THE DESIGN OF A CMOS COLOUR PALETTE INTEGRATED CIRCUIT

By

ROBERT JAMES MIELCARSKI

B.A.Sc., The University of British Columbia, 1981

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
ELECTRICAL ENGINEERING

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 1983

© Robert James Mielcarski, 1983
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of **ELECTRICAL ENGINEERING**

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date **18-OCT**
ABSTRACT

This thesis is concerned with the design, fabrication, and testing of an oxide-isolated complementary metal-oxide-silicon (ISO-CMOS) colour palette integrated circuit. Concepts in computer aided design (CAD) are discussed and the tools required for a complete CAD system are detailed. The fabrication steps, design rules, and electrical parameters of the ISO-CMOS technology are described. The colour palette function and its applications in graphics display systems are explained. The remainder of the thesis details the design philosophy, the design procedure, the simulation results, and the testing performed on the colour palette circuit.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title Page</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>Acknowledgment</td>
<td>ix</td>
</tr>
<tr>
<td>Chapter 1: Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Microelectronics Overview</td>
<td>1</td>
</tr>
<tr>
<td>Project Objectives and Accomplishments</td>
<td>4</td>
</tr>
<tr>
<td>Chapter 2: Review of IC Design Concepts and Tools</td>
<td>6</td>
</tr>
<tr>
<td>IC Design Overview</td>
<td>6</td>
</tr>
<tr>
<td>VLSI Complexity Management</td>
<td>10</td>
</tr>
<tr>
<td>Classes of Custom Circuits</td>
<td>14</td>
</tr>
<tr>
<td>CAD Tools</td>
<td>17</td>
</tr>
<tr>
<td>The Mead and Conway Revolution</td>
<td>27</td>
</tr>
<tr>
<td>Chapter 3: Selection and Justification of a Custom Chip Project</td>
<td>35</td>
</tr>
<tr>
<td>Background Information</td>
<td>35</td>
</tr>
<tr>
<td>The Colour Palette Function</td>
<td>36</td>
</tr>
<tr>
<td>Colour Palette Applications</td>
<td>37</td>
</tr>
<tr>
<td>The Colour Palette Market Potential</td>
<td>37</td>
</tr>
<tr>
<td>The Current MPR Colour Palette Implementation</td>
<td>39</td>
</tr>
<tr>
<td>Colour Palette Specifications</td>
<td>40</td>
</tr>
</tbody>
</table>
Appendices ................................................................. 121

A. Schematics and Block Diagrams of the Microtel . . 121
   Pacific Research Ltd. Discrete Component
   Colour Palette Implementation

B. Colour Palette Specifications . . . . . . . . . . . . . . . . 126

C. GTE ISO-CMOS Process Sequence . . . . . . . . . . . . 138

D. The CDS Layout Language . . . . . . . . . . . . . . . . . 156

E. Top Level Cell Source Listings . . . . . . . . . . . . . . . 163
LIST OF TABLES

I. ISO-CMOS Capacitance Data ........................................... 52
II. ISO-CMOS Resistivity Data ............................................. 52
III. ISO-CMOS Simplified Design Rules ................................. 53
IV. Memory Control and Data Signals ..................................... 77
V. Memory Cell Bus Loading .................................................. 81
VI. Pipeline Register Bus Loading .......................................... 85
VII. Critical Path Simulation Results ...................................... 95
VIII. Chip #1 Memory Test Summary ....................................... 107
LIST OF FIGURES

1. Number of Components per Integrated Circuit ............... 2
   Versus Time

2. The Four Levels of Circuit Abstraction .................... 6

3. The Three Major Design Activities .......................... 8

4. Programmable Logic Array ................................. 16

5. CAD System Overview ..................................... 18

6. Example of Caltech Intermediate Form ...................... 30

7. Caltech Intermediate Form Relationships .................... 31

8. Colour Palette Block Diagram .............................. 41

9. Examples of True Complementary CMOS Circuits .............. 44

10. CMOS Inverter Characteristics ............................ 45

11. Example of a Dynamic CMOS Circuit ....................... 46

12. ISO-CMOS Fabrication Sequence ............................ 51

13. Layout of an Inverter with 2 Enable Signals ............... 54

14. CDS Layout Program Example .............................. 61

15. Stick Diagram Example .................................... 65

16. Colour Palette Cell Hierarchy ............................. 71

17. Traditional Static CMOS Memory Cell ...................... 75

18. Memory Bus Structure ..................................... 78

19. Memory Cell Logic Schematic ............................. 80

20. Memory Cell Stick Diagram ................................. 82

21. Memory Cell Layout ........................................ 83

22. Pipeline Register Logic Schematic ......................... 85

23. Pipeline Register Stick Diagram .......................... 86

24. Pipeline Register Layout .................................. 87

25. Dynamic DAC Layout ....................................... 90
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Static DAC Layout</td>
<td>91</td>
</tr>
<tr>
<td>27</td>
<td>Input Pad Layout</td>
<td>92</td>
</tr>
<tr>
<td>28</td>
<td>Process Monitor Layout</td>
<td>93</td>
</tr>
<tr>
<td>29</td>
<td>The First Iteration Chip</td>
<td>96</td>
</tr>
<tr>
<td>30</td>
<td>Static First Iteration Test Circuit</td>
<td>97</td>
</tr>
<tr>
<td>31</td>
<td>Process Monitor Inverter Characteristics</td>
<td>98</td>
</tr>
<tr>
<td>32</td>
<td>Dynamic DAC Test Circuit</td>
<td>99</td>
</tr>
<tr>
<td>33</td>
<td>Dynamic DAC Output with Static Inputs</td>
<td>100</td>
</tr>
<tr>
<td>34</td>
<td>Dynamic DAC Output at 1 MHz</td>
<td>100</td>
</tr>
<tr>
<td>35</td>
<td>The Second Iteration Chip</td>
<td>105</td>
</tr>
<tr>
<td>36</td>
<td>Second Iteration Test Circuit</td>
<td>106</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

I would like to thank my supervisors, Lawrence Young, Peter Lawrence, and G. Schrack for their wise counsel. Professor Young deserves special note for orchestrating dialogue with industry and for providing the means to attend several courses on integrated circuit design. This thesis would not have been written without Professor Young's encouragement. I would also like to thank my colleague, Gordon Cheng, for his contributions on the colour palette project.

I extend my deepest gratitude to the people at Microtel Pacific Research Ltd. for their generosity in providing the tools necessary to implement the colour palette project. In particular I would like to thank Warren Snyder, Mike Pejskar, Mel Phillips, Paul Thiel, and especially Gerhard Schmiing.

I would like to thank Anatek Electronics Ltd., the British Columbia Institute of Technology, the British Columbia Telephone Company, Glenayre Electronics Ltd., and Microtel Pacific Research Ltd. for their contributions to the B.C. Electronics Industry Graduate Fellowship in Microelectronics. This fellowship made it possible for me to continue my studies and provided funds for several worthwhile visits to integrated circuit design and fabrication facilities.

I would also like to thank the Natural Sciences and Engineering Research Council of Canada for awarding me a scholarship.

Last but not least I would like to thank Heather Sharkey for her infinite patience and support.
CHAPTER 1: INTRODUCTION

Microelectronics Overview

Microelectronics is one of man's great technological achievements. The Industrial Revolution leveraged man's physical power, microelectronics has leveraged man's intellectual power.

Microelectronics technology has had an exhilarating history [WEBER 80, JONES 82]. Bardeen, Brattain, and Shockley invented the bipolar transistor in 1947 while working at Bell Laboratories. The new device was met with skepticism because of its poor reliability and high cost. These reservations were rapidly overcome as the technology improved and because of the transistor's inherent low power and small size. In 1958 Jack Kilby of Texas Instruments realized that all of the components necessary for an electronic circuit could be made from a semiconductor. He then went on to develop the first integrated circuit (a simple oscillator). The level of sophistication rapidly accelerated as shown by Figure 1. This curve supports the prediction in 1964 by Gordon E. Moore, who was then director of research at Fairchild, that the number of components per integrated circuit would double every year. Today, only 36 years after the transistor was invented, Hewlett Packard has succeeded in integrating 450,000 transistors on a single semiconductor substrate [CANEPA 83].
An integrated circuit (IC) has been defined [GLASER 77] as "a group of inseparably connected circuit elements fabricated in place on and within a substrate". There are several important reasons for the widespread use of IC's in electronic systems. Integrated circuits have an extremely low cost per function ratio because of the batch nature of their fabrication. Although many expensive steps are required to fabricate an IC, these steps are applied simultaneously to many silicon wafers. Each wafer in turn has many identical IC's patterned on it. The fabrication costs are therefore distributed over thousands of IC's resulting in a low per unit cost.
The cost per function ratio has continually declined at a rate unprecedented by other manufactured goods. For example, the cost per bit of memory in 1973 was about 0.4 cents and by 1977 it was down to 0.1 cents [NOYCE 77]. Today the cost per bit of memory is only about 0.005 cents. It costs the manufacturer little more to fabricate a complex IC than a simple IC provided that the yield (percentage of defect-free IC's per batch) can be kept reasonably high. The steps required to fabricate a complex IC are identical to those required to fabricate a simple IC.

Integrated circuits also reduce system support costs. IC's consume less space and power than discrete components that perform the same function, thereby reducing the cost of floor space, mounting racks, power supplies, and cooling fans. Labor costs are reduced because there are fewer components to assemble. Finally IC's improve the system reliability because the number of solder joints and other defect-prone factors are reduced.

Now that it is possible to fabricate very large scale integrated (VLSI) circuits, which are usually defined to contain more than 100,000 transistors, the emphasis has shifted to the problems associated with designing such an IC. The components available for designing a simple integrated circuit are identical to those used in a VLSI circuit. These include transistors, wires, resistors, and capacitors. The difficulty in designing VLSI circuits therefore arises not from the complexity of the individual components, but rather from the vast number of components used.
One of the key design problems is interconnecting the components on a VLSI circuit because the number of interconnections increases faster than the number of devices [SEQUIN 81]. Other problems include proving the correctness of a design and maintaining documentation on a VLSI circuit. Much research is being done to develop better computer-aided design (CAD) tools to solve these and other design problems [FEUER 83]. VLSI has advanced and will continue to advance only as new ideas emerge in computer-aided design.

Project Objectives and Accomplishments

The first objective of this project was to become familiar with computer-aided design tools and concepts. It was decided that the best way to accomplish this would be to design an actual integrated circuit. It was clearly desirable to choose a circuit that had some practical application so that the full cycle of specification, design, fabrication, testing, and implementation could be observed. The circuit chosen was a colour palette which has many applications in video circuitry. In particular the circuit is targeted for the Microtel Pacific Research Ltd. Telidon system. The second objective of this project was to become acquainted with a current semiconductor technology. Fortunately we had the opportunity to work with the popular ISO-CMOS technology. The final objective of this project was to become familiar with IC simulation and testing problems.
We succeeded in designing a 4 by 4 bit colour palette prototype integrated circuit. The circuit was fabricated and test results show that it is functional with the exception of a few minor design errors. Several test circuits were included on the prototype chip to facilitate the debugging process. These circuits turned out to be well worth the extra effort required to implement them. A process monitor was designed and included on the chip for evaluating the fabrication process.

Using knowledge gained from the prototype circuit the colour palette was redesigned into its full 12 by 16 bit configuration. The circuit contains about 6000 transistors and is 4070 x 6398 microns in size. Critical paths on the circuit were simulated to estimate performance. A new static digital to analog converter (DAC) was designed and included on the chip in addition to the dynamic DAC used on the prototype. The full size colour palette has been fabricated and test results show that all circuit modules are functional. Processing problems however appear to be the cause of our inability to locate a chip on which all modules operate simultaneously.
IC Design Overview

One of the key differences between integrated circuit design and the design of other electrical systems is that IC design involves a greater number of circuit representations. Integrated circuit design can be viewed as an iterative mapping between four levels of circuit abstraction (Figure 2): functional, logical, electrical, and physical [NEWTON 81].

Figure 2. The Four Levels of Circuit Abstraction
The functional level displays the IC's architecture and specifies its macroscopic properties such as speed, size, and power consumption. A black box diagram or high level language is often used to present the functional information. Suppression of detail permits the designer to experiment with different architectures and algorithms.

The logical level expresses each of the modules defined at the functional level in terms of logic symbols or other abstraction such as register transfer notation. It often becomes clear at this stage that the functional representation needs further subdivision or reorganization to achieve an efficient design. For example, an excessive number of communication lines may be the result of a poor choice of module boundaries.

The electrical representation expresses the logical level in terms of the circuit components: transistors, resistors, capacitors, and wires. The logical level often has to be modified to improve the circuit performance. For example, NOR-NOR logic is faster than NAND-NAND logic because of the way that these functions are implemented with MOS devices. Excessive loading of transistors may also force a revision at the logical level.

The physical level contains the geometric mask data that implements the electrical level components. The unpleasant realities of layout, such as the restrictive two-dimensional nature of IC's, often cause problems. For example, it may prove impossible
to route an interconnection as specified at higher levels. IC design clearly tends to be an iterative process.

There are three major design activities required at each level of abstraction: synthesis, evaluation, and validation [LATTIN 81]. Figure 3 shows how these activities interact.

![Diagram showing the three major design activities: Synthesis, Evaluation, and Validation.]

Figure 3. The Three Major Design Activities

Synthesis is the mapping of a description at one level into a description at the next lower level. Synthesis is a non-trivial task because of the widely differing descriptions. For example, the physical level description tends to be graphical in nature whereas the functional description tends to be textual.
Evaluation determines if the description meets the specifications and constraints of the particular level. At the physical level, for example, one is interested in verifying that there are no geometric design rule violations and that capacitive loading does not excessively degrade performance.

Validation is the process of verifying that a particular description implements the higher level descriptions correctly. Validation can be thought of as the inverse of the synthesis transformation. For example, given the electrical description extract the logic and compare against the logical description.

All of the rich variety of design techniques and methodologies that have been proposed or are in use can be fitted into the above scenario of three activities in each of four levels of description. Techniques and methodologies can be distinguished by the descriptions they use, and by their tools (or absence of them) that perform the activities.

The first integrated circuits were designed without aid of computers. Evaluation of the logic was usually performed by bread-boarding the circuit with discrete components. Synthesis of the physical level involved drawing the circuit shapes onto rubylith (plastic) which was then photographically reduced to produce the masks. A painstaking visual check constituted validation of the masks.
Computers were first used to simulate discrete circuits but by the early 1970's integrated circuits were also being simulated. The next application of CAD was to assist in the synthesis of the physical level [AVENIER 83]. Designs were (and often still are) hand drawn on mylar plastic and then digitized into a database. Interactive graphics can then be used to edit the layout. When complete, a pattern generator produces the masks from the graphics database. First generation CAD systems, such as the Calma GDS I system, were essentially computerized drafting machines as described above. Tools to assist with validation and evaluation (with the exception of simulators) were non-existent until later in the 1970's. Design rule checkers (DRC) for evaluating the mask layout were the next tools to become prevalent. The DRC was followed by electric rule checkers (ERC) and circuit extractors for validation. An overview of a complete CAD system and a more detailed look at each tool will be presented later in this chapter.

VLSI Complexity Management

Design of a VLSI circuit can take several man-years and requires efficient coordination between design team members. Locating an error is difficult because of the complexity, and correcting it can be very expensive because of the cost of mask making and fabrication. VLSI design therefore requires the exploitation of regularity and hierarchy.
A regularity factor has been defined as the ratio of the number of devices drawn to the total number of devices on the circuit [LATTIN 79]. Read only memory (ROM) and programmable logic arrays (PLA) are examples of circuits with high regularity. There are several important advantages of regularity [FAIRBAIRN 82]. Regular structures can often be generated by software if the interconnections between cells are well defined. Software for generating PLA's is commonplace [LANG 79]. The designer need only specify a few parameters such as the number of inputs, outputs, and minterms, and provide a bit map for programming the AND and OR planes of the PLA [GLASSER 80]. Regularity reduces the complexity of a chip and facilitates testing by reducing the number of unique elements. The new emphasis on regularity becomes clear when one compares state-of-the-art circuits such as the Intel iAPX 432 microprocessor [LATTIN 81] with earlier circuits such as the Intel 8080 microprocessor [INTEL 79].

Hierarchy has been defined as "the ability to view and manipulate a design data-base at various levels of abstraction" [JENNE 82]. The idea is to divide and conquer a circuit into smaller manageable modules while at the same time maintaining enough information to reconstruct the whole. Much has been written on the advantages of hierarchical design [TRIMBERGER 81, NIESSEN 83]. Exploitation of hierarchy reduces design time because the circuit can be assembled from a library of previously defined cells. The probability of achieving a correct design should increase because the amount of detail that the designer must cope with at any given time is reduced and because the
library cells are presumably error free. Hierarchical design encourages localized communication between cells, thus avoiding the "spaghetti layouts" of the past. In theory, hierarchical design should increase the speed of some CAD tools. For example, once a lower level module has been checked for design rule violations it need not be rechecked each time it is used [WHITNEY 81]. Hierarchical design may result in a smaller data-base because only one instance of each module must be stored.

There are, however, several potential problems with hierarchical design. Working from a fixed library of modules may result in poorer performance and increased chip area because the modules are not customized to the application. Given a good library this restriction may be no worse than that imposed on a printed circuit board (PCB) design using standard TTL components. Some workers suggest that proponents of hierarchical design have never attempted to implement such a CAD system [SOUKUP 81]. Several serious implementation problems relating to data-base management and module boundary conditions are pointed out.

There are very strong analogies between software engineering and VLSI complexity management [SEQUIN 83]. Consider the history of software. In the beginning memory was expensive and the hardware performance was poor. Programmers therefore wrote in low level languages such as assembler to extract the maximum performance with the least amount of memory. Programmer efficiency was not a priority. As memory became cheaper and hardware performance improved, more powerful but unstructured languages such as
Fortran became prevalent, and it then became possible to write very large and complex programs. Programmer efficiency and software reliability fell off because of the difficulty in maintaining and modifying such large programs. New software concepts were sought and the result has been the boon in structured programming and supporting languages such as Pascal, C, and Ada. Structured software improves productivity of the programmer and reliability of the program at the expense of performance and memory.

Now consider VLSI complexity management. In the beginning low fabrication yields and poor device density meant that only a few functions could be integrated on a single chip. As a result designs tended to be flat (no hierarchy) and highly customized (poor regularity) to achieve the maximum performance in a given chip area. Fabrication yield and device density have since greatly improved resulting in VLSI circuits. Designer productivity and a priori circuit verification are now very strong issues. Regular hierarchical design, the structured programming analogy, has been proposed as a solution. Note however that a VLSI circuit still contains the same basic components that were present on the first integrated circuit, just as the computer still executes the same basic instructions that it did when first invented. Nothing fundamental has changed except the complexity.
Classes of Custom Circuits

Within the framework of IC design presented earlier, there are three classes of custom circuits: programmable arrays, standard cells, and fully custom circuits. Each class of circuits can be designed with a variety of CAD tools and methodologies, and each class is best suited to certain applications. It should be emphasized that often all three classes of circuits are used on the same integrated circuit.

A programmable array can be defined as an array of components in fixed locations whose function is determined by customizing the geometry on specific masks. The common types of programmable arrays are gate arrays, programmable logic arrays (PLA), storage logic arrays (SLA), and read only memories (ROM).

Gate arrays are known by a variety of names such as master slice or uncommitted logic arrays (ULA), and are available in several technologies such as CMOS, bipolar, and integrated injection logic [NEWTON 81, ROFFELEN 81, TRIMBERGER 82]. A gate array is an array of transistors or gates with periodic spaces or channels provided for wiring. The circuit function is determined by customizing the final layer (or two) of interconnection metal. Gate array blanks can be produced in volume and stockpiled because the fabrication sequence is standard up until the final masks. As a result gate arrays provide the least expensive and fastest method of obtaining custom parts. Gate array users design at the logic level and therefore avoid the necessity of under-
standing the fabrication process. The highly regular structure permits successful use of placement and routing tools for layout of the interconnection masks [WERNER 81]. The main disadvantage of gate arrays is a low circuit density because it is rare to utilize 100 percent of the gates available. There has also been some concern expressed about the lack of standardization and second sources of gate arrays [SCHOFIELD 81].

Programmable logic arrays (PLA) provide a means of mapping irregular boolean functions onto a regular circuit structure [WEINBERGER 67, FLEISHER 75, COOK 79]. PLA's are most often used as "glue circuitry" or as finite state machines for control of other circuit blocks. A PLA contains two arrays of transistors as shown in Figure 4, and performs a sum of products function. The first array is called the AND plane and it forms the required product terms from the inputs. The second array is called the OR plane it forms the desired sum of products for output. PLA's require no internal routing and their speed and density are good. As stated earlier in this chapter PLA's can be generated automatically by software [LANG 79].

Structured logic arrays (SLA) are a relatively new approach to programmable array design [PATIL 79, SMITH 82]. The SLA is similar to the PLA except that storage elements (flip-flops) are distributed throughout the array. Advantages include better utilization of the PLA's gates and higher functionality. Disadvantages include poor density and a lack of design aids. Industry is just beginning to use the SLA. Intel uses an SLA on the iAPX
computer [BAYLISS 81] and Boeing has done an evaluation with favorable results [GOATES 82].

Figure 4. Programmable Logic Array

The design of IC's with standard cells is analogous to the design of PC boards with TTL circuits. The designer is provided with a library of standard functions and therefore is divorced from many of the layout and fabrication details. The majority of his effort is spent interconnecting the modules. Standard cells provide a good compromise between the fully custom approach and gate arrays. Savings are achieved in the design cycle because of the predesigned modules however all of the masks must be customized and therefore the fabrication savings achieved by gate arrays are not possible. Standard cell libraries and design tools vary greatly. Some systems fix the cell pitch in one dimension
only whereas others fix both dimensions. Wiring is sometimes confined to predefined channels and other times the designer has complete wiring freedom. One of the problems with standard cell design is that to achieve good density one requires a large library with multiple cell implementations of each function. For example, efficient routing in a variety of situations may require several choices of cell input and output locations. Another disadvantage is that a standard cell library can be very expensive to purchase. Standard cells are widely used by industry [HAYDAMACK 81].

Fully custom design is constrained only by the design rules and the talents of the designer. The advantages are high density and high performance. The disadvantages of course are the high cost of design and debugging. Fully custom design is usually the approach taken for high volume circuits that can make the increased density profitable. Referring back to the earlier software analogy, fully custom design is equivalent to assembly language programming.

CAD Tools

An overview of a complete CAD system is shown in Figure 5. A discussion of each tool in detail is a thesis unto itself, therefore the functions and features of the tools will only be summarized.
Figure 5. CAD System Overview
Common tools available for use at the synthesis stage of design include digitizers, interactive graphics editors, algorithmic layout programs, placement and routing tools, and the silicon compiler. Synthesis tools have traditionally concentrated on layout of the physical level of representation (i.e. the masks), however, the silicon compiler is a recent exception to this rule.

Digitizers are used to enter the coordinates of vertices directly from an enlarged hand-drawing of the masks. Although this method is primitive compared to interactive graphics data entry, it is still commonly used [NEWTON 81, AVENIER 83]. The main reason for this is that there are usually more IC designers than there are graphics work stations and therefore the stations cannot be tied up for bulk data entry. Interactive graphics is reserved for editing the masks after they have been entered via a digitizer. Many designers also seem to prefer doing their initial layouts on paper.

Interactive graphics editors are available from a variety of suppliers such as Applicon, Avera, Calma and Mentor with prices ranging from $50,000 to $450,000 [WERNER 83]. The three main approaches to interactive graphics layout are geometric, symbolic, and standard cells.

Geometric layout systems are essentially drawing machines. The designer is provided with a variety of commands for manipulating geometric shapes such as MOVE and ROTATE. Often more
powerful operations such as the ability to stretch a cell about some axis are provided [HEWLETT-PACKARD 81]. Geometric layout systems require complex post processing tools for evaluation and validation because there is usually no functional or electrical information entered with the geometric data.

Symbolic layout systems also generate the mask layout but superfluous detail is hidden from the designer by the use of symbolic representations for components [BLACK 81]. "Sticks" is a common symbolic approach, and one which was used in the design of the colour palette circuit. The economy of displayed detail permits the designer to see more of the circuit at one time and therefore yields a better overall perspective of the circuit. Structural and behavioral information can be made inherent to the symbols which therefore simplifies the evaluation and validation tasks. Most symbolic layout systems operate on either a fixed or relative grid. Fixed grid systems permit placement of symbols only at specific grid points which are chosen to eliminate any possible design rule violations. Relative grid systems permit a more flexible placement of symbols and a compaction program is then run to minimize the area without violating any design rules [NEWTON 81B].

Standard cell layout systems provide the designer with the ability to place and move around previously designed circuits which are treated as black boxes. A very common approach is to design cells using a geometric or symbolic layout system. After checking for design violations the cell is fabricated and charac-
Algorithmic layout programs such as LAP, PLAP, and AHF require the designer to express the layout with a programming language \cite{LANG 79, CHENG 82, HEUFT 83}. Examples of commands include WIRE, BOX, and PLA. An algorithmic language was used to design the colour palette and more details on its syntax and examples will be presented later. The main advantage of algorithmic layout is that all of the data and control structures inherent to the language are available for use by the designer \cite{HON 80}. This makes for elegant implementations of regular structures such as PLA's and ROM's since arbitrarily large arrays can be created by changing the parameters in a loop that steps and repeats a primitive cell. Another advantage is the low cost of algorithmic layout systems. The system can be embedded in any available language. A simple text editor can be used for input and a standard plotter can be used for output. The need for expensive interactive graphics hardware is thus eliminated. The main disadvantage is that it is sometimes difficult to see the correlation between the output plot and the input text. Liberal usage of comments and parameterized nodes rather than absolute coordinates can alleviate this problem.

Placement and routing algorithms are extremely valuable layout aids and are actively researched \cite{PREAS 79, SOUKUP 81}. The placement problem involves finding optimal locations for circuit building blocks subject to some spacing and interconnect
constraints. A typical goal is to minimize the total routing length of the interconnections. Placement is a two step process. The first step is to find an initial "good" placement and the second step is to iteratively improve on it by making local modifications [GOTO 81].

When placement is complete it is necessary to route the interconnections between circuit blocks. It is usually assumed that there are only two layers available for interconnects, and that vertical and horizontal segments are confined to separate layers [YOSHIMURA 82, BURSTEIN 83]. The common goal is to achieve all of the interconnections with a minimum of horizontal tracks and a fixed number of vertical tracks. Placement and routing are very computation intensive and hence heuristics are often employed. Specialized parallel processing hardware has been suggested to speed up the routing solution [HONG 83].

The silicon compiler is a new concept in CAD that promises to be the ultimate design tool if it is successful. The ideal silicon compiler would span all levels of representation by accepting as input a functional description of the circuit and then automatically generate the mask data as output. There is considerable controversy over the feasibility of silicon compilers [WERNER 82A]. Proponents point to the analogies between geometric layout and assembly language and at the success of high level software compilers. Opponents feel that software is one dimensional and therefore trivial compared to the multi-dimensional complexity of integrated circuits. They argue that a
useful silicon compiler would first have to solve the artificial intelligence problem. Several silicon compilers have been implemented, however, they are limited to specific architectures and usually produce less than optimal designs [TRIMBERGER 81, THOMAS 81, SHIVA 83]. The automated PLA generator discussed earlier can be viewed as a successful simplified silicon compiler.

The high cost and sometimes lengthy turn-around time of fabrication requires the designer to employ every possible means of guaranteeing that his design is correct. Evaluation tools such as simulators, design rule checkers, and electric rule checkers are therefore essential.

The three main classes of simulators are characterized by the domain that they operate in: functional, logical, or electrical. In general the higher-level simulators run faster and consume less memory but perform a less detailed analysis. Two comprehensive reviews of simulation techniques are available [RUEHLI 83, HACHTEL 81].

Circuit simulators operate at the electrical level of representation and solve the set of coupled nonlinear ordinary differential equations that describe the circuit. SPICE is the most well known circuit simulator [VLADIMIRESCU 80]. It was developed at the University of California at Berkeley and has been distributed to over 2000 installations. Circuit simulators tend to be slow because of the heavy computation required to solve for the exact state of each node for each time increment. SPICE, for
example, running on an IBM 370/168 requires 4 ms/device/clock/point. Circuit simulations are therefore rarely run on circuits containing more than a few hundred nodes. The accuracy of a simulator is strongly dependent on the accuracy of the parameters used in the device models. These parameters are in turn dependent on the fabrication facility and may vary from one batch to the next. One must therefore be conservative when interpreting simulation results unless someone from the fabrication facility is supporting the device models. Although SPICE is probably the most refined CAD program in the public domain it is interesting to note that modifications are usually required before it can be used in an industrial setting [DWIVEDI 82, HEWLETT-PACKARD 80].

Logic simulators simplify the analysis by permitting only a fixed number of node states such as logic-one, logic-zero, high impedance, and undefined. The speedup over circuit simulators can be three orders of magnitude [NEWTON 79]. Early logic simulators were based on the unidirectional boolean logic gate model. MOS devices however are bidirectional and the newer logic simulators take this fact into account [BRYANT 81].

Functional simulators abstract the circuit into modules whose functions are described by a hardware description language or behavioral language [HEPLER 79, RAETH 81]. Functional simulation is of course most useful in the early stages of design when global communication and algorithms used in the circuit must be verified.
The complexity of VLSI has motivated much work on improving the performance of simulators. Lookup tables can be exploited to reduce the repetitive solution of device models. Specialized array processing hardware can be employed. Simulation of temporarily inactive areas of a circuit can be avoided. A very promising technique is the mixed mode simulator [NEWTON 78]. The idea is to combine all three types of simulators into one package: circuit analysis for areas that have critical timing or strong feedback loops, logic analysis for areas with less critical timing, and functional analysis for areas of the circuit that are well understood.

Design rule checkers (DRC) accept as input the geometric description of the masks and output a list of design rule violations. The designer must still decide what caused the error. For example, two wires with insufficient clearance may or may not have been intended to connect. A typical DRC algorithm is to bloat a mask by an amount equal to the spacing required between it and another layer. Then by performing boolean AND operations between polygons on the two mask layers one can easily detect encroachment violations [TUCKER 81]. Several techniques are used to improve the performance of DRC's. Polygons can be sorted and windowed to avoid comparison of polygons that are separated by too great a distance. Hierarchy in the circuit can be exploited to avoid repetitive checking of the same cell [WHITNEY 81].
Electric rule checkers (ERC) look for a variety of common electrical violations in the masks such as floating gates, unterminated wires, shorts between power and ground, and incorrect pull-up/pull-down ratios. If additional information is available such as which buses are clocks and which are data then the ERC can also check for type mismatches [CORBIN 81].

Validation tools such as circuit extractors provide the greatest level of confidence that a design is correct. A circuit extractor accepts as input the geometric mask data and outputs an electrical or logical description of the circuit. This description can then be fed into the same simulator used during the evaluation stage of the design and the results compared. The design loop is thus closed which avoids any possibility that the evaluation was biased by the designer's expectations [TUCKER 81].

Testability analysis is another important facet of validation. In the past it was possible to probe a circuit and deduce the faults from observation of its internal operation. Today's complexity and small line widths make this almost impossible. A designer must be acutely aware of how he plans to diagnose faults that are sure to appear after the first iteration. A review of testability is available [WILLIAMS 83].

Design for testability techniques are divided into two categories: ad hoc and structured. The ad hoc techniques are specific to the particular chip and do not attempt to solve the general testing problem. Examples include partitioning the chip
into modules that can be individually tested, providing test points for probes (see for example [CANEPA 83]), and making use of internal buses for stimulating the circuit modules. Another popular ad hoc method is signature analysis which applies a signal to the circuit and then checks if the compressed response is correct.

Structured techniques attempt to solve the more general testability problem. The usual approach is to add extra circuitry that can halt the circuit and then shift in a known vector or shift out the current state for analysis [TSUI 82]. By halting the circuit the problem is reduced to debugging combinatorial logic rather than the intractability of debugging a dynamic finite state machine. Another approach assumes that the most common fault is a single "stuck-at" device and then adds circuitry to detect all such faults [MERCER 80]. Finally some of the new complex IC's are incorporating on board self test circuitry and software [CANEPA 83].

The Mead and Conway Revolution

The connotations of the word revolution are distasteful, however, when one considers the impact that Carver Mead and Lynn Conway have had on IC design it is an apt description. Integrated circuit design was for many years a black magic art because most advances occurred in industry and therefore were proprietary and unpublished. In the fall of 1978 Lynn Conway of Xerox PARC
offered a VLSI system design course at M.I.T. [CONWAY 79]. The course resulted in a custom NMOS chip containing a variety of student designs of which at least 3 functioned correctly. The course textbook was a pre-published version of Mead and Conway's "Introduction to VLSI Systems" [MEAD 80]. The publication of this book in 1979 started a revolution. A more complete history is available [CONWAY 81].

The results have been amazing. There are now over 60 universities in the U.S. and Europe that offer VLSI design courses based on the Mead and Conway text [FAIRBAIRN 82]. Many Canadian universities, including U.B.C., are offering design courses which are being coordinated by the VLSI Implementation Centre (VLSIIC) at Queen's University [GALE 83]. A wide selection of short courses on VLSI design are now available to industry. Several new journals including IEEE Computer-Aided Design, VLSI Design, and Integration have sprung up. There are at least 13 recent start-up companies offering custom IC services [SZIROM 83].

What then did Mead and Conway actually do? Mead and Conway removed the mystery and intimidation from IC design by simplifying and formalizing the design process. At the same time they pointed out many potentially fruitful topics of research. More specifically they introduced the silicon foundry, the multi-project chip (MPC), Caltech Intermediate Form (CIF), simplified design rules, a simplified MOS transistor model, and a structured design methodology.
A silicon foundry is a facility that fabricates an integrated circuit from a design supplied by an independent party [JANSEN 81]. The silicon foundry provides those users that cannot afford the high cost of fabrication equipment (approx. 15 million dollars) with custom circuit capability. An ideal silicon foundry should have a clean and standardized interface so that the designer does not have to be concerned with mask making or fabrication details such as fiducial marks, mask polarity, and bloats or shrinks. The foundry concept is entirely analogous to the commonplace processing of photographic film. There are at least 38 silicon foundries now operating [WERNER 82B]. Minimum lots of 10-20 wafers cost about 10,000 dollars, however, most foundries require a minimum yearly expenditure of 100,000 dollars. This has led to the formation of silicon brokers that specialize in interfacing between silicon foundries and designers [COHEN 82]. By representing several designers the minimum expenditure can be reduced. The chip designed as part of this thesis was fabricated by GTE in Phoenix and should cost approximately 42 dollars each in quantities of 10,000.

The multi-project chip (MPC) is a method of amortizing mask making and foundry costs over several designs [HON 80]. If, for example, 10 projects can be placed on a 5000 micron square chip then a minimum 10,000 dollar run costs each designer only 1000 dollars. MPC's do require someone to coordinate and pack the designs prior to fabrication, and to look after packaging and distribution when the wafers return (hence the VLSIIC at Queen's University).
Caltech Intermediate Form (CIF) is a low-level graphics language for specifying the geometry of integrated circuits [HON 80]. CIF was designed to be a standard interface format between the designer and foundry as shown in Figure 7. Most schools (including U.B.C.) and foundries are indeed using CIF. Typical CIF primitives include define mask layer, draw wire, and draw polygon. CIF also supports hierarchy by permitting one to define cells which can be later called with appropriate transformations applied to it. CIF is in plain text format, as shown in Figure 6, and therefore can be read and understood by both people and computers. Finally, CIF is independent of fabrication processes.

```
DS2; 9 PadBlank; (define symbol #2 with name PadBlank)
(4 items); (bounding box 0,0 to 26500, -26500);
L NM; B L 26500 W 2000 C 13250, -1000; (Vdd line);
L NM; B L 20500 W 2000 C 13250, -25500; (ground line);
L NM; B L 13500 W 13500 C 13250, -13250; (metal pad);
L NG; B L 11500 W 11500 C 13250, -13250; (overglass window)
DF; (end definition)

DS 11; 9 PadSample; (example input pad)
C 6 R 0,1 T 0, -106000;
C 7 R 0,1 T 0, -79500;
C 5 R 0,1 T 0, -53000;
C 2 R 0,1 T 0, -26500;
L NM; B L 77750 W 2000 C 38875, -105000;
   B L 10250 W 2000 C 24875, -60500;
   B L 2000 W 49500 C 25500, -24750;
   B L 28750 W 2000 C 38875, -1000;
   B L 14250 W 2000 C 31625, -102000;
   B L 2000 W 97000 C 31000, -51500;
C 9 R 0, -1 T 77750, -53000;
C 8 R 0, -1 T 77750, -26500;
   B L 2000 W 11250 C 37750, -97375;
   B L 6500 W 2000 C 40000, -92750;
C 4 R 0, -1 T 77750, -79500;
C 10 R 0, -1 T 77750, -79500;
   B L 2000 W 103000 C 52250, -51500;
DF;
```

Figure 6. Example of Caltech Intermediate Form [HON 80]
Design System
Standard Cells
Place & Route
CIF Output

Design System
Interactive Graphical Layout
CIF Output

Design System
Artwork Language e.g. LAP
CIF Output

Design System
Text Editor

Designer

Design Data Base

CIF

Check Plot

Fabrication
Main PG
Reticles
Exposure
Wafers

Fabrication
E-Beam Exposure
Wafers

Fabricator

Figure 7. Caltech Intermediate Form Relationships [HON 80]
Simplified design rules are another cornerstone of the Mead and Conway design philosophy [LYON 81]. Design rules are specified by the fabrication facility and constrain the design to certain minimum (or maximum) widths, separations, extensions, or overlaps of geometric patterns on and between masks. Design rules normally tend to be quite complex, particularly if the designer wishes to squeeze every possible micron out of the layout. Under such tight tolerances there are many special cases, each requiring a design rule. The first problem with complex design rules is that they are hard to learn and it is difficult to check for violations. The second problem is that special case rules tend to change as the technology improves, and therefore a complete redesign may be required to take advantage of the improvements. A third problem is that special case rules are very dependent on the individual characteristics of each fabrication facility. Mead and Conway believe that the design rules should be relaxed and be based on a generic fabrication process such as depletion load silicon gate NMOS. In addition, all rules should be in terms of a dimensionless quantity called lambda. Mead and Conway noted that a fundamental parameter in any fabrication process is the minimum feature size and they made lambda equal to one half of this quantity. Lambda based design rules make logical sense. For example, if one wishes to guarantee that two wires do not cross then a 2 lambda spacing must be provided to ensure separation in the event that each wire moves the maximum 1 lambda error towards the other. Since the simplified rules are based on a generic fabrication processes and are relaxed to avoid special cases they should be reasonably transportable between foundries.
Finally, as geometries decrease one should be able to simply resubmit the original design with a new lambda specified. This of course has limitations because some features such as bonding pads do not scale. Experience shows that simplified rules do indeed scale to a degree. The colour palette was originally submitted for fabrication with lambda = 2.5 microns. It can now be resubmitted unchanged with lambda = 2.0 microns, however, lambda = 1.65 microns will require a redesign of as yet unknown magnitude. Mead and Conway's simplified rules were originally specified for NMOS but similar rules have recently been specified for CMOS [Snyder 82, Griswold 82] and for bipolar technology [Elmasry 83].

It is interesting to note that the majority of work by universities on VLSI design has been performed by computer scientists and not electrical engineers. One of the reasons for this is that Mead and Conway presented a simplified model of the MOS transistor which gave people lacking a device physics background enough information to begin designing integrated circuits. The MOS transistor is described as a voltage-controlled current switch that is formed when a polysilicon wire crosses the diffusion layer. Current through the device equals the charge induced in the channel by the applied gate voltage divided by the transit time or average time required for an electron to travel from the source to the drain. A simple expression for the transit time is derived from the channel length and the electron velocity. The induced charge is found by treating the gate as a simple capacitor and using the definition of capacitance. A second MOS device model is also presented in the Mead and Conway book by Carlo
Sequin. This model appeals more to intuition by describing charge as a fluid and potential as depressions or wells into which the fluid can accumulate.

Mead and Conway refer to the transit time as "the fundamental time unit of the entire integrated system". All delays throughout the integrated circuit can be expressed as multiples of the transit time. For example, the time required for a transistor to charge the gate of another transistor twice its size is two times the transit time. Using this simple idea one can make crude estimates of the circuit performance without elaborate simulation tools.

Mead and Conway have been widely touted for introducing a structured design methodology which is essentially the exploitation of hierarchy and regularity to attack the problems associated with complex designs [TRIMBERGER 81]. Hierarchy and regularity have already been discussed earlier in this chapter. Mead and Conway feel that digital processing systems should be hierarchically decomposed into combinations of register-to-register transfer paths, controlled by finite state machines. They also promote the use of dynamic circuitry with a two phase nonoverlapping clock scheme. The difficulty in designing a complex dynamic part that actually works is, however, not addressed.
CHAPTER 3: SELECTION AND JUSTIFICATION OF A CUSTOM CHIP PROJECT

Background Information

In keeping with our desire to design a novel yet useful integrated circuit we solicited advice and impetus from local electronics firms. Several interesting and promising projects were suggested to us. These included a low power programmable read-only memory for use in a rail car location system, a synthetic aperture radar processing element, and a colour palette circuit.

Our decision to proceed with the colour palette project suggested by Microtel Pacific Research Ltd. (MPR) was influenced by several factors. First, MPR could offer us access to a silicon foundry. Second, to the best of our knowledge the colour palette function has not yet been integrated onto a single chip, and therefore it is a novel project. We do know of a hybrid implementation but this cannot be considered a truly integrated colour palette [INTECH 82]. Third, the colour palette combines digital and analog circuitry which makes for a more interesting design than a purely digital circuit, and adds to its novelty. Finally, while targeted for the MPR Telidon system, the colour palette is a general purpose circuit that has important applications in other graphics systems.
The Colour Palette Function

Given the three primary colours on his palette an artist can mix an infinite number of hues, however, his palette has room for only a finite number of colours at any one time. An electronic colour palette serves a similar function except that a colour monitor replaces the canvas.

The main components of a colour palette are a dual port register array and three digital-to-analog converters (DAC). The analog output of each DAC is connected to one of the monitor colour guns. The intensity of each colour and their resultant combined hue is therefore determined by the bit pattern that is input to the DAC's. Data is written into the registers through one of the ports by a graphics system processor. Pixel data is presented by the CRT controller to the other port which selects one of the registers and causes it to drive its data into the DAC's. Each register in the array is divided into three equal-width bit fields, one for each DAC. If each register field is n bits wide, and if the DAC's have n bits of resolution, then each register can specify one of \(2^{3n}\) colours. Typically n equals 4 in which case each register can specify one of 4096 unique colours. The number of colours that are available at any one time equals \(m\), the number of registers in the array. For obvious reasons \(m\) is chosen to be a power of 2 and is normally equal to 16. In summary then, the colour palette permits \(\log m\) bits to select from \(m\) colours, each of which can be chosen from a pool of \(2^{3n}\) different hues.
Colour Palette Applications

We have explained the colour palette function, but what is its utility? Assume that a colour graphics system has a high quality monitor that is capable of displaying 4096 different hues in each of its pixels. If one wanted to be able to display all colours at once, each pixel would require 12 attribute bits. In a high resolution system this can be an excessive amount of memory. The colour palette provides a compromise solution in that all colours are still available but not at the same time, and the number of bits for each pixel is reduced to log m.

Another important application of the colour palette is in colour graphics animation. Given a complex image one can dramatically alter its appearance by simply writing different words into the colour palette registers. Rapid and striking animation sequences are possible because updating a register is a quick operation. Without a colour palette the attributes of each pixel in the video bit map would have to be rewritten.

The Colour Palette Market Potential

A new and important standard for the exchange of text and graphical information has recently emerged. The North American Presentation-Level-Protocol Syntax (NAPLPS) evolved from the Canadian Telidon system and has been standardized by the American National Standards Institute (ANSI) and the Canadian Standards
Association (CSA) [CSA 82, CBEMA 83]. Many people feel that NAPLPS will play a very important role in the near future by encouraging the production of information-based products [LAX 83]. This in turn will make Telidon-like systems more palatable to the general public.

One of the many rich and powerful features of NAPLPS is its treatment of colour. NAPLPS supports three different colour modes. Mode 0 is the most primitive in that a colour is selected by specifying the intensities of its three primary colours. Modes 1 and 2 however make use of a colour palette. NAPLPS supports advanced animation by providing a command called BLINK that can be programmed to periodically change the colour palette register contents.

Jim Fleming, one of the authors of NAPLPS, states in a review of NAPLPS that a major drawback of modes 1 and 2 is the specialized colour palette hardware that is required [FLEMING 83]. He further goes on to say, "This drawback was known at the time that NAPLPS was designed, but it was determined that because of the incredible special effects that can be achieved using these modes they would be included". This strongly implies that an inexpensive single chip colour palette would have tremendous market potential, particularly when Telidon like systems, fueled by NAPLPS, appear in the consumer market.
The Current MPR Colour Palette Implementation

Microtel Pacific Research Ltd. has a discrete component colour palette in its current Telidon system. Schematics for the circuit and a block diagram showing how it fits into the overall system can be found in Appendix A. This colour palette contains at least 14 integrated circuits, occupies approximately 9 cm by 10 cm of printed circuit board space, and costs about 37 dollars for parts. It is difficult to obtain an estimate of the labor costs for assembly because the colour palette is part of a larger system.

A considerable amount of the circuitry is associated with multiplexing the address buses of the microprocessor, which writes the desired colours into the palette registers, and the CRT controller, which cycles through the video bit map and presents the contents of each pixel to the palette. Multiplexing is required because the register array is formed from standard single port memory. We will later see that our implementation uses dual port registers and therefore eliminates the need for multiplexing circuitry.

In addition to increasing the circuit complexity, multiplexing between the processor and the CRT controller forces all palette register updates to occur during the vertical blanking period of the monitor. Our implementation permits the processor to update the palette registers at any time and asynchronously to the CRT controller operation.
The advantages of a single chip colour palette circuit are numerous. It would significantly reduce the MPR Telidon part count, printed circuit board area, and assembly cost. In large enough volumes it would probably also reduce the parts cost. A single chip colour palette would help protect MPR's design from being copied by a competitor. It could also improve the system performance as demonstrated by the multiplexing issue discussed earlier. Finally, a single chip colour palette has strong market potential as a product by itself.

Colour Palette Specifications

As shown in Figure 8 the desired colour palette contains 16 registers, each 12 bits wide. Separate address buses and control circuitry are provided for the CRT controller and the processor. The important specifications will be summarized here. More detailed timing specifications and signal definitions are available in Appendix B [SCHMIING 82].

On the video side the important number is the pixel clock cycle time which is 180 nsec. This means that the palette must be capable of accepting pixel data from the CRT controller and generating the appropriate analog voltages at a rate of 5.56 MHz. Also important is the DAC settling time, which is defined as the time required for the analog output to reach within 1/2 LSB from its final value, and equals 50 nsec.
Figure 8. Colour Palette Block Diagram
On the processor side the important number is the read and write cycle time which is 500 nsec. This means that the palette registers must be capable of being read or written at a rate of 2 MHz. Read data must be valid no later than 150 nsec after the rising processor clock edge. Write data is valid at worst case for 20 nsec prior to the falling edge of the processor clock. Critical timing clearly occurs on the video side because of the higher clock rate and this fact is taken into account in the design.
CHAPTER 4: THE ISO-CMOS TECHNOLOGY

An Overview of CMOS

Complementary metal-oxide-silicon (CMOS) technology is distinguished from other MOS technologies by the presence of both p and n channel devices. For many years it was thought that CMOS was inherently slow and wasteful of chip area and therefore did not justify its extra processing complexity except for the few circuits that required its special properties [POSA 81]. This viewpoint became a self-fulfilling prophecy since the majority of research effort was spent on improving n channel MOS (nMOS) technologies. Two main factors led to a renewed interest in CMOS. First, as nMOS densities increased excessive power dissipation became a problem. Second, the processing complexity of the high performance nMOS technologies approached or exceeded that of CMOS. Subsequent research has made CMOS as fast as nMOS and almost as dense.

CMOS has many advantages over nMOS. The presence of a complementary gate gives the designer an extra degree of freedom. The traditional CMOS design approach uses one p channel device for every n channel device as shown in Figure 9. This type of CMOS circuit consumes very little static power because when one device is on its complement is off and there is thus never a DC path between Vdd and Vss.

43
CMOS circuits do however dissipate power during logic transitions because energy is required to charge and discharge capacitances, and because there is a brief time during the switching cycle when current can flow from Vdd to Vss as shown in Figure 10. High speed CMOS circuits dissipate about as much power as an equivalent nMOS circuit. In general CMOS results in a power savings because it is rare for a complex circuit to be completely active at all times. This is especially true for memory circuits.
One of the disadvantages of true complementary circuits is the decrease in effective chip density caused by the extra p channel devices. To alleviate this problem many designers use dynamic circuitry in which p channel devices are used sparingly for precharging buses (see Figure 11) [POSA 81]. To avoid glitches in dynamic circuits one must wait until all inputs have stabilized before activating the gate. Four phase clocks are often used to deal with the glitch problem, however, a new approach called domino logic has been suggested [KRAMBECK 82].
Another important advantage of CMOS is that the rise and fall times of a gate do not depend on the ratio of its pull-up to pull-down transistor sizes. With ratioed logic, such as nMOS, one desires a large pull-up resistance for rapid pull-down, and a small pull-up resistance for rapid pull-up. The designer must therefore trade off rise time for fall time with the result that neither is optimal. CMOS does not suffer from this trade off since a switch rather than a resistor is used to pull-up the output.
CMOS is more immune to noise than nMOS because signals swing fully from Vdd to Vss. A low level nMOS output, on the other hand, is determined by the resistance ratio of its pull-up to pull-down transistors and typically only reaches 0.3 volts.

Although an nMOS transistor is a bidirectional switch, it is more efficient at pulling a high node to ground than a low node to Vdd. An nMOS device will turn off when its gate voltage is no longer greater than one threshold above its source voltage. Thus the source can only be pulled to within one threshold of Vdd. For a similar reason pMOS transistors are better suited to pulling low nodes high. By connecting an nMOS transistor in parallel with a pMOS transistor and driving their gates with reverse polarities one obtains a very efficient switch that is usually referred to as a transmission gate. CMOS transmission gates reduce many of the design trade offs inherent to nMOS only switches.

CMOS is not without its disadvantages. The main disadvantage is the decrease in functional density. As stated earlier the density has been improved through the use of dynamic circuitry and fewer p channel devices. Early methods of isolating the n and p channel devices, such as n+ and p+ guard rings, also wasted chip area. New CMOS technologies reduce this wasted area through the use of oxide isolation [MITEL 81].

CMOS circuits can suffer from latch-up which has been defined as a state of high excess current accompanied by a collapsing or low-voltage condition [ESTREICH 82]. A CMOS circuit
has numerous SCR like p-n-p-n paths that can conduct given the following conditions. The emitter-base junctions must become forward biased, the product of the n-p-n and p-n-p transistor gains must exceed unity, and the power supply must be capable of sourcing enough current to maintain latch up. Latch-up is usually caused by a transient such as occurs during power up. Several processing techniques are available for reducing the latch-up tendency. These include gold doping to reduce minority carrier lifetimes, dielectric isolation such as silicon on sapphire (SOS), and buried n+ and p+ layers which help reduce the parasitic transistor gain. Several techniques are also available to the IC designer for reducing latch-up and these will be discussed in a later section.

True complementary CMOS circuits have higher input capacitances than their nMOS counterparts because both a p and an n channel gate must be driven. Since the p type devices are often made twice as large as the n type devices the input capacitance can be 3 times greater. Given this fact one would expect CMOS to be slower than nMOS. This is not the case because the extra capacitance is offset by the ability of CMOS to pull-up and pull-down harder than nMOS [KRAMBECK 82].

There are a wide variety of CMOS technologies which is one reason that simplified design rules were first prepared for nMOS. Their most important distinguishing features are the starting substrate type, the gate material, the number and type of interconnection layers, and the n and p channel isolation method.
If one starts with an n type substrate then a p well must be created to form the n channel devices, and vice versa for a p type substrate. There is considerable debate over the relative merits of p and n well CMOS [GULETT 81]. Proponents of n well CMOS point out that a greater percentage of n channel transistors are used because of their higher performance, and therefore the technology should be tailored to their optimization, as is n well CMOS [CHWANG 81]. Another potential advantage of n well CMOS is that the wafers can be processed on an nMOS fabrication line since they both start with the same substrate type. Twin tub CMOS side steps the issue by creating both n and p wells in lightly doped epi on an n+ substrate [PARRILLO 80]. Twin tub CMOS is claimed to optimize both the n and p channel devices and the n+ substrate helps reduce latch-up. A final approach called silicon-on-sapphire (SOS) grows a thin layer of silicon on an insulating substrate [KINOSHITA 81]. This results in low parasitic capacitances and eliminates latch-up. The main disadvantages of CMOS SOS are its high cost and low yield.

Early CMOS technologies used metal gates while the more advanced technologies use polysilicon gates. A polysilicon gate process results in self aligned sources and drains which reduces parasitic capacitances [GLASER 77].

Some CMOS technologies use a single metal and two polysilicon layers for interconnect [LONDON 83]. Capacitance between the two polysilicon layers is high which can be an advantage if one requires capacitors in a design. The second layer of polysilicon
is sometimes made highly resistive for applications such as memory arrays that require high density pull-ups. Other CMOS technologies use a single polysilicon and two metal layers for interconnect. Thicker oxide is required to isolate the metal layers and therefore they do not form high value capacitors. Double metal processes may be better suited for high performance applications because metal is a better conductor than polysilicon.

The final major distinguishing feature between CMOS technologies is the method used to isolate the n and p channel devices. As stated earlier, the newer technologies use oxide isolation [MITEL 81] while others use n+ and p+ guard rings [SMITH 83].

ISO-CMOS Fabrication and Parametric Data

The technology we had the opportunity to work with is GTE's ISO-CMOS. ISO-CMOS is a 5 micron, p well, polysilicon gate process, with one metal and two polysilicon levels of interconnect, and oxide isolation. Appendix C contains a series of diagrams showing the fabrication sequence [SIMMONS 82]. A condensed version is shown in Figure 12. ISO-CMOS is an 8 mask process with additional masks required if the second polysilicon layer is used. Rather than explain each diagram in detail I have decided to let them speak for themselves. An introduction to fabrication procedures is available [GLASER 77].
The diagrams (right) show the process stages for Mitel's ISO-CMOS process. This process uses a total of 8 masks. P-type transistors are formed directly in the N-type substrate which will be biased to the most positive voltage used (VDD). N-type transistors are formed in P-type wells formed by inverting the N-type substrate with a high concentration of P-type dopant. The P-well will be biased to the most negative supply voltage (Vss). The process has self-aligning gates, as the polysilicon gate can be used to mask the source/drain diffusion areas. Interconnect between the transistors is primarily with metal. However, the polysilicon is used as a second interconnection medium, greatly facilitating interconnect design. Isolation to prevent the formation of spurious transistors in the field-oxide region is achieved by using a very thick, recessed oxide.

The 5-micron design rules coupled with oxide-isolation and self-aligning gates gives the process major speed and packing-density advantages over metal-gate CMOS. While maintaining similar power dissipation characteristics to metal-gate CMOS, packing densities approach those of NMOS in similar geometries and speeds are comparable with low-power-Schottky technology.

Figure 12. ISO-CMOS Fabrication Sequence [MITEL 81]
Detailed ISO-CMOS process parameters and device characteristics were not available to us. The data we have is presented in Tables I and II.

Note: $\lambda = 2.5$ microns

<table>
<thead>
<tr>
<th>Component</th>
<th>Capacitance pF/$\lambda^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly wire</td>
<td>$1.5 \times 10^{-4}$</td>
</tr>
<tr>
<td>n junction</td>
<td>$2.1 \times 10^{-3}$</td>
</tr>
<tr>
<td>p junction</td>
<td>$1.3 \times 10^{-3}$</td>
</tr>
<tr>
<td>metal</td>
<td>$1.2 \times 10^{-4}$</td>
</tr>
<tr>
<td>poly gate</td>
<td>$2.6 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

Table I. ISO-CMOS Capacitance Data

<table>
<thead>
<tr>
<th>Component</th>
<th>Resistivity ohms/square</th>
</tr>
</thead>
<tbody>
<tr>
<td>n poly</td>
<td>25</td>
</tr>
<tr>
<td>p poly</td>
<td>50</td>
</tr>
<tr>
<td>n active</td>
<td>12</td>
</tr>
<tr>
<td>p active</td>
<td>100</td>
</tr>
<tr>
<td>metal</td>
<td>0.023</td>
</tr>
</tbody>
</table>

Table II. ISO-CMOS Resistivity Data

ISO-CMOS Design Constraints

Design constraints include the design rules, techniques for avoiding latch-up, fan-out limitations, and mobility compensation. Table III lists the design rules and Figure 13 demonstrates them graphically with the layout of an inverter that has two enable signals.
Notes: - all dimensions are in lambda ( = 2.5 microns)
- x means no rule

Minimum Spacing Rules

<table>
<thead>
<tr>
<th>Active</th>
<th>Polyl</th>
<th>Poly2</th>
<th>Metal</th>
<th>Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>2</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Polyl</td>
<td>1</td>
<td>2</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Poly2</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>Metal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>Cut</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Minimum Overlap Rules

<table>
<thead>
<tr>
<th>Active</th>
<th>Polyl</th>
<th>Poly2</th>
<th>Metal</th>
<th>Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>Polyl</td>
<td>1.6</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>Poly2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>Metal</td>
<td>x</td>
<td>0</td>
<td>-.6</td>
<td>x</td>
</tr>
</tbody>
</table>

Minimum Width Rules

<table>
<thead>
<tr>
<th>Active</th>
<th>Polyl</th>
<th>Poly2</th>
<th>Metal</th>
<th>Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Minimum Overlap of Active by:

<table>
<thead>
<tr>
<th></th>
<th>N+</th>
<th>P+</th>
<th>Pwell</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.4</td>
<td>1.4</td>
<td>2</td>
</tr>
</tbody>
</table>

Miscellaneous Minimum Spacing Rules

N+ Active to P+ Active = 7  (different substrates)
N+ Active to P+ Active = 3  (same potential and substrate)
Pwell to P+       = 3  (different substrates)
Pwell to Pwell = 8  (different potentials)

Table III. ISO-CMOS Simplified Design Rules [Snyder 82]
Figure 13. Layout of an Inverter with 2 Enable Signals [SNYDER 82]
As Table III shows most design rules are integers and are therefore easy to learn and remember. The only slight complication are the rules concerning the separation of p and n type transistors. Fortunately there are only a few of these more difficult rules.

There are three design guidelines for avoiding latch-up in CMOS circuits [ESTREICH 82]. First, the lateral current which flows along potential latch-up paths must be minimized. This can be achieved by wiring all of the p wells to Vss via a p+ contact and a metal strap. Any injected charge will be shunted to ground through the low resistance path. The n substrate should similarly be tied to Vdd with n+ contacts. Contacts do occupy valuable space and therefore the decision on how many contacts to use is often nebulous. One suggested rule of thumb states: if there is space, put in a contact; if there isn't space, consider making some [GRISWOLD 82].

A second design guideline for avoiding latch-up is to reduce the parasitic resistances of the p well and substrate regions. This reduces the voltage drop across the emitter-base junctions of the parasitic devices for a given level of injected current and thus makes it harder to turn the devices on. One technique for reducing the parasitic resistances is to place as much p+ and n+ diffusion in the p well and substrate regions respectively as is possible [LIPMAN 82].
A third design guideline for avoiding latch-up suggests prudent placement of potential parasitic emitters. Spacing between the p well p+ and nMOS source contacts should be minimized to help collect minority carriers near the parasitic emitter-base junctions [LIPMAN 82, SMITH 83]. Similarly, the spacing between the substrate n+ and pMOS drain contacts should be minimized.

We tried to follow the latch-up prevention guidelines as closely as possible throughout the colour palette design. The memory cell, for example, contains four n+ contacts and five p+ contacts.

Fan-out limitations are another important design constraint. Consider a large capacitive bus that must be driven with the output signal of a minimum sized gate. The gate is only capable of sourcing and sinking a relatively small amount of current. If we connect it directly to the bus the time to charge and discharge the bus will be excessive and may degrade the overall system performance. A better approach is to buffer the signal with successively larger devices until the final device is large enough to optimally drive the bus. Each buffer stage however introduces an additional gate delay. It is therefore intuitively clear that there should be some optimal fan-out factor.

It is easy to show using Mead and Conway's simplified timing model that the optimal fan-out factor is $e (= 2.718)$ [MEAD 80]. In other words, each buffer should drive a capacitive load that is 2.718 times as large as its own gate capacitance. One problem
with using the optimal fan-out factor is that the final buffer is often very large which reduces the circuit density. Furthermore, not all paths in a circuit are critical and therefore using the optimal fan-out ratio may not improve the overall system performance unless something is done to reduce the critical path delays.

The most efficient design approach in terms of circuit area and performance is to locate the critical paths and then reduce their delays with more favorable fan-outs and by trying different circuit configurations. Simulation can play an important role at this stage. Due to time constraints we opted for a very conservative design approach that sacrificed area for performance and used a near optimal fan-out ratio of 3 or 4 throughout the colour palette. Simulation was used to check the delay on several of the colour palette critical paths.

Mobility compensation is the final design constraint to be discussed. N channel devices have a carrier mobility that is approximately twice that of p channel devices [SZE 69]. This means that for a given size, n channel devices can charge or discharge a node approximately twice as fast as p channel devices. This is one reason for the prominence of nMOS over pMOS technologies.

It is desirable that the rise and fall times of gates be approximately equal in digital circuits. This can be achieved in CMOS by making the p channel devices twice as wide as the n
channel devices. There are other advantages to maintaining a pMOS to nMOS width ratio of 2. Assuming a fan-out of 3 and worst case process parameter fluctuations it has been shown that a width ratio of 2 optimizes both the propagation delay and the noise margin [KANG 81]. For these reasons, and in keeping with our conservative approach to design, we made every p channel device twice as wide as its n channel counterpart on the colour palette.
The MPR CMOS Design System

The colour palette chip was laid out with the Microtel Pacific Research CMOS Design System (CDS) which is an algorithmic layout package written by Warren Snyder [SNYDER 82]. CDS belongs to the same class of programs as LAP [LANG 79] and PLAP [CHENG 82]. CDS contains approximately 2000 lines of Hewlett Packard BASIC code and runs on the 68000 based HP98x6 desk top computers.

CDS has two main modules: the compiler and the plotter. The compiler inputs a file containing one or more cell layouts written in the CDS language and outputs a file containing the corresponding CIF code. The CDS language provides the standard algorithmic layout commands such as BOX, WIRE, and the ability to call a cell with suitable transformations applied to it. The plotter program uses the CIF code generated by the compiler to plot the desired cells. Any combination of mask layers can be plotted and the output can be directed to either the HP98x6 screen for a quick black and white check plot, or to the HP7820 eight pen plotter for coloured hardcopy. The plotter also has a filter option that plots all of the features of the top hierarchical level and only the bounding boxes of the cells on the next lower level. This permits the designer to see the top level interconnections without cluttering the plot with unnecessary detail, and greatly reduces the plotting time.
CDS has several unique and useful features. Figure 14 shows an example CDS layout program and Appendix D contains the full CDS language specification. The DEFN command permits one to give a meaningful label to a node within a cell. The node coordinates can then be retrieved by a calling cell with the LOCN command. If the lower level cell had any transformations applied to it when it was called they are automatically applied to the node coordinates, as one would expect. Algorithmic layout, by its nature, does not give immediate visual feedback when the layout program is written. The designer must therefore be very careful when computing the coordinates required to interconnect two cells, particularly at higher levels when much information is displayed on the plots and the display resolution is reduced because of the larger cell sizes. The DEFN and LOCN commands remove all uncertainty from the interconnection process. Drawing a wire to the coordinates returned by a LOCN command guarantees that the connection has been made, and does not just appear to be connected on the plot. The designer must of course still be on guard for design rule violations. One difficulty that arises when using DEFN and LOCN is that nodes are only available to the next higher hierarchical level. This means that to carry a node through from a leaf cell to the top level one must execute a LOCN-DEFN sequence at each level, which can become tedious in complex hierarchies. We made very liberal use of node definitions in the colour palette layout.
This example demonstrates some of the more powerful features of CDS. The code shown was extracted from the top most palette cell definition.

Place the x-decoder beneath the memory array

Place("bigxdcod",168,-81,Xdecod)

Dimension data bus coordinate arrays

DIM Rdpx(4),Rdpy(4) ! red nibble  
DIM Gdpx(4),Gdpy(4) ! green nibble  
DIM Bdpx(4),Bdpy(4) ! blue nibble

Get x-decoder node coordinates

FOR I=0 TO 3
    Locn(Xdecod,"rdp"&VAL$(I),Rdpx(I+1),Rdpy(I+1))  
    Locn(Xdecod,"gdp"&VAL$(I),Gdpx(I+1),Gdpy(I+1))  
    Locn(Xdecod,"bdp"&VAL$(I),Bdpx(I+1),Bdpy(I+1))
NEXT I

Get miscellaneous node coordinates

Locn(Xdecod,"xsigleft",Xsigx,Xsigy)  
Locn(Xdecod,"btmleft",Btmleftx,Btmlefty)  
Locn(Xdecod,"toprght",Toprghtx,Toprghty)

Check for cell placement error

IF Btmlefty <> Rdpy(1) THEN
    BEEP
    PRINT "node error"
END IF

Connect the Vss bus to the x-decoder

Pitch=168  
Btrightx=Btmleftx+(Pitch*6)

Layer(Metal)  
Wire(12,Btmleftx-2,Btmlefty-10)  
    X(Btrightx+2)

FOR I=0 TO 6
    Wire(4,Btmleftx+(Pitch*I),Btmlefty)  
        Dy(-10)
NEXT I

etc....

Figure 14. CDS Layout Program Example
Another nice feature of CDS is the BBOX command which returns the bounding box coordinates of a cell. These coordinates can be used to make an intelligent decision on where to place the cell. This is especially true if the origin of the cell is not at a well defined location such as its lower left hand corner.

Since CDS is embedded in BASIC all of BASIC’s features can be used in the layout program. For example, we made extensive use of subscripted arrays and math operations within FOR-NEXT loops. We also made cells self documenting by having BASIC print out important coordinate pairs and other useful information at the end of each compilation. Errors often become very obvious when these print outs are studied and compared.

CDS has several disadvantages that are caused by its hardware. One problem is that compilation and plotting is relatively slow compared, for example, to the PLAP system at UBC. Another problem is the limited floppy disk capacity which was almost exceeded by the colour palette project. Perhaps these drawbacks are not so serious when one considers the independence provided by a desk top computer.

The Design Procedure

Our design procedure consisted of eight distinct steps: system specification, module partitioning, floor plan design, logic design, loading analysis and simulation, stick diagram
layout, mask layout, and data entry. Using the Chapter 2 discussion of IC design as a basis we note that system specification, module partitioning, and floor plan design take place at the functional level of representation. Logic design of course takes place at the logical level of representation. Stick diagram layout, loading analysis, and simulation are at the electrical level of representation. Mask layout and data entry are at the physical level of representation. Virtually all of the synthesis, validation, and evaluation steps were performed manually.

The system specification was developed by G. Schmiing using the MPR discrete component colour palette as a foundation (Appendices A & B) [SCHMIING 82]. Timing characteristics and signal definitions were constrained by the MPR Telidon system.

Partitioning the system into modules was relatively easy because of the clean colour palette architecture. All module boundaries are distinct and interconnections are well defined. A block diagram was shown earlier in Figure 8.

Floor plan design involved estimating the size of each module and then placing them such that the interconnection complexity and chip area were minimized. The floor plan evolved as the modules were laid out because the size estimates often had to be updated.

Logic design involved implementing each module with boolean gates, and was one of the most interesting stages of the design.
Several configurations were tried for each module in an attempt to minimize the number of gates and increase the performance. It was necessary to consider fan-out constraints at this stage.

The objective of the loading analysis was to determine the gate sizes required to achieve optimal fan-outs and still maintain a good circuit density. This tended to be an iterative process because many of the gates were interrelated and therefore a change in the size of one gate often required revisions to many others. All gate sizes were specified as multiples of a minimum sized inverter. More details on the loading analysis will be presented in the next chapter. Several of the critical paths on the chip were simulated with SPICE to determine if the drivers were large enough to meet the timing specifications.

The next stage of the design was to draw stick diagrams from the logic diagrams. Stick diagrams represent mask shapes with line segments and each mask layer is assigned a unique colour [MEAD 80]. Figure 15 shows a stick diagram with its corresponding logic diagram. Stick diagrams are very convenient for experimenting with mask topologies because the designer is not burdened with design rule details. Stick diagrams can clearly indicate whether or not a proposed layout is efficient in terms of area and interconnection complexity. They also quickly point out any impossible routing schemes. Because of their convenience we went through many revisions of stick diagrams before finalizing the design.
Mask layout was a tedious process of drawing the mask geometry on grid paper from the stick diagrams. Revisions were difficult because often the entire layout had to be redrawn to accommodate a small change. Much effort was spent trying to make the cells as small as possible, particularly for those that were to be repetitively used such as in the memory array. The editing and compaction of cells was where we most wished for some form of interactive graphics layout tool.
The final step, data entry, involved writing CDS programs for the grid paper layouts. This can be an error prone process because one must accurately describe a complex geometric shape with a textual language. The CDS program was then entered into the computer with a visual editor for compilation and plotting. The plots were checked for correspondence with the hand drawn layouts and for design rule violations. If an error was found the CDS program had to be edited and recompiled.

The Design Philosophy

We adhered to a conservative design philosophy. All circuits are completely static with the exception of the prototype digital to analog converter (DAC). We felt that this would ensure a greater probability of success because static design reduces the timing and circuit complexity. The disadvantage of static design is that the circuits are larger. Precharging and dynamic sense amplifiers could not be used to reduce the memory cell driver sizes. At the time we did not feel that size was a major constraint. As we will see, the colour palette's large size may have contributed to processing yield problems.

All circuits are true CMOS, that is, for every gate there is both an n channel and a p channel device. We felt that this was the best way to keep the performance high without going to dynamic circuitry. The bidirectional transmission gates were also constructed with both n and p channel devices to improve their...
switching characteristics. True CMOS design results in a larger chip because of two reasons. First and most obvious, twice as many transistors are required to implement each gate. Second, true CMOS requires the complement of every signal to be present. This means that extra area is required for the complement buses and for the drivers that generate the complements.

All gate fan-outs were restricted to 3 or 4. As was discussed earlier the optimal fan-out is $e$ (≈2.718), however the delay curve is fairly flat near its minimum and therefore a fan-out of 3 to 4 is a good design compromise. The low fan-out resulted in many of the final drivers being quite large.

Contacts from Vss to the p-wells, and Vdd to the n substrate were liberally used. As was discussed earlier this is a technique for reducing latch-up.

The usual methods for improving performance were used. These included keeping the dimensions to a minimum and using metal instead of polysilicon wherever possible. We also avoided long runs of metal over polysilicon which can cause capacitively coupled crosstalk.
CHAPTER 6: THE COLOUR PALETTE DESIGN

The Development Synopsis

We anticipated that the colour palette chip would require at least three design iterations before it would meet the specifications. It is extremely rare for literature on new integrated circuits to disclose the number of design iterations that were required. We feel that three iterations is by no means abnormal and probably is rather optimistic.

The first iteration was designed during the Microtel Pacific Research Ltd. ISO-CMOS Design Course of October 1982. Multiple student projects were to be included on each chip and therefore our circuit size was constrained to 3750 by 3750 microns. This was sufficient area to implement a 4 by 4 bit colour palette prototype with one DAC. (Recall that the full size palette has 12 by 16 bits with three DAC's.) Although scaled down, the prototype contained every cell required for the full size palette, and each cell was designed to handle the loading that it would experience in the full size palette. The prototype therefore provided an excellent test vehicle while still maintaining upward compatibility of the circuit modules. We had enough additional chip area to include a process monitor and a separate test circuit for each module as debugging aids. The prototype was fabricated and tested and the majority of the circuit worked, however, several layout errors were detected. Detailed test results will be presented in the next chapter.
Microtel Pacific Research was kind enough to offer us another fabrication run at no charge and so the second iteration design began on April 5, 1983. Our objective was to correct the problems detected in the prototype and to expand the design into the full size colour palette. The design was completed on May 13, 1983 and the chip was fabricated and returned to us on July 8, 1983. Test results will be presented in the next chapter.

The third iteration will refine the design and correct any of the presumably small problems detected during the second iteration tests. The current project status is that second iteration chip has been resubmitted unchanged for fabrication because of suspicions of processing problems.

Module Partitions and Division of Labor

Referring back to the block diagram of Figure 8 on page 41 we see that the colour palette nicely decomposes into 8 distinct modules. They are: the digital to analog converter, pipeline register, memory array, bidirectional data buffer, y decoders, x decoder, control circuitry, and the I/O pads. The colour palette contains a great deal of regularity. Each 4 bit slice of the palette is almost identical. This justifies our decision to first design the scaled down 4 bit wide prototype. Each 4 bit slice in turn has its own regularity. For example, each bit in the memory array is identical as is each bit in the bidirectional buffer and pipeline register. The colour palette is therefore an excellent
candidate for a hierarchical design approach. We did indeed exploit its hierarchy as is shown by the tree structure in Figure 16. Each node in the tree represents a cell definition in our CDS layout program. Each link represents a CALL by the higher cell to the lower cell. The highest node represents the entire colour palette chip. Our highly structured design greatly facilitated the expansion of the prototype to the full size colour palette.

The colour palette project was large and the potential for error great. It was therefore almost a necessity that more than one designer work on the project. It is invariably easier to spot a logic error or design rule violation in someone else's work than it is in one's own work. It was my fortune to have the opportunity to work with Gordon Cheng of UBC and Gerhard Schmiing of MPR.

The first three stages of the design, namely system specification, module partitioning, and floor plan design were joint efforts by Cheng, Schmiing, and myself. The system specification and module partitioning have already been discussed. The floor plan design is actually inherent to the block diagram of Figure 8 on page 41. The relative placement of the modules deviated very little from this block diagram.

The modules were divided between Cheng and myself, and each of us looked after the remaining stages of design (logic, loading analysis, stick diagrams, mask layout, and data entry) for our own modules. Frequent interaction was of course required to
Figure 16. Colour Palette Cell Hierarchy
ensure that the modules would operate correctly together and that the interconnections would be efficient. For example the pitch (width) and bus locations of the bidirectional buffer had to match those of the memory array. We also cooperated in checking each other’s work.

Cheng’s module contributions were the x and y decoders, the bidirectional data buffers, and the control signal generator. My contributions were the memory array, pipeline register, tri-state input pad, and the process monitor. For the first iteration a dynamic DAC designed by Warren Snyder of MPR and modified slightly by myself was used. The second iteration used the same dynamic DAC in parallel with a new static DAC which was a joint effort between Schmiing, Cheng, and myself. We also used a protected input pad and buffered output pad supplied by MPR. Cheng was responsible for simulating the critical paths. I was responsible for the global routing. First iteration testing was a joint effort between Schmiing, Cheng, and myself. Second iteration testing was a joint effort between Schmiing and myself. I intend to focus the discussion on my own contributions, however contributions by Cheng and Schmiing will be discussed where necessary to maintain cohesion.

Dual Port Memory

The memory array dominates the colour palette circuit. All other modules are dependent on the memory array’s size and con-
configuration of control and data buses. The memory was therefore the first module to be designed. I will first review the specifications and traditional static memory designs and then discuss our own implementation.

The most important specification to note is that both the processor (CPU) and the CRT controller (CRTC) can asynchronously read from the memory array, but only the CPU can write to the memory. This means that the memory must be dual ported if we are to avoid the multiplexing circuitry used on the discrete MPR implementation. Dual porting requires the memory cell to have separate data and control buses for the CPU and CRTC. No arbitration is performed between the CPU write cycle and the CRTC read cycle. The probability of both devices simultaneously accessing the same nibble is low but should it happen the timing is such that a maximum of two pixels will be disrupted for one video scan and will therefore not be noticed by the user. Simultaneous CPU and CRTC read cycles on the same nibble should have no effect on the output.

A traditional 6 transistor cross coupled static CMOS memory cell is shown in Figure 17. A subset of the address lines are decoded to enable one of the word lines and hence one row in the memory array. When its word line is true, the memory latch is connected to the data bus. The remaining address lines are decoded to select which of the data lines running vertically through the array are to be operated on. If a write cycle is in progress then the desired bit state is placed on the data bus by
a driver that is sufficiently large to force the state of the cross coupled latch to comply. If a read cycle is in progress then a high impedance sense amplifier is used to determine the bit state. The advantage of this design is that the memory cell can be kept very small, particularly if a sophisticated sense amplifier capable of detecting small voltage fluctuations is used. The disadvantage of this approach is that the design parameters such as the transistor sizes are very critical and require careful optimization and simulation [PULFREY 83]. Also, many externally static designs employ internally dynamic circuitry which we tried to avoid [OCHII 82]. Another problem with this approach is that to make the 6 transistor memory cell dual ported would require the addition of 2 transistors and another word line. Extremely careful design would be required to prevent simultaneous bit accesses by the CPU and CRTC from disrupting the bit state. For traditional memory applications small size is by far the most important design consideration and therefore one accepts the design complications that come with the small size. Our application, on the other hand, requires a memory array of only 12 by 16 bits. Our most important design constraint was to ensure that the circuit worked and met the specifications on the first or second iteration. We were willing to sacrifice area to achieve a working circuit.
The first step in the memory design was to determine the control and data signals required. The CRTC presents 4 address lines which are decoded on the palette to select 1 of the 16 rows, each of which corresponds to a colour. When a row is selected each of its 12 bits must place their states on the CRTC data buses. The data is then latched into the pipeline register for input to the DAC's. The important points to note are that the CRTC does not write to the memory and x decoding is not required because the data is read out one row at a time. A single control signal Yv (video y select) can therefore be used to force the memory cell to place its data on the CRTC data bus.
The CPU, on the other hand, both reads and writes to the memory a nibble at a time. Signals are therefore required to select the specific nibble and to specify if the operation is a read or a write. The MPR specifications (Appendix B) called for the read data to be held on the data bus for 20-50 nsec after the processor enable clock (E) goes false. The CPU read signal (R) was generated from the off chip signals as follows:

\[ R = CS' \times R/W \times A \]

where  
- \( CS = \) chip select 
- \( R/W = \) processor read/write 
- \( A = \) decoded processor address (A4-A5)

The CS and R/W signals are true for only 10 nsec after E goes false, however we feel that the data bus capacitance and the buffer delays will hold the data valid for the additional 10 nsec required to meet the specification. The CPU write cycle requires that the data be valid when it is latched into the memory. The signal available for latching the data is the negative edge of the processor clock (E). The write signal was therefore generated from the off chip signals as follows:

\[ W = CS' \times R/W' \times A \times E \]

where  
- \( E = \) processor enable clock 
and the other signals are defined above

Note that the CPU read and write signals are not simply complements. We therefore had two choices for implementing the CPU control signals. The first approach was provide the cell with x
and y signals to select the cell and a third signal to specify read or write. The second approach was to supply a signal to select a row in the array and provide separate read and write signal lines. If both the read and write lines were false the cell would be disabled and thus the read and write lines could double as column selects. We chose the second approach because it was much easier to accommodate the differences between the read and write signals discussed above, and because both approaches require the same number of control lines. Table IV lists the control and data signals required in the memory array. Notice that the compliments of all the control signals are required because of our decision to use true complementary circuitry.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dp</td>
<td>processor (CPU) data</td>
</tr>
<tr>
<td>Dy</td>
<td>video (CRTC) data</td>
</tr>
<tr>
<td>Yv Yv'</td>
<td>video row select</td>
</tr>
<tr>
<td>Yp Yp'</td>
<td>processor row select</td>
</tr>
<tr>
<td>R R'</td>
<td>processor read</td>
</tr>
<tr>
<td>W W'</td>
<td>processor write</td>
</tr>
</tbody>
</table>

Table IV. Memory Control and Data Signals

The next step in the design was to decide on the memory bus structure. The read and write lines must of course run at right angles to the row select lines because they double as column select signals. The problem then was to decide which to place in metal and which to place in polysilicon. We decided to place the read and write lines in metal because the write timing is more
Figure 18. Memory Bus Structure
critical than the row select timing. That is, the negative edge of W is used to latch the data. Furthermore, the array y dimension is larger than its x dimension and therefore the columns should be in metal to minimize the propagation delay. Power buses must be in metal and the previous decision therefore forced Vdd and Vss to run vertically through the array. The data buses must run vertically which is fortunate because it meant that they could also be placed in metal.

We tried many different bus configurations and Figure 18 shows the one we finally decided upon. Notice that W, W', R, and R' are shared between adjacent bits of each nibble. Also note that the right two bits of each nibble can be formed by reflecting the left two bits of the nibble about the nibble's central Vss bus. This is in fact how we generated the array. Two bits were laid out and reflected to form a nibble using the CDS rotate and place command. The nibble was then called three times with suitable translations to form the row. Finally the row was called 16 times with translations to form the array.

The logic schematic for the memory cell is shown in Figure 19. The number on each gate indicates its size in terms of minimum sized gates. Note that all fanouts are about 3 and that inverters Q2 and Q6 have also been sized large enough to drive the data buses with a fanout of 3. Inverters Q4 and Q5 store one bit via the feedback loop through transmission gate Q3. During the CRTC read cycle Q6 is turned on by Yv and drives the data onto the Dv bus. Similarly, during the CPU read cycle Q2 is
turned on by \((Yp \times R)\) which drives the data onto the Dp bus. Note that the CPU and CRTC read cycles can occur simultaneously without any problems. During the CPU write cycle Q2 is turned off and the Q3 feedback loop is broken. Q1 is turned on by \((Yp \times W)\) which drives the new data into Q4 and Q5. When \((Yp \times W)\) goes false the new data is latched by Q3. Note that there is the possibility of a glitch on Dv if Yv goes true during a CPU write cycle but this was discussed earlier and is not a problem.

![Figure 19. Memory Cell Logic Schematic](image-url)
The stick diagram for the memory cell is shown in Figure 20. Much effort was spent minimizing the number of polysilicon bridges required when two metal lines must cross. We confirmed the often stated maxim that time spent at the stick diagram stage always pays off during the layout stage. The layout for two bits of the memory is shown in Figure 21. Note that all buses enter and exit on the same longitude or latitude so that the cell can be repeated in both directions and still maintain bus continuity. Also note that Vss contacts to the p wells and Vdd contacts to the substrate have been liberally used to avoid latch-up problems. The size of the two bit cell is 244 by 440 microns.

Given the layout we calculated the capacitive loading on each bus in the array in terms of minimum gate loads and the data is presented in Table V. These figures were used to design optimal drivers in the row and column decode circuitry. The loading is quite significant on several of the buses. This is one of the deleterious effect of a fully static, true CMOS design. Fortunately our memory array is small and is not required to be expandable.

<table>
<thead>
<tr>
<th>Bus</th>
<th>Loading (# of minimum gate loads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dv</td>
<td>0 &lt;---+</td>
</tr>
<tr>
<td>Dp</td>
<td>3</td>
</tr>
<tr>
<td>W'</td>
<td>6 -- multiply by the number of rows</td>
</tr>
<tr>
<td>W</td>
<td>6</td>
</tr>
<tr>
<td>R'</td>
<td>36</td>
</tr>
<tr>
<td>R</td>
<td>16 &lt;---+</td>
</tr>
<tr>
<td>Yp'</td>
<td>21 &lt;---+</td>
</tr>
<tr>
<td>Yp</td>
<td>11 -- multiply by the number of bits per row</td>
</tr>
<tr>
<td>Yv'</td>
<td>10</td>
</tr>
<tr>
<td>Yv</td>
<td>4 &lt;---+</td>
</tr>
</tbody>
</table>

Table V. Memory Cell Bus Loading
Figure 20. Memory Cell Stick Diagram
Figure 21. Memory Cell Layout
Pipeline Register

The pipeline register is a two stage register with the function of ensuring that the DAC output is valid for a full cycle of the pixel clock (Pclk). Appendix B contains the timing diagrams and Figure 22 shows the pipeline register logic schematic. Again, the number on each gate represents its size as a multiple of minimum sized gates. Note that minimal loading is presented to the Dv bus and that Q4 is large enough to drive the DAC's with near optimal fanout. Inverters Q1 and Q2 with their feedback loop form the first register stage. Data is passed through to Q1 when Pclk is false and is latched in the first stage in the rising edge of Pclk. The new data stored in the first stage is passed through to Q3 while Pclk is true. This data is then latched in the second stage on the negative edge of Pclk thereby guaranteeing that the DAC input is valid for an entire Pclk cycle.

The bus structure of the pipeline register is dictated by the memory array. Data and power flow vertically on metal and the control signals Pclk and Pclk' flow horizontally on polysilicon. The pipeline register stick diagram is shown in Figure 23. Note that all of the transmission gates are cleanly formed in two straight diffusion wires. The layout requires only two metal-polysilicon bridges. A plot of the pipeline register layout is shown in Figure 24. The horizontal pitch of the register is fixed by the power and data rails coming from the memory array. The size of the pipeline register cell is 229 by 216 microns. The
capacitive loading on each bus was calculated in terms of minimum gate loads and is presented in Table VI.

<table>
<thead>
<tr>
<th>Bus</th>
<th>Loading (# of minimum gate loads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pclk</td>
<td>6</td>
</tr>
<tr>
<td>Pclk'</td>
<td>6</td>
</tr>
<tr>
<td>Dv</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table VI. Pipeline Register Bus Loading**

![Diagram](image)

**Figure 22. Pipeline Register Logic Schematic**
Figure 23. Pipeline Register Stick Diagram
Figure 24. Pipeline Register Layout
Digital to Analog Converter

The MPR specification called for the DAC to have 4 bits of resolution, operate at 5.5 MHz, settle to within ±1/2 LSB within 50 nsec, and use a 1 volt reference. There are a variety of ways to implement a DAC. One common approach, called the R-2R ladder network, generates binary weighted currents which are then summed or shunted to ground depending on the digital input [GUY 82]. The current sum can then be used to form the analog voltage. Another approach uses an analog ramp generator, a counter-comparator, and a sample and hold circuit [MACK 82]. The analog ramp and counter are reset to zero and then allowed to increase. When the counter value equals the digital input the sample and hold circuit is triggered to freeze the analog voltage. Another approach uses a resistor divider network to generate all possible voltages. The desired voltage can then be selected by an analog switch. Various configurations of resistor dividers are possible [POST 83].

The DAC used on the first iteration prototype was designed by Warren Snyder at MPR and is shown in Figure 25. A diffusion wire is used as the resistor ladder with sixteen equally spaced analog switches placed along its length. The analog switches are formed with wide nMOS transistors, and one of the sixteen switches is turned on by the input decoder. A clock signal is used to precharge the decoder array to ground since the decoder was implemented with pMOS, rather than CMOS to conserve area. As the test results will show, the dynamic precharge signal caused some
noise on the analog output. The accuracy of the DAC depends on process uniformity and the output node must be terminated with a high impedance to avoid shunting any current from the resistor ladder. The DAC presents an input load of 28 minimum sized gates to the pipeline register and its size is 334 by 550 microns.

One minor modification was made to the MPR DAC. The negative voltage reference on the voltage divider was disconnected from the Vss bus and given its own pad. This was done to reduce the output noise caused by the digital circuitry and to permit us to shift the output voltage if required.

A new DAC was designed for the second iteration colour palette and is shown in Figure 26. The decoder was implemented in nMOS, rather than CMOS, again to save area. This time, however, the dynamic precharge circuitry was replaced with a pull-up resistor formed by a long p channel transistor with its gate tied to Vss. The nMOS analog switch was also replaced with a true CMOS transmission gate to improve its characteristics.

Both the dynamic DAC and the static DAC were connected in parallel on the second iteration chip to maximize our probability of success. This increased the routing complexity but did not present any other problems because the memory array pitch was sufficient to accommodate the 3 pairs of DAC's. The new DAC did not present significant additional loading to the pipeline register so we did not have to increase the driver sizes.
Figure 25. Dynamic DAC Layout
Figure 26. Static DAC Layout
Input Pad

A simple unbuffered input pad with a CMOS transmission gate was designed and is shown in Figure 27. The signals en and en' can be used to disable the pad.

Figure 27. Input Pad Layout
Figure 28. Process Monitor Layout
Process Monitor

A process monitor was designed for inclusion on the first iteration prototype. The objective was to be able to measure device characteristics and to confirm that the fabrication process did not have any gross problems. The process monitor contained a 21 stage ring oscillator with a buffered output, a large CMOS inverter, a pMOS transmission gate, and an nMOS transmission gate. It was hoped that the ring oscillator would provide information on the process speed, and that the inverter and transmission gates would provide device characteristics. A plot of the process monitor is shown in Figure 28.

Simulation

A SPICE simulation on several of the colour palette critical paths was performed and the results are summarized in Table VII [CHENG 83]. A test circuit with a hand calculated delay was simulated to verify the model and all results were increased by 50 percent to be conservative. All circuits were reduced to equivalent chains of inverters with capacitive loading. The results show that there should be a sufficient safety margin in our design to meet all of the design specifications.
Critical Path Simulation Results

<table>
<thead>
<tr>
<th>Critical Path</th>
<th>Simulated Delay nsec</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yv'</td>
<td>57</td>
<td>120</td>
</tr>
<tr>
<td>Yp'</td>
<td>72</td>
<td>150</td>
</tr>
<tr>
<td>R'</td>
<td>120</td>
<td>150</td>
</tr>
<tr>
<td>W</td>
<td>87</td>
<td>150</td>
</tr>
</tbody>
</table>

Table VII. Critical Path Simulation Results

Global Routing

Appendix E contains CDS source listings of the three top level cells: pldac, mempldac, and palette. These listings show how the CDS language was used to interconnect the modules. Pldac connects 3 of the dynamic DAC's and 3 of the static DAC's to the 12 bit pipeline register. Mempldac connects pldac to the memory array. Finally, palette pulls together all of the remaining components to form the colour palette chip. Note the heavy use of DEFN and LOCN, and BBOX commands. Also note how diagnostics are printed at the end of each compilation.
CHAPTER 7: RESULTS

First Iteration Results

Figure 29 shows the first iteration prototype chip. Note the seven test circuits that are situated on the prototype periphery. They are, clockwise from the top left, control signal generator cell (xsigtst), bidirectional buffer cell (bdbtest), pipeline register cell (testpl), process monitor (proctest), x decoder cell (xdctest), two bit memory cell (memtst), and a y decoder cell (ydctst). Because of the test circuitry there are far too many pads to be bonded in one package. We were therefore given 3 packages, each with different portions of the chip bonded.

Figure 29. The First Iteration Chip

96
Figure 30. Static First Iteration Test Circuit
The circuit shown in Figure 30 was first used to test the process monitor to see if any devices on the circuit were alive. Static tests confirmed that the n and p channel transmission gates, and the inverter were working correctly. Figure 31 shows the measured inverter characteristics. Unfortunately for some unknown reason we could not make the ring oscillator work. A careful review of the design as well as a careful inspection under the microscope revealed no errors. One possibility is that there is excess loading and insufficient gain around the loop to cause the device to oscillate.

![Graph showing inverter characteristics](image)

Figure 31. Process Monitor Inverter Characteristics
Tests were performed by G. Schmiing on the dynamic DAC which was included as a test circuit on a separate chip. A 4 bit counter and several buffers, as shown in Figure 32, were used to cycle the DAC through all of its states. Figure 33 shows the analog output voltage as a function of static binary inputs. Note that the device shows very good linearity. Next a 1 MHz clock was used to test the DAC's dynamic response. Figure 34 shows that noise greater than 1/2 LSB is coupled through to the output. It was decided that to reduce this noise the analog and digital power supplies should be isolated. Also the n channel analog switches could be enlarged to reduce their on resistance.

Figure 32. Dynamic DAC Test Circuit
Figure 33. Dynamic DAC Output with Static Inputs

Figure 34. Dynamic DAC Output at 1 MHz
Static tests performed on each of the remaining test modules showed that they were functionally correct. These test circuits were not capable, however, of predicting dynamic performance.

The final stage in the test procedure was to test the prototype palette. A similar counter circuit to that shown in Figure 32 was used to cycle the CRTC inputs through all of their states. The CPU inputs were set up with DIP switches to permit static loading of the memory array. Several immediate layout problems were detected. Vss and Vdd on the CPU output pads were connected backwards which resulted in a short circuit. The remedy was to use a probe to cut the output pad power lines thereby making it impossible to test the CPU read cycle. Another problem was that the p-well of the CPU input pad was not tied to Vss. This meant that the input pads could not be turned fully on and therefore the data bus could not be pulled to Vss. Fortunately the memory array powered up in the "0" state and it was still possible to write "1"'s into it. Another problem was a metal to metal design rule violation in the y decoder which resulted in unreliable accesses of odd numbered memory locations. Finally, as the prior DAC tests indicated, there was significant coupling of digital noise to the DAC analog output.

Despite all of these problems the test results were encouraging. Values written into the memory were confirmed by the analog output voltages. It was possible to run the circuit in excess of 13 MHz and still obtain a reasonable output. Tests also showed that the DAC settled in about 3 nsec with a 15 pf load.
To summarize then, the first iteration prototype demonstrated that all circuit modules were functional and that the specification for the CRTC read cycle was met. All problems detected were trivial to correct on the second iteration chip.

Second Iteration Refinements

For the second iteration we increased the memory array to its final 12 by 16 bit configuration. This of course required enlargements to all of the support circuitry such as the x and y decoders. Since the modules were designed to be upward compatible from the first iteration we did not have to make any significant modifications to the individual modules. As was discussed earlier, a new static DAC was designed and included in parallel with the dynamic DAC. This resulted in there being six DAC's on the chip, two for each colour. The design rule violations detected in the first iteration were corrected. Power bus sizes were increased and the analog power supply for the DAC's was separated from the digital supply in an attempt to decrease the output noise. Protected input pads were used and more attention was given to pad placement to minimize the difficulty in bonding the chip. Test circuits for each module were not included because the first iteration test results had shown the modules to be functional and because there was insufficient area for their inclusion. Signal paths and module placements were tidied up which resulted in some reduced path lengths and decreased the chances of us making a routing error.
The second iteration chip has approximately 6000 devices and is 4070 by 6398 microns in size. This is significantly larger than the first iteration chip and therefore required the use of two new CAD tools. A section plotter running on a VAX 750 was used to give blow-up plots of critical sections on the chip. These were invaluable for error checking at module interfaces such as where the memory array connected to the y decoders. Also, as a final check we took our data file to Pacific Microcircuits Ltd. where a large Versatek plot of the entire chip was generated.

Another important enhancement made on the second iteration was the creation of a standard form of cell documentation. We found that when several designers work on the same project standard documentation is a necessity. Integrated circuit design is such that each person has multiple files with many external references and all of the files must be correctly assembled at the end of the project. As a first pass we made a standard file header that detailed the cell name, function, the designer's name, the cell size and bounding box coordinates, external references, node names, and revision history. An obvious enhancement to this documentation would be cell performance and simulation data. It is hoped that our documentation will assist whomever works on future iterations of the colour palette at MPR.
Second Iteration Results

We received eight bonded chips on July 8, 1983. Figure 35 shows a photograph of the second iteration chip layout and Figure 36 shows the circuit used to test the chip. Processor data, address, and control lines were set up with DIP switches. Video addresses were cycled through all possible values with a 4 bit counter. The pipeline register clock was operated at twice the frequency of the video address generator so that we would have a better chance of observing video access problems.

Cursory tests revealed that each of the eight chips had different characteristics. Chip numbers 2, 7, and 8 had short circuits and could not be tested. Chips 3, 4, and 6 had control circuitry or address bus problems rendering all three colour guns inoperative. Chips 1 and 5 exhibited some life and the majority of tests concentrated on these two chips.

The problem of non-uniform results was also present on individual chips. Tests on chip #1, for example, demonstrated that while the majority of memory cells were working, defects were randomly scattered throughout the array and included bits that would not change state, bits that would change state only once, and bits that seemed to affect the state of other bits. Table VIII summarizes the test results on the memory in chip #1. Determining the cause of the defects was extremely difficult. If a memory design error existed one would expect the errors to occur in the same locations of each of the three columns of
Figure 35. The Second Iteration Chip
Figure 36. Second Iteration Test Circuit
nibbles, since they are exact duplicates of each other. This was not the case as no logical pattern of errors was obvious.

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Gun 15 A4=0 A5=1</th>
<th>Gun 17 A4=1 A5=0</th>
<th>Gun 19 A4=0 A5=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bit 3 bad</td>
<td>bad</td>
<td>bad</td>
</tr>
<tr>
<td>1</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>2</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>3</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>4</td>
<td>bit 2 bad</td>
<td>bad</td>
<td>ok</td>
</tr>
<tr>
<td>5</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>6</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>7</td>
<td>bad</td>
<td>bit 4 = 0</td>
<td>ok</td>
</tr>
<tr>
<td>8</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>9</td>
<td>ok</td>
<td>bit 3 = 0</td>
<td>ok</td>
</tr>
<tr>
<td>10</td>
<td>bad</td>
<td>bit 3 = 0</td>
<td>ok</td>
</tr>
<tr>
<td>11</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>12</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>13</td>
<td>ok</td>
<td>bits 0 once only</td>
<td>affects others</td>
</tr>
<tr>
<td>14</td>
<td>ok</td>
<td>bits 0 once only</td>
<td>ok</td>
</tr>
<tr>
<td>15</td>
<td>bad</td>
<td>bits 0 once only</td>
<td>bad</td>
</tr>
</tbody>
</table>

Table VIII. Chip #1 Memory Test Summary
Chip #5 demonstrated even more inconsistent results than chip #1. With chip #1 it was possible to read back written data to confirm that the memory contained what we expected. Read back data from chip #5 on the other hand had no correlation to the written data.

Only one design error was confirmed on the second iteration chip. The polysilicon clock line to the dynamic DAC inadvertently crosses the dynamic DAC's analog power line which is routed in diffusion. The resulting normally closed switch made the dynamic DACs inoperative. The error was in part caused by the use of filter plots to do the global routing. Filtering removes excess, and obviously sometimes critical, information from the plots. Fortunately the new static DAC works correctly as was demonstrated by our ability to generate each of the sixteen possible output voltages on chip #1.

While there appears to be more than one cause of the memory defects we did come up with a promising hypothesis to explain the interaction between bits. It was noticed that the apparent state of a defective bit often depended on the previous state of the data bus. More specifically, if the data bus was high just prior to accessing the defective bit it appeared to be stuck high, whereas if the data bus was low prior to accessing the defective bit it appeared to function correctly. Our hypothesis was that the defective bits had problems with the n channel halves of their output drivers and therefore were unable to pull the data bus down to ground. Armed with the pull-down hypothesis we went
back and reviewed our test results to see if any of the problems could be explained. Some of the previously incomprehensible results from chip #5 were very neatly explained.

To further substantiate our pull-down hypothesis we used a photodiode effect. First, zeroes were written into all memory locations. This made the bus state low prior to accessing each of the defective bits and the outputs therefore correctly displayed the lowest analog voltage. Next, the lid on the chip was removed and a bright incandescent light was shone on the chip. The known defective bits immediately displayed a high state. When the light was turned off the bits reverted to their correct zero state. The light apparently generated enough charge carriers on the data bus to cause it to float high when the bits with defective pull-downs were accessed.

We then returned to our layout plots of the memory cell and by eliminating those sections that we knew from tests were working correctly we pin-pointed where on the chip we thought the pull-down problem was occurring. Since no design error was apparent from the plots we suspected a fabrication problem. The chips were placed under a microscope and the area in question scrutinized. Unfortunately no problem was visible, however, many other fabrication defects were obvious elsewhere on the chips. These defects included partially broken metal lines, short circuits, deep scratches and gouges on the surface, abnormal contacts, foreign material, and improperly stripped photoresist.
Many of the problems apparently were introduced during the wafer slicing and bonding procedures. We therefore requested and received a new set of 16 unbonded chips that came from a different wafer. Before bonding these chips we thought it wise to look at them under the microscope for any obvious problems. Every one of these chips exhibited serious surface defects of the nature described above.

To summarize the results then, we demonstrated that each module functioned correctly, however no one chip contained modules that all functioned simultaneously. There are two obvious explanations. First, a design error may exist, however, since the problems are not consistent from chip to chip it must be a marginal violation. I feel that this explanation, while possible, is not probable because of our very conservative design approach. The second possibility is that there are problems with the fabrication of the chip. This might explain the inconsistent results and is surely substantiated by our microscope observations. A third less obvious possibility is a combination of the two. Yield decreases rapidly as the chip size increases. It may therefore be a case of too large a chip and too small a test sample.

We feel that the most probable explanation is the that there were problems with the chip fabrication. Our design has therefore been resubmitted unchanged for fabrication to see if any of the problems disappear. It is likely that our design will still require fine tuning to meet the performance specifications but at
least we may obtain consistent results from which we can locate the areas that require improvements.
CHAPTER 8: CONCLUSIONS

This thesis has made a variety of useful contributions. Integrated circuit design is a new field of study and as such a detailed overview of the topic with emphasis on computer aided design was warranted. ISO-CMOS, a relatively new fabrication process that is becoming a standard, was also discussed.

A novel integrated circuit, the colour palette, was designed, fabricated, and tested in two stages. The first stage resulted in a scaled down prototype with test circuitry, and the second stage resulted in a full size colour palette. The colour palette combines digital and analog circuitry on the same substrate and consists of a dual port static memory array with three digital to analog converters. The full size colour palette contains about 6000 devices and was the largest chip designed to date at Microtel Pacific Research Ltd.

Unlike many novel circuits reported in the literature, our test results were disclosed. All circuit modules were functional on the first iteration, however, several design errors were detected and corrected for the second iteration. Second iteration test results showed that each of the modules functioned correctly, albeit not simultaneously on the same chip. Most of the evidence points to fabrication problems and as such the chip has been resubmitted for fabrication.
In addition to being an excellent vehicle for testing new concepts in computer aided design, such as a regular structured design methodology, the colour palette has many applications and has a very strong market potential.
BIBLIOGRAPHY


Gale, Dan, "VLSI in Canada", VLSI Implementation Centre, Queen's University, vol. 1, no. 1, pp. 1-4, April 1983.


APPENDIX A

Schematics and Block Diagrams of the Microtel Pacific Research Ltd.
Discrete Component Colour Palette Implementation
APPENDIX B

Colour Palette Specifications
HARDWARE FUNCTIONAL SPECIFICATION
FOR
COLOR PALETTE INTEGRATED CIRCUIT

Prepared by
G. Schmiing

August 17, 1982
CONTENTS

1. GENERAL DESCRIPTION ................................................................. 2
   1.1 Color Palette Pin Description ...................................................... 2

2. PIN DESCRIPTION .............................................................................. 3
   2.1 PROCESSOR INTERFACE ............................................................... 3
      2.1.1 Data Bus (D0-D3) .................................................................. 3
      2.1.2 Enable (E) ............................................................................ 3
      2.1.3 Chip Select (CS) .................................................................... 3
      2.1.4 Address Bus (A0-A5) .............................................................. 3
      2.1.5 Read/Write (R/W) .................................................................. 3
   2.2 VIDEO INTERFACE ........................................................................ 4
      2.2.1 Pixel Data (P0-P3) ................................................................. 4
      2.2.2 Pixel Clock (PC) ..................................................................... 4
      2.2.3 Display Enable (DE) ............................................................... 4
      2.2.4 Video Output (V0-V2) ............................................................. 4
      2.2.5 Reference Voltage (Vref) ......................................................... 4
      2.2.6 Color Palette Memory Organization ....................................... 5

3. VIDEO TIMING .................................................................................... 6
   3.1 VIDEO TIMING CHARACTERISTICS ............................................... 6

4. PROCESSOR TIMING .......................................................................... 7
   4.1 PROCESSOR TIMING CHARACTERISTICS ....................................... 7

5. BLOCK DIAGRAM ............................................................................... 8
1. GENERAL DESCRIPTION

The Color Palette basically consists of a multiported register array whose outputs directly drive three video DAC's. The processor, independent of the video access, has full read, write capability of the internal registers.

The register array appears as 16 color values at 12 bits per value to the video interface, and as 64 words at 4 bits per word to the host processor. See fig. 1 Memory Organization.

Color translation is performed by the 4 bit pixel data (P0-P3), addressing one of the 16 registers in the lookup table. The color value stored in that register drives the three video DAC's, which produce an analog video signal proportional to the digital value. This video signal is used to drive the RED, GREEN, BLUE inputs of a color monitor.

As a result the Video Controller can now display any 16 colors out of a repertoire of 4096.

1.1 Color Palette Pin Description
2. PIN DESCRIPTION

2.1 PROCESSOR INTERFACE

The Processor interfaces to the Color Palette via a bidirectional data bus (D0-D3) using (A0-A5), CS, E and R/W for control signals.

2.1.1 Data Bus (D0-D3)

The bidirectional data lines (D0-D3) allow data transfers between the Color Palette register and the processor. Data bus drivers are high impedance until a read cycle.

2.1.2 Enable (E)

The Enable signal is a high impedance, TTL compatible input which clocks data to and from the Color Palette. The high to low transition is the active edge.

2.1.3 Chip Select (CS)

The CS signal is a high impedance, TTL compatible input which selects the Color Palette, when low, to read or write to the internal register file.

2.1.4 Address Bus (A0-A5)

The Address Bus signals (A0-A5) are high impedance, TTL compatible inputs which selects one of the 48 registers to be written or read. Data transfers are then performed under the control of CS, E, R/W signals.

2.1.5 Read/Write (R/W)

The R/W signal is a high impedance, TTL compatible input which determines whether the internal registers are written or read. A write is defined as a low level.
2.2 VIDEO INTERFACE

The Video interface accepts pixel data (P0-P3) from the Video Controller, and translates the data through the Color Lookup table to a 12 bit color value. This 12 bit color value is converted by three video DAC's to the analog signals (RED, GREEN, BLUE).

2.2.1 Pixel Data (P0-P3)

The Pixel Data signals (P0-P3) are high impedance, TTL compatible inputs which act as address inputs to the color lookup table.

2.2.2 Pixel Clock (PC)

The PC clock signal is a high impedance, TTL compatible input which loads the color pipeline register after the access time of the lookup table. The low to high transition is the active edge.

2.2.3 Display Enable (DE)

The DE signal is a high impedance, TTL compatible input which disables the video output by clearing the pipeline register. This input is active low.

2.2.4 Video Output (V0-V2)

The three video outputs are analog signals generated by the three video DAC's. Inputs of a color monitor.

2.2.5 Reference Voltage (Vref)

This input applies a user supplied voltage reference to the resistive divide chain of the digital to analog converter.
### 2.2.6 Color Palette Memory Organization

<table>
<thead>
<tr>
<th>PROCESSOR ACCESS</th>
<th>DATA BUS</th>
<th>VIDEO ACCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td><strong>COLOR WORD 1</strong></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td><strong>COLOR WORD 2</strong></td>
</tr>
<tr>
<td>0</td>
<td>****</td>
<td><strong>COLOR WORD 16</strong></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>RED</strong></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>GREEN</strong></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>BLUE</strong></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><strong>STD</strong></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td><strong>RED</strong></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td><strong>GREEN</strong></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td><strong>BLUE</strong></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td><strong>STD</strong></td>
<td></td>
</tr>
</tbody>
</table>
3. VIDEO TIMING

3.1 VIDEO TIMING CHARACTERISTICS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>CHARACTERISTIC</th>
<th>MAX</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tp</td>
<td>Pixel clock cycle time</td>
<td>180 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Tad</td>
<td>Address after Pclk</td>
<td>20 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Ta</td>
<td>Register access time</td>
<td>20 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Th</td>
<td>Color data hold time</td>
<td>20 nsec</td>
<td>20 nsec</td>
</tr>
<tr>
<td>Ts</td>
<td>DAC settling time +/- 1/2 LSB</td>
<td>50 nsec</td>
<td>-</td>
</tr>
</tbody>
</table>

Resolution 4 bits
Output resistance 1 K ohm max.
Output Voltage 0 - 4 volts
4. PROCESSOR TIMING

4.1 PROCESSOR TIMING CHARACTERISTICS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>CHARACTERISTIC</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcyc</td>
<td>E cycle time</td>
<td>500 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Tas</td>
<td>Address setup time before E</td>
<td>40 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Tah</td>
<td>Address hold time</td>
<td>10 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Tcs</td>
<td>R/W, CS setup time before E</td>
<td>40 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Tch</td>
<td>R/W, CS hold time</td>
<td>10 nsec</td>
<td>-</td>
</tr>
<tr>
<td>Tddr</td>
<td>Read data delay time</td>
<td>0 nsec</td>
<td>150 nsec</td>
</tr>
<tr>
<td>Trh</td>
<td>Read data hold time</td>
<td>20 nsec</td>
<td>50 nsec</td>
</tr>
<tr>
<td>Tds w</td>
<td>Write data setup time</td>
<td>50 nsec</td>
<td>20 nsec</td>
</tr>
<tr>
<td>Thw</td>
<td>Write data hold time</td>
<td>60 nsec</td>
<td>-</td>
</tr>
</tbody>
</table>
5. BLOCK DIAGRAM
APPENDIX C

GTE ISO-CMOS Process Sequence
ISO - CMOS

PROCESS SEQUENCE
APPENDIX D

The CDS Layout Language from "CMOS Design System 1982" Microtel Pacific Research Ltd. by Warren Snyder
A cell in CDS is defined via statements in the CDS layout language as opposed to being created via a graphics editor. This allows a designer to parameterize his designs, such as dynamically variable cell sizes to suit varying load conditions. It also allows for easy creation of structures by the system itself, given just a few simple values. The best example of this in automatic synthesis of PLA's, ROMS, and control units for microprocessors. The CDS system currently has an automatic PLA generator from truth tables (see Appendix 2).

Statements are either SYSTEM commands such as, FOR,NEXT,END, and assignments, XOR layout primitives such as Wire, Polygon, Defc. All text is entered in lowercase and if the keyword is a system command it will be all capitalized (by the EDITor) else only the first character will be. Only one statement per line is allowed.

example

Defc("test") !gives name to cell
X1=10 !assignment to a variable
Layer(Active) !primitive specifying layering.
Polygon(0,0) !polygon with starting point at 0,0
  Xty(X1,10) !continues to points (X1,0) (X1,10)
  Xy(0,10)  !continues to point (0,10) =>box
Endc  !closes definition of cell
END   !last line of every file

The example shows that primitives such as Polygon (and Wire) are usually continued on a new line with continuation primitives (Xty). One can continue a polygon (or wire) indefinitely until the next primitive is stated. A complete description of all commands is given in Appendix 1. The listings at this manual's end best illustrate the nature of the CDS layout language.

All coordinates given as arguments to primitive commands are translated by the system relative to an origin determined by the two variables Xbase,Ybase. The actual coordinate is determined by adding Xbase to every x value and Ybase to every y value. This allows setting up local origins within cells. These variables are reset to 0 by every Defc primitive.
To use a cell that has been previously defined one uses the 'Place' command. The cell whose name is given as argument to 'Place' can be optionally first mirrored OR rotated about its axes then translated to a given location within the current definition. Rotations currently are limited to 0, 90, 180, XOR 270 degrees about the origin (0,0). Mirroring is about the y-axis 'mx' (x values are negated) XOR about the x-axis 'my' (y values negated).

The interface between cells can be implicit in cases where two cells are intended to but against each other or explicit via 'Nodes'. A 'Node' is a point defined within a cell which other cells can interconnect to, and are labeled by the designer with names (need not be unique outside of the cell).

To interconnect any two nodes one uses the 'Locn' command to obtain the current coordinates of the desired nodes and via a 'Wire' command route a conductor between them. The location of a node is obtained from its name and owner cell, freeing the designer from remembering coordinates.

Three primitive contact cells exist for specifying metal to active, metal to poly, or metal to poly to active contacts (BC butting contact): ma, mp, bc respectively. Their use is optional but can simplify most designs (remember to compile "Lib" if you wish to use them). The orientation of the poly-active in a BC is specified via four directions; be=East, bn=North, bw=West, bs=South. The direction is along a vector from active to poly.
CDS PRIMITIVES

xi,yi specify arguments to primitives and may be expressions. Arguments within () denote optional coordinates pairs.

be, bn, bw, bs
=> be(x1,y1, (x2,y2, x3,y3, x4,y4, x5,y5, x6,y6))
- Butting contacts formed by Ma-Mp contact pairs. Origins at center of Ma contact. East, North, West, South locations for Mp.

box
=> box(x1,y1, x2,y2)
- Rectangle with lower left at x1,y1 and upper right at x2,y2 (on current layer).

bbox
=> bbox("test", x1,y1, x2,y2)
- Interface with data-base to find the bounding box of a previously compiled cell. Returns coordinates of cell’s rectangular outline. x1,y1,x2,y2 must be variable names.

cut
=> cut(x1,y1, (x2,y2, x3,y3, x4,y4, x5,y5, x6,y6))
- Sets current layer to 'Cut' if not already and places 2x2 squares centered at the given coordinates.

defc
=> defc("test")
- Starts a new cell definition and names that cell. Resets xbase and ybase to 0. The current layer is undefined.

defn
=> defn("in1", x1, y1)
- Defines a node within a cell. The name and point are paired together.

dx
=> dx(x1)
- Continuation path increment.
  \[(x,y) \rightarrow (x,y) + (dx,0)\]
dy

=>dy(y1)
- Continuation path increment.
\((x, y) <= (x, y) + (0, dy)\)

dxy

=>dxy(x1,y1)
- Continuation path increment.
\((x, y) <= (x, y) + (dx, dy)\)

dy

end
- Very last line of a design file.

den
- Terminates a cell definition.

layer

=>layer(Active)
- Sets the current layer, which remains valid till modified.

locn

=>locn(cell#, "node", x, y)
- Returns the coordinates of a node in the cell, given by the cell#, as it appears after all transformations. The cell# is set via the Place command which defined the cell's location.

ma

=>ma(x1,y1,(x2,y2,x3,y3,x4,y4,x5,y5,x6,y6))
- Places metal-active contacts centered at the given coordinates.

mp

=>mp(x1,y1,(x2,y2,x3,y3,x4,y4,x5,y5,x6,y6))
- Places metal-poly contacts centered at the given coordinates.

pla

=>pla("bitfile",x,y)
- see appendix 3
place
  =place("cell(.options)",x.y.(cell#))
- Cells are placed with their origins at
  x.y with optional transformations;
  mx XOR my for mirroring about yaxis,
  xaxis respectively; r90, r180, XOR r270
  for rotations about their origins.
  An optional variable can be used to
  record the cells place number for
  use in locn commands.
  eg. place("test.mx.r90",10,10,test)

polygon
  =polygon(x1,y1,(x2.y2,x3.y3,x4.y4,x5.y5,x6.y6))
- Starts a new polygon at x1,y1 on current
  layer. Optional vertices can be specified
  in same command or path continuation commands
  can follow. The polygon automatically closes
  up on itself so the first vertex need not be
  repeated.

wire
  =wire((width),x1.y1,(x2.y2,x3.y3,x4.y4,x5.y5,x6.y6))
- Starts a wire of default width 2 at x1,y1.
  The path of a wire is along its center line
  and starts and ends flush with its end points.
  Any path continuation command can be used.

x
  =$x(x1)
- Continues a path. (x.y)<=(x1.y)

xty
  =$xty(x1,(y1.x2.y2,x3.y3,x4.y4,x5.y5,x6.y6))
- Continues a path with orthogonal bends.
  Performs successive x(xi) y(yi+1) commands.
  Produces a jog in x, then y, then x etc..

xy
  =$xy(x1,y1,(x2.y2,x3.y3,x4.y4,x5.y5,x6.y6))
- Continues a path along given coordinates.
  (x.y)<=(xi,yi)

y
  =$y(y1)
- Continues a path. (x.y)<=(x,y1)
AUTOMATIC PLA GENERATION

A Programmable Logic Array may be synthesized from a truth table via the 'Pla' command. To do so one must first create a file of the truth table. Create a design file under some name (eg. "truthtable") into which the terms of the boolean logic can be put.

To see how this is done follow through with this example.

```
!3,2,5  ! # variables,results,minterms
!001 10 ! term 1
!101 01 ! term 2
!110 11 ! term 3
!x11 00 ! term 4
!111 01 ! term 5
```

Each line must begin with a '!' followed by data. The first line specifies the size of the PLA. Each successive line represents the terms of the truth table. Input conditions followed by outputs. An 'x' specifies that the input variable is ignored. Note! an 'x' can not occur in an output column. There must a single space between the input and output sections with comments allowed anywhere after the outputs.

Once a file has been created it can be used as in.

```
Defc("pl1")
 . !other statements can be
 . !used for custom additions.
Pla("pl1table",x,y)
 .
Endc
```

The pla generator is a system macro that expands a truth table into layout commands within a cell definition.

For each input 'i' there will be defined an associated node "in"sub 'i' ("in1","in2" etc.). Likewise for all outputs ("out1","out2", etc.). This allows easy interfacing to PLAs.
APPENDIX E

Top Level Cell Source Listings
Comments: 3 of Snyder's DACs connected in parallel with 3 of Schmiing's DAC's to the 12 bit pipeline reg

Cell name: pldac

-function . see above comment
-cells called . pipein, dcell, dacrbuf, pvs
-nodes defined . leftp (0, 2) = left pipeline reg clk
     . rightp (1008, 2) = right pipeline reg clk
     . leftp/ (0.82)
     . rightp/ (1008, 82)
     . dacvddl (-31.7, 97) = left end of vdd bus
     . dacvddr (1049.8, 97) = right end of vdd bus
     . dacvssl (-31.7, 494) = left end of vss bus
     . dacvssr (1049.8, 494) = right end of vss bus
     . vref1 (-31.7, 508) = left end of Snyder's voltage reference bus
     . vreflr (1049.8, 508) = right end of Snyder's voltage reference bus
     . vref2l (-31.7, 522) = left end of Schmiing's voltage reference bus
     . vref2r (1049.8, 522) = right end of Schmiing's voltage reference bus
     . clkl (-31.7, 531) = left end of clock bus
     . clkr (1049.8, 531) = right end of clock bus
     . out1 ... out6 = outputs

-bounding box . (-36.1, -1.4, 1049.8, 534)

Revision History: (date/name/mod)

pipeline register and the 6 dacs
DIM Ploutx(12)
DIM Plouty(12)
DIM Vrefx(6)
DIM Vrefy(6)
DIM Outx(6)
DIM Outy(6)
DIM Dacvssx(6)
DIM Dacvssy(6)
DIM Dacvss2x(3)
DIM Dacvss2y(3)
DIM Clkx(3)
DIM Clky(3)
DIM Dacvddx(6)
DIM Dacvddy(6)

Defc("pldac")

FOR I=1 TO 12
  Locn(Pipeln."plout"&VALS(I),Ploutx(I),Plouty(I))
NEXT I

Locn(Pipeln."leftp",Leftpx,Leftpy)
Defn("leftp",Leftpx,Leftpy)

Locn(Pipeln."rightp",Rightpx,Rightpy)
Defn("rightp",Rightpx,Rightpy)

Locn(Pipeln."leftp/",X,Y)
Defn("leftp/",X,Y)

Locn(Pipeln."rightp/",X,Y)
Defn("rightp/",X,Y)

Get bounding box info so we can figure out where to place the dacs

Bbox("dcel",Dcellx1,Dcelly1,Dcellx2,Dcelly2)
Bbox("dacbuf",Dacbufx1,Dacbufy1,Dacbufx2,Dacbufy2)

the bottom of the dacs must not go below Y0

Y0=Plouty(1)+6+12*4+(3*6)+4

We allow 10 lambda space between dacs

Pitch=(Dcellx2-Dcellx1)+(Dacbufx2-Dacbufx1)+(2*10)

Find the left edge X0 such that the 6 dacs are nicely centered

X0=(Rightpx+Leftpx)/2-(Pitch*3)/2+5
1210 XR=X0+3*Pitch ! right edge
1220 !
1230 !******************************************************************************
1240 !
1250 !Vdd bus
1260 !
1270 Layer(Metal)
1280 Wire(12,X0,Plouty(1)+12)
1290 X(Xr)
1300 Defn("dacvdd1",X0,Plouty(1)+12)
1310 Defn("dacvddr",Xr,Plouty(1)+12)
1320 !
1330 ! connect pipeline register to vdd bus
1340 !
1350 Plpitch=168
1360 !
1370 FOR I=0 TO 5
1380 Wire(4,Plpitch/2+I*Plpitch,Plouty(1)-3)
1390 Y(Plouty(1)+12)
1400 NEXT I
1410 !
1420 !******************************************************************************
1430 !
1440 ! place the dac's above the pipeline
1450 ! register and connect
1460 !
1470 DIM DcellK3,Dacbuf(3)
1480 !
1490 FOR I=1 TO 3
1500 ! place Snyder's dac
1510 !
1520 !
1530 Y=Y0-Dcelly1
1540 X=X0+(I-1)*Pitch-Dcelix1
1550 !
1560 Place("dcell",X,Y,Dcell(I))
1570 !
1580 ! get all the dac coordinates for
1590 ! future use
1600 !
1610 Locn(Dcell(I),"vref",Vrefx(I*2-1),Vrefy(I*2-1))
1620 Locn(Dcell(I),"vss",Dacvssx(I*2-1),Dacvssy(I*2-1))
1630 Locn(Dcell(I),"clk",Clkx(I),Clky(I))
1640 Locn(Dcell(I),"vdd",Dacvddx(I*2-1),Dacvddy(I*2-1))
1650 Locn(Dcell(I),"out",Outx(I*2-1),Outy(I*2-1))
1660 !
1670 ! place Schmiing's dac
1680 !
1690 !
1700 Y=Y0-Dacbufy1
1710 !
1720 Place("dacbuf",X,Y,Dacbuf(I))
1730 !
1740 ! get all the dac coordinates for
1750 ! future use
1760 !
1770 Locn(Dacbuf(I),"vref",Vrefx(I*2),Vrefy(I*2))
1780 Locn(Dacbuf(I),"vss1",Dacvssx(I*2),Dacvssy(I*2))
1790 Locn(Dacbuf(I),"vdd",Dacvddx(I*2),Dacvddy(I*2))
1800 Locn(Dacbuf(I),"vout",Outx(I*2),Outy(I*2))
Locn(Dacbuf(I),"vss2",Dacvss2x(I),Dacvss2y(I))

! connect the data lines

FOR J=1 TO 4
  K=(I-1)*4+J
  Mx(Ploutx(K),Plouty(K)+2)
  H=Plouty(K)+22+(J-1)*6
  Mx(Ploutx(K),H)
  Layer(Poly1)
  Wire(Ploutx(K),Plouty(K)+2)
  Y(H)
! Locn(Dcell(I),"in"&VALS(J-1),X,Y)
Layer(Metal)
Wire(Ploutx(K),H)
  X(X)
  Y(Y)
  ! Locn(Dacbuf(I),"in"&VALS(J),X,Y)
Layer(Metal)
Wire(Ploutx(K),H)
  X(X)
  Y(Y)
  ! Locn(Dacbuf(I),"in"&VALS(J),X,Y)
Layer(Metal)
Wire(Ploutx(K),H)
  X(X)
  Y(Y)
NEXT J
! connect the vdd bus to the dacs
Layer(Metal)
Wire(8,Dacvddx(I*2-1),Dacvddy(I*2-1))
  Y(Plouty(1)+12)
Wire(8,Dacvddx(I*2),Dacvddy(I*2))
  Y(Plouty(1)+12)
NEXT I
! busses above the dac

IF Outy(2)>Outy(1) THEN
  Top=Outy(2)
ELSE
  Top=Outy(1)
END IF

Layer(Metal)
Wire(12,X0,Top+12)
  X(Outx(1)-3)
Wire(12,Outx(6)+3,Top+12)

167
2410 X(Xr)
2420 Defn("dacvssl",X0,Top+12)
2430 Defn("dacvssr",Xr,Top+12)
2440 !
2450 FOR I=1 TO 5
2460 Wire(12,Outx(I)+3,Top+12)
2470 X(Outx(I+1)-3)
2480 NEXT I
2490 !
2500 Layer(Active)
2510 FOR I=1 TO 6
2520 Wire(12,Outx(I)-7,Top+12)
2530 X(Outx(I)+7)
2540 Ma(Outx(I)-5,Top+8)
2550 Ma(Outx(I)-5,Top+12)
2560 Ma(Outx(I)-5,Top+16)
2570 Ma(Outx(I)+5,Top+8)
2580 Ma(Outx(I)+5,Top+12)
2590 Ma(Outx(I)+5,Top+16)
2600 NEXT I
2610 !
2620 !*******************************************************************************
2630 !
2640 ! vref1 and vref2
2650 !
2660 Layer(Metal)
2670 Wire(12,X0,Top+26)
2680 X(Outx(1)-3)
2690 Wire(12,Outx(6)+3,Top+26)
2700 X(Xr)
2710 Defn("vref1",X0,Top+26)
2720 Defn("vref1",Xr,Top+26)
2730 !
2740 Layer(Metal)
2750 Wire(12,X0,Top+40)
2760 X(Outx(1)-3)
2770 Wire(12,Outx(6)+3,Top+40)
2780 X(Xr)
2790 Defn("vref2",X0,Top+40)
2800 Defn("vref2",Xr,Top+40)
2810 !
2820 FOR I=1 TO 5
2830 Wire(12,Outx(I)+3,Top+26)
2840 X(Outx(I+1)-3)
2850 !
2860 Wire(12,Outx(I)+3,Top+40)
2870 X(Outx(I+1)-3)
2880 NEXT I
2890 !
2900 Layer(Active)
2910 FOR I=1 TO 6
2920 Wire(12,Outx(I)-7,Top+26)
2930 X(Outx(I)+7)
2940 Ma(Outx(I)-5,Top+22)
2950 Ma(Outx(I)-5,Top+25)
2960 Ma(Outx(I)-5,Top+30)
2970 Ma(Outx(I)+5,Top+22)
2980 Ma(Outx(I)+5,Top+25)
2990 Ma(Outx(I)+5,Top+30)
3000 !
Wire(12, Outx(I)-7, Top+40)
X(Outx(I)+7)
Ma(Outx(I)-5, Top+36)
Ma(Outx(I)-5, Top+40)
Ma(Outx(I)-5, Top+44)
Ma(Outx(I)+5, Top+36)
Ma(Outx(I)+5, Top+40)
Ma(Outx(I)+5, Top+44)

NEXT I

Layer(Poly1)
Wire(2, X0, Top+49)
X(Xr)
Defn("clk1", X0, Top+49)
Defn("clkr", Xr, Top+49)

connect the dacs to the busses
FOR I=1 TO 3

Layer(Poly1)
Wire(2, Clkx(I), Clky(I))
Y(Top+49)

vss2 used by Schmiing's dac
Layer(Metal)
Wire(8, Dacvss2x(I), Dacvss2y(I))
Y(Top+12)

vref1 to Snyder's dac
Layer(Active)
Wire(4, Vrefx(I*2-1), Vrefy(I*2-1))
Y(Top+26)
Ma(Vrefx(I*2-1), Top+22)
Ma(Vrefx(I*2-1), Top+26)
Ma(Vrefx(I*2-1), Top+30)

Layer(Nplus)
Box(Vrefx(I*2-1)-3.5, Vrefy(I*2-1)-3.5, Vrefx(I*2-1)+3.5, Top+30+

vref2 to Schmiing's dac
Layer(Active)
Wire(4, Vrefx(I*2), Vrefy(I*2))
Y(Top+40)
Ma(Vrefx(I*2), Top+36)
Ma(Vrefx(I*2), Top+40)
Ma(Vrefx(I*2), Top+44)

Layer(Nplus)

169
3610 Box(Vrefx(I*2)-3.5,Vrefy(I*2)-3.5,Vrefx(I*2)+3.5,Top+44+3.5)
3620 NEXT I
3630 !
3640 FOR I=1 TO 6
3650 !
3660 ! vss
3670 !
3680 Layer(Metal)
3690 Wire(8,Dacvssx(I),Dacvssy(I))
3700 Y(Top+12)
3710 !
3720 ! output
3730 !
3740 Layer(Metal)
3750 Wire(2,Outx(I),Outy(I))
3760 Y(Top+52)
3770 !
3780 Defn("out"&VAL$(I),Outx(I).Top+52)
3790 !
3800 Layer(Nplus)
3810 Box(Outx(I)-5-3.5,Top+8-3.5,Outx(I)+5+3.5,Top+44+3.5)
3820 NEXT I
3830 !
3840 !******************************************************************************
3850 ! Extend the pwell region
3860 ! around the active bus jumpers
3870 !
3880 FOR I=1 TO 3
3890 !
3900 ! Place the pwell around Snyder's dac
3910 ! and tie to vss
3920 !
3930 Layer(Pwell)
3940 Box(Vrefx(I*2-1)-4,Vrefy(I*2-1)-4,Outx(I*2-1)+5+4,Top+44+4)
3950 !
3960 X=(Outx(I*2-1)-5+Vrefx(I*2-1))/2
3970 Y0=Outy(I*2-1)+10
3980 !
3990 Layer(Metal)
4000 Wire(4,X,Top+12)
4010 Y(Y0)
4020 !
4030 Cnt=INT((Top+12-Y0)/10)
4040 FOR J=1 TO Cnt
4050 ! Place("pvs",X,Y0+2+(J-1)*10)
4060 NEXT J
4070 !
4080 ! Place the pwell around Schmiing's
4090 ! dac and tie to vss
4100 !
4110 Layer(Pwell)
4120 Box(Outx(I*2)-5-4,Outx(I*2)-4,Vrefx(I*2)+4,Top+44+4)
4130 !
4140 X=(Outx(I*2)+5+Vrefx(I*2))/2
4150 ! Place("pvs",X,Top+12)
4160 !
4170 NEXT I
4180 !
4190 ENDc
4200 END
Filename: mempldac

Designer: R. Mielcarski    Date: 4-May-83

Comments: The pipeline and DACs are connected to the memory array.

Cell name: mempldac

-function . see above comment
-cells called . pldac, array
-nodes defined . dacvddl, dacvddd, dacvssl, dacvssr, vrefl, vref1r, vref2l, vref2r, clk1, clk1r, leftp, rightp, leftp/, rightp/, vss1 ... vss7, dp1 ... dp12, dv1 ... dv12, rl ... r6, r/l ... r/l6, vdd1 ... vdd6, w1 ... w6, w/l ... w/l6, yp1 ... yp16, yp/l ... yp/l6, yv1 ... yv16, yv/l ... yv/l6 omitted because of compiler overflow.

-bounding box . (-36.1,0,1049.8,2070)

Revision History: (date/name/mod)

Defc("mempldac")

 DIM Outx(6)
 DIM Outy(6)
FOR I = 1 TO 6
   Locn(Pldac, "out" & VAL$(I), Outx(I), Outy(I))
   Defn("out" & VAL$(I), Outx(I), Outy(I))
NEXT I
!
Locn(Pldac, "dacvddl", Dacvddl.x, Dacvddl.y)
Defn("dacvddl", Dacvddl.x, Dacvddl.y)
!
Locn(Pldac, "dacvddr", Dacvddrx, Dacvddry)
Defn("dacvddr", Dacvddrx, Dacvddry)
!
Locn(Pldac, "dacvssl", Dacvsslx, Dacvssly)
Defn("dacvssl", Dacvsslx, Dacvssly)
!
Locn(Pldac, "dacvssr", Dacvssrx, Dacvssry)
Defn("dacvssr", Dacvssrx, Dacvssry)
!
Locn(Pldac, "vref1l", Vrefllx, Vreflly)
Defn("vref1l", Vrefllx, Vreflly)
!
Locn(Pldac, "vref1r", Vref1rx, Vref1ry)
Defn("vref1r", Vref1rx, Vref1ry)
!
Locn(Pldac, "vref21", Vref2lx, Vref2ly)
Defn("vref21", Vref2lx, Vref2ly)
!
Locn(Pldac, "vref2r", Vref2rx, Vref2ry)
Defn("vref2r", Vref2rx, Vref2ry)
!
Locn(Pldac, "clk", Clklx, Clkly)
Defn("clk", Clklx, Clkly)
!
Locn(Pldac, "clk", Clkrx, Clkry)
Defn("clk", Clkrx, Clkry)
!
! Carry the pipeline register nodes through to the next level
!
Locn(Pldac, "leftp", Leftpx, Leftpy)
Defn("leftp", Leftpx, Leftpy)
!
Locn(Pldac, "rightp", Rightpx, Rightpy)
Defn("rightp", Rightpx, Rightpy)
!
Locn(Pldac, "leftp/", Leftpnx, Leftpny)
Defn("leftp/", Leftpnx, Leftpny)
!
Locn(Pldac, "rightp/", Rightpnx, Rightpny)
Defn("rightp/", Rightpnx, Rightpny)
!
! Carry memory array nodes through to the next level
!
DIM Vssx(7)
DIM Vssy(7)
DIM Dpx(12)
DIM Dpy(12)
DIM Dvx(12)
DIM Dvy(12)
DIM Rx(6)
1210 DIM Ry(6)
1220 DIM Rnx(6)
1230 DIM Rny(6)
1240 DIM Vddx(6)
1250 DIM Vddy(6)
1260 DIM Wx(6)
1270 DIM Wy(6)
1280 DIM Wnx(6)
1290 DIM Wny(6)
1300 DIM Ypx(16)
1310 DIM Ypy(16)
1320 DIM Ypnx(16)
1330 DIM Ypny(16)
1340 DIM Yvx(16)
1350 DIM Yvy(16)
1360 DIM Yvnx(16)
1370 DIM Yvny(16)
1380 FOR I=1 TO 7
1390 Locn(Mem,"vss"&VAL$(I),Vssx(I),Vssy(I))
1400 Defn("vss"&VAL$(I),Vssx(I),0)
1410 NEXT I
1420 FOR I=1 TO 12
1430 Locn(Mem,"dp"&VAL$(I),Dpx(I),Dpy(I))
1440 Defn("dp"&VAL$(I),Dpx(I),0)
1450 Locn(Mem,"dv"&VAL$(I),Dvx(I),Dvy(I))
1460 Defn("dv"&VAL$(I),Dvx(I),0)
1470 NEXT I
1480 FOR I=1 TO 6
1490 Locn(Mem,"r"&VAL$(I),Rx(I),Ry(I))
1500 Defn("r"&VAL$(I),Rx(I),0)
1510 Locn(Mem,"r/"&VAL$(I),Rnx(I),Rny(I))
1520 Defn("r/"&VAL$(I),Rnx(I),0)
1530 Locn(Mem,"vdd"&VAL$(I),Vddx(I),Vddy(I))
1540 Defn("vdd"&VAL$(I),Vddx(I),0)
1550 Locn(Mem,"w"&VAL$(I),Wx(I),Wy(I))
1560 Defn("w"&VAL$(I),Wx(I),0)
1570 Locn(Mem,"w/"&VAL$(I),Wnx(I),Wny(I))
1580 Defn("w/"&VAL$(I),Wnx(I),0)
1590 NEXT I
1600 Xmax=Vssx(7)
1610 FOR I=1 TO 16
1620 Locn(Mem,"yp"&VAL$(I),Ypx(I),Ypy(I))
1630 Defn("yp"&VAL$(17-I),0,Ypy(I))
1640 NEXT I
1650 NEXT I
1660 FOR I=1 TO 16
1670 Locn(Mem,"yp/"&VAL$(I),Ypnx(I),Ypny(I))
1680 Defn("yp/"&VAL$(17-I),0,Ypny(I))
1690 NEXT I
1700 FOR I=1 TO 16
1710 Locn(Mem,"yy"&VAL$(I),Yvx(I),Yvy(I))
1720 Defn("yy"&VAL$(17-I),Xmax,Yvy(I))
1730 NEXT I
1740 FOR I=1 TO 16
1750 Locn(Mem,"yy/"&VAL$(I),Yvnx(I),Yvny(I))
1760 Defn("yy/"&VAL$(17-I),Xmax,Yvny(I))
1770 NEXT I
1780 ! yy/ cannot be def noded because too many nodes bombs the compiler.
```
1810 NEXT I
1820 Endc
1830 !
1840 PRINT "***************"
1850 Bbox("mempldac",X1,Y1,X2,Y2)
1860 PRINT "mempldac bounding box=",X1,Y1,X2,Y2
1870 PRINT "***************"
1880 PRINT "dacvddl x,y=".Dacvddlx,DacvddlY
1890 PRINT "dacvddr x,y=".Dacvddrx,Dacvddry
1900 PRINT "***************"
1910 PRINT "dacvssl x,y=".Dacvsslx,Dacvssly
1920 PRINT "dacvssr x,y=".Dacvssrx,Dacvssry
1930 PRINT "***************"
1940 PRINT "out x="
1950 PRINT Outx(+)
1960 PRINT "y="
1970 PRINT Outy(+)
1980 PRINT "***************"
1990 PRINT "vref11 x,y=".Vref11x,Vref11y
2000 PRINT "vref1r x,y=".Vref1rx,Vref1ry
2010 PRINT "vref21 x,y=".Vref21x,Vref21y
2020 PRINT "vref2r x,y=".Vref2rx,Vref2ry
2030 PRINT "***************"
2040 PRINT "clk1 x,y=".Clk1x,Clk1y
2050 PRINT "clk1r x,y=".Clk1rx,Clk1ry
2060 PRINT "***************"
2070 PRINT "leftp x,y=".Leftpx,Leftpy
2080 PRINT "***************"
2090 PRINT "rightp x,y=".Rightpx,Rightpy
2100 PRINT "***************"
2110 PRINT "vss x="
2120 PRINT Vssx(*)
2130 PRINT "y="
2140 PRINT Vssy(*)
2150 PRINT "***************"
2160 PRINT "vdd x="
2170 PRINT Vddx(*)
2180 PRINT "y="
2190 PRINT Vddy(*)
2200 PRINT "***************"
2210 PRINT "dp x="
2220 PRINT Dpx(*)
2230 PRINT "y="
2240 PRINT Dpy(*)
2250 PRINT "***************"
2260 PRINT "dv x="
2270 PRINT Dvx(*)
2280 PRINT "y="
2290 PRINT Dvy(*)
2300 PRINT "***************"
2310 PRINT "r x=
2320 PRINT Rx(*)
2330 PRINT "y=
2340 PRINT Ry(*)
2350 PRINT "***************"
175```
PRINT "r/ x ="
PRINT Rnx(*)
PRINT " y ="
PRINT Rny(*)
PRINT "********************"
PRINT Wx(*)
PRINT " y ="
PRINT Wy(*)
PRINT "********************"
PRINT Wnx(*)
PRINT " y ="
PRINT Wny(*)
PRINT "********************"
PRINT Wpx(*)
PRINT " y ="
PRINT Wpy(*)
PRINT "********************"
PRINT Wpnx(*)
PRINT " y ="
PRINT Wpny(*)
PRINT "********************"
PRINT Wpx(Xmax)
PRINT " y ="
PRINT Wpy(Xmax)
PRINT "********************"
mempldac has been compiled

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacvdlx,y =</td>
<td>-31.7</td>
<td>1633</td>
</tr>
<tr>
<td>dacvdrx,y =</td>
<td>1049.8</td>
<td>1633</td>
</tr>
<tr>
<td>dacvsslx,y =</td>
<td>-31.7</td>
<td>2030</td>
</tr>
<tr>
<td>dacvssrx,y =</td>
<td>1049.8</td>
<td>2030</td>
</tr>
<tr>
<td>out x =</td>
<td>73.8</td>
<td>232.3</td>
</tr>
<tr>
<td></td>
<td>2070</td>
<td>2070</td>
</tr>
<tr>
<td>vrefflx,y =</td>
<td>-31.7</td>
<td>2044</td>
</tr>
<tr>
<td>vrefflx,y =</td>
<td>1049.8</td>
<td>2044</td>
</tr>
<tr>
<td>vreff2lx,y =</td>
<td>-31.7</td>
<td>2058</td>
</tr>
<tr>
<td>vreff2lx,y =</td>
<td>1049.8</td>
<td>2058</td>
</tr>
<tr>
<td>clklx,y =</td>
<td>-31.7</td>
<td>2058</td>
</tr>
<tr>
<td>clkrx,y =</td>
<td>1049.8</td>
<td>2067</td>
</tr>
<tr>
<td>leftpx,y =</td>
<td>0</td>
<td>1536</td>
</tr>
<tr>
<td>leftpx,y =</td>
<td>0</td>
<td>1618</td>
</tr>
<tr>
<td>rightpx,y =</td>
<td>1008</td>
<td>1538</td>
</tr>
<tr>
<td>rightpx,y =</td>
<td>1008</td>
<td>1618</td>
</tr>
<tr>
<td>vsslx,y =</td>
<td>0</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>vddx,y =</td>
<td>84</td>
<td>252</td>
</tr>
<tr>
<td></td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>dpzx,y =</td>
<td>11</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>683</td>
<td>829</td>
</tr>
<tr>
<td></td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>dvy,x,y =</td>
<td>6</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>678</td>
<td>834</td>
</tr>
<tr>
<td></td>
<td>1536</td>
<td>1536</td>
</tr>
<tr>
<td>rrx,y =</td>
<td>95</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>1536</td>
<td>1536</td>
</tr>
</tbody>
</table>

177
<table>
<thead>
<tr>
<th>$r/ x$</th>
<th>246</th>
<th>426</th>
<th>582</th>
<th>762</th>
<th>918</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>36</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$u/ x$</th>
<th>258</th>
<th>414</th>
<th>594</th>
<th>750</th>
<th>930</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$v/ x$</th>
<th>263</th>
<th>409</th>
<th>599</th>
<th>745</th>
<th>935</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$w/ x$</th>
<th>263</th>
<th>409</th>
<th>599</th>
<th>745</th>
<th>935</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
<td>1536</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$wp/ x$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$wp/ x$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$wp/ x$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$w/ x$</th>
<th>1508</th>
<th>1412</th>
<th>1316</th>
<th>1220</th>
<th>1124</th>
<th>1028</th>
<th>932</th>
<th>836</th>
<th>740</th>
<th>644</th>
<th>548</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>740</td>
<td>644</td>
<td>548</td>
<td>452</td>
<td>356</td>
<td>260</td>
<td>164</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$w/ x$</th>
<th>1442</th>
<th>1346</th>
<th>1250</th>
<th>1154</th>
<th>1058</th>
<th>962</th>
<th>866</th>
<th>770</th>
<th>674</th>
<th>578</th>
<th>482</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>674</td>
<td>578</td>
<td>482</td>
<td>386</td>
<td>290</td>
<td>194</td>
<td>98</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$wp/ x$</th>
<th>1466</th>
<th>1370</th>
<th>1274</th>
<th>1178</th>
<th>1082</th>
<th>986</th>
<th>890</th>
<th>794</th>
<th>698</th>
<th>602</th>
<th>506</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>698</td>
<td>602</td>
<td>506</td>
<td>410</td>
<td>314</td>
<td>218</td>
<td>122</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- not all nodes are labelled
Cell name: palette

-function . see above comments

-cells called mempldac, bigxdocod,
iopad, newvgen,
newyvgen, newctrl,
inpad, logo.

-nodes defined . none

-bounding box . (-353.389,1275,2170)

Revision History: (date/name/mod)

Colour palette chip containing the dacs, the pipeline register, the memory array, the x and y decoders, the control circuitry, and the I/O pads.

Defc("palette")

Place the dacs, pipeline register and memory array.

Place("mempidac",0.0,Mem)

! DAC nodes

DIM Outx(6)
DIM Outy(6)
DIM Ctrlx1(8),Ctrlly1(8)
DIM Ctrlx4(8),Ctrlly4(8)

! FOR I=1 TO 6
Locn(Mem,"out"&VAL$(I),Outx(I),Outy(I))

NEXT I
Locn(Mem, "dacvdd", Dacvddx, Dacvddy)
Locn(Mem, "dacvdd", Dacvddrx, Dacvddry)
Locn(Mem, "dacvssl", Dacvsslx, Dacvssly)
Locn(Mem, "dacvssr", Dacvssrx, Dacvssry)
Locn(Mem, "vref1", Vref1lx, Vref1ly)
Locn(Mem, "vref1", Vref1rx, Vref1ry)
Locn(Mem, "vref2", Vref2lx, Vref2ly)
Locn(Mem, "vref2", Vref2rx, Vref2ry)
Locn(Mem, "clkl", Clklx, Clkly)
Locn(Mem, "clkr", Clkrx, Clkry)
Pipeline register nodes
Locn(Mem, "leftp", Leftpx, Leftpy)
Locn(Mem, "rightp", Rightpx, Rightpy)
Locn(Mem, "leftp", Leftpx, Leftpy)
Locn(Mem, "rightp", Rightpx, Rightpy)
Place the x-decoder
Place("bigxdcod", 168, -81, Xdecod)
DIM Rdpx(4), Rdpy(4)
DIM Gdpx(4), Gdpy(4)
DIM Bdpx(4), Bdpy(4)
FOR I=0 TO 3
Locn(Xdecod, "rdp" & VAL$(I), Rdpx(I+1), Rdpy(I+1))
Locn(Xdecod, "gdp" & VAL$(I), Gdpx(I+1), Gdpy(I+1))
Locn(Xdecod, "bdp" & VAL$(I), Bdpx(I+1), Bdpy(I+1))
NEXT I
Locn(Xdecod, "xsigleft", Xsigx, Xsigy)
Locn(Xdecod, "btmleft", Btmleftx, Btmlefty)
Locn(Xdecod, "toprght", Toprghtx, Toprghty)
Check if file updated correctly
IF Btmlefty <> Rdpy(1) THEN
BEEP
PRINT "node error"
END IF
Vss bus
Pitch=168
Btrightx=Btmleftx+(Pitch+6)

Layer(Metal)
Wire(12,Btmleftx-2,Btmlefty-10)
X(Btrightx+2)

Connect Vss bus to the xdecoder
FOR I=0 TO 6
   Wire(4,Btmleftx+(Pitch*I),Btmlefty)
   Dy(-10)
NEXT I

Processor data busses
DIM Datay(4)
FOR I=1 TO 4
   Datay(I)=Btmlefty-20-(I-1)*6
   ! Horizontal metal busses
   Layer(Metal)
   Wire(2,Btmleftx-2,Datay(I))
   X(Btrightx+2)
   ! Poly jumpers over the Vss bus
   Layer(Poly1)
   Mp(Rdpx(I),Rdpy(I),Rdpx(I),Datay(I))
   Wire(2,Rdpx(I),Rdpy(I))
   Y(Datay(I))
   Mp(Gdpx(I),Gdpy(I),Gdpx(I),Datay(I))
   Wire(2,Gdpx(I),Gdpy(I))
   Y(Datay(I))
   Mp(Bdpx(I),Bdpy(I),Bdpx(I),Datay(I))
   Wire(2,Bdpx(I),Bdpy(I))
   Y(Datay(I))
NEXT I

input enable busses
input enable /

Note that the length of the i/o pads is 800
1810 Ennleftx=Brightx+2-800
1820 Ennlefty=Datay(4)-6
1830 !
1840 Layer(Metal)
1850 Wire(2,Ennleftx,Ennlefty)
1860 X(Brightx+2)
1870 ! enable
1880 !
1890 Enleftx=Ennleftx
1900 Enlefty=Ennlefty-6
1910 !
1920 Layer(Metal)
1930 Wire(2,Enleftx,Enlefty)
1940 X(Brightx+2)
1950 !
1960 !
1970 !
1980 ! Place the processor I/O pads
1990 !
2000 Iovssy=Ennlefty-10
2010 Iovdyy=Iovssy-113
2020 !
2030 Place(\"iopad.mymy\",Enleftx,Iovssy,Iopad)
2040 ! Connect the en, en/, in and out/ signals to their respective busses
2050 !
2060 !
2070 !
2080 FOR I=1 TO 4
2090 !
2100 ! enable
2110 !
2120 Locn(Iopad,\"en\"&VAL$(I),x,y)
2130 Mp(x,Ennlefty)
2140 Layer(Poly1)
2150 Wire(2,x,y)
2160 Y(Enlefty)
2170 !
2180 ! enable /
2190 !
2200 Locn(Iopad,\"en/\"&VAL$(I),x,y)
2210 Mp(x,Ennlefty)
2220 Layer(Poly1)
2230 Wire(2,x,y)
2240 Y(Enlefty)
2250 !
2260 ! input
2270 !
2280 Locn(Iopad,\"in\"&VAL$(I),x,y)
2290 Mp(x,Datay(I))
2300 Layer(Poly1)
2310 Wire(2,x,y)
2320 Y(Datay(I))
2330 !
2340 ! output
2350 !
2360 Locn(Iopad,\"out/\"&VAL$(I),x,y)
2370 Mp(x,Datay(I))
2380 Layer(Poly1)
2390 Wire(2,x,y)
2400
Y(Datay(I))

NEXT I

Place("newypgen",-129,0,Ypgen)

DIM Ainpx(4), Ainpy(4)

FOR I=1 TO 4
    Locn(Ypgen."ain"&VAL$(I), Ainpx(I), Ainpy(I))
NEXT I

Locn(Ypgen."vdd", Ypvddx, Ypvddy)
Locn(Ypgen."vss", Ypvssx, Ypvssy)

Connect VSS to newypgen

Layer(Metal)
Wire(4, Ypvssx, Ypvssy)
X(Btmleftx)

Connect Vdd to newypgen

Layer(Metal)
Wire(4, Ypvddx, Ypvddy)

Y(Dacvddly)

Wire(12, Dacvddx, Dacvddiy)
X(Ypvddx-2)

Place("newyvgen",1017,0,Yvgen)

DIM Ainvx(4), Ainvy(4)

FOR I=1 TO 4
    Locn(Yvgen."ain"&VAL$(I), Ainvx(I), Ainvy(I))
NEXT I

Locn(Yvgen."vdd", Yvvddx, Yvvddy)
Locn(Yvgen."vss", Yvvssx, Yvvssy)

Connect VSS to newyvgen

Layer(Metal)
Wire(4, Yvvssx, Yvvssy)
X(Toprghtx)

Connect Vdd to newyvgen

Layer(Metal)
Wire(4, Yvvddx, Yvvddy)
Place the control signal generator

The top right hand corner of 'newctrl'
is located at (117.37) relative to its origin.

We allow an x-spacing of 40 from the left hand edge of the x-decoder to route the poly busses.

We allow a 5 lambda spacing from the bottom of the y-decoder.

Newctrlx=0-40-117
Newctry=Ypvssy-5-37
Place("newctrl",Newctrlx,Newctry,Ctrl)

Locn(Ctrl,"ctrlvdd",Ctrlvddx,Ctrlvddy)
Locn(Ctrl,"ctrlvss",Ctrlvssx,Ctrlvssy)
Locn(Ctrl,"cs",Csx,Csy)
Locn(Ctrl,"rw",Rwx,Rwy)
Locn(Ctrl,"e",Ex,Ey)
Locn(Ctrl,"a4",A4x,A4y)
Locn(Ctrl,"a5",A5x,A5y)
Locn(Ctrl,"busbt",Busbtmx,Busbtmy)
Locn(Ctrl,"ctrlrgt",Ctrlrgtx,Ctrlrgty)
Locn(Ctrl,"tri",Trix,Tny)
Locn(Ctrl,"triout",Trioutx,Triouty)
Locn(Ctrl,"trioutb",Trioutbx,Trioutby)

Connect control bus to x decoder

Wsep=4
Layer(Poly1)

FOR I=1 TO 8
Locn(Ctrl,"xdbcus"&VAL$(I),X1,Y1)
X1=X1
Y1=Y1
X2=X1+(9-I)*Wsep
Y2=Y1
X3=X2
Y3=Xsigy+(I-1)*(Wsep+1)
X4=Xsigx
Y4=Y3
Wire(2,X1,Y1)
X(X2)
Y(Y3)
X(X4)
Ctrlx1(I)=X1
Ctrlx1(I)=Y1
3610  Ctrlx2(I)=X2
3620  CtrlY2(I)=Y2
3630  Ctrlx3(I)=X3
3640  CtrlY3(I)=Y3
3650  Ctrlx4(I)=X4
3660  CtrlY4(I)=Y4
3670  NEXT I
3680  !
3690  !*************************************************************************!
3700  ! Connect the input pad enable and enable/ signals
3710  ! en/
3720  !
3730  ! Layer(Metal)
3740  Wire(2,Trioutbx,Trioutby)
3750  X(Btmleftx)
3760  Y(Ennlefty)
3770  X(Ennleftx)
3780  !
3790  ! en
3800  !
3810  ! Layer(Metal)
3820  Wire(2,Trioutx,Triouty)
3830  X(Btmleftx+4)
3840  Y(Enlefty)
3850  X(Enleftx)
3860  !
3870  !**************************************************************************
3880  ! Connect power and ground to the control circuitry (newctrl)
3890  !
3900  ! Vss
3910  !
3920  ! Layer(Metal)
3930  Wire(4,Ctrlvssx,Ctrlvssy)
3940  Y(Ctrlvssy-4)
3950  X(Btmleftx+9)
3960  Y(Iovssy) ! Vss rail of i/o pads
3970  X(Enleftx)
3980  !
3990  ! Vdd
4000  !
4010  ! Layer(Metal)
4020  Wire(4,Ctrlvddx,Ctrlvddy)
4030  Y(Ctrlvddy-4-6)
4040  X(Btmleftx+9+6)
4050  Y(Iovddy)
4060  X(Enleftx)
4070  !
4080  !**************************************************************************
4090  ! Place and route the protected input pads
4100  !
4110  DIM Inpad(17) ! we require 17
4120  DIM Padoutx(17) ! input pads
4130  DIM Padouty(17)
Right hand bus

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

X(pluckx(5))
X(pluckx(6))
X(y+10)

M(2-Pluckx,Pluckx)
layer Polly
*/

Route p

X(pluckx(5))
X(pluckx(6))
X(y+10)

M(2-Pluckx,Pluckx)
layer Polly

for i=1 to 2

y0=decadary-z*padptic

pipeline register p and p

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

X(pluckx(I))

X(pluckx(I))

X(pluckx(I))

M(2,Pluckx(I),AnyY(I))
layer Polly

X(Anyx(I)+r+rpad)
layer Polly

M(2-Pluckx,Anyx(2)+r-pad)

for i=1 to 4

y0=Anyx(3)+Anyy(2)+z-pad

video decoder inputs

def: pads along the right hand side

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

busoffst=13 i distance from input node

padoptic=100

r=1
4810  !
4820  Layer(Metal)
4830  Wire(12,Btrightx+2,1ovssy)
   X(X)
4850  Y(Vsstop)
4860  ! Jumper to join the iopad vss bus
4870  ! with the x decoder vss bus
4880  !
4890  !
4900  Layer(Metal)
4910  Wire(12,Btrightx+2,Btmlefty-10)
4920  Dx(10)
4930  Y(Iovssy)
4940  ! Vss pad
4950  !
4960  !
4970  Layer(Metal)
4980  Box(X-6,1ovssy+6-50,X-6+50,1ovssy+6)
4990  !
5000  !********************
5010  !
5020  ! Right hand vdd bus
5030  !
5040  ! Layer(Metal)
5050  Wire(12,Btrightx+2,1ovddy)
5060  X(X+113)  ! distance between vss
5070  Y(Dacvddry)  ! and vdd of pads is 113
5080  X(Dacvddrx)
5090  !
5100  !
5110  !
5120  Layer(Metal)
5130  Box(X+113+6-50,Dacvddry+6-50,X+113+6,Dacvddry+6)
5140  !
5150  !********************
5160  !
5170  ! dac voltage reference pads
5180  !
5190  ! Layer(Metal)
5200  Wire(12,Vref2rx,Vref2ry)
5210  X(X+113+6-50)
5220  Box(X+113+6-50,Vref2ry-25,X+113+6,Vref2ry+25)
5230  !
5240  ! Layer(Metal)
5250  Wire(12,Vref1rx,Vref1ry)
5260  X(X)
5270  Y(Vref2ry-Padpitch)
5280  X(X+113+6-50)
5290  Box(X+113+6-50,Vref2ry-Padpitch-25,X+113+6,Vref2ry-Padpitch+25)
5300  !********************
5310  !
5320  ! Place the logo in the gap between
5330  ! the dac voltage reference pad and
5340  ! the dacvdd rail.
5350  !
5360  X0=X+113
5370  Xh=X0-Dacvddrx
5380  Y0=Dacvddry
5390  Yh=Vref2ry-Padpitch-Y0
5400  !
Bbox("logo",Logox1,Logoy1,Logox2,Logoy2)

X=X0-Xh/2+(Logox2-Logoy1)/2+Logoy1
Y=Y0+Yh/2-(Logox2-Logoy1)/2-Logoy1

Place("logo.r90",X,Y)

Pads along the left hand side

Bbox("newctrl",Bbx,Byy,Bbrx,Bbry)
X=Newctrlx+Bbx ! left edge of the
! control generator
X=X-26-Busoffset ! space for 6 lines

Input signals tri,a4,a5,e,ru,cs
and the processor y decoder inputs
Y=Iovddly
FOR I=1 TO 10
Y=Y0+(I-1)*Padpitch
Place("inpad.r90",X,Y,Inpad(I+6))
Locn(Inpad(I+6),"out",Padoutx(I+6),Padouty(I+6))
NEXT I
Ystop=Y+(2*Padpitch)
Route tri
Layer(Metal)
Wire(2,Trix,Triy)
X(Padoutx(7)+3)
Y(Padouty(7))
X(Padoutx(7))

Route a5

Layer(Metal)
Wire(2,A5x,A5y)
X(Padoutx(8)+7)
Y(Padouty(8))
X(Padoutx(8))

Route a4

Layer(Metal)
Wire(2,A4x,A4y)
X(Padoutx(9)+11)
Y(Padouty(9))
X(Padoutx(9))

Route e

Layer(Metal)
Wire(2,Ex,Ey)
X(Padoutx(10)+15)
6010 \hspace{1cm} \text{Y}(\text{Padouty}(11))
6020 \hspace{1cm} \text{X}(\text{Padoutx}(10))
6030 \hspace{1cm} !
6040 \hspace{1cm} \text{E} \hspace{0.5cm} 0
6050 \hspace{1cm} ! \text{Route rw}
6060 \hspace{1cm} \text{Layer(Metal)}
6070 \hspace{1cm} \text{Wire}(2, \text{Rwx}, \text{Rwy})
6080 \hspace{1cm} \text{X}(\text{Padoutx}(11)+19)
6090 \hspace{1cm} \text{Y}(\text{Padouty}(11))
6100 \hspace{1cm} \text{X}(\text{Padoutx}(11))
6110 \hspace{1cm} !
6120 \hspace{1cm} ! \text{Route cs}
6130 \hspace{1cm} !
6140 \hspace{1cm} \text{Layer(Metal)}
6150 \hspace{1cm} \text{Wire}(2, \text{Csx}, \text{Csy})
6160 \hspace{1cm} \text{X}(\text{Padoutx}(12)+23)
6170 \hspace{1cm} \text{Y}(\text{Padouty}(12))
6180 \hspace{1cm} \text{X}(\text{Padoutx}(12))
6190 \hspace{1cm} !
6200 \hspace{1cm} ! \text{Route processor y decoder inputs}
6210 \hspace{1cm} !
6220 \hspace{1cm} \text{Layer(Metal)}
6230 \hspace{1cm} \text{FOR I}=1 \text{ TO } 4
6240 \hspace{1cm} \text{Mp}(\text{Ainpx}(I)-2, \text{Ainpy}(I))
6250 \hspace{1cm} \text{Wire}(2, \text{Ainpx}(I)-2, \text{Ainpy}(I))
6260 \hspace{1cm} \text{Dx} (-5-(4-I)*4)
6270 \hspace{1cm} \text{Y}(\text{Padouty}(I+12))
6280 \hspace{1cm} \text{X}(\text{Padoutx}(I+12))
6290 \hspace{1cm} \text{NEXT I}
6300 \hspace{1cm} !
6310 \hspace{1cm} ! \text{****************************}
6320 \hspace{1cm} !
6330 \hspace{1cm} ! \text{dac clock input}
6340 \hspace{1cm} !
6350 \hspace{1cm} \text{Y}=\text{Dacvssly}-2*\text{Padpitch}
6360 \hspace{1cm} \text{Place}(\text{"inpad,r90"}, \text{X, Y, Inpad}(17))
6370 \hspace{1cm} \text{Locn(Inpad}(17), \text{"out"}, \text{Padoutx}(17), \text{Padouty}(17))
6380 \hspace{1cm} !
6390 \hspace{1cm} \text{Layer(Poly1)}
6400 \hspace{1cm} \text{Wire}(2, \text{Clklx}, \text{Clkly})
6410 \hspace{1cm} \text{X}(\text{Padoutx}(17)+7)
6420 \hspace{1cm} \text{Y}(\text{Padouty}(17))
6430 \hspace{1cm} \text{X}(\text{Padoutx}(17))
6440 \hspace{1cm} !
6450 \hspace{1cm} ! \text{****************************}
6460 \hspace{1cm} !
6470 \hspace{1cm} ! \text{Left hand Vss bus}
6480 \hspace{1cm} !
6490 \hspace{1cm} \text{Layer(Metal)}
6500 \hspace{1cm} \text{Wire}(12, \text{X}, \text{Iovddy})
6510 \hspace{1cm} \text{Y}(\text{Vsstop})
6520 \hspace{1cm} !
6530 \hspace{1cm} ! \text{Vss pad}
6540 \hspace{1cm} !
6550 \hspace{1cm} \text{Layer(Metal)}
6560 \hspace{1cm} \text{Box}(\text{X}+6-50, \text{Vsstop}-50, \text{X}+6, \text{Vsstop})
6570 \hspace{1cm} ! \text{Box}(\text{X}+6-50, \text{Iovssy}+6-50, \text{X}+6, \text{Iovssy}+6)
6580 \hspace{1cm} !
6590 \hspace{1cm} ! \text{****************************}
6600 \hspace{1cm}!
! Left hand vdd bus

Layer(Metal)
Wire(12,X-113,iovddly)
Y(Y)

Layer(Metal)
Wire(12,X-113,Dacvddly)
X(Dacvddlly)

Layer(Metal)
Wire(12,X-113,Dacvssly)
X(X-113-6+50)
Box(X-113-6,Dacvssly-25,X-113-6+50,Dacvssly+25)

! Vdd pad (This is not really needed
! because the vdd bus extends over to
! to right hand side)

Layer(Metal)
Box(X-113-6,Dacvddly+6-50,X-113-6+50,Dacvddly+6)

Layer(Metal)
Box(X-113-6,Dacvssly-25,X-113-6+50,Dacvssly+25)

! connect vss to the clk input pad

Layer(Metal)
Wire(12,X,Dacvssly)
Y(Dacvssly-Padpitch)

! Place and route the dac output pads

Layer(Metal)
FOR I=1 TO 6
Wire(2,Outx(I),Outy(I))
Y(Outy(I)+50)
Box(Outx(I)-25,Outy(I)+50,Outx(I)+25,Outy(I)+50+50)
NEXT I

! Place and route the dac output pads

Layer(Metal)
FOR I=1 TO 6

PRINT "******"
7210 PRINT "vref21 x,y=".Vref21x,Vref21y
7220 PRINT "vref2r x,y=".Vref2rx,Vref2ry
7230 PRINT "***********"  
7240 PRINT "clk1 x,y=".Clk1x,Clk1y
7250 PRINT "clk2 x,y=".Clk2x,Clk2y
7260 PRINT "***********"  
7270 PRINT "leftp x,y=".Leftpx,Leftpy
7280 PRINT "***********"  
7290 PRINT "leftp/ x,y=".Leftpnx,Leftpny
7300 PRINT "***********"  
7310 PRINT "rightp x,y=".Rightpx,Rightpy
7320 PRINT "***********"  
7330 PRINT "rightp/ x,y=".Rightpnx,Rightpny
7340 PRINT "***********"  
7350 PRINT "rdp x=".Rdpx(*)
7360 PRINT "  y=".Rdpy(*)
7370 PRINT "***********"  
7380 PRINT "gdp x=".Gdpx(*)
7390 PRINT "  y=".Gdpy(*)
7400 PRINT "***********"  
7410 PRINT "bdp x=".Bdpx(*)
7420 PRINT "  y=".Bdpy(*)
7430 PRINT "***********"  
7440 PRINT "ainv x=".Ainvx(*)
7450 PRINT "  y=".Ainvy(*)
7460 PRINT "***********"  
7470 PRINT "ainp x=".Ainpx(*)
7480 PRINT "  y=".Ainpy(*)
7490 PRINT "***********"  
7500 PRINT "cs x,y=".Cex,Csy
7510 PRINT "rw x,y=".Rwx,Rwy
7520 PRINT "e x,y=".Ex,Ey
7530 PRINT "a4 x,y=".A4x,A4y
7540 PRINT "a5 x,y=".A5x,A5y
7550 PRINT "***********"  
7560 PRINT "btmleft (vss) x,y=".Btmleftx,Btmlefty
7570 PRINT "***********"  
7580 PRINT "xsigleft x,y=".Xsigx,Xsigy
7590 PRINT "***********"  
7600 PRINT "ctrlrgt x,y=".Ctrlrgtx,Ctrlrgty
7610 PRINT "***********"  
7620 PRINT "iovddr x,y=".Btrightx,Iovddy
7630 PRINT "iovddl x,y=".Enleftx,Iovddy
7640 PRINT "iovssl x,y=".Enleftx,Iovssy
7650 PRINT "iovssr x,y=".Btrightx,Iovssy
7660 PRINT "***********"  
7670 PRINT "ctrlvss x,y=".Ctrlvssx,Ctrlvssy
7680 PRINT "ctrlvdd x,y=".Ctrlvddx,Ctrlvddy
7690 PRINT "***********"  
7700 PRINT "padout x=".Padoutx(*)
7710 PRINT "  y=".Padouty(*)
7720 PRINT "***********"  
7730 ! FOR I=1 TO 8
7740 ! PRINT "ctrl1("&VAL$(I),") x,y=".Ctrlx1(I),Ctrly1(I)
7750 ! PRINT "ctrl2("&VAL$(I),") x,y=".Ctrlx2(I),Ctrly2(I)
7760 ! PRINT "ctrl3("&VAL$(I),") x,y=".Ctrlx3(I),Ctrly3(I)
7770 ! PRINT "ctrl4("&VAL$(I),") x,y=".Ctrlx4(I),Ctrly4(I)
7780 ! PRINT "  
7790 ! NEXT I
7800 ! PRINT "***********"
7810 !
7820 PRINTER IS 1
7830 !
7 0 END
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-84</td>
<td>-377</td>
</tr>
<tr>
<td>-149</td>
<td>-377</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1144</td>
<td>1144</td>
</tr>
<tr>
<td>-222</td>
<td>-222</td>
</tr>
<tr>
<td>-222</td>
<td>-222</td>
</tr>
<tr>
<td>-204</td>
<td>-104</td>
</tr>
<tr>
<td>596</td>
<td>1880</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1144</td>
<td>1144</td>
</tr>
<tr>
<td>-222</td>
<td>-222</td>
</tr>
<tr>
<td>99</td>
<td>299</td>
</tr>
<tr>
<td>96</td>
<td>196</td>
</tr>
</tbody>
</table>

**Note:** The numbers in the table represent coordinates or values, but the context is not clear from the image alone.