A Digital Color Translation System

by

Edward Ashford Lee

B.S. Yale University
(1979)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS OF THE
DEGREE OF
MASTER OF SCIENCE IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

October 1980
(1.8. February, 1981)

© Edward Ashford Lee 1980

The author hereby grants to M.I.T. permission to reproduce and
to distribute copies of this thesis document in whole or in part.

Signature of Author

Department of Electrical Engineering and Computer Science
October 30, 1980

Certified by

William F. Schreiber
Thesis Supervisor

Accepted by

Chairman, Departmental Graduate Committee

ARCHIVES
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
MAY 6 1981
LIBRARIES
# TABLE OF CONTENTS

List of Illustrations ............................................. 3  
Abstract .................................................................... 4  
Acknowledgements ....................................................... 5  
Forward ..................................................................... 6  
Introduction .................................................................. 7  
Color Translation .......................................................... 9  
Color ......................................................................... 14  
  Luminance / Chrominance Color Spaces ......................... 16  
  Characteristics of the L-C1-C2 Color Space ..................... 24  
Cartesian to Polar Conversion .......................................... 28  
Polar to Cartesian Conversion .......................................... 35  
Selective Color Correction .............................................. 40  
Special Color Correction ................................................ 52  
  Notes on Design ....................................................... 57  
The System .................................................................... 68  
Suggestions for Further Work ........................................... 78  
Footnotes ..................................................................... 83  
Bibliography .................................................................. 84  

**APPENDICES**

A. Pictures ................................................................. 85  
B. Computer Programs - Simulation and PROM contents generators .. 88  
C. Schematics ............................................................... 104
LIST OF ILLUSTRATIONS

1. Color Translation Module Block Diagram ......................... 13
2. Photopic Luminosity Function .................................... 21
3. l,c1,c2 Primaries in r,g,b Space .............................. 22
4. r,g,b Hexahedron in l,c1,c2 space ............................. 23
5. Cross Sections of r,g,b Hexahedron in l,c1,c2 Space ............ 27
6. A Digital Color Wheel ............................................ 33
7. Cartesian to Polar Converter Functional Block Diagram ...... 34
8. Relative Domains of Polar and Rectangular Representations ... 38
9. Polar to Cartesian Converter Functional Block Diagram ...... 39
10. Control Panel Diagram ........................................... 48
11. Blue and Cyan Weighting functions ............................. 49
12. Even and Odd Weighting Functions (SCC) ....................... 50
13. Selective Color Correction Functional Block Diagram ......... 51
14. User Defined Chromatic Neighborhood .......................... 65
15. Special Color Correction Functional Block Diagrams
   Knob Processor .................................................. 66
   Video Processor .................................................. 67
16. Color Translation Displays ....................................... 74
17. First Test Configuration .......................................... 75
18. Second Test Configuration ....................................... 76
19. Backplane Configuration ......................................... 77
A DIGITAL COLOR TRANSLATION SYSTEM

by

EDWARD ASHFORD LEE

Submitted to the Department of Electrical Engineering
and Computer Science on Oct. 31, 1980
in partial fulfillment
for the Degree of Master of Science in
Electrical Engineering and Computer Science

ABSTRACT

A technique has been designed and developed for the precise control of colors in polychromatic imaging systems. A study of the theory of color spaces showed that a reasonable system could be constructed using digital hardware to permit an operator to alter precisely the chromatic characteristics of an image with real time visual feedback on a color T.V. monitor. A system was designed to modify selectively the hue and saturation of colors in images encoded digitally in a Luminence/Chrominance color space. A transformation of the chrominance information from cartesian to polar form made hue and saturation information available to the rest of the system. Using this information, a Selective Color Correction (SCC) module individually controlled the chrominance of pels (pixels) falling in each of seven hue domains. A Special Color Correction (SpCC) module controlled the chrominance of pels falling within a user defined chromatic neighborhood. A prototype was built and demonstrated.

Thesis Supervisor: Dr. William Schreiber

Title: Professor of Electrical Engineering
and Computer Science
ACKNOWLEDGEMENTS

I am deeply indebted to Esteban Peynado whose meticulous and organized working style was invaluable to our partnership in this project. Gratitude is also due to Prof. William Schreiber and Finley Shapiro for their preliminary work on the problem, and to Sam Goldwasser and Jim Cyr for their assistance during the prototype construction phase of the project. Credit for the design and construction of the control panel used to demonstrate the system goes to Sudhindra Nath Mishra.
FORWARD

The project described, because of its magnitude and the limited time available for completion, was divided between the author and another M.I.T. graduate student, Esteban Peynado. Although there was certainly a division of responsibilities, close corroboration was essential because of the unified nature of the project. The system is discussed here in its totality, but discussions of the hardware implementations concentrate on those for which the author was responsible. See the S.M. thesis of Mr. Peynado for further discussions of the hardware.
INTRODUCTION

Color is such an integral part of our everyday existence that we frequently do not notice the rich and subtle nuances that appear in even the most mundane of our surroundings. Normal human vision has the ability, however, to discern amazingly slight differences in color. Subtle shades can, in certain circumstances, strongly effect our perception of a scene, just as subtle shades of meaning in a word can change our understanding of a sentence. The remarkable acuity of human vision presents quite a challenge to technologies attempting to accurately reproduce colors, such as television, photography, and color printing.

In fact, truly accurate color reproduction, especially in large quantities, is a monumental task. The difficulty is not strictly limited, however, to the high color resolution of our color perception. The task is further complicated by the versatility of human vision in adapting to itself to widely varying light conditions. For example, although incandescent light differs in spectrum from white light, we do not generally notice serious color distortion under such light because our vision adapts to the different spectral distribution. A photograph taken under incandescent light, however, shows considerable color distortion. When we view the photograph, our vision is adapted to the ambient light, not to the light being reproduced by the photograph. Human vision has the ability, it would appear, to alter its reference point in color definition whereas straightforward color reproduction generally responds to fixed colors in a fixed way.
Most of the color reproduction technologies introduce color distortion simply because of the physical properties of the materials or processes used. Kodachrome film, for example, darkens the blues of the sky and brightens most other colors. Color printing generally involves multiple processes, each of which introduces its own form of distortion. In gravure printing for example, photographic processes begin the distortion. The scanning process, which converts the image into electrical signals, and the engraving and plating of the copper cylinder add to the distortion. The transfer of ink onto the cylinders and then onto the paper as well as the chemical interactions of the different colored inks further distort reproduction. Finally, the color and texture of the paper can also degrade the image. Needless to say, an image processed in this way rarely looks the same on paper as it did in life. The printing industry generally uses complex photographic masking techniques to compensate for the deficiencies of the system. Even skilled craftsmen with considerable experience must frequently rely on trial and error to arrive at the proper compensation.
COLOR TRANSLATION

The Digital Color Translation System is part of a larger project supported by the Providence Gravure Company in Rhode Island. The goal of the project is to facilitate color control in color printing using digital image processing techniques. The system permits an operator to alter precisely the colors in an image via a control panel and to observe immediately on a color monitor the results of his changes. The possible applications of such a system, however, extend well beyond the printing industry into all forms of picture processing, including television. For this reason, in the design, less emphasis is placed on applications orientation, and maximum flexibility is maintained. Nonetheless, since demand for such capability is greatest in the printing industry, its needs form the basis for the applications model.

The uses of color translation are actually threefold:

1. It compensates for systematic deficiencies inherent in the transfer of an image from one medium to another.

2. It compensates for color errors in the source image such as the original photograph.

3. It permits aesthetic changes in the image.

The color translation system separates these needs. Compensation for systematic errors (called company translation) will be accomplished digitally via a large lookup table which specifies the ink densities required to produce a color defined in some color space. The generation of this table is the subject of a separate graduate thesis. The system
discussed here is concerned with the second and third of these needs. It assumes that systematic errors due to the printing process will be corrected, and allows the user to make aesthetic or subjective changes in the image which can also be used to correct distortion. Most importantly, the system will permit the user to observe his modifications on a T.V. monitor in real time. The question of ensuring that the image on the monitor accurately represents the result of a printing run is part of the company translation project.

The Color Translation Module (CTM) must be able to process picture information at a rate of 10 million picture elements (pixels or pels) per second (10 MHZ). This provides the resolution needed for a good quality image on a standard T.V. monitor. The user defines the parameters for processing via a control panel with knobs for the various functions. A knob processor must handle the information from the control panel at a rate fast enough so that the user observes instantaneous response to his commands. It must make the information available to a video processor which modifies pels at 10MHZ. The fast (video) operations are implemented using TTL technology and pipelining. The system consists of the following components (see fig. 1):

1. A control panel.

2. A Gradation Module: allows the user to make changes in the characteristics of the image which are independent of color (e.g. contrast). It resembles existing modules in monochrome systems and does not form part of this thesis.

3. Neutral Color Balance: allows the user to change the overall color cast of an image as well as the color casts in the highlights, midranges, or shadows of the picture. This module
has been designed and built by Finley Shapiro.

4. Cartesian to Polar Converter (CPC): converts color information into a form usable by the following two modules.

5. Selective Color Correction (SCC): allows the user to make changes to all pels in an image which fall within any of seven hue domains (Magenta, Red, Orange, Yellow, Green, Cyan, and Blue). For example, saturation of all reds in the picture may be reduced, or the oranges may be made more yellow.

6. Special Color Correction (SpCC): allows the user to define his own hue domain (or saturation domain) to be altered. In this manner, changes to very specific colors in an image may be made.

7. Polar to Cartesian Converter (PCC): reverses the process of the Cartesian to Polar Converter.

The full range of applications for such a system has yet to be explored. When the system was first demonstrated (see Appendix A for pictures) the most obvious application was in generating special effects. The kind of flexibility the system demonstrated could overcome many of the obstacles to creative control of an image. More importantly, however, subtle changes are well within the realm of possibility. If the system is put under computer control, then there is no inherent limit to its resolution capability. Because of speed considerations the resolution of the monitored image is limited if the real time response feature is to be maintained. Once a translation has been decided on by monitoring a lower resolution image, however, the same transformation may be applied to an image of arbitrarily high resolution. With high resolution and precise control of color nuances, the
system becomes usable for virtually any imaging application.
COLOR

An understanding of the theory of color spaces is essential to any discussion of electronic processing of color. In order to be processed, a color must be represented in some way comprehensible to the processor. Color is fundamentally a human perception of an electromagnetic waveform with some power spectrum, \( C(f) \), where \( f \) denotes a range of frequencies within the visible spectrum. The function \( C(f) \) is a continuous function in frequency and is therefore an awkward way to represent a color. One property of the human visual system, however, makes digital representation of colors considerably easier. Any color with a power spectrum \( C(f) \) can be exactly visually matched by a linear combination of three other properly selected colors called primaries. This means that colors need not have identical power spectrums to appear identical to human eyes. It implies that if the three primary colors which are selected are considered to be basis functions for a vector space encompassing all colors, then properly selected basis functions should span the entire visible color space \(^1\). Thus a color can be represented as a triplet or a vector referenced to a defined basis. Suppose, for example, that the three primaries selected are purely monochromatic colors represented by the symbols \( \chi, \upsilon, \) and \( \zeta \). Then any arbitrary color \( (c) \) can be represented by the triplet \((X, Y, Z)\) such that

\[
    c \equiv X\chi + Y\upsilon + Z\zeta
\]

where "\( \equiv \)" represents a color match rather than an exact spectral equivalence. In practice, primaries which span the entire visible color
space do not exist. Any real system must compromise by selecting primaries which span most of a subspace of interest. Color television displays work on this principle. Phosphors are selected which, when struck by an electron beam, glow red, green, or blue. All other colors are formed by the additive effects of the three phosphors. Neutral tones, such as white, are formed by the additive effects of equal amounts of the three colors. Any real phosphor, however, does not emit purely monochromatically, so its color is limited in saturation (spectral purity). Consequently, the range of colors reproducible by the T.V. phosphors is limited by the lack of saturation in the primaries, in addition to being limited by the use of a finite number of primaries. In practice, however, the range is large enough to be useful. If we represent the colors emitted by the phosphors when struck by some unit electron beam as \( r, g, \) and \( b \), then any color reproducible on a T.V. screen can be represented as a triplet or vector \((R,G,B)\) such that

\[
c \equiv R \times r + G \times g + B \times b
\]

where again, "\( \equiv \)" refers to a color match.
LUMINANCE / CHROMINANCE COLOR SPACES

In the above example, the primaries were determined by the physical characteristics of a television screen. We can, however, select any other set of primaries based on any other criterion, and still use these as a basis for a vector space spanning a range of colors. Representations based on phychovisual properties rather than T.V. phosphors, for example, might be useful. One such representation is referred to as the L-a-b cube-root color coordinate system and is intended to provide a relatively accurate measure of color in agreement with a psychovisually oriented color system called the Munsell color system. The first element, L, is approximately proportional to brightness sensation. The second and third elements, a and b, jointly describe the hue and saturation of the color. This agrees with Munsell color space in which the vertical axis represents rising brightness; in the horizontal planes, the distance from the vertical axis represents the saturation, and the angle around the axis is related to the hue. The coordinates a and b define the location of a color within this plane. L-a-b space has the disadvantage, however, that it is non-linearly related to r-g-b space. Color signals are likely to originate in r-g-b space (or some linear transformation thereof), and hardware limitations make it difficult to perform this nonlinear transformation at video speeds. Linear transformations are computationally much more practical, and in some cases can be quite useful. The N.T.S.C. Coordinate system (called Y-I-Q) used to transmit color television signals is such a linear transformation of r-g-b.

For use in the Color Translation Module (CTM) a similar coordinate
system called L-C1-C2 was developed. The objective was that, as in the
Y-I-Q system, one element (Y in this case, L in L-C1-C2) would be
approximately related to the luminance of a color, and the remaining two
elements would approximately jointly specify the hue and saturation of
the color. The transformation to such Luminance/Chrominance coordinates
from r-g-b is equivalent to the linear transformation:

\[
\begin{bmatrix}
L \\
C1 \\
C2
\end{bmatrix}
= B
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

where B is a 3x3 matrix. The triplet (L,C1,C2) specifies a color in
terms of a new set of primaries which we will denote l, c1, and c2. The
primaries l, c1, and c2 form a new basis which spans the color space.
The color c is therefore matched as follows:

\[
c \equiv L \times l + C1 \times c1 + C2 \times c2 \\
\equiv R \times r + G \times g + B \times b
\]

To compute the contents of the matrix B, the chromaticities of the phos-
phors of the 51cm BARCO Standard NTSC tube, and for a reference white a
5000 K light source such as the Macbeth F40T12/PL50 were used. Recall
that we wish to design the transformation so that L is related to the
luminance (the physical counterpart of perceptual brightness sensation)
of a color. The luminous flux of a color measured in lumens is defined
as:

\[
F = K \int_0^\infty C(\lambda)V(\lambda)d\lambda
\]

where C(\lambda) is the spectral energy distribution of the color as a func-
tion of the wavelength \( \lambda \), and \( K \) is 685 lumen/W. \( V(\lambda) \) is called the pho-
toptic luminosity function (or the relative luminous efficiency) and is
defined to be the spectral sensitivity of the C.I.E. "standard observer"
(see fig. 2)\textsuperscript{4}. It describes the relative sensitivity of a human observer to monochromatic light as a function of wavelength. For a meaningful definition of luminance, however, we reference it to a standard white. If A (W), A (R), and A (B) refer to the relative amounts of the three primaries r, g, and b needed to match our reference white, then we define the luminosity coefficients of the primaries as:

\begin{align*}
L(r) &= \int_A^{} A(W)r(\lambda)V(\lambda)d\lambda \\
L(g) &= \int_A^{} A(R)g(\lambda)V(\lambda)d\lambda \\
L(b) &= \int_A^{} A(B)b(\lambda)V(\lambda)d\lambda
\end{align*}

where \(r(\lambda), g(\lambda),\) and \(b(\lambda)\) describe the power spectral distribution of the primary colors and \(V(\lambda)\) is the photopic luminosity function. The luminance of a color \(c\) matched by the primaries is

\[L(c) = R \times L(r) + G \times L(g) + B \times L(b)\]

With these references and phosphors, the luminance of a color represented by an \((R,G,B)\) triplet was found to be

\[L = 0.299 \times R + 0.621 \times G + 0.920 \times B\]

Two chrominance signals (similar in concept to the ones used by the PAL system\textsuperscript{5} for television transmission in Europe) were defined as:

\begin{align*}
J &= R - L = 0.701 \times R - 0.621 \times G - 0.080 \times B \\
K &= B - L = -0.2999 \times R - 0.621 \times G + 0.920 \times B
\end{align*}

The chrominance signals C1 and C2 are simply scaled versions of J and K such that C1 and C2 lie within a range which can be represented by 8 bits (-127 to 127). The \((R,G,B)\) to \((L,C1,C2)\) transformation is therefore\textsuperscript{6}:

\[\text{-18-}\]
\[
\begin{bmatrix}
L \\
C1 \\
C2
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.621 & 0.080 \\
0.498 & -0.442 & -0.056 \\
-0.162 & -0.336 & 0.498
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

The inverse transformation is:

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
1 & 1.406 & 0 \\
1 & -0.679 & -0.219 \\
1 & 0 & 1.844
\end{bmatrix}
\begin{bmatrix}
L \\
C1 \\
C2
\end{bmatrix}
\]

The ranges of the digital signals are:

\[0 < R, G, B, L < 256\]
\[-127 < C1, C2 < 127\]

It becomes a simple matter now to determine what the relationship is between the r-g-b basis and the l-c1-c2 basis. To find a triplet in r-g-b space representing l we simply set

\[(L, C1, C2) = (1, 0, 0)\]

and solve for (R, G, B). We find that

\[(R, G, B) = (1, 1, 1)\]

Which means that

\[l \equiv r + g + b\]

where again "\(\equiv\)" refers to a match rather than an equivalence. By the same approach, we find that c1 can be represented as the triplet \((1.407, -0.677, 0)\) and c2 as \((0, -0.237, 1.848)\) in r-g-b space. These triplets are plotted in fig. 3. Observe that the primaries c1 and c2 are not actually realizable colors, as they contain negative amounts of the r, g, and b primaries. Furthermore, their luminance is zero. Also, we see that if we consider r, g, and b to be orthogonal basis functions,
then 1, c1, and c2 do not form an orthogonal basis. Note that the orthogonality of the basis functions is an arbitrary attribute without physical significance. Because of this, we can, with equal validity, consider 1, c1, and c2 to be orthogonal. The point is that it makes little sense when considering the relationship between the two vector spaces to consider both sets of basis functions orthogonal.
(a) The Photopic Luminosity Function (also known as the Relative Luminous Efficiency) sketched here specifies the relative sensitivity of a standard human observer to light as a function of wavelength under conditions of high illumination.

(b) The Scotopic Luminosity Function specifies the relative sensitivity of a standard human observer under conditions of low illumination. It is not used as frequently in imaging system design as the Photopic Luminosity Function.

Fig. 2: Photopic and Scotopic Luminosity Functions (From Pratt, p. 53)
Fig. 3: l, c1, and c2 primaries in r-g-b space
Fig 4: r-g-b hexahedron in l-cl-c2 space
CHARACTERISTICS OF L-C1-C2 SPACE

Observe in fig. 3 that limits on the ranges of the R,G, and B signals constrain the representable color space to a cube. The mapping of this cube into l-c1-c2 space determines the ranges of the \((L,C1,C2)\) triplet which can be reproduced by the television phosphors. If we assume that the \((R,G,B)\) triplet is to be represented by three eight bit binary numbers, then any set of values for \((L,C1,C2)\) which leads to an R,G, or B greater than 255 or less than zero clearly cannot be represented in r-g-b space. Since the transformation is linear, the cube must map into another hexahedron in l-c1-c2 space. Furthermore, the vertices of the hexahedron are simply the mappings of the vertices of the cube. This mapping is tabulated below and plotted in fig. 4.

<table>
<thead>
<tr>
<th>VERTEX</th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>L</th>
<th>C2</th>
<th>C1</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>black</td>
</tr>
<tr>
<td>1</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>76</td>
<td>127</td>
<td>-41</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>158</td>
<td>-113</td>
<td>-86</td>
<td>green</td>
</tr>
<tr>
<td>3</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>235</td>
<td>14</td>
<td>-127</td>
<td>yellow</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>20</td>
<td>-14</td>
<td>127</td>
<td>blue</td>
</tr>
<tr>
<td>5</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>97</td>
<td>113</td>
<td>86</td>
<td>magenta</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>179</td>
<td>-127</td>
<td>41</td>
<td>cyan</td>
</tr>
<tr>
<td>7</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>white</td>
</tr>
</tbody>
</table>

If l-c1-c2 is considered a rectangular coordinate system, then the hexahedron is certainly not a cube.

Consider for a moment planes of constant \(L\). These planes are parallel to the plane formed by the \(c1\) and \(c2\) axes. Each plane cuts a cross section in the hexahedron. Some cross sections are sketched in fig. 5. Observe that the origin of the \(c1,c2\) coordinate plane lies at the point where the L-axis intersects this plane. Colors along the L-axis \((C1 = C2 = 0)\) are neutral colors, ranging from black at \(L=0\) to
white at maximum L.

If we convert the (C1,C2) coordinates into polar notation, then the resultant angle, $\phi$, is related to the hue of the represented color in a one-to-one fashion. Furthermore, the magnitude, $S$, is related to the saturation (spectral purity) of the color. As the (C1,C2) vector gets further from the origin, it gets closer to the boundaries of the hexahedron. The greatest saturation is achieved at the corners. Observe, however, that different magnitudes, $S$, can represent the same saturation, if the Luminance differs. This is easiest to see if one considers a locus of maximum spectral purity, such as $(R,G,B) = (k,0,0)$ where $k$ is allowed to vary over the range of $R$. Clearly, this color is red, and has the maximum saturation because only the red phosphor contributes to the color. The saturation, since it is maximum for any $k$, should not depend on $k$. Only the luminance (brightness) should depend on $k$. Saturation is limited only by the physical characteristics of the phosphor, as mentioned before. The magnitude, $S$, computed from the $(L,C1,C2)$ representation of this vector, is, however, dependent on $k$, so we see that $S$ is not exactly representative of the saturation. $L$ is also dependent on $k$. From the matrix transformation given above, we have

$$L = 0.299k$$

$$C1 = 0.498k$$

$$C2 = -0.162k$$

$$S = \sqrt{C1^2 + C2^2} = 0.524k$$

Since $S$ and $L$ are both linearly dependent on $k$, then dividing $S$ by $L$ yields a number independent of $k$: 

-25-
$S/L = 1.753$

This number is related to the true saturation of the color in a one-to-one fashion. For historical reasons, we refer to $S$ as "saturation" and $S/L$ as "true saturation."
Fig. 5: Cross Sections of r-g-b hexahedron in l-c1-c2 space
CARTESIAN TO POLAR CONVERTER

The Selective Color Correction (SCC) module and the Special Color Correction (SpCC) module can be most easily implemented if they operate on the hue and saturation of a pel (pixel) rather than on C1 and C2, the chrominance components. If we consider C1 and C2 to be rectangular coordinates representing a point in a cartesian color space, and S and H to be polar coordinates representing the same point, then we can extract hue and saturation information from H and S, as discussed above. H represents the angle and has a one-to-one correspondence with the hue of a pel represented by C1 and C2. S represents the magnitude and has a one-to-one correspondence with the saturation of a pel multiplied by the luminance of the same pel. Thus if we convert C1 and C2 into polar coordinates, H and S, and divide S by the luminance L, then we will have representations of hue and saturation. In practice, in most circumstances in this system, we can use S directly without the division.

A digital hardware implementation of the transformation to polar coordinates was proposed by Stephen Guattery in his S.B.Thesis and was later modified by Finley Shapiro as part of his S.B.Thesis. That implementation is presented here virtually unchanged.

The basic expressions used in the transformation are:

\[ H = \arctan(C1/C2) \]
\[ S = C1 / \sin(H) \]
\[ = C2 / \cos(H) \]
Guattery proposed the use of logarithms, obtained from lookup tables stored in PROMs, for implementation of the multiplications, divisions, and trigonometric functions. The advantages of this approach are two-fold:

1. We can maintain control over the precision of the numbers processed at each stage by introducing scaling factors in the lookup tables. Thus, if 8 bits is sufficient to represent \( \ln(C1) \) at the desired precision, the lookup table from which it is obtained can be structured to utilize the full dynamic range. Such scaling factors can be used whenever a lookup table is used to implement arithmetic of this type, as long as care is taken to maintain internal consistency.

2. We avoid the use of multipliers, which are more expensive and more difficult to use than adders.

The implementation of the conversion is thus accomplished according to the following expressions:

\[
H = \arctan(\exp(\ln(C1) - \ln(C2)))
\]

\[
\ln(S) = \ln(C1) - \ln(\sin(H))
= \ln(C1) - \ln(\sin(\tan^{-1}(\exp(\ln(C1) - \ln(C2)))))
= \ln(C2) - \ln(\cos(H))
= \ln(C2) - \ln(\cos(\tan^{-1}(\exp(\ln(C1) - \ln(C2)))))
\]

Any situation in which we find ourselves taking a logarithm of zero must be treated as a special case.

Guattery shows that if \( H \) is represented by a ten bit two's complement number, then errors due to conversion are minimal. This was veri-
fied using a computer simulation (see Appendix B) of the Cartesian to
Polar Converter (CPC) and the Polar to Cartesian Converter (PCC). The
simulation showed that there was never more than a one bit error when
both conversions were cascaded.

To simplify the circuitry, the two highest order bits of \( H \) are
determined logically rather than arithmetically (see fig. 7). In fig. 6
we have a representation of the values of \( H \) assuming that \( H \) is four bit
number. This corresponds with considering only the top four bits of \( H \),
the ten bit number. It is evident that the top two bits specify the
quadrant and are thus easy to determine logically.

Since the Selective and Special Color Correction Modules will be
using \( \ln(S) \), there is no need for the Cartesian to Polar Converter to
calculate \( S \) from \( \ln(S) \). However, when \( C1 \) and \( C2 \) both are zero, then
\( \ln(S) \) does not exist. \( C2 \), as shown in fig. 7, is an extra bit transmit-
ted which indicates this condition.

In generating the contents of the PROMs, scaling factors are used
to make the output of the PROMs meaningful. For example, \( C1 \) has a range
of values from \(-128\) to \( 127 \). \( \ln|C1| \) therefore has a range of values from
minus infinity to about 4.85. We represent minus infinity with a spe-
cialized bit (generated by the zero detect circuitry in fig. 7). The
remaining range of values goes from zero to 4.85. We multiply this
range by a constant and represent the result as an eight bit word. To
utilize the full dynamic range of eight bits the constant is about
52.56. Observe that we can also simply calculate the logarithm to some
base other than e. The effect is identical to scaling but somewhat more
difficult to visualize. Observe that all scaling factors must be mutu-
ally consistent. In other words, if multiplication is to be performed
by addition of two scaled logarithms, the logarithms must have been scaled by the same constant. Furthermore, the same constant must be used when the inverse logarithm operation is performed. The determination of scaling constants and the wordsizes needed by the CPC are summarized below:

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>RANGE</th>
<th>WORDSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>-128 to 127</td>
<td>8 bits</td>
</tr>
<tr>
<td>C2</td>
<td>-128 to 127</td>
<td>8 bits</td>
</tr>
<tr>
<td>ln</td>
<td>C1</td>
<td></td>
</tr>
<tr>
<td>ln</td>
<td>C2</td>
<td></td>
</tr>
<tr>
<td>K1*ln</td>
<td>C1</td>
<td></td>
</tr>
<tr>
<td>K1*ln</td>
<td>C2</td>
<td></td>
</tr>
<tr>
<td>A=K1*(ln</td>
<td>C1</td>
<td>-ln</td>
</tr>
<tr>
<td>arctan(exp(A/K1))</td>
<td>.008 to 1.56</td>
<td>NA</td>
</tr>
<tr>
<td>K2*arctan(exp(A/K1))</td>
<td>1 to 255</td>
<td>8 bits</td>
</tr>
<tr>
<td>K1*ln(sin(arctan(exp(A/K1))))</td>
<td>18 to 0</td>
<td>5 bits</td>
</tr>
<tr>
<td>K1*ln(cos(arctan(exp(A/K1))))</td>
<td>18 to 0</td>
<td>5 bits</td>
</tr>
<tr>
<td>K1*ln(S)</td>
<td>0 to 273</td>
<td>9 bits</td>
</tr>
</tbody>
</table>

K1 = 127/4.85203 = 52.55532
K2 = 256/(π/2) = 162.97467

Using these constants, it is instructive to map the vertices of the R-G-B cube into L-C1-C2 space and then convert C1 and C2 into polar coordinates. Below we find tabulated the vertices of the hexahedron in terms of L,LNS, and H. The values shown below are the ones actually generated by the CPC (obtained from the computer simulation listed in appendix B) and may be off by a bit from the true value.
<table>
<thead>
<tr>
<th>VERTEX</th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>L</th>
<th>LNS</th>
<th>H</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>BLACK</td>
</tr>
<tr>
<td>1</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>76</td>
<td>258</td>
<td>3CE</td>
<td>RED</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>158</td>
<td>260</td>
<td>26B</td>
<td>GREEN</td>
</tr>
<tr>
<td>3</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>235</td>
<td>255</td>
<td>312</td>
<td>YELLOW</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>20</td>
<td>255</td>
<td>112</td>
<td>BLUE</td>
</tr>
<tr>
<td>5</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>97</td>
<td>260</td>
<td>06B</td>
<td>MAGENTA</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>179</td>
<td>258</td>
<td>1CE</td>
<td>CYAN</td>
</tr>
<tr>
<td>7</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>*</td>
<td>0</td>
<td>WHITE</td>
</tr>
</tbody>
</table>

* - LNS is indeterminate. A flag (SZ) indicating this condition is set.
Fig. 6: A Digital Color Wheel
POLAR TO CARTESIAN CONVERTER

After the processing by the selective color correction module and the special color correction module are complete, it is necessary to reconvert the picture information from polar form to L, C1, and C2 components. Again multiplications can be implemented by adding logarithms. Any situation in which we find ourselves trying to take a logarithm of zero must be treated as a special case. In the calculation of C2 (the X component, for historical reasons) the special case occurs when H represents ± 90 degrees. In the calculation of C1, the special case occurs when H represents 0 or 180 degrees. In fig. 9, a block diagram of the implementation is shown. The approach is based on simple trigonometry, but it proved to be simpler in terms of hardware to treat each quadrant differently. The calculations are done according to the following tables, where only the low order 8 bits of H (denoted by HL) are used in the arithmetic, and logic functions are used to properly account for the two high order bits, which denote the quadrant.

CALCULATION OF C2 (X COMPONENT)

<table>
<thead>
<tr>
<th>quadrant</th>
<th>value</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>C2=S*cos(HL)</td>
<td>C2=exp(ln(cos(HL))+ln(S))</td>
</tr>
<tr>
<td>II</td>
<td>C2=-S*sin(HL)</td>
<td>C2=-exp(ln(sin(HL))+ln(S))</td>
</tr>
<tr>
<td>III</td>
<td>C2=-S*cos(HL)</td>
<td>C2=-exp(ln(cos(HL))+ln(S))</td>
</tr>
<tr>
<td>IV</td>
<td>C2=S*sin(HL)</td>
<td>C2=exp(ln(sin(HL))+ln(S))</td>
</tr>
<tr>
<td>special</td>
<td>C2=0</td>
<td>C2=0</td>
</tr>
</tbody>
</table>
### Calculation of C1 (Y Component)

<table>
<thead>
<tr>
<th>Quadrant</th>
<th>Value</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>( C_1 = S \cdot \sin(HL) )</td>
<td>( C_1 = \exp(\ln(\sin(HL)) + \ln(S)) )</td>
</tr>
<tr>
<td>II</td>
<td>( C_1 = S \cdot \cos(HL) )</td>
<td>( C_1 = \exp(\ln(\cos(HL)) + \ln(S)) )</td>
</tr>
<tr>
<td>III</td>
<td>( C_1 = -S \cdot \sin(HL) )</td>
<td>( C_1 = -\exp(\ln(\sin(HL)) + \ln(S)) )</td>
</tr>
<tr>
<td>IV</td>
<td>( C_1 = -S \cdot \cos(HL) )</td>
<td>( C_1 = -\exp(\ln(\cos(HL)) + \ln(S)) )</td>
</tr>
<tr>
<td>Special</td>
<td>( C_1 = 0 )</td>
<td>( C_1 = 0 )</td>
</tr>
</tbody>
</table>

These algorithms can be verified by examining the digital color wheel (fig. 6) and observing basic trigonometric identities.

Observe that since the two modules immediately preceding the polar to cartesian converter make modifications to \( \ln(S) \) & \( H \) it is possible that the resultant \( \ln(S') \) and \( H' \) cannot be represented by eight bits of \( C_1 \) and \( C_2 \) at the same resolution as the input \( C_1 \) and \( C_2 \). This problem is represented graphically in fig. 8. The circle represents the domain of the LMS-H representation and the square represents the domain of the \( C_1,C_2 \) representation. In printing applications this problem is not necessarily serious because colors on the periphery of the circle represent saturations which in all probability are unattainable using conventional printing inks, and a non-printable color detector would somehow inform us of the error. Upon occasion, however, a pel may move outside of the square as a consequence of a desired change on some other set of pels. Thus, if we were simply not to allow changes which cause this overflow condition, then we could end up restricting the utility of the device. Furthermore, in applications other than printing, this issue cannot be ignored on the basis of printing ink limitations.

Simply clamping whichever of \( C_1 \) or \( C_2 \) goes over its range is an inelegant solution because it would cause a shift in hue which might or might not be visible. An alternative solution which is quite simple to implement is included in the block diagram in fig. 9. This solution
calls for clamping $S$ to a maximum value dependent on $H$ (from a lookup table) before conversion is performed.

As with the CPC, scaling constants are used in the PROM contents. These, of course, must be compatible with those in the CPC in the sense that the PCC must reverse the process of the CPC. Thus, the same values for $K1$ and $K2$ are used. See appendix B for listings of the programs used to generate the contents of the PROMs.

Note that the CTM has the capability to operate in the so called "transparent" mode. In this mode, the knob settings on the control panel have no effect on an image. The Selective Color Correction and Special Color Correction Modules simply pipeline their inputs without effecting them. The CPC and PCC, however, are operating as always, so an error of as much as a bit may appear on the output. When tested, the CTM showed no visible degradation of the image in this mode, as expected.
FIG 8: RELATIVE DOMAINS OF POLAR AND RECTANGULAR REPRESENTATIONS
SELECTIVE COLOR CORRECTION

The Selective Color Correction (SCC) permits a user to alter the hue or saturation of all pels of a certain color in an image. There are seven such colors:

- M - Magenta
- R - Red
- O - Orange
- Y - Yellow
- G - Green
- C - Cyan
- B - Blue

For example, a user could desaturate the oranges in an image, or make the reds magenta. (See appendix A for pictures generated by the working prototype.) This is accomplished first by dividing an image into seven hue domains, denoted M, R, O, Y, G, C, and B, and assigning a set of controls on the control panel to each domain (see fig. 10). In this implementation, there are two controls for each hue domain; one modifies the saturation, and the other modifies the hue. There should be some overlap between the hue domains. A pel might be somewhere between blue and cyan, for example. In this case, we would like both the blue controls and the cyan controls to effect the pel, although both should effect it less than they would a precisely blue or precisely cyan pel. The hue of a pel is represented by the value H generated from the chrominance information by the Cartesian to Polar Converter. In order to define the hue domains, it is helpful to calculate the values for H corresponding to the eight corners of the r-g-b hexahedron (see fig. 4). These values are tabulated on page 32. Note that the corners corresponding to black
and white are irrelevant because they have zero chrominance, and should not therefore be effected by the SCC. The other six points, however, give us reasonable values for the centers of the magenta, red, yellow, green, cyan, and blue domains. In this implementation we assumed that the center of the orange domain, the only one not defined by a corner of the r-g-b hexahedron, falls half way between the red and yellow centers. Note that this mapping of colors into H does not divide the digital color space into seven equal parts. This implies that incremental changes in the value H are not perceptually equal everywhere in the color space. The hue domains are defined in the prototype by the contents of EPROMs (2708s), and can therefore be easily changed.

It is also necessary to define the boundaries between the domains of adjacent color controls. Abrupt boundaries which have no overlap with neighboring domains may add undesirable artifacts to the image. Boundaries can be defined in terms of weighting functions \( W(H) \) which are multiplied by the desired change in hue or saturation. Thus if the hue of a pel is beyond the range of the blue controls, the blue weighting function will be zero and the setting of the blue control will have no affect on the pel. If the hue of a pel is precisely blue, then the blue weighting function will be unity and the blue control will effect the pel without attenuation. If the pel lies between cyan and blue, we would like the sum of the effects of the cyan and blue controls to equal the effect that the blue control would have on a purely blue pel. Therefore, the weighting functions of two neighboring hue domains should add to unity where the two domains overlap. One smooth function which satisfies this criterion is the raised cosine or cosine squared function. Let \( B \) and \( C \) denote the values for \( H \) at the centers of the blue and cyan
domains, respectively. The the blue weighting function would be

\[ W_B(H) = 1 + \cos(\pi(H-B)/(C-B)) \]
\[ = 2\cos^2(\pi(H-B)/(2(C-B))) \]

and the cyan weighting function could be

\[ W_C(H) = 1 + \cos(\pi(H-C)/(C-B)) \]
\[ = 2\cos^2(\pi(H-C)/(2(C-B))) \]

These functions and their sum are plotted in fig. 11. Note that their sum is unity between the two center values.

Observe that any given value for H can lie within no more than two hue domains at once. This property greatly simplifies the design. In the implementation shown in fig. 13, the seven weighting functions corresponding to the seven hue domains are combined into two, called \( W_{\text{EVEN}}(H) \) and \( W_{\text{ODD}}(H) \) shown in fig. 12. (See appendix B for the computer program used to generate these curves.) The magenta weighting function is split between the two functions. Two multiplexers are used to select the two control signals relevant to the values of H in question. The two weighting functions are applied and the effects of the two control signals are summed to yield the total effect on the pel. This is done first for hue shifts and then for saturation shifts.

It is desirable also to be able to translate equally the hue or saturation of all pels in an image. The block diagram in fig. 13 shows an implementation of such overall hue and saturation controls. An overall hue shift corresponds to rotating the entire C1,C2 space. In
order to maintain flexibility, we assign a weighting function $W_0(H)$ to the overall hue and saturation shifts. This function can be used to compensate for the problem noted above in which equal shifts may not be perceptually equal everywhere in the color space. Since considerable psychovisual research would be required to determine appropriate values for this weighting function, the prototype demonstrated simply uses a constant.

In the implementation shown in fig. 13, we precompute the hue and saturation translation for every possible value of $H$; these translations are stored in RAM, using $H$ as an address. In video time, then, the circuit simply needs to extract from the RAMs the translations relevant to the current pel, using $H$ as an address, and use these values to modify $H$ and $S$. The modifications are done according to the following algorithms:

$$H' = H + W_{\text{EVEN}}(H) \times H_{S1} + W_{\text{ODD}}(H) \times H_{S2} + W_0(H) \times H_{S0}$$

where $H_{S1}$ and $H_{S2}$ represent the desired hue shifts of whichever two knobs have an effect for the given $H$, and $H_{S0}$ represents the desired overall shift. The change in saturation is ideally:

$$S' = S (1 + W_{\text{EVEN}}(H) \times S_{S1} + W_{\text{ODD}}(H) \times S_{S2} + W_0(H) \times S_{S0})$$

where $S_{S1}$ and $S_{S2}$ represent the desired saturation shifts specified again by whichever two knobs have an effect for the given $H$, and $S_{S0}$ represents the desired overall saturation shift. The implementation becomes considerably simpler if instead we use the expression

$$LNS' = LNS + W_{\text{EVEN}}(H) \times S_{S1} + W_{\text{ODD}}(H) \times S_{S2} + W_0(H) \times S_{S0}.$$
The difference is that in the latter case, the control knobs have a logarithmic effect on the saturation of the pels instead of linear effect. Since there was no reason to believe that a linear effect was more desirable than a logarithmic effect, we proceeded with the simpler approach.

Clearly, it would be difficult for the writing and reading of the RAMs to be going on simultaneously. During the horizontal retrace, which occurs every $65.5$ microseconds, no picture elements should appear at the input to the system. During this time, we could easily load values into the RAMs. The Selective Color Correction (SCC) module of the CTM requires 2048 RAM writes to completely update the RAMs which are used to perform the translation at video speeds. For each RAM write, a set of signals from the control panel undergoes some analog processing and an A/D conversion. The simplest approach, and the one shown in fig. 13, is to execute one conversion and one RAM write for every horizontal trace. This approach has the advantage that the precalculation circuitry has at least 50 microseconds to settle, and therefore need not be terribly fast. If one write is executed during each retrace then the total time for 2048 writes is $1/7.5$ sec. At 512 scan lines per frame, this translates into a worst case response time for the controls of four frames. The average response time in $1/15$ sec. or two frames. When testing the prototype, this delay was not bothersome. If, however, an application arises in which this delay is objectionable, there are a number of ways to improve it.

1. The A/D converter selected for the SCC executes one 12 bit conversion in 25us. We need only 11 bits, so with the addition of a counter, the time can be reduced by one or two
microseconds. The settling time of the analog preprocessing is approximately 5μs. It is possible, therefore, to complete two of these 29μs conversion cycles within a single 63.5μs trace cycle. The response time would be halved by such an approach, but a worst case response time of 1/15 sec. may still be objectionable. Note, however, that the five microsecond settling time is approximate, and we only have about a microsecond margin for error in each conversion cycle. If we use opamps with a larger gain-bandwidth product, we may be able to improve on this. Nonetheless, implementation of this approach would significantly complicate the circuitry and would involve a major re-design, so it is not recommended.

2. A second approach to the problem is to allow the A/D conversion sequence to run continuously and asynchronously (i.e. not bound to the retrace signal). RAM writes are performed whenever a cycle is completed by simply stealing a cycle from the video data stream. The pel lost in the process might be replaced by the previous pel. The timing for this alternative is simpler than that for the alternative in (1), but the additions to the present design are non-trivial. Although the artifacts introduced by this approach may not be visible, it seems preferable to use a method which introduces no artifacts, if possible.

3. An elegant solution to the problem is to include circuitry which detects changes in the control settings and updates only the relevant portions of the RAMs. Detection of changes could be implemented simply by comparing the values to be written
into the RAMs with the corresponding values stored previously in the RAMs. If no change is detected, then the counter could be incremented some predetermined amount which would then put it into a new hue domain. Again, the circuitry is complicated by this approach, perhaps even more so than above. The response time, however, would easily be reduced to under a frame.

4. Another alternative actually involves a simplification of the circuitry. If the RAM size is decreased from 2048 to 1024 or 512, the response time is reduced by a factor of two or four, respectively. The only side effect of this change would be a decrease in the resolution of the weighting function applied to the control knobs. In other words, if the RAM size is 512, four neighboring values of the hue (a ten bit number) will be affected identically by any setting on the control panel. This effect might not be visible, and the worst case response time will have been reduced to one frame instead of four. There is not much point in reducing it any further. This approach has the added advantage that as long as the system is designed for full resolution capability, it is trivially easy to change the resolution in the hardware and experiment with different values. For this reason, the prototype described here was designed with full resolution. Furthermore, the highly simple timing of the present design can be maintained.

It seems unreasonable to proceed with any of the complex alternatives without first testing the simpler one; simplicity is a huge advantage and should be maintained if possible.
It was mentioned above that we needed 11 bits from the A/D conversion, even though the value being altered is no longer than ten bits (for \( H \)). The reason for this is simple. In the design of this prototype, for flexibility, we wished to maintain both maximum resolution and maximum range for the color controls. Any value for \( H \) can be altered by one bit or by ten bits by the two controls pertaining to the hue domains in which \( H \) falls. So the signal due to these controls should have ten bit resolution. The overall hue control, however, needs equal resolution and range. That implies a ten bit resolution on this control signal also. Since these two signals are added in analog form before they are converted to digital form, the A/D converter must have 11 bit resolution to preserve the integrity of the two signals. This means, for example, that while we can use the hue controls to alter a color by as little as one LSB, we can also complement it with the overall hue control, and complement it again with the color selective controls. The current design of the control panel does not support this kind of flexibility, however. See the "Suggestions for Further Work" section for recommendations on how to improve the control panel.

This kind of flexibility, however, is not necessary for all conceivable applications. In the printing industry, only relatively small changes will probably be used, so the full range is not necessary. For special effects, only relatively large changes are useful, so full resolution is not necessary. The prototype described here is designed to operate in both environments.
FIG 11: BLUE AND CYAN WEIGHTING FUNCTIONS AND THEIR SUM
The Special Color Correction module (SpCC) permits the user to define his own color to be altered. This is done by defining an area of the color space, called a chromatic neighborhood, within which the color (or colors) of interest fall (see fig. 14). There are four knobs on the control panel which define the chromatic neighborhood. These are:

CH - Center Hue
HW - Hue domain width
CS - Center Saturation
SW - Saturation width

Changes which affect this neighborhood are specified with two more controls:

HS - Desired Hue Shift
SS - Desired Saturation Shift

In the definition of the saturation range it is important that the true saturation of a pel be used. Among the desired capabilities of the SpCC is the ability to pick out an object in an image based on its chrominance and modify its hue or saturation without affecting the rest of the image. In general, the hue and saturation of the image of an object are constant throughout the object. Shadows and highlights represent variations in luminance, but not usually in chrominance. Recall, however, that S must be divided by L before we have a reasonable representation of the true saturation. Naturally, this extra division complicates the implementation. Recall also that chrominance information has lower resolution than luminance information. C1 and C2 (and therefore LNS and H) are sampled at 5MHz vs. 10MHz for L. In this implementation, we
simply assume that subsampling the Luminance is sufficient for the calculation of the true saturation. The block diagram of the SpCC (fig. 15) shows the specifics of the implementation.

As in the case of the selective color correction, we need to apply a weighting function to the control signals, so that there are not sharp boundaries between colors affected and colors not affected. Observe, however, that since the size of the chromatic neighborhood varies with the settings on the control panel, a straightforward application of a weighting function is not possible, because the weighting function must vary in width along with the hue and saturation domains. In fact, the variable size of the chromatic neighborhood does not present a serious problem in the calculation of this weighting function.

Let

\[ K = \left(\frac{1}{HW}\right) \times (H - CH) \]

and

\[ M = \left(\frac{1}{SW}\right) \times (S/L - CS/L) \]

where \( S \) and \( H \) are the saturation and hue of a pel, and \( L \) is the luminance. A single weighting function \( W(X) \) can be used where \( X = K \) or \( X = M \) for hue or saturation weightings respectively. The weighting relevant to a specific pel is then the product of these two weighting functions. Observe that if a pel is precisely equal in chrominance to the center of the chromatic neighborhood, then

\[ H = CH \]

and

\[ S/L = CS/L \]

consequently
\[ K = 0 \]
and
\[ M = 0. \]
Thus, we need a weighting function such as a raised cosine centered at zero, so that both \( W(K) \) and \( W(M) \) equal one when the pel in question is precisely in the center of the chromatic neighborhood. When either \( H \) or \( S/L \) get beyond their respective domain widths, \( W(K) \) or \( W(M) \) should go to zero so that such a pel is not affected by the SS or HS values. When one or the other of these values goes to zero, the product goes to zero, so a pel that is out of range in either the saturation dimension or the hue dimension will not be altered.

The modifications to a pel are given by:

\[ H' = H + HS \times W(K) \times W(M) \]
and
\[ LNS' = LNS + SS \times W(K) \times W(M) \]
where as in the SCC we are allowing the SS control to affect a pel in a non-linear fashion. These expressions can be implemented by periodically updating three RAMs which contain precomputed values to be accessed and processed at video speeds. These are defined as follows:

<table>
<thead>
<tr>
<th>RAM</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>( HS \times W(K) ) for all 1024 possible values of ( H )</td>
</tr>
<tr>
<td>B</td>
<td>( SS \times W(K) ) for all 1024 possible values of ( H )</td>
</tr>
<tr>
<td>C</td>
<td>( W(M) ) for all 512 possible values of ( S )</td>
</tr>
</tbody>
</table>

This approach is quite similar in principle to that of the Selective Color Correction, but a bit more complicated. The circuitry is divided
into two parts, a knob processor (fig. 15a) and a video processor (fig. 15b). In the knob processor, a ten bit counter walks through all possible values of \( H \) and \( S/L \). \( K \) is computed using the counter value instead of \( H \), and \( M \) is computed using the counter value instead of \( S/L \). Thus,

\[
K = \left(1/HW\right) \times \left(COUNT - CH\right)
\]

\[
M = \left(1/SW\right) \times \left(COUNT - CS/L\right).
\]

The multiplications are performed using multiplying digital to analog converters (MDACs) in the analog signal path. The results, \( K \) and \( M \), are amplified by the gains \( G1 \) and \( G2 \) (see note 11 below for reason) and converted to digital form. Using PROMs (2708s), we compute the weighting function values \( W(K) \) and \( W(M) \). \( W(K) \) is multiplied by the digitized HS and SS signals, where the multiplication is performed by adding logarithms. Three results, denoted \( SS\text{SHIFT} \), \( H\text{SHIFT} \), and \( WM \) in fig. 15, are then written into the RAMs during the next retrace. The counter is then incremented, and the computation is repeated for the new value of \( COUNT \).

Each RAM, at address \( COUNT \), contains the following actual values:

<table>
<thead>
<tr>
<th>RAM CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ( \ln</td>
</tr>
<tr>
<td>HLTZ - a bit indicating that ( HS &lt; 0 ).</td>
</tr>
<tr>
<td>HEZ - a bit indicating that ( HS = 0 ) or ( W(K) = 0 ).</td>
</tr>
<tr>
<td>B ( \ln</td>
</tr>
<tr>
<td>SLTZ - a bit indicating that ( SS &lt; 0 ).</td>
</tr>
<tr>
<td>SEZ - a bit indicating that ( SS = 0 ) or ( W(M) = 0 ).</td>
</tr>
<tr>
<td>C ( \ln(W(M)) ) (ten bits)</td>
</tr>
<tr>
<td>ZERO - a bit indicating that ( W(M) = 0 ).</td>
</tr>
</tbody>
</table>

where \( K \) and \( M \) are computed from \( COUNT \) as above.

The second part of the SpCC, called the video processor, first calculates the value of \( S/L \) from the incoming values LNS and L. An
additional signal, denoted \textcolor{red}{BLACK} in fig. 15b, indicates the condition L = 0, when clearly the S/L division is meaningless. When L = 0, the pel has neither luminance nor chrominance, so it is passed unaltered. The value S/L is used to address the RAM C when the RAM is read. H is used to address RAMs A and B. The results are added, corresponding to the multiplications \((H\times W(K)) \times W(M)\) and \((S\times W(K)) \times W(M)\). Combinatorial logic handles the cases of negative or zero values. The results correspond to the desired shift in LNS and H, so they are simply added to LNS and H. The OVFL signal indicates that LNS' has overflowed and should be clamped to its maximum value (dependent on H) by the PCC.

The speed of response considerations for the SpCC are virtually identical to those for the SCC, so they need not be repeated. Should it prove desirable, the response time can be improved by decreasing the size of the RAMs and losing resolution only in the weighting function, as in the SCC.

A number of complications arose when designing the actual implementation. The following list of notes is intended to explain aspects of the design which might be unclear to someone simply trying to read the schematics or the block diagram. (See appendix C for the schematics of the digital parts.) The numbers in front of the notes refer to specific locations labeled on the block diagram in fig. 15 parts a and b.
NOTES ON SpCC DESIGN

(SEE BLOCK DIAGRAM)

1. The inputs from the control panel representing the center hue position (-CH) and the center "true" saturation position (-CS/L) are considered the negative of CH and CS as defined above. Note that this simply reverses the sense of the controls - higher voltages correspond to smaller (or more negative) numbers. This need not affect the user because the leads connected to the potentiometers on the control panel can be reversed. The circuitry is simplified by eliminating the need to perform a two's complement operation, or an analog inversion with an offset.

2. In performing the A/D conversion, we encounter a problem due to having reversed the sense of the CH and CS/L signals. CH has the range -512 to 511. This implies that -CH should have the range -511 to 512. Such a range is not representable in a two's complement ten bit number. The number -CH-1, however, has a range of -512 to 511 and is therefore compatible with a ten bit format. We can simply consider the output of the A/D converter to represent -CH-1, and at the addition operation later on, use the carry input of the adder to add the missing one. The effect of this does not appear to the operator. We treat -CS/L in a similar fashion. The output of the A/D converter is considered to represent -CS/L-1, and a compensating addition of one is implemented at the adders.
3. A single A/D converter is used to convert four control values. Four registers are maintained and updated every fourth convert cycle. Since a convert cycle occurs once during every horizontal trace, (or once every 61 microseconds) the addition to the worst case response time is only 260 microseconds, certainly not a noticeable figure.

4. The input to this adder, $-CS/L-1$, is a unipolar negative number with a range from $-1024$ to $-1$. For $-CS/L-1$, the A/D converter provides us with a number from 0 to 1023, and if we simply append an eleventh bit which is always one, then this range maps into $-1024$ to $-1$. As expected, the highest voltages applied to the A/D converter correspond to $-1$ and the lowest voltages correspond to $-1024$ (see note 1). The appended eleventh bit is referred to as a ghost bit. We can easily visualize the effect of this if we consider two bit numbers instead of ten bit numbers. All possible values of these two bit numbers are tabulated below.

<table>
<thead>
<tr>
<th>CS/L</th>
<th>-CS/L</th>
<th>-CS/L-1</th>
<th>S/L (CCUNT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-3</td>
<td>-4  (100)</td>
<td>3 (011)</td>
</tr>
<tr>
<td>2</td>
<td>-2</td>
<td>-3  (101)</td>
<td>2 (010)</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-2  (110)</td>
<td>1 (001)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1  (111)</td>
<td>0 (000)</td>
</tr>
</tbody>
</table>

**GHOST BIT**

The possible results of the addition (with and without the additional one (1) referred to in note 2) are tabulated below:
It is easy to see that the ghost bit and the additional one combine to yield the correct result. It is not necessary to use a ghost bit for the adder operating on $-CH-1$ because this is a bipolar number with an appropriate range. Simple sign extension suffices.

5. Computation of the logarithm is accomplished here via a PROM lookup. A scaling constant is multiplied by the logarithm of the input to ensure that the dynamic range of the output fully utilizes the available dynamic range determined by the output wordsize. Since the output in this case does not interact with any previously computed logarithms (such as LNS), we are completely free to choose the scaling constant as we see fit. Observe that the input to the PROM can represent either SS or HS, depending on the state of the counter. In both cases the input is a ten bit offset binary number representing the range from $-512$ to $511$. (Offset binary is simply two's complement with the highest order bit inverted.) Since we are computing the logarithm of the absolute value of the input, the effective input range is from $0$ to $512$. If the output is to be an 8 bit number (range $0$ to $255$), we would want an input of $512$ to yield an output of $255$. 

-59-
K3 \* ln|512| = 255
K3 = 40.88

Observe, however, that for this value of K3 we do not have a one to one mapping between the input and the output. Different values of the input yield the same value for the output, so the operation is not reversible, and information is lost.

K3 \* ln |512| = 255
K3 \* ln |511| = 255
K3 \* ln |510| = 255
K3 \* ln |505| = 254

To ensure a one to one mapping, the output must be at least 12 bits wide. This wordsize, however, is cumbersome, and would seriously complicate the implementation. Observe that because of the properties of the logarithm, if a wordsize of less than twelve bits is used, the process loses its one to one mapping only for higher values of input. In the 8 bit case above, for lower values of input, a one to one mapping is preserved.

K3 \* ln |20| = 122
K3 \* ln |21| = 124
K3 \* ln |22| = 126

The effect of using a wordsize of less than twelve bits is therefore that resolution is lost for larger values of SS or HS. Thus, when large changes in hue or saturation are requested, the changes cannot be fine tuned to the extent that smaller changes can be. This seems a reasonable compromise.

The largest wordsize which we can implement without unduly expanding the hardware is ten bits. This way we can use two bits from a 12 bit PROM (or rather three four bit PROMs) to
generate the SZ and LTZ signals. A new value for K3 must be calculated.

\[
K3 = \ln |512| = 1023
\]
\[
K3 = 163.98634
\]

With this wordsize, a maximum of three adjacent large magnitude inputs yield the same output value. The one to one correspondence is valid for all values of input less than 416 in magnitude. See appendix B for a listing of the program which generates the PROM contents.

6. The computation of the logarithms here must be compatible with that described in note 5, so the same scaling constant, K3, is used. The particular weighting function used is intended to be changeable, but a reasonable starting point seems to be a raised cosine. The input is a ten bit offset binary number, and we want the non-zero range of the weighting function to span approximately from 412 to 612 (corresponding to -100 to 100 in binary). (see note 11 for reason). To accomplish this we use the following expression:

\[
W(x) = \begin{cases} 
(1 + \cos((x - 512) \pi / 100))/2 & 412 < x < 612 \\
0 & \text{otherwise}
\end{cases}
\]

where 'x' is either K or M. This function has the property that its value is one near 512 (corresponding to zero) and decreases smoothly to zero at higher and lower values of the input. By trigonometric identities, we see that this is equivalent to a squared cosine function.
\[
W(x) = \begin{cases} 
(\cos((x - 512) \pi / 200))^2 & 412 < x < 612 \\
0 & \text{otherwise}
\end{cases}
\]

Observe that when we compute the logarithm, we have a number which is always negative or zero. The contents of the PROMs are therefore

\[
W(x) = \begin{cases} 
K3 \times \log((\cos((x-512) \pi / 200))^2) & 412 < x < 612 \\
0 & \text{otherwise}
\end{cases}
\]

Again, in appendix B we find the program used to generate these values.

7. For the purposes of computing the true saturation of a video pel, L is subsampled at 5MHz so that it is compatible with LNS and H.

8. The calculation of the logarithm here must be consistent with the calculation yielding LNS, so the scaling constant, \( K1 = 52.555 \), is used.

9. We have shown that within the hexahedron of permissible colors in the l-c1-c2 color space (i.e. those colors which do not cause an overflow when converted to r-g-b color space) the quantity \( S/L \) has a maximum value (about 6.45). We cannot, of course, ignore the possibility that the pel arriving at the SpCC falls outside of this hexahedron and yields a quantity for \( S/L \) which exceeds this maximum value. The Neutral Balance Module and the Selective Color Correction module have both operated on the pel in such a way that we can no longer be sure that the maximum value is valid. It is true, however, that we
are interested principally in operating on pels which yield \( S/L \leq 6.45 \). It is reasonable therefore that the number representing \( S/L \) should have the best resolution in the region of interest. Our approach is to represent \( S/L \) in the range of zero to 6.45 by a ten bit number in the range from 0 to 1022. All values of \( S/L \) greater than 6.45 will be represented by the ten bit number 1023. This PROM, therefore, is used to first calculate \( S/L \), and then multiply it by a constant which brings it up to the desired resolution. Values greater than 6.45 are thus treated as a special case.

\[
\begin{align*}
K_4 &= \exp(93/K_1) = 1022 \\
K_4 &= 6.45 = 1022 \\
K_4 &= 158.35114
\end{align*}
\]

10. The 1/HW and 1/SW inputs are inverted sense from Hue Width and Saturation Width. This means that a 10V input on either line corresponds to the smallest possible Hue or Saturation domain width, and a 0V input on either line corresponds to an infinitely wide Hue or Saturation domain width. This approach simplifies the algorithm which uses this information and is transparent to the user.

11. A non-unity gain is used here for reasons of resolution. The MDACs preceding this stage are actually digitally controlled attenuators, rather than true multipliers. If a unity gain was used here, then PROMs B and C would have to contain the weighting function \( W(x) \) as it would look for the narrowest possible Hue and Saturation domain widths. If we allow reasonably narrow domains, then the PROMs would have very poor resolution in
their outputs. This gain allows us to widen the weighting function stored in the PROMs by a factor equivalent to the gain constants G1 and G2. We do not, however, want G1 and G2 to be too large, because they will amplify noise as much as signal. As a starting point, we set

\[ G1 = G2 = 10 \]

This means that if the minimum Hue and Saturation domain widths we are allowing is 20, then the weighting function is 200 wide. This yields a reasonable resolution. In the prototype, we were able to reduce G2 to unity gain without serious hampering the selectivity in the saturation domain, and could thus eliminate some of the noise inherent in this type of design.
FIG 14: USER DEFINED CHROMATIC NEIGHBORHOOD
THE SYSTEM

Fig. 16 shows some pictures taken from oscilloscope traces during the final debugging stages. Fig. 16a shows an x-y trace of the output of the system in transparent mode, where the horizontal axis represents C2 (positive to the right) and the vertical axis represents C1 (positive downwards). The test configuration is shown in fig. 17. The input was fed directly to the SCC (bypassing the CPC) and consisted of a constant for LNS and a counter output for H. Thus, the effect is a series of pels with constant saturation but varying hue. The gap in the circle is simply due to an incomplete count during each trace, and was used to readily identify the zero H point on the circle. The dot in the center is the result of the blanking imposed during the retrace. Fig. 16b shows the circle enlarged by the overall saturation control. Note the clamping performed by the PCC. Fig. 16c shows the result of decreasing the saturation of the yellows using the SCC. Fig. 16d shows the result of an overall hue rotation (note the new position of the gap) plus a saturation increase and positive hue shift in the magenta domain. Fig. 16e shows the special color correction selecting a narrow hue domain in the red and increasing its saturation. Observe that here we also have an overall hue shift performed by the SCC; recall that the SpCC operates on pels already processed by the SCC, so the reds being selected here were originally in the orange domain. Note that except in the transparent case (a), the circles displayed are not particularly smooth. The reason is simple. In the current design of the control panel, there is no easy way to set individual knobs so they have no effect. Each
knob, therefore, is generating a slight hue or saturation shift in its area, and the circle becomes distorted. In the "Suggestions for Further Work" section, the author recommends possible improvements to the control panel which would alleviate this problem.

Appendix A contains some pictures of television images taken during the final test. The test configuration used was sub-optimal, because some of the external hardware which will eventually be used to operate the system (such as the digital L-C1-C2 to R-G-B converter) have not been fully developed. The second test configuration is shown in fig. 18, and the backplane configuration for the final system is shown in fig. 19. The specifications for the inputs and the outputs of the system are described below.

The Color Translation Module takes L, C1, and C2 as inputs; it outputs L unchanged and new values for C1 and C2. Observe that even if the input values correspond to a pel within the hexahedron representable in r-g-b space (see fig. 4,) the output pel may not. There is no reasonable way to constrain the output to lie within this space without actually executing a conversion from (L,C1,C2) to (R,G,B) representations. Circuitry is being developed to perform this transformation, and its approach when presented with a pel falling outside the hexahedron is simply to clamp whichever values of R, G, or B that overflow or underflow. Note that this has the consequence that a clamped pel will elicit a change in hue; this is nonetheless preferable to no clamping at all which would probably cause overflowing pels to undergo much more drastic changes.

The inputs and outputs for the system should have the following characteristics.
1. HCLK is the 10MHz system clock.

2. CLK5 is the 5MHz system clock, which may be generated by dividing HCLK, but any case, its rising edge should coincide with a rising edge of HCLK. In the tests discussed here, CLK5 is generated on the Digital Receiver Card (see fig. 19) built by Shapiro.

3. HVB is intended to be the inverse of a composite horizontal and vertical blanking signal. Such a signal is generated in the C.I.P.G. Video Chain. A similar signal, called HVRTC, for Horizontal Vertical Retrace, is generated by the C.I.P.G. Video Synch Generator used in the tests discussed here, but this signal contains a glitch (about 100ns after the vertical retrace begins) which can cause the special color correction to introduce artifacts into the image. In any case, the fundamental requirements on this signal are:

   1. Transitions must be synchronized to the rising edge of the 5MHz clock,

   2. When the signal goes to zero volts, it must remain there at least 10 microseconds, and

   3. The frequency of the signal is related to the worst case response time of the CTM to changes in the control panel settings. If the response time is denoted by T then

      \[ T = 1024 \times P \]

where P is the average period of HVB. In this prototype, the worst case response time was 1/7 second, and
the delay was not noticeable. See SCC section above for a further discussion of response time.

4. Comp. Sync. is simply pipelined through the CTM. It is intended to be used for the synchronization signals needed to control a monitor.

5. C1 and C2 are 3MHz chrominance signals. These signals are modified by the CTM. They are also blanked (forced to zero) whenever HVB is low.

6. L is the 10MHz luminance signal. Note that L should be blanked when HVB is low; the CTM does not blank this signal.

All signals coming in should be LS TTL compatible. CLK5 and HCLK have a fan out low enough to be driven by a single LS TTL chip, but additional loads should not be added. All other input signals drive no more than one LS TTL load. Output signals come from LS TTL buffers, and should be passed through cable driver circuitry before being transmitted to another system unit. In the first and second test configuration, the outputs of the PCC were transmitted through D/A converters on coax cables. These D/A converters should be replaced by cable drivers where digital outputs are required.

The CTM also receives analog signals from the control panel. The signals are processed by C.I.P.G. standard analog receiver boards which accept differential inputs and alter the offset and range of the incoming signals. The signals coming out of the receiver cards should meet the following specifications:

1. The 16 signals (seven hue and seven saturation shift signals, plus two overall shift controls) coming to the SCC must be zero to ten volt signals with zero shift represented by a five volt
value.

2. The two shift controls coming to the SpCC (SS and HS) have the same requirements as above.

3. The center chromatic neighborhood controls (CH and CS) must have a range from zero to ten, but their sense is inverted. In the case of CH, 10V corresponds to an H of -512, and 0V corresponds to an H of 511. In the case of CS, 10V corresponds to an S of zero, and 0V corresponds to an S of 511.

4. The chromatic neighborhood width controls (HW and SW) also have an inverted sense, but their meaning is slightly different. In both cases the domain width is proportional to the inverse of the input voltage. Thus, an input voltage of zero corresponds to a domain width of infinity, while an input voltage of ten volts corresponds to the smallest domain width allowed, which is determined by the gains G1 and G2 in the analog circuitry of the SpCC.

One further note might be useful to future users of the system. The SCC analog board contains several potentiometers for fine adjustment of the system response (see appendix C for schematics). These are used as follows:

1. R1 and R2 control the gains in the two channels of analog information which divide the knobs into even and odd domains. The gains of the two channels should be identical. Adjustments can be made by observing that the input to the A/D converter should be smooth in spite of the discontinuities in the weighting functions (see fig. 12).

2. R3 controls the gain in the overall hue and saturation control...
channel and should be adjusted to equal the gains of the other two channels. This adjustment, however, is not critical.

3. R4 adjust the overall gain after the three channels are added. This gain should be approximately 3/2 so R4 should approximately equal 5K. Again, this adjustment is not critical.

4. R5 and R6 adjust the zero and offset of the A/D converter. See the specifications of the Analog Devices AD572 for the adjustment procedure.
Fig. 16: C2 vs. C1 displays showing various aspects of the CTM operating on a series of pels with constant saturation and varying hue.
FIG. 17: FIRST TEST CONFIGURATION
<table>
<thead>
<tr>
<th>SLOT</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>SERIAL INTERFACE CARD - COLOR BALANCE MODULE</td>
<td>COLOR BALANCE MODULE - ANALOG BOARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>DIGITAL RECEIVER CARD</td>
<td>COLOR BALANCE MODULE - DIGITAL BOARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>BLANK</td>
<td>CARTESIAN TO POLAR CONVERTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>RECORDING CARD NO. 1</td>
<td>SELECTIVE COLOR CORRECTION - DIGITAL BOARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>BLANK</td>
<td>SPECIAL COLOR CORRECTION - ANALOG BOARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>RECORDING CARD NO. 2</td>
<td>SPECIAL COLOR CORRECTION - KNOB COMPUTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>RECORDING CARD - SSL (Posing)</td>
<td>SPECIAL COLOR CORRECTION - VIDEO COMPUTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>SSL - ANA/LOG - (Posing)</td>
<td>SELECTIVE LUMINANCE CORRECTION - DIGITAL BOARD (Future)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>DIGITAL TRANSMITTER CARD</td>
<td>POLAR TO CARTESIAN CONVERTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIG 19: COLOR TRANSLATION MODULE BACKPLANE CONFIGURATION**
SUGGESTIONS FOR FURTHER WORK

The Color Translation Module described here is intended to form a first prototype from which enough can be learned to design a highly refined system for use in color printing. A number of steps need to be taken towards that goal; some of those are listed below. The list is inherently incomplete, because each refinement in the list may expose a need for further refinements not listed.

1. The most obvious requirement for further work is an adequate source of images coded in L-C1-C2 format and appropriate hardware for displaying modified images. The configuration used for the second test (fig. 18) was not adequate because the analog matrix used to perform the linear transformation from l-c1-c2 space to r-g-b space was not compatible with the CTM. The transformation was barely even a rough approximation of the transformation described on page 19, and the synchronization was not compatible. A digital transformation circuit designed to run at video speeds has been designed but not completely implemented. This should be used in a configuration in which the video signal may be taken from a computer and returned to the computer for storage and display.

2. Once an adequate source of signals has been implemented, then thorough testing of the system can begin. Experiments should be performed to determine resolution and range requirements for the controls. The hue ranges for the SCC could be refined, and
the weighting functions could probably stand improvement.
Also, tolerable noise levels should be determined. Essential
to these refinements, however, is a suitable test setup.

3. In the course of the work done thus far, some possible improve-
ments to the control panel have surfaced. Undoubtedly, more
will appear as further tests are performed, so these recommen-
dations are again incomplete. The pots used for hue and
saturation changes are single turn pots with a resolution which
we measured at about 1%. The value of the hue is represented
by a ten bit number and the log of the saturation by a nine bit
number. Maximum control is achieved over these numbers only if
the pots have a resolution of .1% for the hue controls and .2%
for the saturation controls. In practice, such fine resolution
may not be necessary, particularly for the saturation controls.
This would need to be determined experimentally, and finer
resolution would be needed to perform the experiment ade-
quately. We can improve the resolution by a number of methods.
One approach is to use multi-turn pots, but this has the disad-
vantage of becoming awkward to use. Another approach is to
provide two (possibly concentric) pots on the control panel, a
coarse control and a fine control. The two pots with differing
resistances could simply be placed in series. In addition,
individual defeat switches for each saturation and hue control
are necessary. It is difficult without these to alter any set
of colors without affecting all colors simply because it is
difficult to set all unused knobs to their center position with
enough precision.
4. One quite desirable feature missing from the CTM as currently conceived is a mechanism to alter the luminance of pels of a given chrominance. A bank of luminance controls identical to the hue controls for the SCC should be added to the control panel. The circuitry for implementing the requisite translation will be virtually identical to that of the SCC, with the difference that the SCC processes hue and saturation at 5MHz while the Selective Luminance Correction (SLC) would process Luminance at 10MHz. The differing resolutions of the chrominance and luminance components might introduce undesirable artifacts in the image, however, such as halos. Some scheme for avoiding these problems may need to be devised. In any case, the SLC is quite desirable.

5. By putting the CTM under computer control, the flexibility of the device can be enhanced considerably. For example, if images with resolution higher than 512x512 are to be processed, a reasonable and simple approach would encompass the following steps:

1. Subsample the image to obtain a 512x512 full frame image;

2. Process this image through the CTM until the knob settings yield the desired color effects;

3. Finally, pass the full resolution image through the CTM in chunks of 512x512 pels and store the results. Another possible use of computer control would be to read and store the contents of the RAMs in the SCC and SpCC as a permanent record of the transformation performed. The space
required in memory is much less than that necessary to store
the entire modified image. This would require additional
hardware interfacing.

6. For more flexibility and color control, the SpCC could be
enhanced with some potentially useful additional features. One
of these could be implemented once the system is under computer
control. This feature would allow the user to delineate an
area of the image or outline an object in the image, using a
cursor and one of the existing routines at C.I.P.G., and per-
form the color translation on that area only. Another possible
feature would allow a user to position a cursor on a color in
an image, and the SpCC would be instructed to perform its
translation only on that color. Both of these would require
some software development, and the second would also require
some additional hardware interfacing to the SpCC.

In addition to the above, some minor hardware improvements might be
incorporated into future generations of the CTM. For example, if the
CTM hardware is rebuilt using printed circuit boards, considerable space
savings can be achieved by using larger PROMs whenever possible instead
of 82S129s, 82S130s, and 82S131s. Larger PROMs were not used in the
prototyping stage because their pin configuration is not well suited to
wirewrap prototyping on DEC standard boards. Also, printed circuit
boards should be designed to house the analog circuitry for both the
SpCC and the SCC. The DEC standard wirewrap modules used to construct
the prototype were not designed for analog hardware and make low noise
operation difficult. Careful attention to layout considerations on
these P.C. boards is essential, because noise problems will appear as noticeable degradations in the output image. This is particularly important because the analog circuits are in close proximity with 5MHz and 10MHz digital signals. Finally, a provision should be added for blanking the luminance signal during each retrace, just as the chrominance signals are blanked. Theoretically, this should not be necessary because the Luminance signal is not being processed by the CTM and should be blanked prior to the arriving at the CTM; however, the blanking is helpful when signals are generated locally for the purposes of testing or demonstrating the system. The PCC was temporarily rigged to accomplish this objective for the tests discussed above, but the scheme is undesirable because it results in the loss of eight pel's of luminance information at the beginning of every horizontal trace. The shift registers performing the luminance pipelining in the PCC could be replaced by individual buffers (such as 74LS174's), making the blanking a trivial task.
FOOTNOTES


3. The N.T.S.C., or National Television Systems Committee, is the group that has defined the U.S. television standards.


5. The P.A.L. system is the European television standard similar to the N.T.S.C. system.

A BRIEF BIBLIOGRAPHY


Walsh, J.W.T., Photometry, Constable, London, 1953

Wyszecki, G. and Stiles, W.S., Color Science, Wiley, New York, 1957
The image above comes from the monitor in the configuration depicted in fig. 18. The CTM is operating in transparent mode, so this image is essentially equivalent to the input to the system. Each square has the same luminance, but a different chrominance, and its position corresponds approximately to the position of its chrominance in the C1-C2 plane. Observe that the central square is a neutral grey and that the saturation increases towards the edges. Also observe that the hue varies with angle. The configuration used was the best available at the time of this test, but it leaves much to be desired. The analog matrix was performing at best a very rough approximation of the l-c1-c2 to r-g-b conversion, so color fidelity is poor. Also, the synchronization requirements of the analog matrix were incompatible with the
synchronization configuration of the CTM. As noted in the text, a prerequisite for further work is an adequate test configuration.

The image above graphically depicts the selectivity of the SCC. All saturation controls were set to their minimum value (thus neutralizing the colors) except for the red control. The square with the greatest saturation in this image is therefore the reddest, although distortion in the l-c1-c2 to r-g-b converter makes it appear more orange. The halos around the squares are believed due to the synchronization problems.
This image depicts the SpCC being used to select a specific color and alter its hue. Again, the halos may be due to the synchronization incompatibilities.
APPENDIX B

The following programs and subroutine files were used to generate
the contents of the PROMs used in the Color Translation Module.
They are included so that the reader can refer to the algorithms.
The code is all in C.

CARTESIAN TO POLAR CONVERTER AND

POLAR TO CARTESIAN CONVERTER

PROM CONTENTS GENERATOR

/*
This program generates the contents of the PROMs needed for the
Cartesian to Polar Converter and the Polar to Cartesian Converter.
The contents are written into files with one PROM location value
per line of the file. File names correspond to the PROM labels
in the block diagram. For example, the file containing the contents
of PROM A1 is called "A1rom".
Note that the PROM sizes are not all the same.

compile as
    $cco -f conen.c -ls
*/
#include <stdio.h>
#include <usr/eal/consub.c>
define TOP 255    /* range of numbers tested */
define K1 52.55531935
#define MASK 0377    /* used for unsigned 8 bit integers */
define MASK1 01    /* low order bit */
define MASK2 02    /* next low order bit */
define MASK3 017   /* four low order bits */
define MASK4 0360  /* next four low order bits */
define MASK5 0400  /* the 9th LOB */
define MASK6 0177000 /* sign extension for 9 bit numbers */
define LOB3 07    /* 3 low order bits */
 extern double sin(),cos(),log(),exp(),abs(),atan();
 FILE *fopen(), *fp,*fpA1,*fpA2,*fpB1,*fpB2;

-88-
main()
{
    struct polar *p, pol;
    struct cart *c, car;
    char c1, c2;
    int dum1, dum2;
    char c3, c4, c5, c6;
    int t, i, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11;

    p = &pol;
    c = &car;
    printf("COLOR TRANSLATION MODULE SIMULATOR\n");
    /* generate files which contain rom values */
    /* ROMs are 512 x 4 for A and B, and 256 x 4 for the rest */
    if ((fpA1 = fopen("A1rom", "w")) == NULL) printf("file A1 error\n");
    if ((fpA2 = fopen("A2rom", "w")) == NULL) printf("file A2 error\n");
    if ((fpB1 = fopen("B1rom", "w")) == NULL) printf("file B1 error\n");
    for (dum1 = 0; dum1 < 512; dum1++){
        if (dum1 > 255) dum2 = dum1 | MASK6;
        else dum2 = dum1;
        i3 = oatanexp(dum2);
        i6 = oscatanexp(dum2);
        i4 = ((i3 & LOB3) << 1) | ((i6 >> 4) & MASK1);
        i5 = (i3 >> 3) & MASK3;
        fprintf(fpA1, "%d\n", i4);
        fprintf(fpA2, "%d\n", i5);
        i4 = i6 & MASK3;
        fprintf(fpB1, "%d\n", i4);
    }
    fclose(fpA1);
    fclose(fpA2);
    fclose(fpB1);
    printf("512x4 ROMs done\n");
    if ((fpC1 = fopen("C1rom", "w")) == NULL) printf("file C1 error\n");
    if ((fpC2 = fopen("C2rom", "w")) == NULL) printf("file C2 error\n");
    if ((fpD1 = fopen("D1rom", "w")) == NULL) printf("file D1 error\n");
    if ((fpD2 = fopen("D2rom", "w")) == NULL) printf("file D2 error\n");
    if ((fpE1 = fopen("E1rom", "w")) == NULL) printf("file E1 error\n");
    if ((fpE2 = fopen("E2rom", "w")) == NULL) printf("file E2 error\n");
    if ((fpF1 = fopen("F1rom", "w")) == NULL) printf("file F1 error\n");
    if ((fpF2 = fopen("F2rom", "w")) == NULL) printf("file F2 error\n");
    if ((fpG1 = fopen("G1rom", "w")) == NULL) printf("file G1 error\n");
    if ((fpG2 = fopen("G2rom", "w")) == NULL) printf("file G2 error\n");
    if ((fpG3 = fopen("G3rom", "w")) == NULL) printf("file G3 error\n");
    c1 = 0;
    for (dum1 = 0; dum1 < 256; dum1++){
        i3 = olnsin(c1);
        i4 = i3 & MASK3;
        i5 = (i3 & MASK4) >> 4;
        fprintf(fpC1, "%d\n", i4);
        fprintf(fpC2, "%d\n", i5);
        i8 = olncos(c1);
        i4 = i8 & MASK3;
    }
i5 = (i8 & MASK4) >> 4;
fprintf(fpD1,"%d\n",i4);
fprintf(fpD2,"%d\n",i5);
i9 = oexp(c1);
i4 = i9 & MASK3;
i5 = (i9 & MASK4) >> 4;
fprintf(fpE1,"%d\n",i4);
fprintf(fpE2,"%d\n",i5);
i10 = oln(c1);
i4 = i10 & MASK3;
i5 = (i10 & MASK4) >> 4;
fprintf(fpF1,"%d\n",i4);
fprintf(fpF2,"%d\n",i5);
i11 = oclamp(c1);
i4 = i11 & MASK3;
i5 = (i11 & MASK4) >> 4;
i6 = ((i11 & MASK5) >> 8) | ((i3 & MASK5) >> 7) | ((i8 & MASK5) >> 6);
fprintf(fpG1,"%d\n",i4);
fprintf(fpG2,"%d\n",i5);
fprintf(fpG3,"%d\n",i6);
c1++;
CPC AND PCC SUBROUTINES

/* compile only in combination with other cpc and pcc programs */
#include <stdio.h>
#define TOP 255 /* range of numbers tested */
#define PI 3.14159
#define K1 52.55531935
#define K2 162.9746617 /* 256/(PI/2) */
#define MASK 0377 /* used for unsigned 8 bit integers */
#define MASK1 01 /* low order bit */
#define MASK2 02 /* next low order bit */
#define MASK3 017 /* four low order bits */
#define MASK4 0360 /* next four low order bits */
#define MASK5 0400 /* the 9th LOB */
#define MASK6 0177000 /* sign extension for 9 bit numbers */
#define LOB3 07 /* 3 low order bits */
#define MAX .7071067812

extern double sin(),cos(),log(),exp(),abs(),atan();
extern FILE *fpA1,*fpA2;
struct polar {
    char hl,hh,sz;
    int lns;
};
struct cart {
    char x,y;
};

/* The following subroutine simulates the Polar to Cartesian Converter.
   It returns the cartesian form results in a structure polar format
   and writes the intermediate results of the simulation into the
   external file pointed to by fpA2. */
struct cart *pcc(a,t)
struct polar *a;
struct cart *t;
{
    int lncoshl,lnsinhl,lnsmax;
    char t1,t2;

    lncoshl = olncos(a->hl);
    fprintf(fpA2, "A %4x0,lncoshl);%n
    lnsinh = olnsin(a->hl);
    fprintf(fpA2, "B %4x0,lnsinhl);%n
    lnsmax = olamp(a->hl);
    fprintf(fpA2, "C %4x0,lnsmax);%n

    return t;
}
if (a->lns < lnsmmax)
    lnsmmax = a->lns;
fprintf(fpA2, " D %4x0,lnsmmax);
lncoohl += lnsmmax;
lnsinhl += lnsmmax;
t1 = lncoohl & MASK;
t2 = lnsinhl & MASK;
if (((a->hh & MASK1) == 0){
    if (lnsinhl >= 0) {
        t->y = oexp(t2);
        fprintf(fpA2, " E %4x0,t2);
        fprintf(fpA2, " F %4x0,t->y);
    }
    else t->y = 0;
    if (lncoohl >= 0) {
        t->x = oexp(t1);
        fprintf(fpA2, " G %4x0,t1);
        fprintf(fpA2, " H %4x0,t->x);
    }
    else t->x = 0;
} else {
    if (lncoohl >= 0) {
        t->y = oexp(t1);
        fprintf(fpA2, " E %4x0,t1);
        fprintf(fpA2, " F %4x0,t->y);
    }
    else t->y = 0;
    if (lnsinhl >= 0) {
        t->x = oexp(t2);
        fprintf(fpA2, " G %4x0,t2);
        fprintf(fpA2, " H %4x0,t->x);
    }
    else t->x = 0;
};
if (((a->hh & 01) ^ ((a->hh & 02) >> 1)) != 0) /* h9 XOR h9 */
t->x = -t->x;
if ((a->hh & 02)! = 0)
t->y = -t->y;
if (a->hl == 0)
    if ((a->hh & 01) != 0)
        t->x = 0;
else
    t->y = 0;
if (a->sz == 1)
    t->x = t->y = 0;
}

/* The following subroutine simulates the Cartesian to Polar Converter.
It returns the polar form results in a structure cart format
and writes the intermediate results of the simulation into the
external file pointed to by fpA1.
*/
struct polar *opc(a,p)
struct cart *a;

-92-
struct polar *p;
{
    int ln1,ln2,t1,t4;
    char t2,t3;

    ln1 = c1n(a->y);
    ln2 = c1n(a->x);
    ln1 = ln1 & MASK;ln2 = ln2 & MASK; /* unsigned integers */
    fprintf(fpA1," A %4x0,ln1);
    fprintf(fpA1," B %4x0,ln2);
    t1 = ln1-ln2;
    fprintf(fpA1," C %4x0,t1);
    t2 = catanexp(t1);
    fprintf(fpA1," D %4x0,t2);
    t3 = cscatanexp(t1);
    fprintf(fpA1," E %4x0,t3);
    if (ln1 >= ln2)
        t4 = ln1;
    else
        t4 = ln2;
    fprintf(fpA1," F %4x0,t4);
    if ((a -> y == 0) || (a -> x == 0)) {
        p->lns = t4;
        t2 = 0;
    } else {
        p->lns = t4 + t3;
        p->sz = ((a->x == 0) && (a->y == 0));
        p->hh = ~((~((a->y<0)<<(a->x<0))||(a->x==0))||(a->y==0))&MASK1;
        if (p->hh==0) p->h1 = t2;
        else p->h1 = -t2;
        p->hh = p->hh + (((a->x<0)&&(a->y==0)||(a->y<0)<<1))&MASK2;
    }
}

/* lncos ROM simulator for pcc */
/* parameter is unsigned 8 bit number representing 0 to PI/2 */
/* returns a ten bit negative number in int with sign extension */
c1nncos(b)
char b;
{
    double t;
    int it;

    if (b == -1) return(-255);
    it = b; /* type conversion */
    t = it & MASK; /* type conversion */
    t = log(cos(t/K2))^K1;
    it = t - .5; /* type conversion */
    return(it);
}

/* lnsin ROM simulator for pcc */
/* parameter is unsigned 8 bit integer representing 0 to PI/2 */
/* returns ten bit negative number in int with sign extension */
c1nnsin(b)
char b;
{
    double t;
    int it;

    if (b == 1) return (-255);
    it = b;
    t = it & MASK;
    t = log(sin(t/K2))*K1;
    it = t - .5;
    return(it);
}

/* exp ROM simulator for cpc */
/* parameter is 8 bit unsigned */
char exp(b)
char b;
{
    double t;
    char c;
    int it;

    if (b == -1) return(127); /* special case to prevent rounding effect */
    else {
        it = b;
        t = it & MASK;
        t = exp(t/K1);
        c = t + .5;
        return(c);
    }
}

/* ln ROM simulator for cpc */
/* parameter is 8 bit signed */
char lnl(b)
char b;
{
    double t;
    char c;

    t=b;
    if (t<0) t = -t;
    t=X1 * log(t);
    c=t + .5;
    return(c);
}

/* atan exp ROM simulator for cpc */
/* parameter is 9 bit signed represented in an integer */
/* with sign extension */
/* Returns an 8 bit unsigned number in char */
char oatanexp(b)
int b;
{
# ln sin atan exp and ln cos atan exp ROM simulator for cpc */
# parameter is 9 bit signed */
# represented in an integer with sign extension */
char oscatanexp(b)
int b;
{
    double t;
    char c;
    t=b;
    if (t >= 0)
        t = -log(sin(atan(exp(t/K1)))) * K1;
    else
        t = -log(cos(atan(exp(t/K1)))) * K1;
    c=t + .5;
    return(c);
}

# clamping ROM simulator */
# parameter is 8 bit unsigned */
# returns a 9 bit positive number in int */
clamp(b)
char b;
{
    double t1,t2,t;
    int it;
    it = b;
    t = it & MASK;
    if (t==0) it = 255;
    else if (t>128)
        it = K1 * log(127.0/sin(t/K2)) + .5;
    else
        it = K1 * log(127.0/cos(t/K2)) + .5;
    return (it);
SELECTIVE COLOR CORRECTION

PR-ROM CONTENTS GENERATOR

/*

This program generates the contents of PROMs A, B, C, and D in the Special Color Correction Module. The outputs are written into the files Arom, Brom, Crom, and Drom, and the format is simply one PROM location (8 bit value) per line in the file.

Compile as
    >oc -f socrom.c -Ls
*/
#include <stdio.h>

/* define the centers hues for each color */
#define MAGENTA 107
#define BLUE 274
#define CYAN 462
#define GREEN 619
#define YELLOW 786
#define ORANGE 880
#define RED 974

#define PI /2 1.570796327

extern double cos();

main()
{
    int in1, out1, out2, out3, out4;

    printf("SELECTIVE COLOR CORRECTION
PR-ROM CONTENTS GENERATOR\n");
    if ((fpA = fopen("Arom", "w")) == NULL) printf("file A error\n");
    if ((fpB = fopen("Brom", "w")) == NULL) printf("file B error\n");
    if ((fpC = fopen("Crom", "w")) == NULL) printf("file C error\n");
    if ((fpD = fopen("Drom", "w")) == NULL) printf("file D error\n");
    for (in1 = 0; in1 < 1024; in1++) {
        out1 = oeewn(w(in1);
        out2 = ooddw(in1);
        out3 = oeverall(in1);
        out4 = ootrol(in1);
        fprintf(fpA, "%d\n", out1);
        fprintf(fpB, "%d\n", out2);
        fprintf(fpC, "%d\n", out3);
        fprintf(fpD, "%d\n", out4);
    }
}
/* returns even weighting function
   input is a ten bit positive number
   output is an eight bit positive number in integer form */

evenw(in)
int in;
{
    double t;
    int i1;

    t = in;
    if(t < MAGENTA)
        t = cos(PIOVER2 * (MAGENTA - t) / (MAGENTA + 1024 - RED));
    else if(t < BLUE)
        t = cos(PIOVER2 * (BLUE - t) / (BLUE - MAGENTA));
    else if(t < CYAN)
        t = cos(PIOVER2 * (t - BLUE) / (CYAN - BLUE));
    else if(t < GREEN)
        t = cos(PIOVER2 * (GREEN - t) / (GREEN - CYAN));
    else if(t < YELLOW)
        t = cos(PIOVER2 * (t - GREEN) / (YELLOW - GREEN));
    else if(t < ORANGE)
        t = cos(PIOVER2 * (ORANGE - t) / (ORANGE - YELLOW));
    else if(t < RED)
        t = cos(PIOVER2 * (t - ORANGE) / (RED - ORANGE));
    else
        t = cos(PIOVER2 * (1024 + MAGENTA - t) / (1024 + MAGENTA - RED));

    i1 = 255 * t + .5;
    return(i1);
}

/* returns odd weighting function
   input is a ten bit positive number
   output is an eight bit positive number in integer form */

oddw(in)
int in;
{
    double t;
    int i1;

    t = in;
    if(t < MAGENTA)
        t = cos(PIOVER2 * (t + 1024 - RED) / (MAGENTA + 1024 - RED));
    else if(t < BLUE)
        t = cos(PIOVER2 * (t - MAGENTA) / (BLUE - MAGENTA));
    else if(t < CYAN)
        t = cos(PIOVER2 * (CYAN - t) / (CYAN - BLUE));
    else if(t < GREEN)
        t = cos(PIOVER2 * (t - CYAN) / (GREEN - CYAN));
    else if(t < YELLOW)
        t = cos(PIOVER2 * (YELLOW - t) / (YELLOW - GREEN));
    else if(t < ORANGE)
t = cos(PIOVER2 * (t - YELLOW) / (ORANGE - YELLOW));
else if(t < RED)
    t = cos(PIOVER2 * (RED - t) / (RED - ORANGE));
else
    t = cos(PIOVER2 * (t - RED) / (MAGENTA + 1024 - RED));
i1 = 255.0 * t * t + .5;
return(i1);

/\ returns overall weighting function
  \ input is a ten bit positive number
  \ output is an eight bit positive number in integer form

overall(in)
int in;
{
    return(255);
}

/\ returns control signals
  \ input is a ten bit positive number
  \ output is a four bit positive number in integer form
  \ low order two bits specify even color
  \ the other two bits specify the odd color

ocontrol(in)
int in;
{
    int i1;

    if(in < MAGENTA)
        i1 = 15; /* select magenta (even) and red (odd) */
    else if(in < BLUE)
        i1 = 0; /* select blue (even) and magenta (odd) */
    else if(in < CYAN)
        i1 = 4; /* select blue (even) and cyan (odd) */
    else if(in < GREEN)
        i1 = 5; /* select green (even) and cyan (odd) */
    else if(in < YELLOW)
        i1 = 9; /* select green (even) and yellow (odd) */
    else if(in < ORANGE)
        i1 = 10; /* select orange (even) and yellow (odd) */
    else if(in < RED)
        i1 = 14; /* select orange (even) and red (odd) */
    else i1 = 15; /* select magenta (even) and red (odd) */
    return(i1);
}
SPECIAL COLOR CORRECTION

PROM CONTENTS GENERATOR

 */
This program generates the contents of the PROMs needed in the Special Color Correction Module. The outputs are written into files with one entry per line of the file. Files are named according to the label of the PROMs; for example, the file corresponding to PROM G3 is called "G3spcc."

compile as
  > cc -f spccrom.c -lS
*/
#include <stdio.h>
#include </usr/eal/specsub.c>
#define K1 52.55531935
#define K3 163.98634
#define K4 158.35114
#define HOB10 010000 /* HOB for 10 bit numbers */
#define HOB11 020000 /* HOB for 11 bit numbers */
#define HOB12 040000 /* HOB for 12 bit numbers */
#define HOB16 0100000 /* HOB for 16 bit numbers */
#define MASK4 017 /* MASK for four bit numbers */
#define MASK8 0377 /* MASK for eight bit numbers */
#define MASK10 017777 /* MASK for 10 bit numbers */
#define MASK12 077777 /* MASK for 12 bit numbers */
#define SE10 0176000 /* Sign extension for 10 bit numbers */
#define PI 3.14159

extern double log(), exp(), cos();
extern oln10(), oexp10(), oln8(), oexp8(), olnw();

main()
{
  int oln10(), oexp10(), oln8(), oexp8();
  int in1, out1, out2, out3, out4, out5;
  int i1;

  printf("SPECIAL COLOR CORRECTION PROM CONTENTS GENERATOR\n");
  if ((fpA1=fopen("A1spcc","w")) == NULL) printf("file A1 error\n");
  if ((fpA2=fopen("A2spcc","w")) == NULL) printf("file A2 error\n");
  if ((fpA3=fopen("A3spcc","w")) == NULL) printf("file A3 error\n");

  -99-
if ((fpB1=fopen("B1spcc","w")) == NULL) printf("file B1 error\n");
if ((fpB2=fopen("B2spcc","w")) == NULL) printf("file B2 error\n");
if ((fpD1=fopen("D1spcc","w")) == NULL) printf("file D1 error\n");
if ((fpD2=fopen("D2spcc","w")) == NULL) printf("file D2 error\n");
if ((fpD3=fopen("D3spcc","w")) == NULL) printf("file D3 error\n");
if ((fpE1=fopen("E1spcc","w")) == NULL) printf("file E1 error\n");
if ((fpE2=fopen("E2spcc","w")) == NULL) printf("file E2 error\n");
if ((fpE3=fopen("E3spcc","w")) == NULL) printf("file E3 error\n");
if ((fpF1=fopen("F1spcc","w")) == NULL) printf("file F1 error\n");
if ((fpF2=fopen("F2spcc","w")) == NULL) printf("file F2 error\n");
if ((fpF3=fopen("F3spcc","w")) == NULL) printf("file F3 error\n");
if ((fpG1=fopen("G1spcc","w")) == NULL) printf("file G1 error\n");
if ((fpG2=fopen("G2spcc","w")) == NULL) printf("file G2 error\n");
if ((fpG3=fopen("G3spcc","w")) == NULL) printf("file G3 error\n");
for(in1 = 0; in1 <= 1023; in1++) {
    out1 = oln10(in1);
    out2 = oexp10(in1);
    out5 = olnw(in1);
    i1 = out1 & MASK4;
    fprintf(fpA1, "\%d\n", i1);
    i1 = (out1 >> 4) & MASK4;
    fprintf(fpA2, "\%d\n", i1);
    i1 = (out1 >> 8) & MASK4;
    fprintf(fpA3, "\%d\n", i1);
    i1 = out5 & MASK8;
    fprintf(fpB1, "\%d\n", i1);
    i1 = (out5 >> 8) & MASK8;
    fprintf(fpB2, "\%d\n", i1);
    i1 = out2 & MASK4;
    fprintf(fpF1, "\%d\n", i1);
    fprintf(fpG1, "\%d\n", i1);
    i1 = (out2 >> 4) & MASK4;
    fprintf(fpF2, "\%d\n", i1);
    fprintf(fpG2, "\%d\n", i1);
    i1 = (out2 >> 8) & MASK4;
    fprintf(fpF3, "\%d\n", i1);
    fprintf(fpG3, "\%d\n", i1);
    i1 = out3 & MASK4;
    fprintf(fpE1, "\%d\n", i1);
    i1 = (out3 >> 4) & MASK4;
    fprintf(fpE2, "\%d\n", i1);
    i1 = (out3 >> 8) & MASK4;
    fprintf(fpE3, "\%d\n", i1);
    if (in1 < 256) {
        out4 = oln8(in1);
        i1 = out4 & MASK4;
        fprintf(fpD1, "\%d\n", i1);
        i1 = (out4 >> 4) & MASK4;
        fprintf(fpD2, "\%d\n", i1);
        i1 = (out4 >> 8) & MASK4;
        fprintf(fpD3, "\%d\n", i1);
    }
}
/* compile only as part of other spcc programs */
#include <stdio.h>

#define K1 52.55531935
#define K3 163.98634
#define K4 158.35114
#define HOB10 010000 /* HOB for 10 bit numbers */
#define HOB11 020000 /* HOB for 11 bit numbers */
#define HOB12 040000 /* HOB for 12 bit numbers */
#define HOB16 01000000 /* HOB for 16 bit numbers */
#define MASK10 017777 /* MASK for 12 bit numbers */
#define SE10 01760000 /* Sign extension for 10 bit numbers */
#define PI 3.14159

extern double log(), exp(), cos();
/* returns K3 * ln |in|
input is a ten bit offset binary number (0 to 1023)
output is a twelve bit positive number returned as integer
where the bottom ten bits are the magnitude of the output,
the 12th bit is zero if the input represents zero
(i.e. equals 512), and
the 11th bit is one if the input represents a negative number
(i.e. is less than 512) */

oln10(in)
int in;
{
  double t;
  int i1;

  if (in == 512) return(0);
  t = in - 512;
  if (t < 0) t = -t;
  t = K3*log(t);
  i1 = t + .5;
  i1 = i1 | HOB12;
  if (in < 512) i1 = i1 | HOB11;
  return(i1);
}

/* returns exp(in/K3)
input is a ten bit positive number
output is a nine bit positive number */
#/ oexp10(in) int in; 
{ 
  double t;
  int i1;

  if (in == 1023) return(511);
  t = in;
  t = exp(t/K3);
  i1 = t + .5;
  return(i1);
}

/# returns -K1 # ln(in) 
input is an 8 bit non-negative number 
output is a twelve bit number where the low order ten bits 
are a negative number representing -K1*ln(in) in two's complement 
the eleventh bit is ignored 
and the 12th bit is zero if the input is zero. 
/# oln8(in) int in; 
{ 
  double t;
  int i1;

  if (in == 0) return(0);
  t=in;
  t=K1*log(t);
  i1 = t+.5;
  i1 = -i1;
  i1 = (i1 & MASK10) | HOB12;
  return(i1);
}

/# returns K4 # exp(in/K1) 
input is a ten bit two's complement number (-291 to 273) 
(with no sign extension) 
output is a ten bit positive number 
/# oexp8(in) int in; 
{ 
  double t;
  int i1;

  if (((in & HOB10) != 0) in = in | SE10;  /* sign extension */
  if (in > 98) return(1023);
  t = in;
  t = K4 * exp(t / K1);
  i1 = t + .5;
  return(i1);
}
returns K3 * ln(W(in))
   K3 * ln(cos((in - 512) * PI / 200) ** 2)  412 < in < 612
   0                               elsewhere

input is a ten bit offset binary number
output is a 16 bit number.
low order 12 bits are a two's complement number in the range
   from zero to -1024.
Next three bits are zero.
High order bit is zero if W(in) is zero (making ln(W(in)) indeterminate)

olnw(in)
int in;
{
   double t;
   int i1;

   if ((in < 412) || (in > 612)) return(0);
   t = in;
   t = cos((in - 512) * PI / 200);
   t = K3 * log(t**t);
   if (t < -1024) return(0);
   i1 = t;
   i1 = (i1 & MASK12) | HOB16;
   return(i1);