milliseconds recognized option with a mandatory fall-back to the 20 milliseconds standard,

   c) 5 milliseconds recognized option with a mandatory fall-back to the 10 milliseconds option and the 20 milliseconds standard,

   d) 0 millisecond recognized option with a mandatory fall-back to the 5 milliseconds option, the 10 milliseconds option and the 20 milliseconds standard, and an optional fall-back to the 40 milliseconds option,

   e) 40 milliseconds recognized option.

2) Alternative 2, where the minimum transmission time of the total coded scan line for the optional higher resolution is half of that for the standard resolution (see Note). These figures refer to the standard resolution:

   a) 10 milliseconds recognized option with a mandatory fall-back to the 20 milliseconds standard,

   b) 20 milliseconds recommended standard,

   c) 40 milliseconds recognized option.

The identification and choice of this minimum transmission time is to be made in the pre-message (phase B) portion of Recommendation T.30 control procedure.

Note – Alternative 2 applies to equipment with printing mechanisms which achieve the standard vertical resolution by printing two consecutive, identical higher resolution lines. In this case, the minimum transmission time of the total coded scan line for the standard resolution is double the minimum transmission time of the total coded scan line for the higher resolution.

3.2 The maximum transmission time of any total coded scan line should be less than 5 seconds. When this transmission time exceeds 5 seconds, the receiver must proceed to disconnect the line.

4 Coding scheme

4.1 *One-dimensional coding scheme*

The one-dimensional run length coding scheme recommended for Group 3 apparatus is as follows:

4.1.1 *Data*

A line of Data is composed of a series of variable length code words. Each code word represents a run length of either all white or all black. White runs and black runs alternate. A total of 1728 picture elements represent one horizontal scan line of 215 mm length.

In order to ensure that the receiver maintains colour synchronization, all Data lines will begin with a white run length code word. If the actual scan line begins with a black run, a white run length of zero will be sent. Black or white run lengths, up to a maximum length of one scan line (1728 picture elements or pels) are defined by the code words in Tables 1/T.4 and 2/T.4. The code words are of two types: Terminating code words and Make-up code words. Each run length is represented by either one Terminating code word or one Make-up code word followed by a Terminating code word.

Run lengths in the range of 0 to 63 pels are encoded with their appropriate Terminating code word. Note that there is a different list of code words for black and white run lengths.

Run lengths in the range of 64 to 1728 pels are encoded first by the Make-up code word representing the run length which is equal to or shorter than that required. This is then followed by the Terminating code word representing the difference between the required run length and the run length represented by the Make-up code.

TABLE 1/T.4

Terminating codes

White run length	Code word	Black run length	Code word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111
15	110101	15	000011000
16	101010	16	0000010111
17	101011	17	0000011000
18	0100111	18	0000001000
19	0001100	19	00001100111
20	0001000	20	00001101000
21	0010111	21	00001101100
22	0000011	22	00000110111
23	0000100	23	00000101000
24	0101000	24	00000010111
25	0101011	25	00000011000
26	0010011	26	000011001010
27	0100100	27	000011001011
28	0011000	28	000011001100
29	00000010	29	000011001101
30	00000011	30	000001101000
31	00011010	31	000001101001
32	00011011	32	000001101010
33	00010010	33	000001101011
34	00010011	34	000011010010
35	00010100	35	000011010011
36	00010101	36	000011010100
37	00010110	37	000011010101
38	00010111	38	000011010110
39	00101000	39	000011010111
40	00101001	40	000001101100
41	00101010	41	000001101101
42	00101011	42	000011011010
43	00101100	43	000011011011
44	00101101	44	000001010100
45	00000100	45	000001010101
46	00000101	46	000001010110
47	00001010	47	000001010111
48	00001011	48	000001100100
49	01010010	49	000001100101
50	01010011	50	000001010010
51	01010100	51	000001010011
52	01010101	52	000000100100
53	00100100	53	000000110111
54	00100101	54	000000111000
55	01011000	55	000000100111
56	01011001	56	000000101000
57	01011010	57	000001011000
58	01011011	58	000001011001
59	01001010	59	000000101011
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111

TABLE 2/T.4

Make-up codes

White run lengths	Code word	Black run lengths	Code word
64	11011	64	0000001111
128	10010	128	000011001000
192	010111	192	000011001001
256	0110111	256	000001011011
320	00110110	320	000000110011
384	00110111	384	000000110100
448	01100100	448	000000110101
512	01100101	512	0000001101100
576	01101000	576	0000001101101
640	01100111	640	0000001001010
704	011001100	704	0000001001011
768	011001101	768	0000001001100
832	011010010	832	0000001001101
896	011010011	896	0000001110010
960	011010100	960	0000001110011
1024	011010101	1024	0000001110100
1088	011010110	1088	0000001110101
1152	011010111	1152	0000001110110
1216	011011000	1216	0000001110111
1280	011011001	1280	0000001010010
1344	011011010	1344	0000001010011
1408	011011011	1408	0000001010100
1472	010011000	1472	0000001010101
1536	010011001	1536	0000001011010
1600	010011010	1600	0000001011011
1664	011000	1664	0000001100100
1728	010011011	1728	0000001100101
EOL	000000000001	EOL	000000000001

Note — It is recognized that machines exist which accommodate larger paper widths whilst maintaining the standard horizontal resolution. This option has been provided for by the addition of the Make-up code set defined as follows:

Run length (black and white)	Make-up codes
1792	00000001000
1856	00000001100
1920	00000001101
1984	000000010010
2048	000000010011
2112	000000010100
2176	000000010101
2240	000000010110
2304	000000010111
2368	000000011100
2432	000000011101
2496	000000011110
2560	000000011111

4.1.2 *End-of-line (EOL)*

This code word follows each line of Data. It is a unique code word that can never be found within a valid line of Data; therefore, resynchronization after an error burst is possible.

In addition, this signal will occur prior to the first Data line of a page.

Format: 000000000001

4.1.3 *Fill*

A pause may be placed in the message flow by transmitting Fill. Fill may be inserted between a line of Data and an EOL, but never within a line of Data. Fill must be added to ensure that the transmission time of Data, Fill and EOL is not less than the minimum transmission time of the total coded scan line established in the pre-message control procedure.

Format: variable length string of 0s.

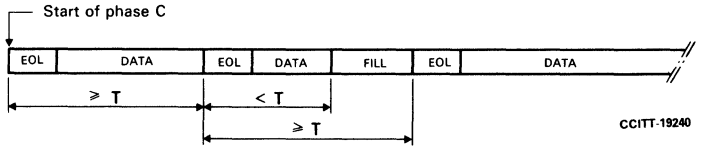
4.1.4 *Return to control (RTC)*

The end of a document transmission is indicated by sending six consecutive EOLs. Following the RTC signal, the transmitter will send the post message commands in the framed format and the data signalling rate of the control signals defined in Recommendation T.30.

Format: 000000000001 000000000001
(total of 6 times)

Figures 1/T.4 and 2/T.4 clarify the relationship of the signals defined herein. Figure 1/T.4 shows several scan lines of data starting at the beginning of a transmitted page. Figure 2/T.4 shows the last coded scan line of a page.

The identification and choice of either the standard code table or the extended code table is to be made in the pre-message (phase B) portion of Recommendation T.30 control procedures.



T Minimum transmission time of a total coded scan line

FIGURE 1/T.4

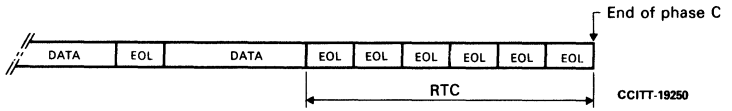


FIGURE 2/T.4

4.2 *Two-dimensional coding scheme*

The two-dimensional coding scheme is an optional extension of the one-dimensional coding scheme specified in § 4.1 and is as follows:

4.2.1 *Data*

4.2.1.1 *Parameter K*

In order to limit the disturbed area in the event of transmission errors, after each line coded one-dimensionally, at most *K*-1 successive lines shall be coded two-dimensionally. A one-dimensionally coded line may be transmitted more frequently than every *K* lines. After a one-dimensional line is transmitted, the next series of *K*-1 two-dimensional lines is initiated. The maximum value of *K* shall be set as follows:

Standard vertical resolution: $K = 2$

Optional higher vertical resolution: $K = 4$.

Note 1 – Some Administrations pointed out that for the optional higher vertical resolution *K* may optionally be set to a lower value.

Note 2 – Some Administrations reserve the right to approve only such apparatus for use in the facsimile service in their respective countries which will be able to produce a visible sign on its received facsimile message indicating that two-dimensional coding has been used in the transmission process.

4.2.1.2 *One-dimensional coding*

This conforms with the description of Data in § 4.1.1.

4.2.1.3 *Two-dimensional coding*

This is a line-by-line coding method in which the position of each changing picture element on the current or coding line is coded with respect to the position of a corresponding reference element situated on either the coding line or the reference line which lies immediately above the coding line. After the coding line has been coded it becomes the reference line for the next coding line.

4.2.1.3.1 *Definition of changing picture elements (see Figure 3/T.4)*

A changing element is defined as an element whose "colour" (i.e. black or white) is different from that of the previous element along the same scan line.

a_0 The reference or starting changing element on the coding line. At the start of the coding line a_0 is set on an imaginary white changing element situated just before the first element on the line. During the coding of the coding line, the position of a_0 is defined by the previous coding mode. (See § 4.2.1.3.2.)

a_1 The next changing element to the right of a_0 on the coding line.

a_2 The next changing element to the right of a_1 on the coding line.

b_1 The first changing element on the reference line to the right of a_0 and of opposite colour to a_0.

b_2 The next changing element to the right of b_1 on the reference line.

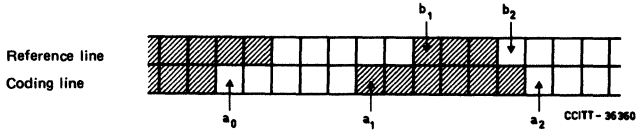


FIGURE 3/T.4
Changing picture elements

4.2.1.3.2 *Coding modes*

One of the three coding modes are chosen according to the coding procedure described in § 4.2.1.3.3 to code the position of each changing element along the coding line. Examples of the three coding modes are given in Figures 4/T.4, 5/T.4 and 6/T.4.

a) *Pass mode*

This mode is identified when the position of b_2 lies to the left of a_1. When this mode has been coded, a_0 is set on the element of the coding line below b_2 in preparation for the next coding (i.e. on a'_0).

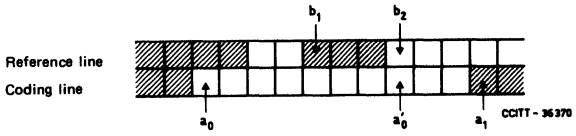


FIGURE 4/T.4

Pass mode

However, the state where b_2 occurs just above a_1, as shown in Figure 5/T.4 is not considered as a pass mode.

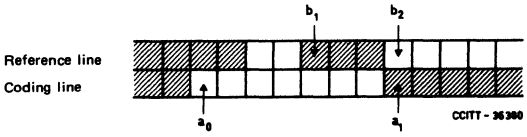


FIGURE 5/T.4

An example not corresponding to a Pass mode

b) *Vertical mode*

When this mode is identified, the position of a_1 is coded relative to the position of b_1. The relative distance a_1b_1 can take on one of seven values V(0), $V_R(1)$, $V_R(2)$, $V_R(3)$, $V_L(1)$, $V_L(2)$ and $V_L(3)$, each of which is represented by a separate code word. The subscripts R and L indicate that a_1 is to the right or left respectively of b_1, and the number in brackets indicates the value of the distance a_1b_1. After vertical mode coding has occurred, the position of a_0 is set on a_1, (see Figure 6/T.4).

c) *Horizontal mode*

When this mode is identified, both the run-lengths a_0a_1 and a_1a_2 are coded using the code words H + $M(a_0a_1)$ + $M(a_1a_2)$. H is the flag code word 001 taken from the two-dimensional code table (Table 3/T.4). $M(a_0a_1)$ and $M(a_1a_2)$ are code words which represent the length and "colour" of the runs a_0a_1 and a_1a_2 respectively and are taken from the appropriate white or black one-dimensional code tables (Tables 1/T.4 and 2/T.4). After a horizontal mode coding, the position of a_0 is set on a_2 (see Figure 6/T.4).

4.2.1.3.3 *Coding procedure*

The coding procedure identifies the coding mode that is to be used to code each changing element along the coding line. When one of the three coding modes has been identified according to Step 1 or Step 2 mentioned below, an appropriate code word is selected from the code table given in Table 3/T.4. The coding procedure is as shown in the flow diagram of Figure 7/T.4.

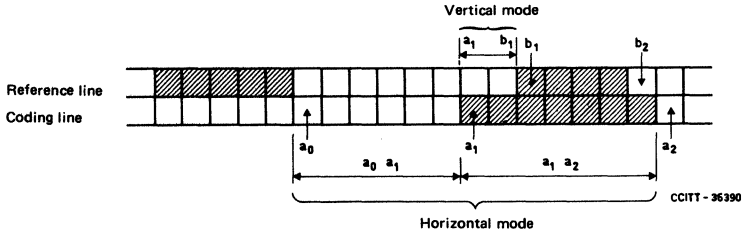


FIGURE 6/T.4

Vertical mode and Horizontal mode

Note – It does not affect compatibility to restrict the use of pass mode in the encoder to a single pass mode. Variations of the algorithm which do not affect compatibility should be the subject of further study.

Step 1

i) If a pass mode is identified, this is coded using the word 0001 (Table 3/T.4). After this processing, picture element a'_0 just under b_2 is regarded as the new starting picture element a_0 for the next coding. (See Figure 4/T.4.)

ii) If a pass mode is not detected then proceed to Step 2.

Step 2

i) Determine the absolute value of the relative distance a_1b_1.

ii) If $|a_1b_1| \leqslant 3$, as shown in Table 3/T.4, a_1b_1 is coded by the vertical mode, after which position a_1 is regarded as the new starting picture element a_0 for the next coding.

iii) If $|a_1b_1| > 3$, as shown in Table 3/T.4, following horizontal mode code 001, a_0a_1 and a_1a_2 are respectively coded by one-dimensional coding. After this processing position a_2 is regarded as the new starting picture element a_0 for the next coding.

4.2.1.3.4 *Processing the first and last picture elements in a line*

a) *Processing the first picture element*

The first starting picture element a_0 on each coding line is imaginarily set at a position just before the first picture element, and is regarded as a white picture element (see § 4.2.1.3.1).

The first run length on a line a_0a_1 is replaced by $a_0a_1 - 1$. Therefore, if the first run is black and is deemed to be coded by horizontal mode coding, then the first code word $M(a_0a_1)$ corresponds to a white run of zero length (see Figure 10/T.4, Example 5).

b) *Processing the last picture element*

The coding of the coding line continues until the position of the imaginary changing element situated just after the last actual element has been coded. This may be coded as a_1 or a_2. Also, if b_1 and/or b_2 are not detected at any time during the coding of the line, they are positioned on the imaginary changing element situated just after the last actual picture element on the reference line.

4.2.2 *Line synchronization code word*

To the end of every coded line is added the end-of-line (EOL) code word 000000000001. The EOL code word is followed by a single tag bit which indicates whether one- or two-dimensional coding is used for the next line.

TABLE 3/T.4

Two-dimensional code table

Mode	Elements to be coded		Notation	Code word
Pass	b_1, b_2		P	0001
Horizontal	a_0a_1, a_1a_2		H	001 + M(a_0a_1) + M (a_1a_2) (see Note)
Vertical	a_1 just under b_1	$a_1b_1 = 0$	V(0)	1
	a_1 to the right of b_1	$a_1b_1 = 1$	$V_R(1)$	011
		$a_1b_1 = 2$	$V_R(2)$	000011
		$a_1b_1 = 3$	$V_R(3)$	0000011
	a_1 to the left of b_1	$a_1b_1 = 1$	$V_L(1)$	010
		$a_1b_1 = 2$	$V_L(2)$	000010
		$a_1b_1 = 3$	$V_L(3)$	0000010
Extension	2-D (extensions) 1-D (extensions)			0000001xxx 000000001xxx (see Note 2)

Note 1 — Code M( ) of the horizontal mode represents the code words in Tables 1/T.4 and 2/T.4.

Note 2 — It is suggested the uncompressed mode is recognized as an optional extension of the two-dimensional coding scheme for Group 3 apparatus. The bit assignment for the xxx bits is 111 for the uncompressed mode of operation whose code table is given in Table 4/T.4.

Note 3 — Further study is needed to define other unspecified xxx bit assignments and their use for any further extensions.

Note 4 — If the suggested uncompressed mode is used on a line designated to be one-dimensionally coded, the coder must not switch into the uncompressed mode following any code word ending in the sequence 000. This is because any code word ending in 000 followed by a switching code 000000001 will be mistaken for an end-of-line code.

TABLE 4/T.4

Uncompressed mode code words

Entrance code to uncompressed mode	On one-dimensionally coded line: 000000001111 On two-dimensionally coded line: 0000001111	
	Image pattern	*Code word*
Uncompressed mode code	1 01 001 0001 00001 00000	1 01 001 0001 00001 000001
Exist from uncompressed mode code	0 00 000 0000	0000001T 00000001T 000000001T 0000000001T 00000000001T

T denotes a tag bit which tells the colour of the next run (black = 1, white = 0).

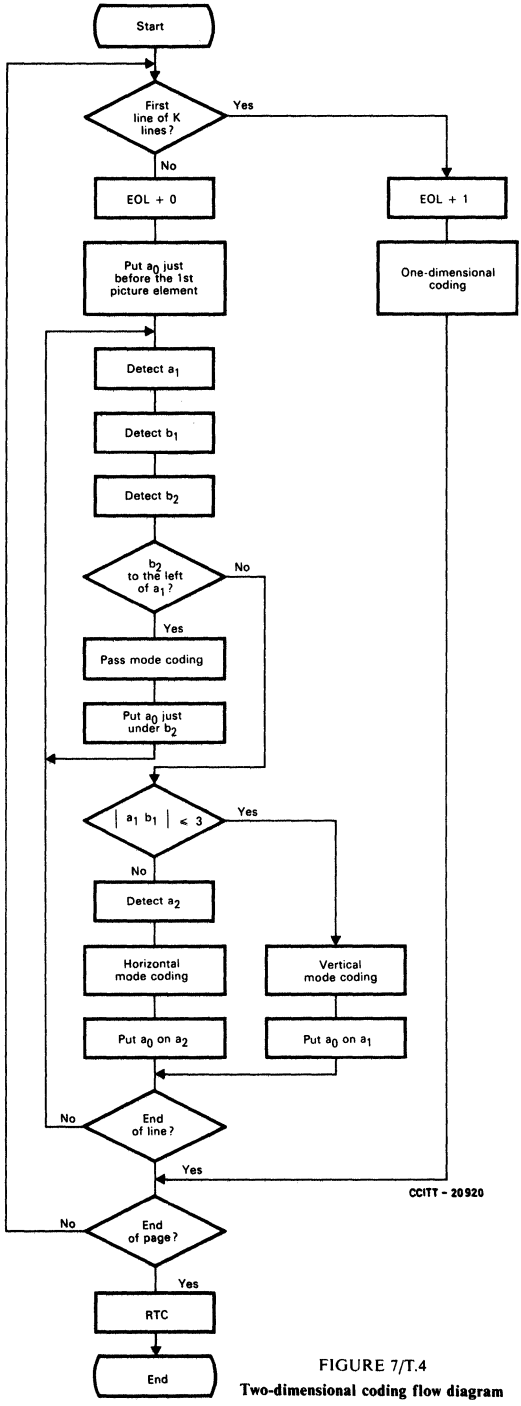


FIGURE 7/T.4

Two-dimensional coding flow diagram

In addition, EOL plus the tag bit 1 signal will occur prior to the first Data line of a page.

Format:

EOL + 1: one-dimensional coding of next line

EOL + 0: two-dimensional coding of next line

4.2.3 *Fill*

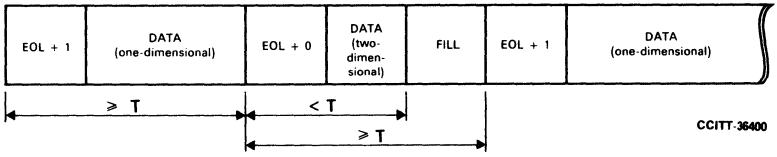
Fill is inserted between a line of Data and the line synchronization signal, EOL + tag bit, but is not inserted in Data. Fill must be added to ensure that the transmission time of Data, Fill and EOL plus tag bit is not less than the minimum transmission time of the total coded scan line.

Format: variable length string of 0 s.

4.2.4 *Return to control (RTC)*

The format used is six consecutive line synchronization code words, i.e., 6 × (EOL + 1).

To further clarify the relationship of the signals defined herein, Figures 8/T.4 and 9/T.4 are offered in the case of K = 2. Figure 8/T.4 shows several scan lines of data starting at the beginning of a transmitted page. Figure 9/T.4 shows the last several lines of a page.



T Minimum transmit time of a total coded scan line

FIGURE 8/T.4

Message transmission (first part of page)

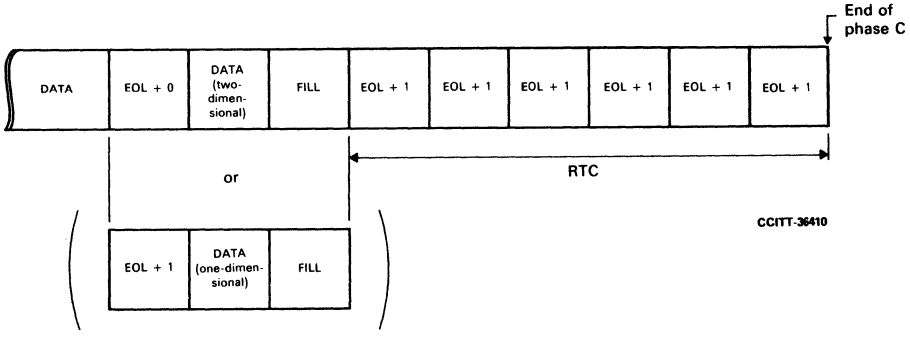


FIGURE 9/T.4

Message transmission (last part of page)

4.2.5 *Coding examples*

Figure 10/T.4 shows coding examples of the first part of scan lines and Figure 11/T.4 coding examples of the last part, while Figure 12/T.4 shows other coding examples. The notations P, H and V in the figures are, as shown in Table 3/T.4, the symbols for pass mode, horizontal mode and vertical mode respectively. The picture elements marked with black spots indicate the changing picture elements to be coded.

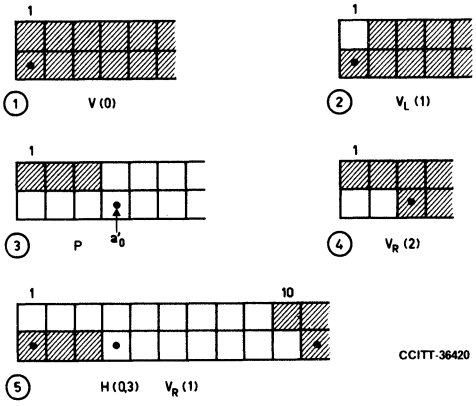


FIGURE 10/T.4

Coding examples: first part of scan line

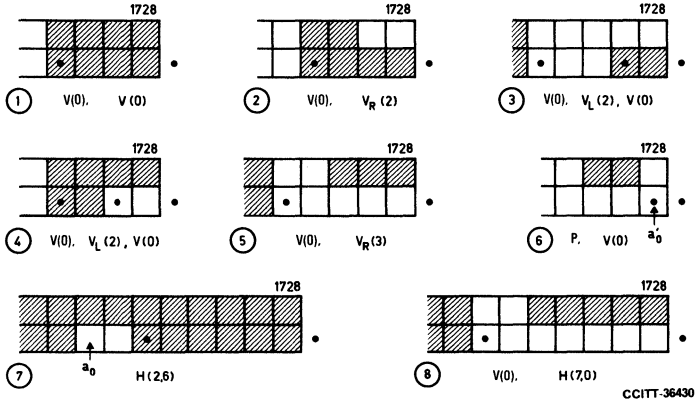


FIGURE 11/T.4

Coding examples: last part of scan line

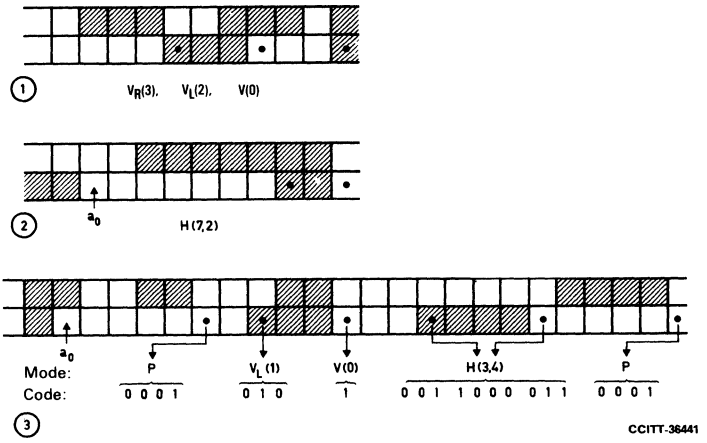


FIGURE 12/T.4
Coding examples

5 Modulation and demodulation

Group 3 apparatus operating on the general switched telephone network shall utilize the modulation, scrambler, equalization and timing signals defined in Recommendation V.27 *ter*, specifically §§ 2, 3, 7, 8, 9, 11 and the Appendix.

5.1 The training signal to be used shall be the long training sequence with protection against talker echo. (See Recommendation V.27 *ter*, §2.5.1, Table 3/V.27 *ter*).

5.2 The data signalling rates to be used are 4800 bit/s and 2400 bit/s as defined in Recommendation V.27 *ter*.

Note 1 – Some Administrations pointed out that it would not be possible to guarantee the service at a data signalling rate higher than 2400 bit/s.

Note 2 – It should be noted that there are equipments in service using, inter alia, other modulation methods.

Note 3 – Where quality of communication service can successfully support higher speed operation, such as may be possible on leased circuits or high-quality switched circuits, Group 3 apparatus may optionally utilize the modulation, scrambler, equalization and timing signals defined in Recommendation V.29, specifically §§ 1, 2, 3, 4, 7, 8, 9, 10 and 11. Under this option the data should be non-multiplexed and limited to the data signalling rates of 9600 bit/s and 7200 bit/s.

6 Power at the transmitter output

The average power should be adjustable from −15 dBm to 0 dBm but the equipment should be so designed that there is no possibility of this adjustment being tampered with by an operator.

Note – The power levels over the international circuits will conform to Recommendation V.2.

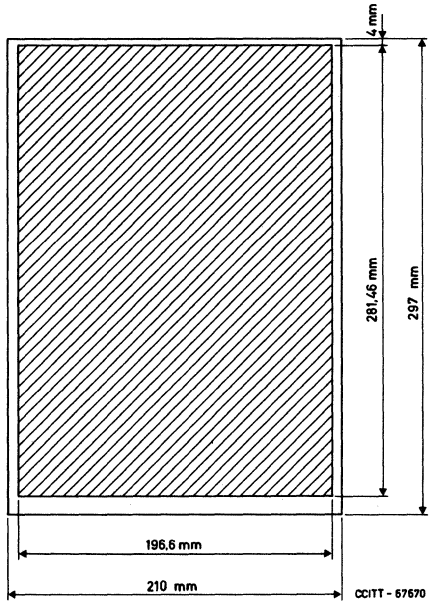
7 Power at the receiver input

The receiving apparatus should be capable of functioning correctly when the received signal level is within the range of 0 dBm to −43 dBm. No control of receiver sensitivity should be provided for operator use.

APPENDIX I

(to Recommendation T.4)

Guaranteed reproducible area for Group 3 apparatus conforming to Recommendation T.4



Note 1 – Paper characteristics (i.e. weight) are important parameters. Lightweight paper may cause additional paper handling errors and may result in a reduced guaranteed reproducible area.

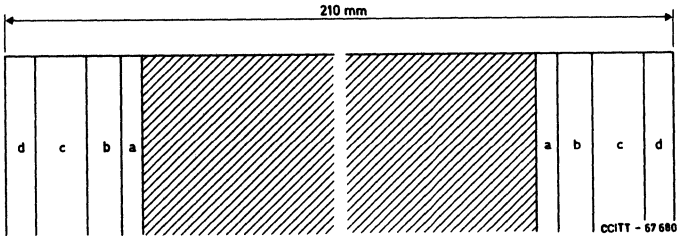
Note 2 – Sheet feed mechanisms may reduce the guaranteed reproducible area.

Note 3 – All calculations were done using worst case values. Using nominal values increases the reproducible area.

Note 4 – The exact horizontal position of this area within the ISO A4 paper size as well as sizes larger than the above are subject to national recommendations and/or definitions.

FIGURE I-1/T.4

Guaranteed reproducible area for Group 3 machines for use on facsimile services referring to ISO A4 paper size



a : Printer/scanner tolerances
b : Loss caused by the enlarging effect due to TLL tolerance
c : Loss caused by skew
d : Record medium positioning errors

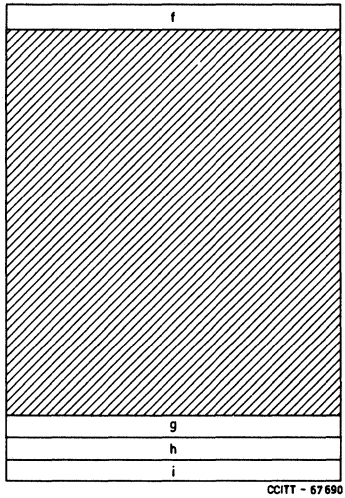
FIGURE I-2/T.4

Horizontal loss

TABLE I-1/T.4

Horizontal losses

Printer/scanner	a	± 0.5 mm
Enlarging	b	± 2.1 mm
Skew	c	± 2.6 mm
Positioning errors	d	± 1.5 mm



f : Paper insertion loss
g : Loss caused by skew
h : Scanning density tolerance
i : Gripping loss

FIGURE I-3/T.4

TABLE I-2/T.4

Vertical losses

Paper insertion	f	4.0 mm
Skew	g	± 1.8 mm
Scan line tolerance	h	± 2.97 mm
Gripping loss	i	2.0 mm

Note – Scanning density tolerance will reduce to 0 mm on roll-fed machines.

Recommendation T.6

FACSIMILE CODING SCHEMES AND CODING CONTROL FUNCTIONS FOR GROUP 4 FACSIMILE APPARATUS

(Malaga-Torremolinos, 1984)

1 General

1.1 *Scope*

1.1.1 Recommendation T.6 defines the facsimile coding schemes and their control functions to be used in the Group 4 facsimile.

1.1.2 This Recommendation should be read in conjunction with the following Recommendations:

T.5 – General aspects of Group 4 facsimile apparatus

T.73 – Document interchange protocol for the Telematic services

T.62 – Control procedures for Teletex and Group 4 facsimile services

T.70 – Network-independent basic transport service for Telematic services

F.161 – International Group 4 facsimile service

In addition, in the case of Group 4 Class II/III (Teletex or mixed mode of operation), the following Recommendations should also be read:

T.60 – Terminal equipment for use in the Teletex service

T.61 – Character repertoire and coded character sets for the international Teletex service

T.72 – Terminal capabilities for mixed mode of operation

1.2 *Fundamental principles*

1.2.1 *Facsimile coding schemes and coding control functions*

Facsimile coding schemes consist of the basic facsimile coding scheme and optional facsimile coding schemes. They are defined in § 2 and §§ 3 and 4, respectively.

Facsimile coding schemes are specified assuming that transmission errors are corrected by control procedures at a lower level.

The basic facsimile coding scheme is the two-dimensional coding scheme which is in principle the same as the two-dimensional coding scheme of Group 3 facsimile specified in Recommendation T.4.

Optional facsimile coding schemes are specified not only for black and white images but also for grey scale images and colour images.

Facsimile coding control functions are used in facsimile user information in order to change facsimile parameters or to invoke the end of facsimile block. They are defined in § 2.4.

2 Facsimile coding schemes and coding control functions for black and white images

2.1 *General*

This section specifies the facsimile coding schemes, and associated control functions for black and white images.

Facsimile coding schemes consist of the basic facsimile coding scheme and optional coding schemes.

The use of the optional facsimile coding schemes is subject to mutual agreement between terminals and shall be initiated by the appropriate procedural steps.

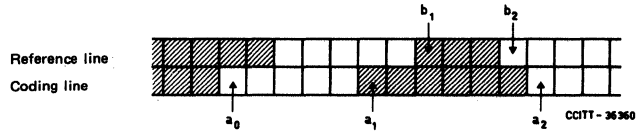
2.2 *Basic facsimile coding scheme*

2.2.1 *Principle of the coding scheme*

The coding scheme uses a two-dimensional line-by-line coding method in which the position of each changing picture element on the current coding line is coded with respect to the position of a corresponding reference element situated on either the coding line or the reference line which is immediately above the coding line. After the coding line has been coded, it becomes the reference line for the next coding line. The reference line for the first coding line in a page is an imaginary white line.

2.2.2 *Definition of changing picture elements* (see Figure 1/T.6)

A changing element is definied as an element whose "colour" (i.e. black or white) is different from that of the previous element along the same scan line.



a_0 : The reference or starting changing element on the coding line. At the start of the line a_0 is set on an imaginary white changing element situated just before the first element on the line. During the coding of the coding line, the position of a_0 is defined by the previous coding mode (see § 2.2.3).

a_1 : The next changing element to the right of a_0 on the coding line.

a_2 : The next changing element to the right of a_1 on the coding line.

b_1 : The first changing element on the reference line to the right of a_0 and of opposite colour to a_0.

b_2 : The next changing element to the right of b_1 on the reference line.

FIGURE 1/T.6

Changing picture elements

2.2.3 *Coding modes*

One of the three coding modes are chosen according to the coding procedure described in § 2.2.4 to code the position of each changing element along the coding line. Examples of the three coding modes are given in Figure 2/T.6, 3/T.6 and 4/T.6.

2.2.3.1 *Pass mode*

This mode is identified when the position of b_2 lies to the left of a_1. (See Figure 2/T.6.)

However, the state where b_2 occurs just above a_1, as shown in Figure 3/T.6 is not considered as a pass mode.

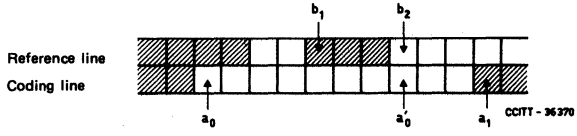


FIGURE 2/T.6

Pass mode

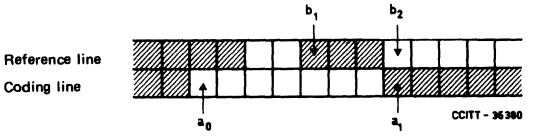


FIGURE 3/T.6

An example not corresponding to a Pass mode

2.2.3.2 *Vertical mode*

When this mode is identified, the position of a_1 is coded relative to the position of b_1. The relative distance a_1b_1 can take on one of seven values V(0), $V_R(1)$, $V_R(2)$, $V_R(3)$, $V_L(1)$, $V_L(2)$ and $V_L(3)$, each of which is represented by a separate code word. The subscripts R and L indicate that a_1 is to the right or left respectively of b_1, and the number in brackets indicates the value of the distance a_1b_1 (see Figure 4/T.6).

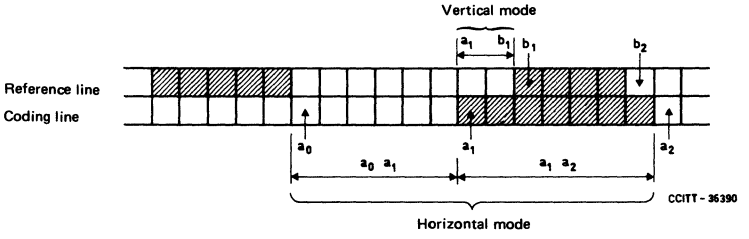


FIGURE 4/T.6

Vertical mode and horizontal mode

2.2.3.3 *Horizontal mode*

When this mode is identified, both the run-lengths a_0a_1 and a_1a_2 are coded using the code words H + $M(a_0a_1)$ + $M(a_1a_2)$. H is the flag code word 001 taken from the two-dimensional code table (Table 1/T.6). $M(a_0a_1)$ and $M(a_1a_2)$ are code words which represent the length and "colour" of the runs a_0a_1 and a_1a_2 respectively and are taken from the appropriate white or black run-length code tables (Tables 2/T.6 and 3/T.6).

TABLE 1/T.6

Code table

Mode	Elements to be coded		Notation	Code word
Pass	b_1, b_2		P	0001
Horizontal	a_0a_1, a_1a_2		H	001 + M(a_0a_1) + M (a_1a_2) (see Note)
Vertical	a_1 just under b_1	$a_1b_1 = 0$	V(0)	1
	a_1 to the right of b_1	$a_1b_1 = 1$	$V_R(1)$	011
		$a_1b_1 = 2$	$V_R(2)$	000011
		$a_1b_1 = 3$	$V_R(3)$	0000011
	a_1 to the left of b_1	$a_1b_1 = 1$	$V_L(1)$	010
		$a_1b_1 = 2$	$V_L(2)$	000010
		$a_1b_1 = 3$	$V_L(3)$	0000010
Extension				0000001xxx

Note – Code M( ) of the horizontal mode represents the code words in Tables 2/T.6 and 3/T.6.

TABLE 2/T.6

Terminating codes

White run length	Code word	Black run length	Code word
0	00110101	0	0000110111
1	000111	1	010
2	0111	2	11
3	1000	3	10
4	1011	4	011
5	1100	5	0011
6	1110	6	0010
7	1111	7	00011
8	10011	8	000101
9	10100	9	000100
10	00111	10	0000100
11	01000	11	0000101
12	001000	12	0000111
13	000011	13	00000100
14	110100	14	00000111
15	110101	15	000011000
16	101010	16	0000010111
17	101011	17	0000011000
18	0100111	18	0000001000
19	0001100	19	00001100111
20	0001000	20	00001101000
21	0010111	21	00001101100
22	0000011	22	00000110111
23	0000100	23	00000101000
24	0101000	24	00000010111
25	0101011	25	00000011000
26	0010011	26	000011001010
27	0100100	27	000011001011
28	0011000	28	000011001100
29	00000010	29	000011001101
30	00000011	30	000001101000
31	00011010	31	000001101001
32	00011011	32	000001101010
33	00010010	33	000001101011
34	00010011	34	000011010010
35	00010100	35	000011010011
36	00010101	36	000011010100
37	00010110	37	000011010101
38	00010111	38	000011010110
39	00101000	39	000011010111
40	00101001	40	000001101100
41	00101010	41	000001101101
42	00101011	42	000011011010
43	00101100	43	000011011011
44	00101101	44	000001010100
45	00000100	45	000001010101
46	00000101	46	000001010110
47	00001010	47	000001010111
48	00001011	48	000001100100
49	01010010	49	000001100101
50	01010011	50	000001010010
51	01010100	51	000001010011
52	01010101	52	000000100100
53	00100100	53	000000110111
54	00100101	54	000000111000
55	01011000	55	000000100111
56	01011001	56	000000101000
57	01011010	57	000001011000
58	01011011	58	000001011001
59	01001010	59	000000101011
60	01001011	60	000000101100
61	00110010	61	000001011010
62	00110011	62	000001100110
63	00110100	63	000001100111

TABLE 3/T.6

Make-up codes between 64 and 1728

White run lengths	Code word	Black run lengths	Code word
64	11011	64	0000001111
128	10010	128	000011001000
192	010111	192	000011001001
256	0110111	256	000001011011
320	00110110	320	000000110011
384	00110111	384	000000110100
448	01100100	448	000000110101
512	01100101	512	0000001101100
576	01101000	576	0000001101101
640	01100111	640	0000001001010
704	011001100	704	0000001001011
768	011001101	768	0000001001100
832	011010010	832	0000001001101
896	011010011	896	0000001110010
960	011010100	960	0000001110011
1024	011010101	1024	0000001110100
1088	011010110	1088	0000001110101
1152	011010111	1152	0000001110110
1216	011011000	1216	0000001110111
1280	011011001	1280	0000001010010
1344	011011010	1344	0000001010011
1408	011011011	1408	0000001010100
1472	010011000	1472	0000001010101
1536	010011001	1536	0000001011010
1600	010011010	1600	0000001011011
1664	011000	1664	0000001100100
1728	010011011	1728	0000001100101

Make-up codes between 1792 and 2560

Run length (black and white)	Make-up codes
1792	00000001000
1856	00000001100
1920	00000001101
1984	000000010010
2048	000000010011
2112	000000010100
2176	000000010101
2240	000000010110
2304	000000010111
2368	000000011100
2432	000000011101
2496	000000011110
2560	000000011111

2.2.4 *Coding procedure*

The coding procedure identifies the coding mode that is to be used to code each changing element along the coding line. When one of the three coding modes has been identified according to Step 1 or Step 2 mentioned below, an appropriate code word is selected from the code table given in Table 1/T.6. The coding procedure is as shown in the flow diagram of Figure 5/T.6.

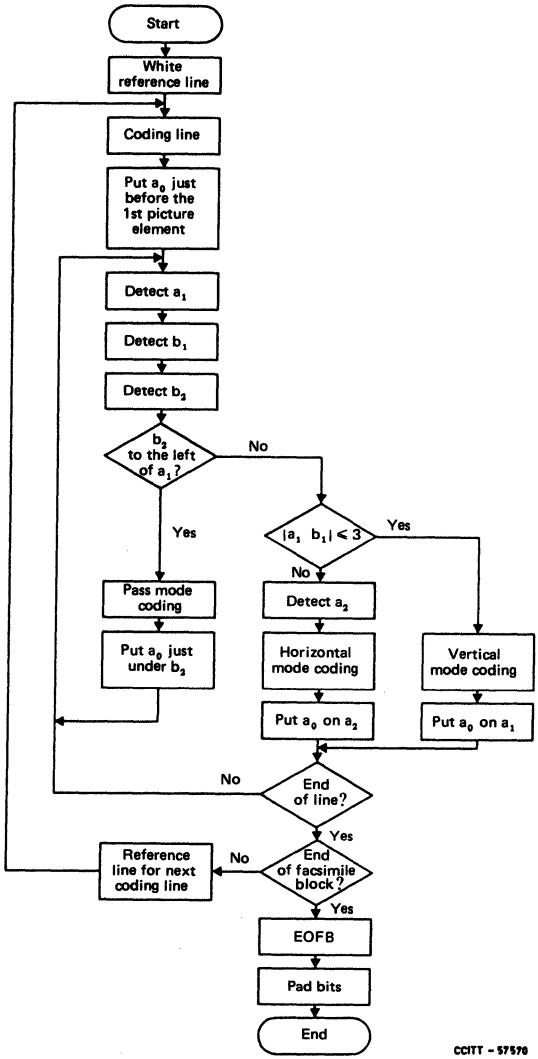


FIGURE 5/T.6

Coding flow diagram

Step 1

i) If a pass mode is identified, this is coded using the word 0001 (Table 1/T.6). After this processing, picture element a'_0 just under b_2 is regarded as the new starting picture element a_0 for the next coding (see Figure 2/T.6).

ii) If a pass mode is not detected, then proceed to Step 2.

Note – It does not affect compatibility to restrict the use of pass mode in the encoder to a single pass mode. Variations of the algorithm which do not affect compatibility should be the subject of further study.

Step 2

i) Determine the absolute value of the relative distance a_1b_1.

ii) If $|a_1b_1| \leq 3$, as shown in Table 1/T.6, a_1b_1 is coded by the vertical mode, after which position a_1 is regarded as the new starting picture element a_0 for the next coding.

iii) If $|a_1b_1| > 3$, as shown in Table 1/T.6, following horizontal mode code 001, a_0a_1 and a_1a_2 are respectively coded by one-dimensional run length coding.

Run lengths in the range of 0 to 63 pels are encoded with their appropriate terminating code word of Table 2/T.6. Note that there is a different list of code words for black and white run lengths. Run lengths in the range of 64 to 2623 pels are encoded first by the make-up code word representing the run length which is nearest, not longer, to that required. This is then followed by the terminating code word representing the difference between the required run length and the run length represented by the make-up code. Run lengths in the range of lengths longer than or equal to 2624 pels are coded first by the make-up code of 2560. If the remaining part of the run (after the first make-up code of 2560) is 2560 pels or greater, additional make-up code(s) of 2560 are issued until the remaining part of the run becomes less than 2560 pels. Then the remaining part of the run is encoded by terminating code or by make-up code plus Terminating code according to the range as mentioned above.

After this processing, position a_2 is regarded as the new starting picture element a_0 for the next coding.

Note – Coding examples are given in Recommendation T.4, § 4.2.5.

2.2.5 *Processing the first and last picture element in a line*

2.2.5.1 *Processing the first picture element*

The first starting picture element a_0 on each coding line is imaginarily set at a position just before the first picture element, and is regarded as a white picture element (see § 2.2.2).

The first run length on a line a_0a_1 is replaced by $a_0a_1 - 1$. Therefore, if the first actual run is black and is deemed to be coded by horizontal mode coding, then the first code word $M(a_0a_1)$ corresponds to an imaginary white run of zero length (see Figure 10/T.4).

2.2.5.2 *Processing the last picture element*

The coding of the coding line continues until the position of the imaginary changing element situated just after the last actual element has been coded. This may be coded as a_1 or a_2. Also, if b_1 and/or b_2 are not detected at any time during the coding of the line, they are positioned on the imaginary changing element situated just after the last actual picture element on the reference line.

2.3 *Optional facsimile coding schemes for black and white images*

2.3.1 *Uncompressed mode*

Uncompressed mode is an optional coding scheme associated to the basic facsimile coding scheme and is used to transmit the image information without data compression techniques as shown in Table 4/T.6.

The extension code in § 2.2.4 with the xxx bits set to 111 is used as an entrance code from the basic coding scheme in § 2.2 to the uncompressed mode.

TABLE 4/T.6

Uncompressed mode code words

Entrance code to uncompressed mode	Basic coding scheme : 0000001111	
	Image pattern	*Code word*
Uncompressed mode code	1 01 001 0001 00001 00000	1 01 001 0001 00001 000001
Exist from uncompressed mode code	0 00 000 0000	0000001T 00000001T 000000001T 0000000001T 00000000001T

T denotes a tag bit which tells the colour of the next run (black = 1, white = 0).

2.4 *Facsimile coding control functions*

2.4.1 *Control functions for basic facsimile coding scheme*

2.4.1.1 *End-of-facsimile block*

The end-of-facsimile block (EDFB) code is added to the end of every coded facsimile block. The format of EOFB is as follows:

Format: 000000000001000000000001
24 bits

2.4.1.2 *Pad bits*

Pad bits may be used after the end-of-facsimile block code if it is necessary to align on octet boundaries or to a fixed block size. The format used is as follows.

Format: Variable length string of 0s.

2.4.1.3 *Extension*

Extension code is used to indicate the change from the current mode to another mode, e.g., another coding scheme.

Format: 0000001xxx,

where xxx = 111 indicates uncompressed mode which is specified in § 2.3.1.

Further study is needed to define other unspecified xxx bit assignments and their use for any further extensions.

3 Optional grey scale facsimile coding schemes and their coding control functions

For further study.

4 Optional colour facsimile coding schemes and their coding control functions

For further study.

Appendix G
STANDARD CCITT COMPRESSION TEST DOCUMENTS

The following eight documents are the test documents used to measure the compression achieved for various types of documents. The results of the tests are given in Table 3-2.

THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/EAC 18th January, 1972.

Dr. P.N. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Figure G-1 Test Document #1

Registered in England: No. 2038
Registered Office: 60 Vicara Lane, Ilford. Essex.

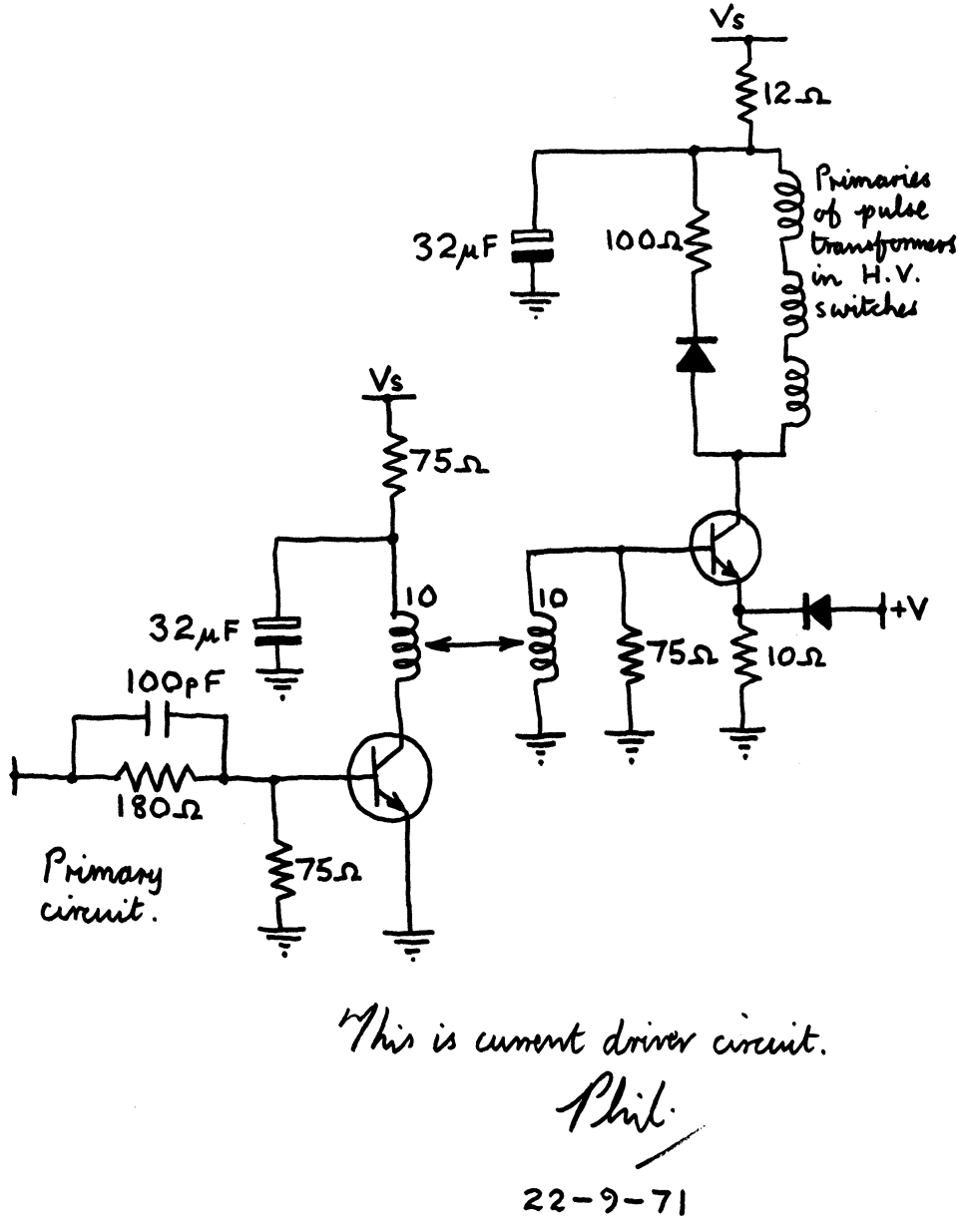


Figure G-2 Test Document #2

ETABLISSEMENTS ABCDEFG
SOCIÉTÉ ANONYME AU CAPITAL DE 300 000 F
20, RUE DU XVUTRSTBSL F 00000 NTBCLAG
Tél. : (35) 24.46.32 Adr. Tg. : NRVLJROLM
Télex : 31596 F IN : 718490070257
Transporteur (ou Transitaire)
M. M. DUPONT Frères
8 quai des bicdfsh F 0000 NTBCLAG

Mot directeur

CLASSEMENT	FACTURE INVOICE	Exemplaire 15	
CODE CLIENT Z 04399	DATE 7-7-74	NUMÉRO 06	FEUILLET 01

Votre commande du 74-2-2 numéro 438
Notre offre AZ/B7 du 74-1-1 numéro 12

LIVRAISON
5, rue XYZ
99000 VILLE

FACTURATION
12, rue ABCD BP 15
99000 VILLE

DOMICILIATION BANCAIRE DU VENDEUR
CODE BANQUE CODE GUICHET COMPTE CLIENT

PAYS D'ORIGINE PAYS DE DESTINATION
CONDITIONS DE LIVRAISON DATE 74-03-03
LICENCE D'EXPORTATION NATURE DU CONTRAT (monnaie)
CONDITIONS DE PAIEMENT (échéance, %...) FAB

TRANSPORTS ORIGINE	DESTINATION	MODE
Pays 1	Etat 2	Air

MARQUES ET NUMÉROS MARKS AND NUMBERS	NOMBRE ET NATURE DES COLIS : DÉNOMINATION DE LA MARCHANDISE NUMBER AND KING OF PACKAGES: DESCRIPTION OF GOODS	NOMENCLATURE STATISTICAL No.	MASSE NETTE NET WEIGHT / MASSE BRUTE GROSS WEIGHT	VALEUR VALUE / DIMENSIONS MEASUREMENTS
74.21.456.44.2 A	1 Composants	U 123/4	5 kg / 8 kg	1400 X 13x10x6

QUANTITÉ COMMANDÉE ET UNITÉ QUANTITY ORDERED AND UNIT	N° ET RÉF. DE L'ARTICLE	DÉSIGNATION	QUANTITÉ LIVRÉE ET UNITÉ QUANTITY DELIVERED AND UNIT	PRIX UNITAIRE UNIT PRICE	MONTANT TOTAL TOTAL AMOUNT
2	AF-809	Circuit intégré	2	104,33 F	208,66 F
10	S8-T4	Connecteur	10	83,10 F	831,00 F
25	ZI07	Composant indéterminé	20	15,00 F	300,00 F

Costs	Débours	Inclus	Non inclus
Packing	Emballages		92,14
Freight	Transport		
Insurance	Assurances		
Total invoice amount	Montant total de la facture		1431,80
Installment	Acomptes		
NET TO BE PAID	NET A RÉGLER		1431,80

Figure G-3 Test Document #3

L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "en bloc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs globaux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 6 à 10 personnes représentant les services les plus divers concernés par le projet,et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 300 TI à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale sur un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartie sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger,et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gèrera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 50 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Figure G-4 Test Document #4

Cela est d'autant plus valable que $T\Delta f$ est plus grand. A cet égard la figure 2 représente la vraie courbe donnant $|\phi(f)|$ en fonction de f pour les valeurs numériques indiquées page précédente.

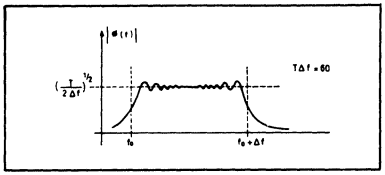


FIG. 2

Dans ce cas, le filtre adapté pourra être constitué, conformément à la figure 3, par la cascade :

— d'un filtre passe-bande de transfert unité pour $f_0 \leqslant f \leqslant f_0+\Delta f$ et de transfert quasi nul pour $f < f_0$ et $f > f_0+\Delta f$, filtre ne modifiant pas la phase des composants le traversant ;

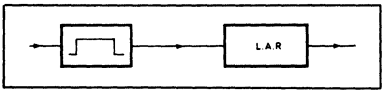


FIG. 3

— filtre suivi d'une ligne à retard (LAR) dispersive ayant un temps de propagation de groupe T_R décroissant linéairement avec la fréquence f suivant l'expression :

$$T_R = T_0+(f_0-f)\frac{T}{\Delta f} \quad \text{(avec } T_0 > T\text{)}$$

(voir fig. 4).

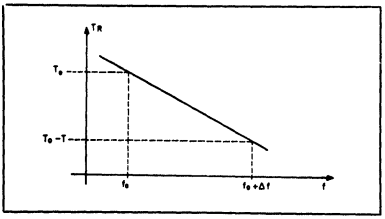


FIG. 4

telle ligne à retard est donnée par :

$$\varphi = -2\pi\int_0^f T_R\,df$$

$$\varphi = -2\pi\left[T_0+\frac{f_0T}{\Delta f}\right]f+\pi\frac{T}{\Delta f}f^2$$

Et cette phase est bien l'opposé de $\underline{/\phi(f)}$,

à un déphasage constant près (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0+\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et dont l'enveloppe a la forme indiquée à la figure 5, où l'on a représenté simultanément le signal $S(t)$ et le signal $S_1(t)$ correspondant obtenu à la sortie du filtre adapté. On comprend le nom de récepteur à compression d'impulsion donné à ce genre de filtre adapté : la « largeur » (à 3 dB) du signal comprimé étant égale à $1/\Delta f$, le rapport de compression est de $\dfrac{T}{1/\Delta f} = T\Delta f$

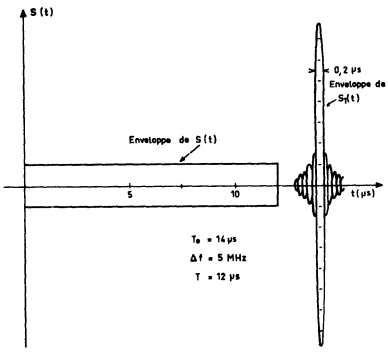
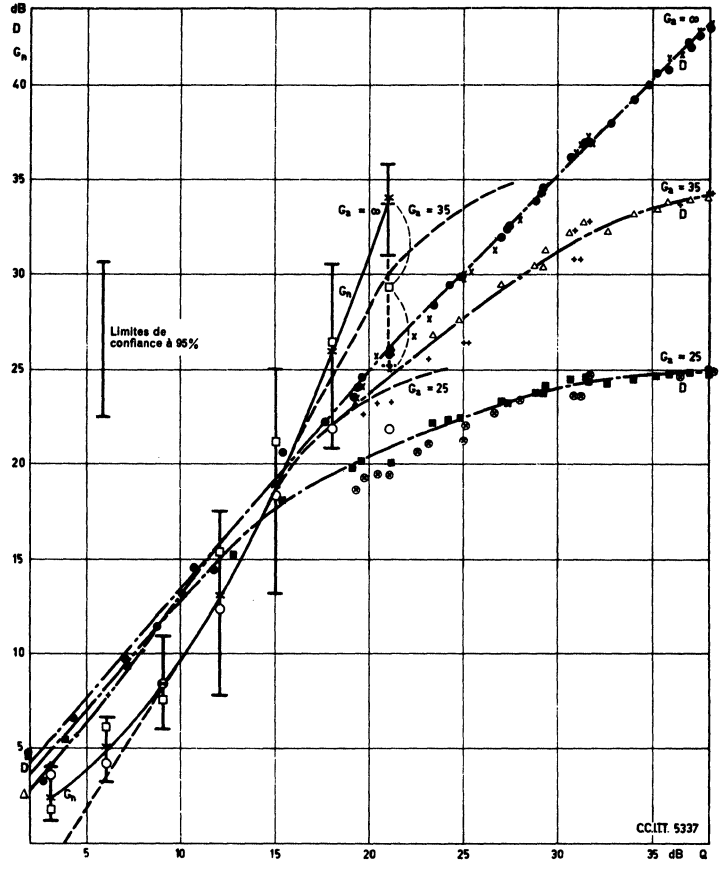


FIG. 5

On saisit physiquement le phénomène de compression en réalisant que lorsque le signal $S(t)$ entre dans la ligne à retard (LAR) la fréquence qui entre la première à l'instant 0 est la fréquence basse f_0, qui met un temps T_0 pour traverser. La fréquence f entre à l'instant $t = (f-f_0)\dfrac{T}{\Delta f}$ et elle met un temps $T_0-(f-f_0)\dfrac{T}{\Delta f}$ pour traverser, ce qui la fait ressortir à l'instant T_0 également. Ainsi donc, le signal $S(t)$

Figure G-5 Test Document #5



Courbes adaptées G_a (essais subjectifs) pour
$G_a = \infty$ $G_a = 35$dB $G_a = 25$dB
[10] ——— × □ ○
[23] — — — —

Points calculés D (Q, G_a) pour
$G_a = \infty$ $G_a = 25$ $G_a = 35$ dB
● ■ △ – dans la partie montante
× ⊕ + – dans la partie descendante

Courbes ——×— —×——△ D (Q, G_a)

FIGURE 3

TOME V — Question 18/XII, Annexe 6

Figure G-6 Test Document #6

ＣＣＩＴＴの概要

沿革

ＣＣＩＴＴは、国際電気通信連合（ＩＴＵ）の四つの常設機関（事務総局、国際周波数登録委員会、ＣＣＩＲ、ＣＣＩＴＴ）の一つとして、ＩＴＵの中でも、世界の国際通信上の諸問題を真先に取上げ、その解決方法を見出して行く重要な機関である。日本名は、国際電信電話諮問委員会と称する。

ＣＣＩＴＴの前身は、ＣＣＩＦ（国際電話諮問委員会）とＣＣＩＴ（国際電信諮問委員会）である。ＣＣＩＦは、１９２４年にヨーロッパに「国際長距離電話通信諮問委員会」が設置され、これが１９２５年のパリ電信電話会議のとき、正式に、「国際電話諮問委員会」として万国電信連合の公式機関となったものである。ＣＣＩＴは、同じく１９２５年の会議のとき、ＣＣＩＦと併立するものとして設置された。

そして、ＣＣＩＦは、１９５６年の12月に第18回総会が開催されたのち、ＣＣＩＴは、同年同月に第８回総会が開催されたのち、併合されて現在のＣＣＩＴＴとなった。このＣＣＩＴＴは、ＣＣＩＦとＣＣＩＴが解散した直後、第１回総会を開催し、第２回総会は、１９６０年にニューデリーで、第３回総会は、１９６４年、ジュネーブで、第４回総会は、１９６８年、アルゼンチンで開催された。

ＣＣＩＦとＣＣＩＴが合併したのは、有線電気通信の分野、とくに伝送路について電信回線と電話回線とを技術的に分ける意味がなくなってきたこと、各国とも大体において、電信部門と電話部門は同一組織内にあること、ＣＣＩＦの事務局とＣＣＩＴの事務局の合併による能率増進等がおもな理由であった。

ＣＣＩＴＴは、上述のように、ヨーロッパ内の国ぐにによって、ヨーロッパ内の電信・電話の技術・運用・料金の基準を定め、あるいは統一をはかってきたので、現在でも、その影響を受け、会合参加国は、ヨーロッパの国が多く、ヨーロッパで生起する問題の研究が多い。たとえば、１９６０年のＣＣＩＴＴ勧告の中で、技術上配慮する距離は約２，５００㎞であったが、これはヨーロッパ内領域を想定したものである。

しかしながら、１９５６年９月に敷設された大西洋横断電話ケーブルは、大陸間電話通信の自動化および半自動化への技術的可能性を与え、ＣＣＩＴＴがこの問題を取り上げるに及び、ＣＣＩＴＴの性格は漸次、汎世界的色彩を実質的に帯びるに至った。この汎世界的性格は第２次世界大戦後目ざましくなったアジア・アフリカ植民地の独立に伴ってＩＴＵの構成員の中にこれらの国が加わり、ＩＴＵの中に新しい意見が導入されたことにも起因して、技術面、政治面の双方から導入されてきた。ＣＣＩＴＴの汎世界化は、１９６０年の第２回総会がニューデリーで開催されたことにもあらわれている。この総会までは、ＣＣＩＴ、ＣＣＩＦのいずれにしろ、アメリカやアジアで総会が開催されたことがなく、ＣＣＩＴＴ委員長も、ニューデリー総会の準備文書で、この点には注目すべきであるとのべている。

任務

ＩＴＵは、全権委員会議、主管庁会議を始めとして、七つの機関をもち、それぞれの機関の権限と任務は国際電気通信条約に明記されている。そこで条約を参照してみるならば、ＣＣＩＴＴの任務は、つぎのとおりとなっている。

「国際電信電話諮問委員会（ＣＣＩＴＴ）は、電信および電話に関する技術、運用および料金の問題について研究し、および意見を表明することを任務とする。」（１９６５年モントルー条約第１８７号）

「各国際諮問委員会は、その任務の遂行に当たって、新しい国または発展の途上にある国における地域的および国際的分野にわたる電気通信の創設、発達および改善に直接関連のある問題について研究し、および意見を作成するように妥当な注意を払わなければならない。」（同第１８８号）

「各国際諮問委員会は、また、関係国の要請に基づき、その国内電気通信の問題について研究し、かつ、勧告を行なうことができる。」（同第１８９号）

上記第１８７号と第１８８号にいわれる「意見」とは、フランス語の Avis から訳したもので、英語では、「勧告（Recommendation）」となっている。ＣＣＩＴＴの表明する意見は、国際法的には強制力をもたないものであって、この点が、条約、電信規則、電話規則等各国を拘束する力をもっているものと異なる。もっとも意見とは称しても、技術的分野では、電信規則のごとき、各国政府が承認してその内容を実施する強制規則をもたないので、実際にある機器の仕様を定める場合には、多くの国の意見が統一されたこの「意見」に従わなければ、円滑な国際通信を行なうことができない場合が多い。この意見（または勧告）は、国際通信を行なう場合各国が直面する問題について、具体的意見を表明するもので、たとえば、大陸間ケーブルで大陸間通話を半自動化しようとする場合、その信号方式や取り扱う通話の種類および料金は、どのようにするかを研究して意見を表明する。したがって、ＣＣＩＴＴの活動は、つねに時代の最先端を行くもので、ＣＣＩＴＴの活動方向は、そのまま世界の国際通信の活動方向であるともいえる。

この意見は、また、電信規則以下のその他の規則のごとく、数年以上の間隔をもって開催される主管庁会議というような大会議の決定をまたなくても表明することができ、また、その改正も容易であるので、現在のように進歩の早い国際通信界では、関係国の意見を統一した国際的見解としては非常に便利である。

Figure G-7　Test Document #7

memorandum

FROM: A.P. Sproggs Research	TO: G.V. Smith Project Planning
TEL: EXTN: 2041	DATE: 1-9-71

We know that, where possible, data is reduced to alphanumeric form for transmission by communication systems. However, this can be expensive, and also some data must remain in graphic form. For example, we cannot key-punch an engineering drawing or weather map.

I think we should realise that high speed facsimile transmissions are needed to overcome our problems in efficient graphic data communication. We need research into graphics data compression.

Any comments?

Albert.

Figure G-8 Test Document #8

Appendix H

DATA ERROR RECOVERY PROCEDURE

This error recovery procedure only applies to Group 3 processing. Group 4 code is assumed to be error free at the chip level. Group 4 error correction is done at a higher level.

The expander checks the expanded data length for Group 3 (EOL = 0). If the expanded data length is not equal to the effective line length of L = (EPWR • 8) • (EWR + 1), then the DER bit in the ESR is set to one. Unrecognizable codes and illegal EOLs also cause the DER bit to be set.

This error recovery procedure is based on the assumption that the expander destination buffer size is an integral multiple of the effective line length, i.e. EDWCR = n • EPWR • (EWR+1). Under this assumption, no premature destination overflow occurs unless a transmission error causes it. Therefore, a premature destination overflow can be handled in the same way as a data error.

The error recovery procedure must accomplish the following tasks:

1. Replace the erroneously decoded line in the destination buffer with a copy of the line preceding it.

2. Clear the decoding registers in the expander by doing a software reset (Set the Operation Control (OC) field in the CMCR to "00").

3. Save the current value of the ESCAR in a variable (named OSCA in this discussion).

4. Set the ESA bit to 1 so that the expansion processor will look for a starting EOL before processing another line.

5. Back up the ESCAR four bytes to place it ahead of the EOL ending the last line (the line with the error) so that this EOL can be used as the starting EOL.

6. Set the GO bit and thereby process another line. If another error does not occur, continue processing. If another error stops processing, proceed with step 7.

7. In case the line with the error is less than four bytes long, backing up the ESCAR by four bytes places it at a point in the line preceding the bad line and the EOL ending this good line will be used to restart the expansion. The result is that the bad line is again processed.

The possibility of this happening must be considered to prevent the program from going into an infinite loop. This can be detected by comparing the value in OSCA with the value in ESCAR after processing another line and receiving another data error. If ESCAR is not greater than OSCA, it means that the same bad line has been processed again. In this case, reduce ESCAR by a value progressively less than four and each time process another line to see if ESCAR becomes larger that OSCA indicating that a new line was processed.

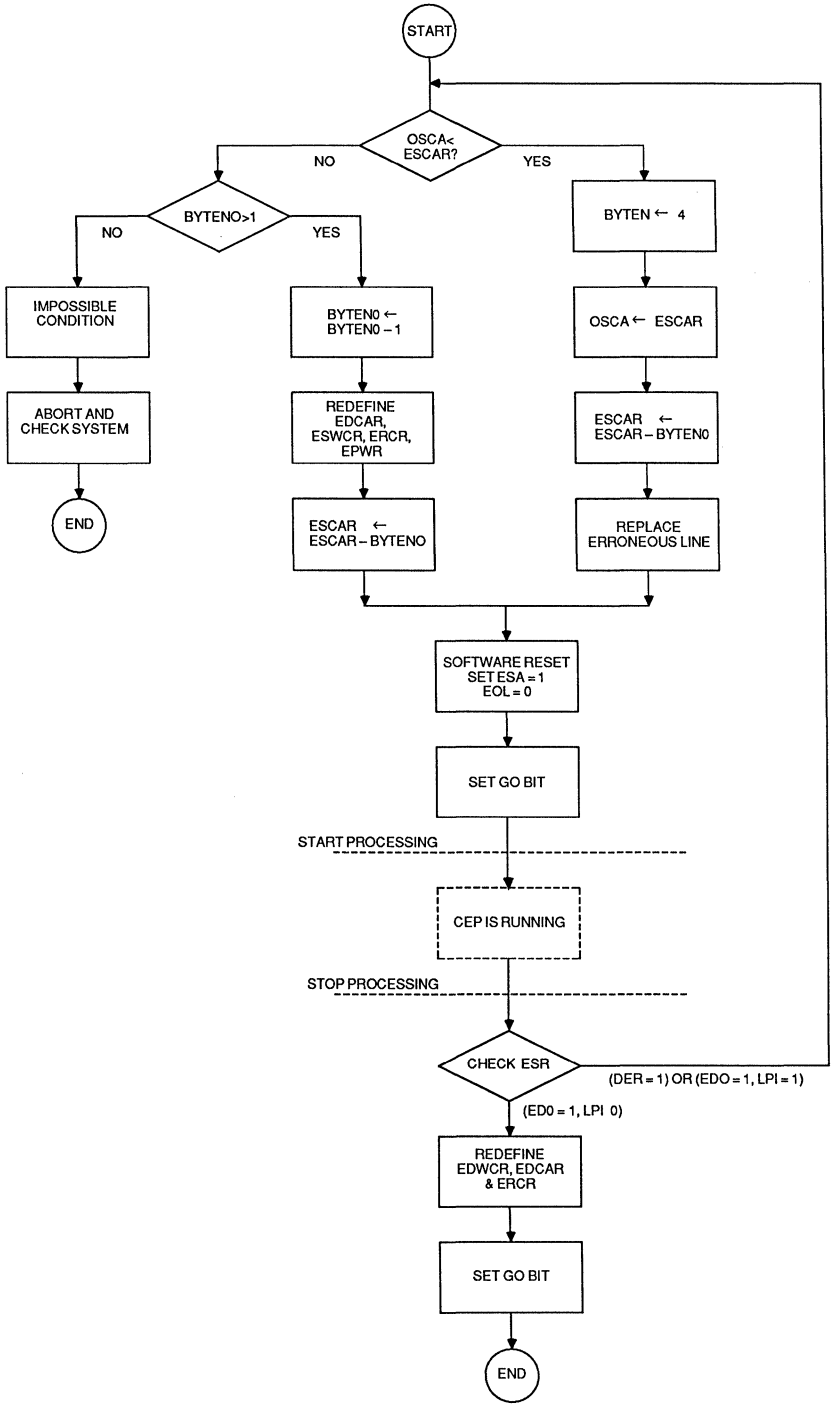


Figure H-1. Error Recovery Flow Diagram

07666A H-1

ADVANCED MICRO DEVICES
DOMESTIC SALES OFFICES

ALABAMA	(205) 882-9122
ARIZONA,	
Tempe	(602) 242-4400
Tucson	(602) 792-1200
CALIFORNIA,	
El Segundo	(213) 640-3210
Newport Beach	(714) 752-6262
San Diego	(619) 560-7030
Sunnyvale	(408) 720-8811
Woodland Hills	(818) 992-4155
COLORADO	(303) 691-5100
CONNECTICUT,	
Southbury	(203) 264-7800
FLORIDA,	
Altamonte Springs	(305) 339-5022
Clearwater	(813) 530-9971
Ft Lauderdale	(305) 484-8600
Melbourne	(305) 254-2915
GEORGIA	(404) 449-7920
ILLINOIS	(312) 773-4422
INDIANA	(317) 244-7207
KANSAS	(913) 451-3115
MARYLAND	(301) 796-9310
MASSACHUSETTS	(617) 273-3970
MINNESOTA	(612) 938-0001
NEW JERSEY	(201) 299-0002
NEW YORK,	
Liverpool	(315) 457-5400
Poughkeepsie (IBM only)	(914) 471-8180
Woodbury	(516) 364-8020
NORTH CAROLINA,	
Charlotte	(704) 525-1875
Raleigh	(919) 847-8471
OREGON	(503) 245-0080
OHIO,	
Columbus	(614) 891-6455
PENNSYLVANIA,	
Allentown (AT&T only)	(215) 398-8006
Willow Grove	(215) 657-3101
TEXAS,	
Austin	(512) 346-7830
Dallas	(214) 934-9099
Houston	(713) 785-9001
WASHINGTON	(206) 455-3600
WISCONSIN	(414) 782-7748

INTERNATIONAL SALES OFFICES

BELGIUM,		
Bruxelles	TEL:	(02) 771 99 93
	FAX:	762-3716
	TLX:	61028
CANADA, Ontario,		
Kanata	TEL:	(613) 592-0090
Willowdale	TEL:	(416) 224-5193
	FAX:	(416) 224-0056
FRANCE,		
Paris	TEL:	(01) 46 87 36 66
	FAX:	(01) 46 86 21 85
	TLX:	202053F
GERMANY,		
Hannover area	TEL:	(05143) 50 55
	FAX:	5553
	TLX:	925287
München	TEL:	(089) 41 14-0
	FAX:	406490
	TLX:	523883
Stuttgart	TEL:	(0711) 62 33 77
	FAX:	625187
	TLX:	721882
HONG KONG,		
Kowloon	TEL:	3-695377
	FAX:	1234276
	TLX:	50426
ITALY, Milano	TEL:	(02) 3390541
	FAX:	3498000
	TLX:	315286
JAPAN, Tokyo	TEL:	(03) 345-8241
	FAX:	3425196
	TLX:	J24064AMDTKOJ
LATIN AMERICA,		
Ft. Lauderdale	TEL:	(305) 484-8600
	FAX:	(305) 485-9736
SWEDEN, Stockholm	TEL:	(08) 733 03 50
	FAX:	7332285
	TLX:	11602
UNITED KINGDOM,		
Manchester area	TEL:	(0925) 828008
	FAX:	827693
	TLX:	628524
London area	TEL:	(04862) 22121
	FAX:	22179
	TLX:	859103

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA	
I^2 INC	OEM (408) 988-3400 DISTI (408) 498-6868
CONNECTICUT	
SCIENTIFIC COMPONENTS	(203) 272-2963
IDAHO	
INTERMOUNTAIN TECH MKGT	(208) 322-5022
INDIANA	
SAI MARKETING CORP	(317) 241-9276
IOWA	
LORENZ SALES	(319) 377-4666
MICHIGAN	
SAI MARKETING CORP	(313) 227-1786
NEBRASKA	
LORENZ SALES	(402) 475-4660
NEW JERSEY	
TAI CORPORATION	(609) 933-2600
NEW MEXICO	
THORSON DESERT STATES	(505) 293-8555
NEW YORK	
NYCOM, INC	(315) 437-8343
OHIO	
Dayton	
DOLFUSS ROOT & CO	(513) 433-6776
Strongsville	
DOLFUSS ROOT & CO	(216) 238-0300
PENNSYLVANIA	
DOLFUSS ROOT & CO	(412) 221-4420
UTAH	
R^2 MARKETING	(801) 595-0631

ADVANCED MICRO DEVICES 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 5388450

ADVANCED MICRO DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale,
California 94088
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

Order #07666A

IH-MU-5M-1/86-0